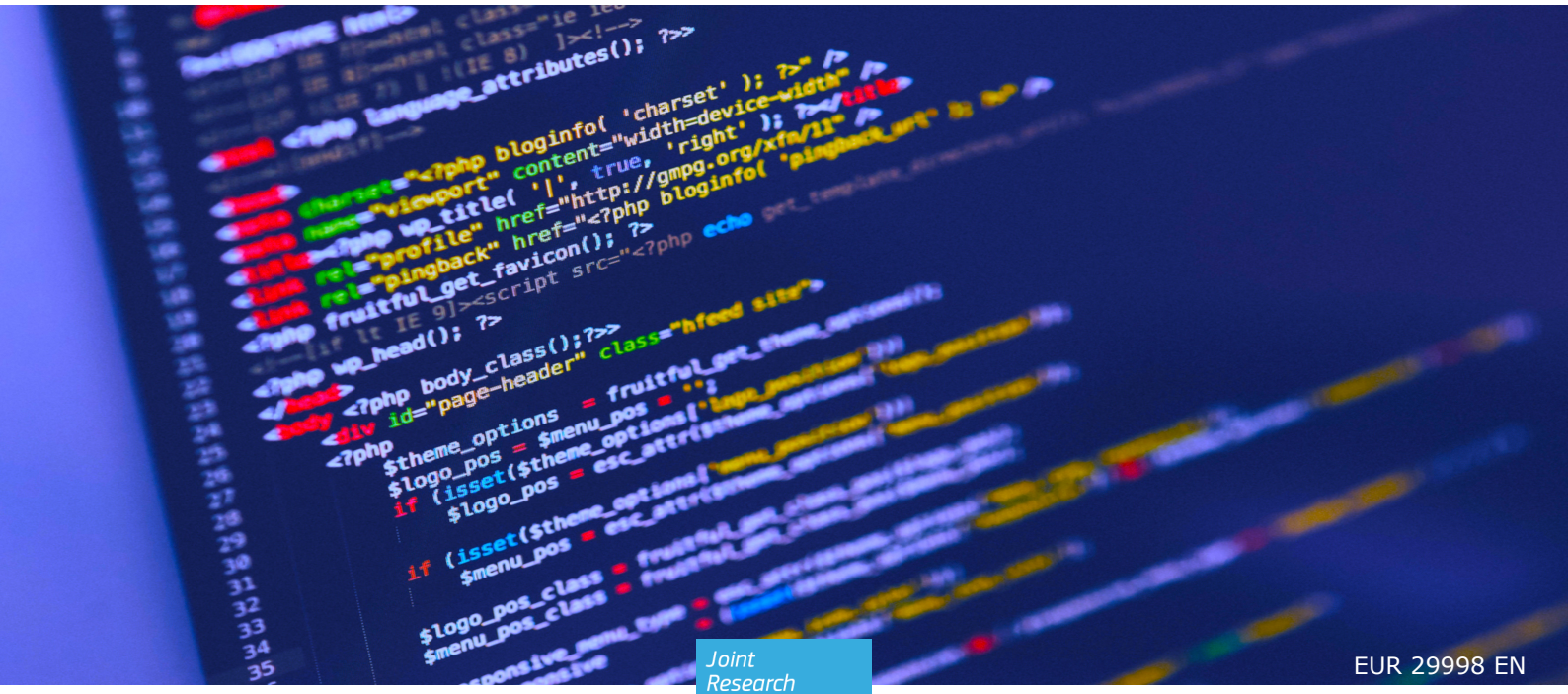
European Commission

JRC TECHNICAL REPORTS

A Distributed Approach to solve Power Flow problems in new emerging scenarios

Methodological aspects, implementation issues and applications

Rinaldo, S. G., Ceresoli, A., Prettico, G.



Joint Research Centre

EUR 29998 EN

Contact Information
Name: S. G. Rinaldo, A. Ceresoli, G. Prettico
Address: Joint Research Centre, Via Enrico Fermi 2749, TP 044, 21027 Ispra (VA), Italy
E-mail: giuseppe.prettico@ec.europa.eu

EU Science Hub
https://ec.europa.eu/jrc

JRC116302

EUR 29998 EN

PDF ISBN 978-92-76-11252-5 ISSN 1831-9424 doi:10.2760/869640

Luxembourg: Publications Office of the European Union, 2019

How to cite this report: Rinaldo, S., Ceresoli, A. and Prettico, G., A Distributed Approach to solve Power Flow problems in new emerging scenarios, EUR 29998 EN, Publications Office of the European Union, Luxembourg, 2019, ISBN 978-92-76-11252-5 (online), doi:10.2760/869640 (online), JRC116302.

A Distributed Approach to solve Power Flow problems in new emerging scenarios

Contents

Acknowledgements

We would like to thank all the colleagues of the C.3. unit - Energy Security, Distribution and Markets - for the support and sharing of precious scientific insights. We also thank Professor D.J.P. Lahaye of TU Delft and Professor M. Merlo of Politecnico of Milano for the trustworthy feedback and valuable advice in conducting this research work.

Authors
Stefano Guido Rinaldo, Andrea Ceresoli, Giuseppe Prettico

Abstract

Distributed Computing is growing up in interest in many applied fields of scientific research. At the same time power system operation is becoming increasingly complex due to the need to integrate a relevant amount of energy production coming from Distributed Energy Resources (DERs) connected at various voltage levels. In this context, the need to automate grid operations is in fact fundamental to ensure adequate levels of reliability, flexibility and cost effectiveness of power systems. This report provides methodological aspects and principles for the solution of the power flow problem through a distributed approach. The focus is on the case in which multiple interacting entities (*utilities*) by sharing a small portion of their grids data with the other parties can improve the solution of the global grid (composed by all the grids involved). The advantage is that the computation is done locally and in parallel with the others without the need to exchange further data. The aim of this report is both to give the reader a comprehensive overview of the software used for the implementation, the Portable Scientific Extensible Toolkit for Scientific Computation (PETSc), and to highlight the principles followed to build the Distributed Power Flow Solver as well as the specific features that diversify our approach from others. Finally, two case studies are presented as potential applications of our model: the capacity calculation between transmission system operators (TSOs), and the transmission-distribution networks coupling between TSOs and Distribution System Operators (DSOs). Open issues and future research studies are discussed in the final part of the present report.

1 Introduction

1.1 Broad Context

Power systems are one of the most complex engineering human-made works. Operating such a sophisticated system requires constant monitoring, high-automation level, continuous corrective actions and planning as close as possible to real-time. In recent years power systems have been facing an arising number of challenges set by the on-going *energy transition*. In Europe, this has been mainly driven by European Commission (EC) energy policies (Third Energy Package - TEP, 2009 and Clean Energy Package - CEP, 2018) that have set important goals for carbon emission reduction and market harmonisation[1] for the member states (MS).

In the past national grids were designed to serve the same scope: that of transmitting electricity produced in big centralised power plants to final consumers at different voltage levels. The power was used to flow solely from generating plants to final consumers passing through meshed transmission networks and radial distribution networks. All grid-related activities were fully under the responsibility of national Transmission System Operators (TSOs), while the distribution grid was built according to a fit-and-forget approach, over-staffed and with passive role. At market level, no competitive auctions were in place (e.g. through a power exchange pool) and all the stages of the electricity supply chain were managed by a single vertical integrated company. This plain and simple organisation of the electricity supply chain ensured an easy and reliable operation of the grid in the past, even though supposedly not in the most economically efficient manner. Moreover this approach proved not suitable to match the novel ways of consuming and producing electricity.

These reasons have led to a radical re-engineering of the power sector in the last couple of decades. The complete unbundling of generation electricity companies from transmission sector realised by the TEP resulted in the introduction of competitive market designs (the Power eXchanges - PXs) at wholesale level. In practical terms, this allows to auction the use of the grid together with the energy trade and reward the less costly generators with network access. This new paradigm for electricity markets from one hand increases market liquidity, lowers prices and provides a framework for more transparent trades. On the other hand, unbundling makes the operation of power systems more complex. Generating companies, system operators, market operators and suppliers must all coordinate with each other and continuously exchange information in order to ensure the security of supply and the efficient management of the power system as a whole.

At the same time in order to cope with a drastic reduction of the carbon emissions, power generation must shift from conventional fossil-based technologies to Renewable Energy Sources (RES). The power production from RES is typically dislocated over the grid and connected at different voltage levels, often at medium and low voltage levels (MV-LV). Furthermore, new actors are entering the competitive markets (e.g. prosumers) and new type of loads are challenging the stability of the system (e.g electric vehicles (EVs)). The intermittent and stochastic nature of such Distributed Energy Resources (DERs) is strongly influencing the grid operation, as the increasing uncertainty on flows prediction and real-time grid state, especially at distribution level, makes it difficult to ensure security of supply and efficient optimisation of resources. In other words, TSOs see volatile flows appearing on their system as a result of the real-time uncertain behaviour of DERs at distribution level and hence are moved to procure more often system services (e.g. balancing, voltage support) and called for corrective actions. At distribution level, investments in proper control systems and monitoring infrastructure will become necessary.

To summarise, future power systems have to withstand both the uncertainties and complexities introduced by DERs at operation level, providing a reliable and cost efficient service, while ensuring

[1] From this point of view, the final goal is to create a fully harmonised Internal Energy Market (IEM) in Europe, offering one single wholesale price for all member states.

the unbundling at different stages of electricity supply chain. In this context, where economic and operational challenges might look in anti-thesis, the integration of *smart* and *intelligent* devices as well as of innovative methodologies for power system analyses play a pivotal role.

1.2 Project Context

Distributed computing is applied in a great number of different research and industry contexts. The growing importance of distributed computing is mainly related to the advantages of de-centralised approaches, typically allowing to better exploit computational resources otherwise not used, to reduce investments, to keep data-locality and to be fault-tolerant. Among the aforementioned appealing features of distributed applications, the application presented in the following focuses on *input data-locality*. This report discusses the methodology to carry out a distributed solution of power flow equations and gives guidance on the chosen implementation based on the parallel computing software (PETSc). The framework under study is that of cross-border flow exchanges. In this context, distributed approaches might find application every time that coordination is needed among system operators managing different parts of a global inter-connected grid. The application aims at automating such a coordination, by only providing local data as input to the power flow simulation. On this purpose, we focus here on two situations: at member states level (thus among neighbouring European countries i.e. at the transmission level) and at national level (at the transmission-distribution interface). A brief description of the current state of the art for both levels is depicted in the following.

At the transmission level, investments in expanding cross-border capacities are one of the main topic of the recently published CEP[2]. This will enable more frequent exchanges of electricity and a more flexible operation between member states. For instance, in cases of unexpected surplus of electricity or in order to be able to cover shortfalls as a result of the unpredictability of RES (wind and solar) national TSOs might rely on neighbouring countries at need. The creation of a proper market design at European level that reflects such exchanges and allocate efficiently cross-border rights is thus needed. At national level cross-border exchanges at day-ahead (DAM) and intra-day (IDM) markets[3] were firstly organised so that capacity and energy were bid explicitly (i.e. in different markets) over different time-frames. However this is not economically efficient, since those agents that procure cross-border capacity ex-ante can exert market power on the spot markets, often causing the prices to increase. Moved by this reason, three European PXs (OMEL, NORDPOOL and EPEX Spot) launched in 2009 the Price Coupling of Regions (PCRs) project, an initiative to promote coordinated (implicit) price formation on their respective spot markets. In other words, grid constraints are modelled into the price formation algorithm and, as the market clears, quantity and prices formed are those that maximise welfare subject to grid constraints. Nowadays, the PCR project involves eight PXs. The market is cleared by solving the market problem centrally, through the EUPHEMIA algorithm.[4] To clear the DA problem, EUPHEMIA needs a set of pre-calculated inputs. TSOs closely coordinate with each other and with Nominated Electricity Market Operators (NEMOs) in order to model the grid constraints and find compatible market equilibrium. The procedure to model cross-border interconnections takes place through a complex and long procedure made up of several steps that conceptually could be simplified into two main ones: 1) a Capacity Calculation (CC)

[2]When we refer to the CEP we refer mainly to the Electricity Regulation (Ele, 2019b) and to the Electricity Directive (Ele, 2019a)

[3]From now on we refer to DAM and IDM as spot markets. According to finance terminology, spot markets are those markets that involve the trading of financial instruments for immediate delivery, in contrast with forward markets, where trade refers to future delivery. Although DAM and IDM are not strictly spot markets, in the electricity sector they are commonly referred to as such.

[4]EUPHEMIA stands for Pan-European Hybrid Electricity Market Integration Algorithm. EUPHEMIA clears the day-ahead market over Europe for more than 50 bidding zones, allocating cross-border transmission capacity for more than 60 inter-connectors in less than 10 minutes.

procedure requiring TSOs coordination; and 2) a Capacity Allocation (CA) mechanism requiring a common market clearing algorithm. During these steps system operators run several times power flow computations, in order to calculate parameters, to define physical limits for electrical components,to foresee contingencies, etc. In this work we focus on step 1, even though distributed approaches can in principle be thought also for step 2.

At national level the framework is considerable different. Even though long-run investments are increasingly being planned, distribution grids' observability and controllability are still very limited (Prettico et al., 2019). The uncertainty of RES production connected at MV-LV levels thus translate into volatile flows at transmission level. Nowadays, TSO and DSOs exchange information about the state of their grids periodically (even though this periodicity differs considerably among MS), though not carrying out coordinated calculations. As a result, TSOs need to estimate the uncertainty on generation and demand pattern down to lower voltage levels and procure balancing reserves ex-ante to account for worst-case scenarios. From the DSOs side, even though the grid is over-staffed, congestions may occur caused by the increasing penetration of RES. In this context we foresee that soon DSOs will collect more detailed and fine grained information and will run power flow equations to improve their knowledge of the actual state of their grids. The inherent uncertainty from DSO to TSO might be mitigated by allowing a coordinated calculation framework for TSO and DSOs that provide data-locality and hence do not prejudice competition.

1.3 Project Scope

This report discusses methodologies, issues and applications of an experimental distributed power flow solver. The aim is to carry out a distributed computation of the load flow problem between two separated parties (bipartite case) sharing a grid boundary. In other words, the goal is to provide two grid operators (TSO-DSO or TSO-TSO) with an application capable of performing exact load flow computations on a joint network by exchanging only a limited piece of information (for instance that at the interfacing nodes between the two networks). This is in line with unbundling policy indications and operational issues briefly presented in 1.1 and 1.2. The application framework investigated is thus that one of cross-border exchanges, among national EU countries, through transmission-distribution interface, or in general among adjoining grid operators. The implementation takes advantage of the Portable Extensible Toolkit for Scientific Computation (PETSc), an API (Application Programming Interface) for writing parallel scientific applications based on the C programming language. For the sake of completeness, we chose to dedicate a whole chapter to the understanding of PETSc functionalities (Section 3), in order to provide the reader with an overview of the software and to build up increasing layers of complexities step-by-step.

1.4 Organisation

This report may be conceptually broken down into four parts. The first part reviews the power flow problem and solvers. Then, the mathematical methodology to compute the power flow equations remotely is introduced. The second part introduces the reader to the PETSc API, providing its general functioning. In the third part, based on the concepts introduced in Section 2 and Section 3, we explain the Distributed Power Flow. Finally, the last part shows two proposed applications frameworks (transmission-transmission interface, transmission-distribution interface).

A more detailed overview of the report organisation is provided in the following.

Section 2 starts from recalling power system basics and the power flow problem. Later, we gradually introduce more complex solvers for the PF equations, reviewing pros and drawbacks. Linear solvers and parallel implementation become the central topic of discussion during the second part of this section. A short section is dedicated to parallel communication analysis of stationary block preconditioned solvers. In the final part, we provide an example of the distributed methodology on case13.m by using the Newton-Krylov-Schwarz (NKS) approach.

Section 3 introduces to the use of the Portable Extensible Toolkit for Scientific Computation (PETSc). This chapter provides a quick hand-guide to the PETSc API and discuss its relevance in this work.

Section 4 discusses the implementation of the Distributed Power Flow in PETSc.

Section 5 provides an in-depth review of two possible fields of application for the distributed methodology: 1) In the context of congestion management at European level, and specifically on the capacity calculation process for transmission-transmission flow control; 2) In the context of transmission-distribution network coupling, touching upon the TSO-DSO interaction.

Section 6 draws conclusions and possible future works on power flow distributed solvers.

2 Power Flow, Numerical Methods & Distributed Power Flow Methodology

In this chapter the relevant electrical variables, the formulation of the power problem (or load flow) and the main numerical methods that are used in the current transmission system operator practice are presented. This background information will turn out useful in the next chapters when the distributed power flow methodology is presented. It is worth mentioning that the aim of this section is not to carry out an extensive review of the mathematical methods used to solve power flow equations, but rather to provide a concise background to the reader willing to grasp the more complex features of the distributed methodology.

2.1 Electrical Variables, Grid Components, Bus Types

In this section we introduce the relevant electrical variables for the formulation of the power flow problem and the classification of buses, in line with regular TSOs practices. In AC operation, electricity is an oscillating flow of electrons at a certain constant frequency[5]. As a result, current, voltage and complex power are time-variant quantities that might be difficult to deal with when writing down equations for grid modelling. Furthermore, in steady-state analysis there is no interest in knowing the instant values of those variables, but rather to assess some difference among them (e.g. nodal voltage difference, current-voltage difference, etc.). For these reasons, for steady-state grid analysis, electrical variables are defined through *phasors*. Phasors are rotating vectors in the complex plane that are used to represent time variant oscillating electrical quantities. Phasors allow to split the information needed to assess an electrical variable into just two components, the *phase* and the *module*. Thus, let us define the following parameters:

The current phasor: $\overline{I} = Ie^{j\delta_I} = I(cos\delta_I + jsin\delta_I)$

The voltage phasor: $\overline{V} = Ve^{j\delta_V} = V(cos\delta_V + jsin\delta_V)$

The complex power phasor: $\overline{S} = Se^{j\vartheta} = S(cos\theta + jsin\theta)$

From now on we refer to V as the *voltage magnitude* and to δ_V as the *voltage angle*. Note that the complex power S is related to the active power P and reactive power Q by the following expression:

$$S = P + jQ \tag{1}$$

Electrical grids may be modeled in many ways. From the mathematical point of view an electric grid can be interpreted as a graph $\mathcal{G}(\mathcal{B}, \mathcal{E})$, where $\mathcal{B}$ and $\mathcal{E}$ denote the set of buses and edges respectively and $\mathcal{G}$ is a mapping between $\mathcal{B}$ and $\mathcal{E}$ providing the connectivity bus-edges[6]. From an engineering and operating side, a power grid is an interconnected network of components that aim at moving electricity from generators to final consumers. Buses act as sinks, injecting or withdrawing power to/from the grid, while edges (i.e. transmission/distribution lines) allow for transportation. As such, electric variables may be branch related or bus related. In this work, we refer to bus variables with capital letters and to branch variables with lowercase letters. Buses can be categorised into three types: those that provide net injections to the grid (PV buses); those that make net withdrawals from the grid (PQ buses); those that have flexible operation and thus compensate for losses (called Slack buses) (see Fig. 1).

[5] In Europe the oscillating frequency is 50Hz.

[6] PETSc extensively makes use of graph theory to store and manage networks. This is shortly deepened in Section 4 and contextualized in power flow.

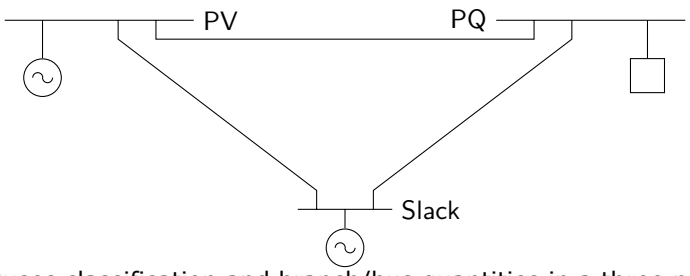


Figure 1: Buses classification and branch/bus quantities in a three node system.

Buses classification into PQ, PV and Slacks relates to the fact that these electrical quantities are known by system operators. Other quantities, like branch currents, power flows over lines, losses and state variables (V,δ) are not quantities known in advance, but rather derived from the information given at PQ, PV and Slacks. This sets the basis for the power flow problem. Summing up, the unknowns and known variables for the power flow problem for each type of bus are:

Bus Type	Known	Unknown
PQ	P,Q	V,$\boldsymbol{\delta}$
PV	P,V	Q,$\boldsymbol{\delta}$
Slack	V,$\boldsymbol{\delta}$	P,Q

Before entering into the power flow problem definition, let us clarify the role of Slack buses. The definition of slack bus serves two functions. First, it sets a reference (1 p.u.[7] for V and zero for $\boldsymbol{\delta}$), so that voltage angles and voltage magnitudes can be computed as deviations. Its second purpose is to balance power losses. In fact, losses are defined only once the power flows over branches are computed, which means only after the load flow problem is solved.[8]

2.2 The Power Flow Problem

Power flow equations are solved continuously over time by grid operators, in order to assess the security state of the grid, assess contingencies, plan long-run investments, ensure the best cost-efficient allocation of resources. This is a fundamental tool for grid operators and research in this field is very active, both from the numerical and methodological side.

In this section we present its mathematical formulation. The power flow problem is the problem of finding the voltage phasor (i.e. voltage magnitude and voltage angle) for all buses. Currents, power flows and losses are dependent variables and can hence easily derived once the load flow is solved.

Let us consider a power network made up of N buses, with one single slack bus. Let us also consider that of the N buses, G, with $G < N$, is the number of PV buses and that the remaining $N - G - 1$ are the number of PQ buses. PQ buses have both the voltage magnitude and the voltage angle unknown, that means PQ buses introduce $2(N - G - 1)$ unknowns. On the other hand, PV buses have only voltage angles which are unknown, introducing then only G unknowns. The total number of unknowns is thus:

$$2(N - G - 1) + G = 2(N - 1) - G \tag{2}$$

[7] p.u. stands for "per unit". In power systems analysis, a per-unit system is the expression of system quantities as fractions of some reference unit quantity. It's usual to take as base quantities a complex power reference and a voltage reference, from which other references are derived, e.g. current reference. Per unit system allows to greatly simplify calculations.

[8] Basically, at least an active power at a bus must be set free to accommodate differences in the active power balance due to losses. In other words, the slack bus is a need to keep the energy balance verified.

The same number of equations (not introducing other unknowns) is needed in order to obtain a unique solution for the mentioned problem. In principle, there's no constraining rule upon the choice of these equations. The most common choice is to write nodal power mismatch equations: we can write the active power balance equation for each PQ and PV buses, for a total of $N-1$ equations. Then we can write a reactive power balance equation for each PQ bus (i.e. $N-G-1$), getting a total number of equations of:

$$N-G-1+N-1=2(N-1)-G \tag{3}$$

Which is equal to the number of unknowns. Let us write them down explicitly for the generic bus i. This is done starting from the net complex power expression injected or withdrawn from bus i:

$$\overline{\mathrm{S}}_i=\overline{\mathrm{V}}_i\underline{\mathrm{I}}_i \tag{4}$$

Where $\overline{\mathrm{V}}_i, \underline{\mathrm{I}}_i$ are the bus voltage and bus current related to the bus i. Notice that the underline of I stands for the complex conjugate operator. The nodal current $\underline{\mathrm{I}}$ may be expressed in terms of incident branch currents by introducing the Kirchhoff's Current Law (KCL). The KCL states that the charge flow cannot be generated, consumed or collected in a bus, so that the current summation at a node must be zero. In other terms, we can express $\underline{\mathrm{I}}$ as:

$$\underline{\mathrm{I}}=\sum_{j=1}^{m}\underline{\mathrm{i}} \tag{5}$$

Where m is the number of incident buses at bus i. By introducing Eq.(5) into Eq.(4) we obtain:

$$\overline{\mathrm{S}}_i=\overline{\mathrm{V}}_i\sum_{j=1}^{m}\underline{\mathrm{i}}_j \tag{6}$$

The currents i_j are unknowns. Introducing the Ohm's Law we can express them in terms of complex voltages obtaining:

$$\overline{\mathrm{S}}_i=\overline{\mathrm{V}}_i\sum_{j=1}^{N}\underline{Y_{ij}V_j} \tag{7}$$

Where Y_{ij} is the ij element of the nodal admittance matrix (see Annex 1). According to the complex expressions introduced in 2.1, the phasors can be expanded as followss:

$$\overline{\mathrm{S}}_i=\sum_{j=1}^{N}V_iV_jY_{ij}e^{j(\delta_i-\delta_j-\theta_{ij})} \tag{8}$$

Being Eq.(8) a complex equation, it can be separated into its scalar components:

$$P_i=\sum_{j=1}^{N}V_iV_jY_{ij}\cos\left(\delta_i-\delta_j-\theta_{ij}\right) \tag{9}$$

$$Q_i=\sum_{j=1}^{N}V_iV_jY_{ij}\sin\left(\delta_i-\delta_j-\theta_{ij}\right) \tag{10}$$

These are the so-called *power flow equations*. Note that we are dealing with nodal quantities, that means, that the δ_i, δ_j are voltage angles related to bus i and j. Whilst θ_{ij} refers to the phase shift given by line parameters connecting bus i with bus j.

The power flow equations relate grid state variables with net injection/withdrawals P_i, Q_i. Since in reality, grid operators know the scheduled generation profiles and the demand forecasts, it can be said that P_i, Q_i are known inputs for their power flow computations. Additionally, the grid operators own all the data about components connected to the grid, e.g. switches state, lines parameters, bus-edges connectivity (i.e. topology). In other words, the only unknowns are the state variables V, δ.

Before introducing the solution methods, let us derive an alternative but equivalent formulation of the power flow equations. Let us consider Eq.(7) and separate the admittance matrix Y_{ij} in its real and imaginary components G_{ij}, B_{ij}. This translates into:

$$\overline{\mathrm{S}}_i = \overline{\mathrm{V}}_i \underline{\mathrm{I}}_i = \overline{\mathrm{V}}_i \sum_{j=1}^{N} \underline{Y_{ij} V_j} = \sum_{j=1}^{N} V_i V_j (G_{ij} - jB_{ij})(\cos\delta_{ij} + j\sin\delta_{ij}) \tag{11}$$

And by separating active and reactive power components:

$$\begin{aligned} -P_i + \sum_{j=1}^{N} V_i V_j (G_{ij}\cos\delta_{ij} + B_{ij}\sin\delta_{ij}) = 0 \\ -Q_i + \sum_{j=1}^{N} V_i V_j (G_{ij}\sin\delta_{ij} - B_{ij}\cos\delta_{ij}) = 0 \end{aligned} \tag{12}$$

This alternative formulation is the one used in the distributed solver discussed in this work. The power flow problem is thus a system of non-linear algebraic equations. In other words, Eq.(12) belongs to the general class of problems:

$$\mathbf{F}(\mathbf{x}) = 0 \qquad \text{with} \qquad \mathbf{x} = \begin{bmatrix} \mathbf{V} \\ \boldsymbol{\delta} \end{bmatrix} = \begin{bmatrix} \mathbf{V_1} \\ \vdots \\ \mathbf{V_{PQ}} \\ \boldsymbol{\delta_1} \\ \vdots \\ \boldsymbol{\delta_{PQ+PV}} \end{bmatrix} \tag{13}$$

Practically speaking this means to look for the roots vector **x** of the non-linear vector-valued function **F**. Note that all *bold* quantities refer to vectors. Finding the roots of an algebraic nonlinear system of equations is not an easy task. Exact (i.e. analytical) solutions exist only for special cases. For all the other cases since this is not possible the solution is investigated through numerical methods. The next sections introduce the numerical approaches used to solve the power flow problem. This has a double aim. From one side, to review the reference methods in the industry practice, from the other side, to establish a basis for the mathematical approach here proposed to carry out the distributed power flow computation.

2.3 Non-Linear Solvers for Power Flow Equations

All over the world industry practice, power flow equations are solved by means of a quite standard approach. An iteration procedure is started to approximate the non-linear problem to a linear one. Then, the arising linear system is further simplified through a factorisation technique and solved in one single step. The procedure restarts as a stopping criteria is fulfilled. This approach combines a

non-linear Newton step with a direct single-step method, giving a good balance between robustness and computational time, which motivates the long-time reference of the industry practice[9].

Research in numerical solution of power flow equations has in recent years explored outer-inner iterative methods as well. This approach combines a Newton iteration and an iterative linear procedure. These methods very often offer good parallel features, better scalability and possibility to be run on distributed platforms.

In the following sections we start by reviewing the Newton's method, moving then to linear solvers, and later to the introduction of a particular numerical approach - the Newton-Krylov-Schwarz algorithm - available in the PETSc libraries and used for a distributed computation of power flow equations.

2.3.1 Newton-Raphson

The Newton-Raphson (NR) method is a standard approach for many classes of non-linear problems. The NR iterative procedure is quite straightforward: given a starting point for the iteration, the problem is approximated locally to a linear problem, solved, and iterated back. The NR iteration scheme can be expressed in general as:

$$x^{(k+1)} = x^{(k)} + \Delta x^{(k)} \tag{14}$$

Where the term $\Delta x^{(k)}$ works as a "corrective" term at each step k. Hence, the convergence will depend from $\Delta x^{(k)}$.

Solving Monodimensional Case

For sake of clarity, let us first derive the NR method for a single algebraic non-linear equation. We want to find the zero of a function $f(x) : R \to R$ in the single variable x, that means to find x such that:

$$f(x) = 0 \tag{15}$$

Figure 2 shows the idea behind the method. The $x^{(0)}$ is the starting point for the iteration scheme. Given $x^{(0)}$ as the starting guess[10], $f(x)$ is approximated locally through a first-order Taylor expansion around the point $(x^{(0)})$ according to:

$$f(x) \approx f(x^{(0)}) + f'(x^{(0)})\Delta x^{(0)} = 0 \tag{16}$$

With:

$$\Delta x^{(0)} = x^{(1)} - x^{(0)} \tag{17}$$

Eq.(16) above is a linear equation in the $\Delta x^{(0)}$ unknown. By reordering it with respect to $\Delta x^{(0)}$, we obtain:

$$\Delta x^{(0)} = -\frac{f(x^{(0)})}{f'(x^{(0)})} \tag{18}$$

[9] Notice that robustness is an essential feature for a power flow solver. Grid operators face short time-frames and security challenges continuously over time so that software reliability must be ensured for a broad range of input sets.

[10] In case an approximated solution of the nonlinear equation is known, it shall be used as first guess. This decreases greatly the number of iterations needed to get the solution.

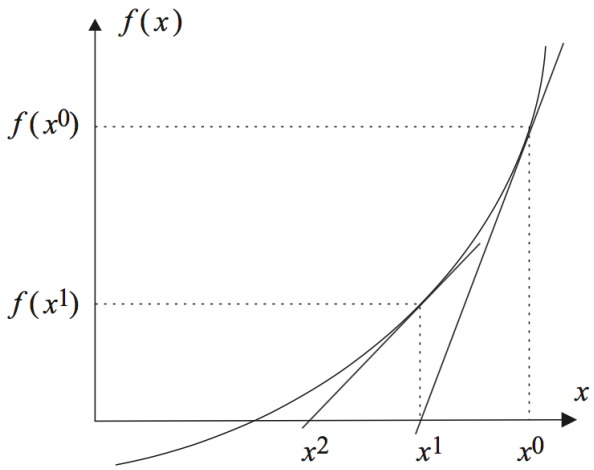


Figure 2: Newton-Raphson method for one-dimensional case

So that the next root $x^{(1)}$ is found as:

$$x^{(1)} = x^{(0)} - \frac{f(x^{(0)})}{f'(x^{(0)})} \tag{19}$$

Generalising at iteration k, Eq.(19) becomes:

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})} \tag{20}$$

Note that, each $f'(x^{(k)})$ must never be equal to zero (i.e. no stationary points have to be encountered) in order to get to convergence.

Solving Multidimensional Cases

In the following , we extend the concept to a system of equations. A system of non-linear equations is defined as a system of generic equations n in m unknowns, where at least one equation is non-linear. In this work, we consider only well-posed problems where the number of equations equal the number of unknowns (i.e. $n = m$) and all equations are non-linear. Furthermore, we consider the *dense* case, where each equation shows dependence on any unknown x_i. We will introduce the sparsity later on, where the Newton methodology will be applied to the power flow problem.

A system of non-linear equations is defined as:

$$\begin{cases} f_1(x_1, \ldots, x_n) = 0 \\ f_2(x_1, \ldots, x_n) = 0 \\ \ldots \quad \ldots \quad \ldots \quad \ldots \\ f_n(x_1, \ldots, x_n) = 0 \end{cases}$$

More concisely:

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \\ f_n(\mathbf{x}) \end{bmatrix} \quad \text{with} \quad \mathbf{x} = (x_1, \ldots, x_n)$$

As the vector-valued function. The solution of a system of non-linear equations can be thus interpreted as:

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}$$

Which means once again to find the zero vector of the non-linear vectorial function **f**. This is the same form obtained in the mono-dimensional case, hence, the same procedure seen in the previous paragraph can be applied, taking into account that in this case scalar variables are substituted by vectors [11]. Using the same approach of the mono-dimensional case, we get:

$$\mathbf{f(x)} = \begin{bmatrix} f_1(\mathbf{x}^{(k+1)}) \\ f_2(\mathbf{x}^{(k+1)}) \\ \vdots \\ f_n(\mathbf{x}^{(k+1)}) \end{bmatrix} \approx \begin{bmatrix} f_1(\mathbf{x}^{(k)}) + \boldsymbol{\nabla} f_1(\mathbf{x}^{(k)}) \cdot \Delta\mathbf{x}^{(k)} = 0 \\ f_2(\mathbf{x}^{(k)}) + \boldsymbol{\nabla} f_2(\mathbf{x}^{(k)}) \cdot \Delta\mathbf{x}^{(k)} = 0 \\ \vdots \quad \vdots \quad \vdots \\ f_n(\mathbf{x}^{(k)}) + \boldsymbol{\nabla} f_n(\mathbf{x}^{(k)}) \cdot \Delta\mathbf{x}^{(k)} = 0 \end{bmatrix} \tag{21}$$

And each equation is solved with respect to $\Delta\mathbf{x}^{(k)}$, whereas $f_i(\mathbf{x}^{(k)})$ and $\boldsymbol{\nabla} f_i(\mathbf{x}^{(k)})$ (with $i = 1, \ldots, n$) are evaluated by means of the solution of the previous iteration.

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \frac{\mathbf{f}(\mathbf{x}^{(k)})}{\nabla \mathbf{f}(\mathbf{x}^{(k)})} \tag{22}$$

Exactly as for the mono-dimensional case. Note that the operator ∇ is the gradient operator. This means that the term $\nabla \mathbf{f}(\mathbf{x}^{(k)})$ is actually a matrix defined as:

$$J = \begin{bmatrix} \boldsymbol{\nabla} f_1(\mathbf{x}) \\ \boldsymbol{\nabla} f_2(\mathbf{x}) \\ \vdots \\ \boldsymbol{\nabla} f_n(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & & \frac{\partial f_2}{\partial x_n} \\ \vdots & & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \tag{23}$$

The coefficient matrix J is called Jacobian matrix and it is the matrix of all first order partial derivatives. Therefore, the solution of equations contained in Eq.(21) with respect to $\Delta\mathbf{x}^{(k)}$ implies to solve the linear system:

$$-J(\mathbf{x}^{(k)})\Delta\mathbf{x}^{(k)} = \mathbf{f}(\mathbf{x}^{(k)}) \tag{24}$$

Note the important result obtained: the non linear problem has been turned into a succession of k linear systems.

In principle, the solution of Eq.(24) can be approached analytically (if the Jacobian matrix is invertible, i.e. $det(J) \neq 0$). By inverting J the exact solution **x** is found. In practice, this becomes soon unfeasible even for relatively small linear systems (i.e. $> 10^1$ unknowns). Let n be the order of J, calculating its inverse requires $\mathcal{O}(n!)$ FLOPS (Floating Point Operation Per Second). Table 1 shows how the computational complexity of the problem scales up quickly by solving linear systems of increasing size, and its relation with computational time.

[11] Notice that the operator $< \cdot >$ is the *dot-product*.

Table 1: Time required to solve a linear system of dimension n through Cramer's rule. "o.o.r." stands for "out of reach". (Alfio Quarteroni, 2012)

	FLOPs		
n	$\mathbf{10^9}$	$\mathbf{10^{12}}$	$\mathbf{10^{15}}$
10	10^{-1}	10^{-4} sec	negligible
15	17 hours	1 min	0.610^{-1} sec
20	4860 years	4.86 years	1.7 days
25	o.o.r	o.o.r.	38365 years

Linear systems arise pretty much in any scientific discipline and sometimes they may contain millions or even billions of unknowns, making these problems unsolvable with Cramer's rule. This has attracted considerable attention on research methods to solve linear systems. Many techniques have been developed over the years depending on the features of the problem (e.g. sparsity, spectrum, positive definiteness, symmetry, etc.), that can be divided into two main categories: *direct methods*, which are those that find a solution of the linear system after a finite number of steps, and *iterative methods*, that might require a theoretical infinite number of steps.

Before going through linear systems techniques, let us apply the NR method to the power flow problem, in order to turn the problem into a sequence of linear systems.

Newton-Raphson & Power Flow Problem

Let us recall the power flow equations from section Section 2.2:

$$P_i = \sum_{j=1}^{N} V_i V_j (G_{ij} \cos \delta_{ij} + B_{ij} \sin \delta_{ij}) \tag{25}$$

$$Q_i = \sum_{j=1}^{N} V_i V_j (G_{ij} \sin \delta_{ij} - B_{ij} \cos \delta_{ij}) \tag{26}$$

The first action is to split the unknown terms from those that are known, according to power flow problem definition given in Section 2.1. That means we can write Eq.(25) as:

$$\begin{cases} P_i - P_{i,comp}(\mathbf{x}) & = 0 \\ Q_i - Q_{i,comp}(\mathbf{x}) & = 0 \end{cases} \tag{27}$$

Where $P_{i,comp}, Q_{i,comp}$ contain the unknown vector $\mathbf{x}$. Let define the *Power Mismatch Function* (PMF) as:

$$\mathbf{F(x)} = \begin{bmatrix} \mathbf{P} - \mathbf{P}(\mathbf{x}) \\ \mathbf{Q} - \mathbf{Q}(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} P_1 - P_{1,comp}(\mathbf{x}) \\ \vdots \\ P_{N_{PQ}+N_{PV}} - P_{N_{PQ}+N_{PV},comp}(\mathbf{x}) \\ Q_1 - Q_{1,comp}(\mathbf{x}) \\ \vdots \\ Q_{N_{PQ}} - Q_{N_{PQ},comp}(\mathbf{x}) \end{bmatrix} = \mathbf{0} \tag{28}$$

The PMF represents the system of non-linear equations in a compact form. From the PMF side, the power flow problem consists in determining the vector of state variables $\mathbf{x}$ so that the net injections/withdrawals P_i, Q_i equal those computed from state variables.

As mentioned in the previous section, the NR procedure starts with the linearisation of the initial problem. This gives:

$$\mathbf{F(x)} = \begin{bmatrix} P_1 - P_1(\mathbf{x}^{(k+1)}) \\ P_2 - P_2(\mathbf{x}^{(k+1)}) \\ \vdots \\ P_n - P_n(\mathbf{x}^{(k+1)}) \\ Q_1 - Q_1(\mathbf{x}^{(k+1)}) \\ Q_2 - Q_2(\mathbf{x}^{(k+1)}) \\ \vdots \\ Q_m - Q_m(\mathbf{x}^{(k+1)}) \end{bmatrix} \approx \begin{bmatrix} P_1 - P_1(\mathbf{x}^{(k)}) + \boldsymbol{\nabla} P_1(\mathbf{x}^{(k)}) \cdot \Delta\mathbf{x}^{(k)} = 0 \\ P_2 - P_2(\mathbf{x}^{(k)}) + \boldsymbol{\nabla} P_2(\mathbf{x}^{(k)}) \cdot \Delta\mathbf{x}^{(k)} = 0 \\ \vdots \quad \vdots \quad \vdots \\ P_n - P_n(\mathbf{x}^{(k)}) + \boldsymbol{\nabla} P_n(\mathbf{x}^{(k)}) \cdot \Delta\mathbf{x}^{(k)} = 0 \\ Q_1 - Q_1(\mathbf{x}^{(k)}) + \boldsymbol{\nabla} Q_1(\mathbf{x}^{(k)}) \cdot \Delta\mathbf{x}^{(k)} = 0 \\ Q_2 - Q_2(\mathbf{x}^{(k)}) + \boldsymbol{\nabla} Q_2(\mathbf{x}^{(k)}) \cdot \Delta\mathbf{x}^{(k)} = 0 \\ \vdots \quad \vdots \quad \vdots \\ Q_m - Q_m(\mathbf{x}^{(k)}) + \boldsymbol{\nabla} Q_m(\mathbf{x}^{(k)}) \cdot \Delta\mathbf{x}^{(k)} = 0 \end{bmatrix} \tag{29}$$

With $n = N_{PQ} + N_{PV}$ and $m = N_{PQ}$. Exactly as before, the gradient terms are all vectors, so that they can be gathered into a matrix of dimension $(n+m)$x$(n+m)$. We call this matrix the Jacobian matrix of the power flow problem J_{PF}. Reordering, the non-linear problem is turned into a sequence of linear systems k as:

$$-J(\mathbf{x}^{(k)})\Delta\mathbf{x}^{(k)} = \mathbf{F}(\mathbf{x}^{(k)}) \tag{30}$$

With the unknown vector defined as:

$$\mathbf{x} = \begin{bmatrix} \mathbf{V} \\ \boldsymbol{\delta} \end{bmatrix} = \begin{bmatrix} V_1 \\ \vdots \\ V_{N_{PQ}} \\ \delta_1 \\ \vdots \\ \delta_{N_{PQ}+N_{PV}} \end{bmatrix} \tag{31}$$

Note that J_{PF} has a block structure. By dividing the derivatives of $\mathbf{P}, \mathbf{Q}$ with respect $\mathbf{V}, \boldsymbol{\delta}$, we recognise a structure made up of *four* blocks:

$$J_{PF} = \begin{bmatrix} \dfrac{\partial \mathbf{P}(\mathbf{x})}{\partial \boldsymbol{\delta}} & \dfrac{\partial \mathbf{P}(\mathbf{x})}{\partial \mathbf{V}} \\ \dfrac{\partial \mathbf{Q}(\mathbf{x})}{\partial \boldsymbol{\delta}} & \dfrac{\partial \mathbf{Q}(\mathbf{x})}{\partial \mathbf{V}} \end{bmatrix} \tag{32}$$

For sake of simplicity from now on we refer to J_{PF} as J. An example of the Jacobian structure is reported in Fig. 3 (case13), to underline its block structure. This case has been created by merging case9 and case5 of Matpower libraries (Zimmerman and Murillo-Sanchez, 2016).

It is worth reporting that most power flow software set up the *reduced* Jacobian, meaning that rows related to slack and PV reactive power equations are not included. For this reason, the Jacobian for case13.m has size 19x19, rather than 26x26. From now on we will use the term *reduced Jacobian* to indicate the first case and the term *full Jacobian* to indicate the latter.

Note that the Jacobian structure might differ from case to case, depending on the implementation. PETSc follows a different rational indeed. The derivatives ordering follows a bus ordering. As a

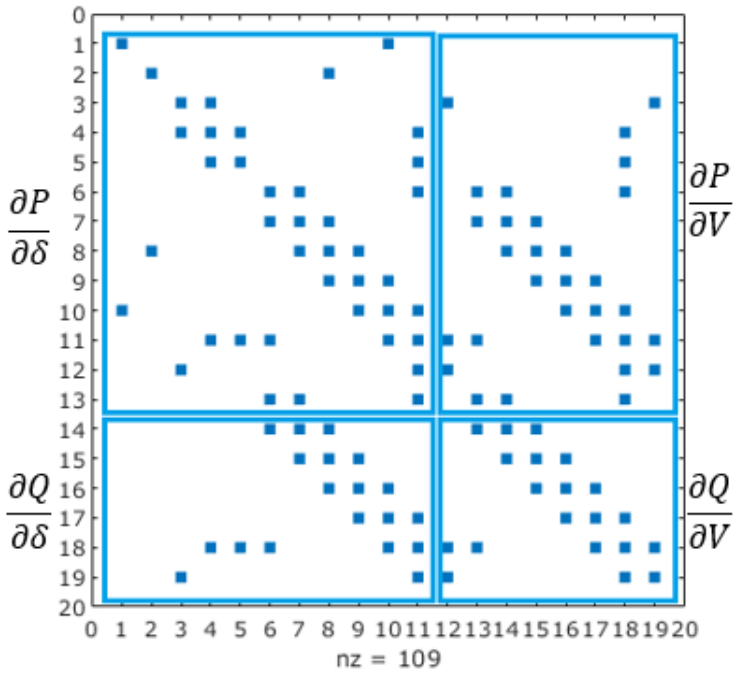
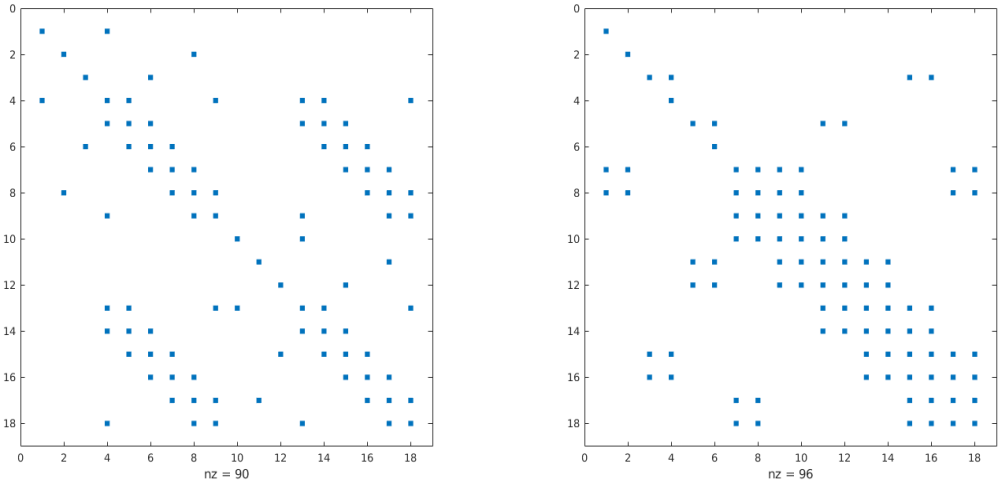


Figure 3: Reduced Jacobian of case13.m according to Matpower (Zimmerman and Murillo-Sanchez, 2016) block-ordering.

result the Jacobian resembles more the structure of Y_{bus}, the bus admittance matrix. Each entry is actually a 2x2 matrices, containing all the four derivatives, with respect to a generic bus i and j. A comparison between the full Jacobian of case9.m in Matpower and PETSc is reported in Fig. 4.



(a) Matpower **(b)** PETSc

Figure 4: Comparison between Matpower and PETSc full Jacobians of case9.

In conclusion, the NR method involves two steps:

1. Solve $J(\mathbf{x}^{(k)})\Delta\mathbf{x}^{(k)} = -F(\mathbf{x}^{(k)})$
2. Update $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)}$

Note that when using an iterative linear solver, step 1) of the procedure should be better called: 'solve approximately $J(\mathbf{x}^{(k)})\Delta\mathbf{x}^{(k)} = -F(\mathbf{x}^{(k)})$' due to the fact that iterative solvers solve the Jacobian linear system up to a given a tolerance. This class of methods are called *Inexact-Newton* methods and are the main matter of this work. In the next sections aspects regarding linear solvers will be dealt in detail. Note that proper tolerances shall be used for the linear (Krylov-Schwarz) iterations: in fact, while an excessive rough solution could lead to an increase of non linear iterations, an excessive precise solution would lead to a waste of computational resources.

2.3.2 Newton-Raphson with Line Search

The NR method presented in the previous sections is a basic but effective approach to solve non-linear problems. The NR converges quadratically, which is good, though may not be robust enough for real applications. In commercial applications, the NR method is implemented with some modifications to improve convergence properties. The two main techniques are that of the *trust-regions* and the *line-search* technique, more common in the context of power flow equations. Here we briefly recall the idea.

Let consider the non-linear update scheme derived in the previous section:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)}, \quad -J(\mathbf{x}^{(k)})\Delta\mathbf{x}^{(k)} = \mathbf{F}(\mathbf{x}^{(k)}) \tag{33}$$

In this expression the updating vector $\Delta\mathbf{x}^{(k)}$ can be interpreted as a correction vector, giving the new direction $\mathbf{x}^{(k+1)}$. In principle we expect $\Delta\mathbf{x}^{(k)}$ to be decreasing over the non-linear iterations, reaching out the zero value when the exact (up to an accepted tolerance) solution is found. This could not be the case for any iteration k and the procedure may even diverge. The line-search makes the NR procedure converging for any starting guess $\mathbf{x}^{(0)}$. When this happens we say NR is globally convergent[12]. The idea of line-search is quite simple. We introduce a parameter σ in the updating scheme according to:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \sigma^{(k)}\Delta\mathbf{x}^{(k)} \tag{34}$$

And set $\sigma^{(k)}$ as the term which minimises:

$$arg\min_{\sigma} \left\| F(\mathbf{x}^{k} - \sigma^{(k)}\Delta\mathbf{x}^{(k)}) \right\| \tag{35}$$

This minimisation problem cannot be solved analytically, but usually an approximated solution is enough to provide the desired result. For more information about line-search and trust region see for instance (Idema and Lahaye, 2014). This approach belongs to the broader category of *gradient* methods. This will be important in the context of non-stationary iterative linear solvers, that will be the main matter of the next sections.

[12]Note that this ensures global convergence at *local* minimum. Which means, in general to the closer solution to $\mathbf{x}^{(0)}$. This shall be strongly taken in mind when interpreting the results.

2.4 Introduction to Linear Solvers

One of the most well-known scopes of computer science is to study algorithms that can ease the solution of complex computational problems. Among the many features that an algorithm should have there are indeed *efficiency*, *elegance* and *stability*. In other words, algorithms shall make minimal use of computational resources, shall be easy to be read and implemented, and shall not magnify numerical errors, potentially induced by particular sets of inputs. While a comprehensive discussion is out of the scope of the present work, a short overview of some notable techniques will be provided to better motivate the linear solver discussion for distributed power flow.

As mentioned in Section 2.3, linear solvers are divided into direct and iterative. The two approaches differ in the formulation as well as in their distinctive features. For instance, one of the most relevant characteristic of direct solvers is their *robustness*. This has lead direct methods to be preferred over iterative in the industry practice, as the property of giving reliable solutions on different sets of inputs is of key importance. On the other hand, iterative methods are mostly still used in research contexts, as they can perform much better when problems scale up.

Remarkably, the reason behind using iterative linear methods in this work is not strictly the speed-up to find the solution of the power flow problem, rather than its parallel features that perfectly fit the scope of a distributed computation. This will gradually become clearer. Here, we start briefly by introducing the general formulation of direct and iterative methods. After that, we will introduce the iterative solver actually used and discuss it in detail.

2.4.1 Direct Methods

Let A be the coefficient matrix and $\mathbf{b}$ the right-hand side of the linear system:

$$A\mathbf{x} = \mathbf{b} \tag{36}$$

The solution of the unknown vector $\mathbf{x}$ is approached by direct methods by factorising the coefficient matrix A into a product of two matrices M, N that gives back A, so that Eq.(36) can be re-written as:

$$MN\mathbf{x} = \mathbf{b} \tag{37}$$

Once the M, N matrices are found, the solution of Eq.(36) is easily found. In practice, the main effort of the direct methods is on the factorisation of the A matrix. A prominent example of direct method is the LU factorisation method, which factorises A into a lower triangular matrix L and an upper triangular matrix U. In practice, the L, U matrices are based on the the Gaussian elimination, a well-known algorithm proposed by Gauss in the '800s. The Gaussian elimination process may fail in some occasions. In order to avoid such occurrences, A rows and columns are permuted before factorisation. This procedure is called *total pivoting* and allows to get to an important result: *given any non-singular coefficient matrix A, it is always possible to find a permutation of its columns and rows so that Gaussian elimination is possible: it is always possible to find the matrices L, U.* This fact gives an immediate proof of the *robustness* of direct methods. On the other hand, direct methods suffer from the so-called *fill-in* process. In practice, it is quite common to deal with *sparse* linear systems[13]; this is usually an advantage, as it lowers the memory storage and simplifies the coupling of equations in the system. The fill-in phenomena occurs during the factorisation process which induce a lot of non-zero elements to show up.

[13] In numerical analysis a sparse system is a matrix in which most of the elements are equal to zero.

2.4.2 Iterative Methods & Preconditioning

Iterative methods seek for the solution of Eq.(36) through consecutive refinements of approximate solutions. A common procedure is to find the generic iterate $\mathbf{x}^{(k+1)}$ by means of a fixed point iteration scheme, that means through the general:

$$\mathbf{x}^{(k+1)} = \Phi(\mathbf{x}^{(k)}) \tag{38}$$

Most of the iterative schemes in the literature are based on a linear map Φ. This represents a simple basis for analysis of the properties of the iterative scheme, without actually precluding its efficiency. More complex features can be added, but also in these cases the updating scheme is still linear. A linear iterative scheme updates the solution through:

$$\mathbf{x}^{(k+1)} = B\mathbf{x}^{(k)} + \mathbf{f} \tag{39}$$

Where $B, \mathbf{f}$ are respectively the *iteration matrix* and a vector, both function of $A, \mathbf{b}$. In principle, iterative schemes converge to the exact solution $\mathbf{x}^* = A^{-1}\mathbf{b}$ only after an infinite number of iterations. In practice this is not possible nor actually needed. Let us introduce the error $\mathbf{e} = \mathbf{x}^k - \mathbf{x}^*$ relative to the step k as the vector depicting the 'distance' from the exact solution. What would be sufficient is to stop the iteration process after the iteration $\overline{k}$ that makes the error norm small enough:

$$\left\|\mathbf{e}^{(\overline{k})}\right\| = \left\|\mathbf{x}^* - \mathbf{x}^{(\overline{k})}\right\| < \epsilon \tag{40}$$

That is, lower than a certain quantity ϵ. Of course, the error is an unknown quantity and can only by *estimated* at each iteration k. In practice we use an *estimator* of the error and we check its value against our tolerance ϵ. The most common estimator used is the *residual*, as it is often already available during the iteration process. We define the residual $\mathbf{r}$ as:

$$\mathbf{r} = \mathbf{b} - A\mathbf{x} \tag{41}$$

And do at each iteration k a check on its norm. The check may involve both the residual norm relative to the right hand side vector $\mathbf{b}$ and its absolute norm.

Note that the residual expression equals zero for the exact solution $\mathbf{x}^*$, exactly as it does the error expression (Eq.(40)). This might give the wrong idea that the residual perfectly estimates the error. Actually, as the machine can only approximate real numbers to floating point numbers, errors are constantly introduced in the iteration process acting as perturbations and eventually leading to false solutions. In other words, we may find a solution that respects our residual tolerance conditions:

$$\|\mathbf{r}\| < \epsilon_1 \qquad and \qquad \frac{\|\mathbf{r}\|}{\|b\|} < \epsilon_2 \tag{42}$$

While actually being far from the exact solution $\mathbf{x}^*$. Linking the error norm and the residual norm is not an easy task. It depends on many factors, among which the coefficient matrix A related properties (e.g. symmetry or positive definiteness), on the numerical perturbation considered, on the iterative scheme, on the norm definitions, etc. Generally, is common to find expressions that relate the error norm to the residual norm through the so-called *condition number* $K(A)$, which as stated in the notation is strictly dependent on the coefficient matrix. For instance, if we consider A symmetric and positive definite (SPD) and we consider numerical perturbations only on the right hand side vector $\mathbf{b}$, the error norm relates to the residual norm at the generic step k through the following expression:

$$\frac{\left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\|}{\left\|\mathbf{x}^{(k)}\right\|} \leq \frac{\left\|\mathbf{r}^{(k)}\right\|}{\|\mathbf{b}\|} \tag{43}$$

With the condition number being equal to:

$$K(A) = \frac{\lambda_{max}}{\lambda_{min}} \tag{44}$$

In general, we can consider the condition number as a measure of how much we expect the error to be small with respect to the residual. The greater the condition number $K(A)$, the greater the difference between the residual and the error. This introduces us to the concept of *preconditioning*. Preconditioning is a widely used technique aiming at facilitating the iterative process by diminishing the condition number, in other words, to diminish the impact and propagation of numerical errors. Actually, any linear system that has lower condition number is in turn easier to solve, therefore we can say that preconditioning helps in general the solution finding process.

Convergence Analysis

Convergence is an important, if not the most important, feature of numerical methods. When building an iterative scheme, for instance taking into account Eq.(39), two of the most important things that we would like to know are:

- Does Eq.(39) converge to the exact solution?
- How quick does the Eq.(39) converge?

To answer these questions we need to derive some properties of the iteration matrix B. In order for the Eq.(39) to converge to the exact solution, it is necessary that the following *consistency condition* is fulfilled:

$$\mathbf{x}^* = B\mathbf{x}^* + \mathbf{f} \tag{45}$$

$$\tag{46}$$

In other words, if we let B to be freely chosen, we are obliged to set $\mathbf{f}$ in order to respect the equality above. Let us now subtract the Eq.(45) from Eq.(39) so that we obtain a recursive expression for the error:

$$\mathbf{e}^{(k+1)} = B\mathbf{e}^{(k)} \tag{47}$$

As any iterative procedure starts with a first guess $\mathbf{x}^{(0)}$, we can also write:

$$\mathbf{e}^{(k)} = B^k\mathbf{e}^{(0)} \tag{48}$$

This gives a clear insight on the role of B: the method converges if and only if B acts as a diminishing factor for the error through iterations. The iteration matrix is the parameter we act on to design the iterative method according to our needs, as it gives all the convergence properties of the iterates. We can write a more formal convergence criterion by using eigenvalues and eigenvector properties. Recall that, given $\lambda \in \mathcal{R}^n$ as a scalar, an eigenvector $\mathbf{w}$ of B is any vector for which the following equality holds:

$$B\mathbf{w} = \lambda\mathbf{w} \tag{49}$$

The λ_p is called an eigenvalue of B. Let us assume that B has a complete set of eigenvectors (for instance this occurs when B is SPD)[14]. This means, given n as the matrix order of B, we have

[14]The result that we will get here can be extended also in the case B does not have a full set of eigenvectors. This is not treated here, but can be found for instance in (Saad, 2003) at section 4.2.1.

a set $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n$ independent vectors that form a basis $\in \mathcal{R}^n$. As n is also the order of $\mathbf{e}^{(0)}$, we can write $\mathbf{e}^{(0)}$ down as a linear combination of the eigenvectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n$:

$$\mathbf{e}^{(0)} = c_1\mathbf{w}_1, c_2\mathbf{w}_2, \ldots, c_n\mathbf{w}_n \tag{50}$$

With $c_1, c_2, \ldots, c_n$ being a unique set of coefficients. Combining Eq.(48), Eq.(49) and Eq.(50) we get:

$$\mathbf{e}^{(k)} = c_1\lambda_1^k\mathbf{w}_1 + c_2\lambda_2^k\mathbf{w}_2 + \cdots + c_n\lambda_n^k\mathbf{w}_n \overset{k \to \inf}{\approx} \max_{p\in\{1,\ldots,n\}} |\lambda_p|^k \tag{51}$$

Which approximates to the term containing the greater eigenvalue in module as $k \to \inf$. We define this term as the *spectral radius* of B. Explicitly this means:

$$\rho(B) \triangleq \max_{\lambda_p\in\{\lambda_1,\ldots,\lambda_n\}} |\lambda_p| \tag{52}$$

Finally, the expression for $\mathbf{e}^{(k)}$ becomes:

$$\mathbf{e}^{(k)} \approx \rho(B)^k \tag{53}$$

And hence it is a necessary and sufficient condition for the convergence that $\rho(B) < 1$. Furthermore, the lower the spectral radius the quicker the convergence.

Basic Iterative Methods

Here we want to derive explicitly some basic iterative schemes. A common way to derive an iterative scheme is through the *splitting* of the coefficient matrix A. In other words we write A as:

$$A = M - N \tag{54}$$

With M necessarily a non-singular matrix. Substituting into $A\mathbf{x} = \mathbf{b}$ we get:

$$\mathbf{x}^{(k+1)} = M^{-1}N\mathbf{x}^{(k)} + M^{-1}\mathbf{b} \tag{55}$$

The matrix M is called *preconditioning* matrix. Recalling Eq.(39), the iteration matrix B is now defined as:

$$B = M^{-1}N \tag{56}$$

Which suggests that the preconditioning matrix actually influences the convergence properties of the iterates. A further rework can be done on Eq.(55) in order to get an expression of B as a function only of M and A. Let us set:

$$N = M - A \tag{57}$$

Eq.(55) can be written as:

$$\mathbf{x}^{(k+1)} = (I - M^{-1}A)\mathbf{x}^{(k)} + M^{-1}\mathbf{b} \tag{58}$$

Which, by comparison with Eq.(39) gives the expression of the iteration matrix:

$$B = I - M^{-1}A \tag{59}$$

As the spectral radius of B gives the convergence rate, the iterative method converges fast in the case we make:

$$\rho(B) = \rho(I - M^{-1}A) \to 0 \tag{60}$$

This happens in the case $M \approx A$[15]. Of course, the extreme case $M = A$ is meaningless since we are inverting A itself and the preconditioning is of no help. In contrast, taking M very different from A may lead to spectral radius values just slightly lower than one, making the convergence very slow. This gives us an important insight: it is important to find the right balance between the application cost of the preconditioner and the actual ease gained from its introduction, i.e. in terms of spectral radius.

This highlights another important fact. Direct and iterative methods are not that far as it may seems, in practice the insights from one are combined into the other. One can make use effectively of a direct technique to build a preconditioner. Many recipes can be designed in this sense, for instance one may carry out an incomplete factorisation of A in order to have $A \approx LU$, set $M = LU$ and iterate through Eq.(58). By this approach, as $M \approx A$, we can obtain convergence in few iterations, while saving at the same time computational resources from a full factorisation.

Preconditioning Formulations

Splitting is just one of the methods to derive an iterative scheme. Depending on the idea as well as on the needs of the iterative solver, one can define the application of the preconditioner through alternative formulations. An example is to transform the linear system $A\mathbf{x} = \mathbf{b}$ into the left-preconditioned:

$$M^{-1}A\mathbf{x} = M^{-1}\mathbf{b} \tag{61}$$

Alternative preconditioning formulations may be based on *right-preconditioning*:

$$AM^{-1}\mathbf{y} = \mathbf{b}, \qquad \mathbf{x} = M^{-1}\mathbf{y} \tag{62}$$

Or *split-preconditioning*. Assume to have M in a factorised form, that means $M = M_L M_R$. The split-preconditioning formulation modify $A\mathbf{x} = \mathbf{b}$ as:

$$M^{-1}AM\mathbf{y} = M^{-1}\mathbf{b}, \qquad \mathbf{y} = M^{-1}\mathbf{x} \tag{63}$$

Split-preconditioning is often used when we want to preserve the symmetry of the problem. For instance, consider having M in the form:

$$M = CC_T \tag{64}$$

The split-preconditioned linear system:

$$M^{-1}AM\mathbf{y} = M^{-1}\mathbf{b}, \qquad \mathbf{y} = M^{-1}\mathbf{x} \tag{65}$$

Is always symmetric.

Right-preconditioning can be used to preserve the residual expression, that can be advantageous for some algorithms for residual check, whereas left-preconditioning uses the preconditioned residual:

$$\mathbf{z} = M^{-1}\mathbf{r} = M^{-1}(\mathbf{b} - A\mathbf{x}) \tag{66}$$

Through iterations. In conclusion, preconditioning shall be applied by following the two-step procedure:

[15] In fact: $M \approx A$ means the product $M^{-1}A \approx I$, hence $\rho(B) \approx 0$

1. Select a preconditioner M that is easy to build and ensure an easy solution of the preconditioned linear system $M\mathbf{z} = \mathbf{r}$.

2. Choose the preconditioning formulation that is the most tailored to the application, properties of the coefficient matrix A (e.g. symmetry), etc.

For more details on preconditioning formulations see (Ferronato, 2012).

Note that we can easily show that Eq.(61) is totally equivalent to a Eq.(39). Let us recall Eq.(58). This must necessarily holds also for the exact solution $\mathbf{x}^*$, that means one can write it down also as:

$$\mathbf{x}^* = (I - M^{-1}A)\mathbf{x}^* + M^{-1}\mathbf{b} \tag{67}$$

Gathering the terms in $\mathbf{x}^*$ the Eq.(61) is obtained. In general, regardless of the preconditioning formulation that we use, we can always lead back to an update scheme of the kind of Eq.(39) and analyse the convergence characteristics by identifying the expression of the matrix B. For right-preconditioning this can be done as:

$$\begin{aligned} AM^{-1}\mathbf{y} &= \mathbf{b} \\ AM^{-1}\mathbf{y} + \mathbf{y} - \mathbf{y} &= \mathbf{b} \\ \mathbf{y}^{(k+1)} &= (I - AM^{-1})\mathbf{y}^{(k)} + \mathbf{b} \end{aligned} \tag{68}$$

Which gives an expression for B almost[16] identical to that of left-preconditioning. The last equation above can be also written as:

$$\begin{aligned} \mathbf{y}^{(k+1)} &= \mathbf{y}^{(k)} + \mathbf{b} - AM^{-1}\mathbf{y}^{(k)} \\ M\mathbf{x}^{(k+1)} &= M\mathbf{x}^{(k)} + \mathbf{r}^{(k)} \end{aligned} \tag{69}$$

Which shows how the right-preconditioning formulation preserves the residual expression.

An interesting question may arise on preconditioning formulations. Does the spectrum of B changes when applying the three different expressions of preconditioning? It can be proved (see (Saad, 2003)) that the spectrum of Eq.(61), Eq.(62) and Eq.(63) is equal and hence one should not expect different convergence properties. This is actually verified only in exact arithmetic. As problems are more ill-conditioned, in practice we can have different performances depending upon the preconditioning formulation chosen. For instance, if A resembles a symmetric matrix the split-preconditioning may be the preferred formulation.

2.4.3 Practical Updating Schemes & Basic Implementation Algorithms

The expressions derived so far about the iteration matrix B are fundamental for convergence analysis and to the understanding of the properties of an iterative method. However they are not tailored for implementation schemes. Here we want to introduce expressions that can be used for implementations. Let us start by reworking a bit the following equations:

[16] The matrix-matrix product is not commutative. This means in general AM^{-1} and $M^{-1}A$ differ with each other.

$$\begin{aligned}\mathbf{x}^{(k+1)} &= M^{-1}(M-A)\mathbf{x}^{(k)} + M^{-1}\mathbf{b} \\ &= (I - M^{-1}A)\mathbf{x}^{(k)} + M^{-1}\mathbf{b} \\ &= \mathbf{x}^{(k)} + M^{-1}(\mathbf{b} - A\mathbf{x}) \\ &= \mathbf{x}^{(k)} + M^{-1}\mathbf{r}^{(k)}\end{aligned} \tag{70}$$

This gives an updating scheme for $\mathbf{x}^{(k)}$ that only involves the residual and the preconditioner. Let define $\mathbf{z}^{(k)} = M^{-1}\mathbf{r}^{(k)}$, conceptually the Eq.(70) consists of three steps: solving the preconditioned linear system in $\mathbf{z}^{(k)}$, update the solution and update the residual for the next iteration. A pseudo-algorithm for a stationary iterative method can be thus:

Algorithm 1 Prototype algorithm for stationary iterative linear solver.

1: **Begin**
2: initialize $\mathbf{x}_0, \mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$
3: **for** $k = 0, 1, \ldots,$ until convergence **do**
4: solve $M\mathbf{z}^{(k)} = \mathbf{r}^{(k)}$
5: update the solution $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{z}^{(k)}$
6: update the residual $\mathbf{r}^{(k+1)} = \mathbf{r} - A\mathbf{x}^{(k+1)}$
7: **end for**
8: **End**

An alternative way of updating the residual is sometimes used:

$$\begin{aligned}\mathbf{r}^{(k+1)} &= \mathbf{b} - A\mathbf{x}^{(k+1)} \\ &= \mathbf{b} - A(\mathbf{x}^{(k)} + \mathbf{z}^{(k)}) \\ &= \mathbf{r} - A\mathbf{z}^{(k)}\end{aligned} \tag{71}$$

From the computational point of view this costs one matrix-vector multiplication exactly as for the previous update scheme. In more complex iterative schemes, algorithms involve much more steps and the product $A\mathbf{z}^{(k)}$ might be already available at the residual update step, making it less expensive.

The algorithm shows us another important thing. At each step k we solve the preconditioned linear system $M\mathbf{z}^{(k)} = \mathbf{r}^{(k)}$. Again: the closer $M \approx A$, the higher the computational cost per iteration will be. Whereas taking M such that it is easier to solve $M\mathbf{z} = \mathbf{r}$ means in general to produce more iterations k before convergence is reached. Generally, the preconditioning matrix M is chosen so that it balances the computational cost per iteration k and the ease to obtain the solution of the linear system. In practice, the design of a proper preconditioner is a complex task and should take into account many factors, such as the physics of the problem or the sparsity of the coefficient matrix A, and depend upon the specific needs of the iterative solver (i.e. decrease computational complexity of the problem, provide numerical stability, enhance condition number of the problem, etc.). For a more detailed discussion on preconditioning see (Chen, 2005), (Saad, 2003), (Benzi, 2002). In the context in exam, it is sufficient to consider the preconditioner as a 'facilitator' for the convergence of the succession Eq.(39), though, as it will be discussed in Section 2.5, it must fulfil other additional requirements.

Non-Stationary Iterative Methods

So far the preconditioning theory has taught us to build an iterative method and to make analysis on its convergence. The basic schemes seen belong tothe family of the *stationary iterative methods*. The term *stationary* is quite straightforward: the Eq.(70) does not change through the iterations. However, these schemes are not tailored for real applications as soon as the number of iterations grows, which is the case of many real problems. To this aim the concept of *dynamic iterative methods* needs to be introduced. These methods make use of some parameters that are optimally chosen at each step k to speed up the convergence.

More formally, let us introduce the scalar $\alpha \neq 0 \in \mathcal{R}$ and modify Eq.(70) as follows:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha \mathbf{z}^{(k)} \tag{72}$$

Which can also be written as:

$$\mathbf{x}^{(k+1)} = (I - \alpha M^{-1} A)\mathbf{x}^{(k)} + \mathbf{z}^{(k)} \tag{73}$$

Due to the introduction of α, the spectral radius is changed into:

$$\rho(B) = \rho(I - \alpha M^{-1} A) \tag{74}$$

As known, the lower the spectral radius the faster the convergence is reached. The following minimisation problem can hence be set:

$$\min_{\alpha} \rho(I - \alpha M^{-1} A) \tag{75}$$

To find an optimal value for α. This simple iterative scheme is known as Richardson's iteration. It is possible to do even better by allowing α to change at each iteration k, meaning that α is *dynamically* updated through the iterations k. From now on we will denote α as $\alpha^{(k)}$ every time we intend it as a dynamic parameter. This introduces us to the gradient method. The gradient method computes $\alpha^{(k)}$ in order to obtain the minimal value for the norm of the residual vector at iteration $k+1$. In short, it sets at each iteration k the minimisation:

$$\min_{\alpha} \left\| \mathbf{x}^{(k)} + \alpha^{(k)} \mathbf{r}^{(k)} \right\| \tag{76}$$

Empirically, among all the linear vectors $\mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{r}^{(k)}$ we are looking for the optimal *step length* $\alpha^{(k)}$ that makes the next residual minimal. In this conception the residual vector is interpreted as the *search direction* along which we move in order to find the next iterate $\mathbf{x}^{(k+1)}$. This scheme can be further improved by choosing alternative directions to the one given by the residual vector. The *conjugate gradient* (CG) algorithm, for instance, takes *conjugate* (i.e. orthogonal) directions to update the iterate according to the following:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)} \mathbf{p}^{(k)} \tag{77}$$

Further insights can be gained by expanding between the first iteration (0) and the generic iteration (k). This gives:

$$\mathbf{x}^{(k)} = \mathbf{x}^{(0)} + \alpha^{(1)}\mathbf{p}^{(1)} + \alpha^{(2)}\mathbf{p}^{(2)} + \cdots + \alpha^{(k-1)}\mathbf{p}^{(k-1)} = \mathbf{x}^{(0)} + \boldsymbol{\delta} \tag{78}$$

Basically, we are looking for a correction vector $\boldsymbol{\delta}$ which is the linear combination of the set of vectors $\mathbf{p}^{(1)}, \mathbf{p}^{(2)}, \ldots, \mathbf{p}^{(k-1)}$. Since these vectors are all conjugate between each other, they form a basis for a subspace of dimension equal to $k-1$. One could think that as k gets to $n-1$, the constructed basis can be used to describe any vector of dimension n, including the solution vector.

This is not completely true, as generic n-dimensional vectors could lie in different spaces. In other words, constructing such a basis is not in general an easy task. For SPD coefficient matrix A we can prove that CG can find the exact solution in such space by minimsing the error in the A-induced norm. In other words, the CG method converge in exactly n iterations, other than numerical errors. For this reason sometimes the CG method is considered a direct method. For a more in-depth derivation of the conjugate gradient method and mathematical proofs of the concepts presented, see for instance (Gutknecht, 2007).

In practice, dynamic schemes are always used with proper preconditioners. In other words, we can interpret dynamic schemes as *accelerators* for the previously introduced stationary schemes. Notice that one can decide to solve the linear system only by means of an accelerator (i.e. without any preconditioner, that means setting $M = I$) or only by means of a stationary iterative scheme (i.e. without any accelerator, that means with $\alpha^{(k)} = 1$ and update direction given by $\mathbf{r}^{(k)}$). Even though when such techniques are combined together give definitely better performances. Another possible choice is the 'order' of the preconditioner-accelerator. In this sense we distinguish preconditioned accelerators and outer-inner dynamic schemes. In other words, we can either left-precondition the linear system through:

$$M^{-1}A\mathbf{x} = M^{-1}\mathbf{b} \tag{79}$$

And apply an accelerator, getting an update scheme as:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)} M^{-1}\mathbf{p}^{(k)} \tag{80}$$

Or use the right-preconditioning to obtain an outer-inner scheme that employ the preconditioner as an inner iteration inside the accelerator. In this configuration some notable mentions go to the outer-inner Krylov solvers and flexible preconditioned Krylov methods. Outer-inner Krylov solvers employ a double Krylov Subspace procedure, one for the outer iteration and one as a preconditioner, which means to solve the preconditioned linear system. On the other hand, flexible preconditioning allow to change the preconditioner inside Krylov iterations.

Krylov Subspace Projection Methods

In this section we further generalize the idea of dynamic iterative methods by introducing Krylov methods. Preconditioned Krylov methods are among the more effective iterative techniques to solve big sparse linear systems. The CG method is for instance part of this large classes of iterative methods. Despite CG is considered to be the best method to solve symmetric positive-definite (SPD) problems, Krylov methods offer efficient approaches also for non-SPD problems. In our context, the Jacobian of the power flow problem is not symmetric[17]. For this reason, more 'general' Krylov methods shall be employed, as the Generalized Minimal REsidual (GMREs) method(Saad and Schultz, 1986). In this section we want to introduce the Krylov methods and present some of their key features.

A Krylov Projection Subspace can be defined in many ways. A common definition for Krylov Subspace is the following. Consider a generic $n \times n$ matrix A and a $n \times 1$ vector $\mathbf{c}$, we define the Krylov Subspace $\mathcal{K}$ generated by the matrix-vector multiplications of $(A, \mathbf{c})$ as any space constructed as follows:

$$\mathcal{K}_v(A, \mathbf{c}) \triangleq \{\mathbf{c}, A\mathbf{c}, \ldots, A^{v-1}\mathbf{c}\} \tag{81}$$

In general, the sequence $\mathbf{c}, A\mathbf{c}, \ldots, A^{v-1}\mathbf{c}$ does not yield a linear independent sequence of vectors, meaning that $dim\{\mathcal{K}_v\} \leq v$. How such a sequence can be used to solve a linear system? Say that

[17] This can be easily noticed by looking at the block structure of the Jacobian Eq.(32). Derivatives in block 1,2 and block 2,1 are different, hence the matrix is generally not symmetric.

$\mathcal{A}$ is the space spanned by the columns of A and $\mathbf{c}$ is a generic $n \times 1$ vector. As we consider only well-posed problems having $det(A) \neq 0$, the dimension of $\mathcal{A}$ is exactly n. The idea behind Krylov methods is to extract a subspace $\mathcal{K} \subseteq \mathcal{A}$ having a dimension $v << n$ so that we can find inside it a 'sufficient good' solution. More precisely we look for the approximate solution $\mathbf{x}_v$ in the affine space $\mathbf{x}_0 + \mathcal{K}_v(A, \mathbf{c})$ which has the form:

$$\mathbf{x}_v = \mathbf{x}_0 + \boldsymbol{\delta} = \mathbf{x}_0 + V_v \mathbf{y}_v \tag{82}$$

If $\mathbf{x}_v$ lies in the affine space, and the dimension v of $\mathcal{K}_v(A, \mathbf{c})$ is small, we can compute $\mathbf{x}_v$ easily. Notice that the correction vector $\boldsymbol{\delta}$ (previously introduced in Eq.(78)) is expressed through the linear combination $V_v, \mathbf{y}_v$, which indicate the matrix for the basis of the Krylov Subspace and a proper vector of coefficients respectively. In general Krylov methods differ among themselves in the manner the basis are built and the weights $\mathbf{y}_v$ are computed.

The GMREs (Generalized Minimal REsidual) method solve a linear system $A\mathbf{x} = \mathbf{b}$ by setting up the following minimisation problem:

$$\min_{y \in \mathcal{K}(A, \mathbf{b})} \|\mathbf{b} - A\mathbf{x}\| \tag{83}$$

In other words, GMREs looks for the vector $\mathbf{y}_v$ that, among all the vectors in $\mathcal{K}(A, \mathbf{b})$, gives back the minimal residual vector. In order to solve the minimisation problem we need a tailored basis for the Krylov Subspace. This could be done by constructing the sequence:

$$\mathcal{K}_v(A, \mathbf{r}^{(0)}) = \{\mathbf{r}^{(0)}, A\mathbf{r}^{(0)}, \ldots, A^{v-1}\mathbf{r}^{(0)}\} \tag{84}$$

As mentioned this is actually not a basis, as such vectors are linear dependent. In principle, we could stop the matrix-vector multiplications as v gives the first dependent vector. In practice this is not done as the sequence is poorly linear independent, leading to an ill-conditioned basis that is not suitable for use. For this reason some ortho-normalisation process must be employed. The GMREs makes use of a modified version of the Gram-Schmidt ortho-normalisation (i.e. the Arnoldi process) in order to obtain the Krylov basis. The GMREs algorithm can be outlined in three conceptual steps:

1. Ortho-normalise the sequence $\{\mathbf{r}^{(0)}, A\mathbf{r}^{(0)}, \ldots, A^{v-1}\mathbf{r}^{(0)}\}$ to create the Krylov basis V_v;
2. Solve the residual minimisation problem to compute $\mathbf{y}_v$;
3. Update the solution with Eq.(82).

The computational cost of the first step scales up much faster than linearly as the dimension of the Krylov Subspace increases. For this reason a fourth step is usually added to the procedure above. This sets the possibility to restart the procedure after some dimension v of the subspace is reached. This modification consist in the Restarted-GMREs (R-GMREs) version of the algorithm. The R-GMREs starts back the procedure in the case the solution obtained does not fulfil the estabklished tolerance. If such situation occurs, R-GMREs takes as the first iterate the solution obtained by the previous subspace, setting then $\mathbf{x}^{(0)} = \mathbf{x}^{(v)}$.

2.4.4 Computational Complexity of Direct and Iterative Methods

Computational complexity can be retained as the amount of computational resources needed to solve a computational problem. An important step when solving scientific computing problems is to assess the computational complexity of the problem in exam. In turn, algorithms are chosen in order to decrease the amount of resources required to solve the problem. Usually, two main resources are

considered: the *computational space* i.e. memory requirements and *computational time* i.e. the time required to solve a given computational problem.

In this section we want to give a short insight on the computational complexity of direct and iterative methods as a function of the size and scale of the problem. This will help us later in identifying the main 'computational features' of the power flow problem. We find the FLOPs (floating point operations) required for a direct and an iterative solver for a *dense*[18] linear system as function of the dimension n of the linear system and draw some straight conclusion.

Let's start considering the LU method. The LU method looks for a factorisation of A in a lower triangular matrix L and an upper triangular matrix U, so that a linear system $A\mathbf{x} = \mathbf{b}$ is turned into the two equivalent linear systems:

$$L\mathbf{y} = \mathbf{b}, \qquad U\mathbf{x} = \mathbf{y} \tag{85}$$

Once L and U are found, the solution of the two systems above can be easily found. The triangular structures of L and U gives a series of equations totally decoupled. For instance, consider the system $L\mathbf{y} = \mathbf{b}$. The first equation contains only the first unknown, the second equation contains only the first and second unknown, hence is only function of the first, the third is only function of the first twos, etc. In other words, we can solve the two triangular linear systems above through backward (U) and forward (L) substitution. From the computational point of view this costs only $n^2 - n$ operations. The expensive stage is the factorisation process. Let's consider the Gaussian elimination algorithm as the process to find the matrices L and U. This algorithm costs roughly $2/3n^3$. A proof for this can be found for instance in (Farebrother, 1988). As n gets sufficiently large, the computational complexity of a direct method for dense matrices can be approximated to $O(n^3)$.

An iterative method updates the solution through the standard linear fixed point iteration scheme (Eq.(39)). This costs at each iteration k at least one MVM: which means $n * (n + (n - 1)) = 2n^2 - n$ operations plus, at least, $2n$ summation operations to update the solution and residual with $\mathbf{f}$. Basically this means that an iterative method cost is in the order of $\mathcal{O}(kn^2)$. This allows us to draw some simple conclusions. Suppose the iterative method converges (i.e. iterations usually are < 100000), problems that scale up over 10^5 unknowns give a computational advantage to iterative methods compared to direct methods. In practice, iterative methods are not implemented according to Eq.(39), though the computational expenditure can still be approximated to the matrix-vector multiplication only.

Assessing in detail the actual computational complexity of an algorithm is not an easy task and it is a study subject by itself, as many other factors come into play (e.g. matrix-vector specific implementations, sparsity, parallelism, ecc.). In the next sections to provide a more comprehensive view to the reader we discuss (qualitatively) the role of *sparsity* and *parallelism* to assess computational complexity.

Sparsity

Let us now introduce the concept of *sparsity*[19]. When dealing with sparse matrices the computational complexity becomes of the order of $\mathcal{O}(nz)$, where nz is the number of non-zero entries of the coefficient matrix. This represents a further advantage for iterative methods for at least two reasons. First, direct methods use factorisation processes that lead the initial sparse matrix to be *filled-in*. The higher the number of non zero-entries of LU matrices, the greater the computational complexity, meaning that more memory storage is required as well as more operations to solve the linear system. The second reason being that in general it is more difficult to develop factorisation

[18] The matrix A which describes the system has only non-zero entries.

[19] There is not strict definition of sparsity of a matrix. Simply speaking, a sparse matrix is a matrix whose entries are mostly zero elements.

algorithms that take fully advantage of zero entries. In practice, sparsity is one of the reason why iterative methods perform better for solving large sparse linear systems.

Parallelism

When dealing with complex scientific problems, solving algorithms are designed to efficiently run in parallel on dedicated computing platforms (e.g. clusters). The benefits from parallelising the source code in which the algorithm are written are usually measured by comparing the simulation sequential time with the simulation parallel time. To this aim the speed-up factor S is defined as:

$$S = \frac{\text{sequential computational time [s]}}{\text{parallel computational time[s]}} \tag{86}$$

The program has linear speed-up if $S = p$ where p is the number of cores used in the executing program. Efficiency is said to be the ratio between speed-up and the number of cores used in the computation $E = S/p$. In practice, E is always less than one, mainly for two reasons. First, problems cannot be completely broken down into sub-tasks, that is, cannot be fully parallelised. For instance, from Amdhal's law one can see that even in the favourable case of a 90% of code parallelisable the maximum theoretical speed-up can be equal to 10. The second reason is that processors need to communicate with each other when sub-tasks are coupled. Let us consider a simple example to clarify this last point.

Example 1. *Let us consider the dot-product operation of two $\mathcal{R}^n$ vectors **v** and **w**. The result λ of the dot-product is a scalar given by the following operation:*

$$\lambda = \langle \boldsymbol{v}, \boldsymbol{w} \rangle_2 = \boldsymbol{v}^T \boldsymbol{w} = \sum_{i=1}^{n} v_i w_i = v_1 w_1 + v_2 w_2 + \cdots + v_n w_n$$

How can we parallelise such an operation? Let consider two processors p_1 and p_2. We can split equally the number of n products and the $n-1$ summations over the two processors. This means p_1 does:

$$\lambda_1 = v_1 w_1 + v_2 w_2 + \cdots + v_m w_m$$

And p_2 does:

$$\lambda_2 = v_{m+1} w_{m+1} + v_{m+2} w_{m+2} + \cdots + v_n w_n$$

In order to compute λ we need to add up the two contributions e.g. `send` λ_2 from p_2 and make p_1 ready to `receive` it. This means we need a global communication *operation. The `send` and `receive` operations are message passing primitives and are at the basis of inter-process communication protocols (IPCs).*

Eventually, one processor computes:

$$\lambda = \lambda_1 + \lambda_2$$

This simple example shows us two things:

1. We need to spend some time passing information among processes;
2. We cannot always exploit the whole computational power of a parallel computing platform, since some operations cannot be further sub-divided.

These two factors limit the speed-up factor to be lower than the actual available computational power (in the example < 2). In other words, parallel computing is based on a 'divide-and-conquer' principle, which may be effective or not depending on the algorithm adopted, the message passing protocols, the parallel architecture, software and hardware, etc. In this context, it is often said that some algorithms are more 'parallelisable' than others, meaning that they contain more *operators* that are prompt to the divide-and-conquer principle. If we take n very large, the amount of operations done locally on a processor exceed the number of operations done together (i.e. the single final addition) of a great extent. In other words, the dot-product operation (i.e. consequently, matrix-vector operations) is well-suited to parallelisation.

Distributed vs Parallel Computing Architectures

From an hardware point of view, the concepts previously introduced may have different implications depending on the type of architecture that is used in the parallel processing. Generally it can be of distributed (or *loosely-coupled* systems) or parallel (*tightly-coupled* systems) type. An example of tightly-coupled systems is the high performance computing platforms that are usually found in research institutions (molecular, aerospace, hydraulic to cite a few), but investments on this field are also growing in industry contexts (e.g. Oil & Gas companies, Investment Banks or High-Tech companies). On the other hand, loosely coupled systems are for instance wide area networks (WANs), such as 4G or the Internet itself. These systems are often highly geographically dislocated and may need to communicate through huge distances. Conceptually, a distributed system architecture is typically characterised by non-shared memory units, that is, each single system owns separately its processing capacity and memory unit. Parallel computers, in contrast, employ high-speed interconnections among CPUs and shared memory systems, in order to provide fast IPC communications and overall increasing the computational efficiency. Notice that the *distributed* and *parallel* nomenclature is not that strict since mixed solutions exist as well. Fig. 5 depicts the physical difference between typical parallel and distributed architectures.

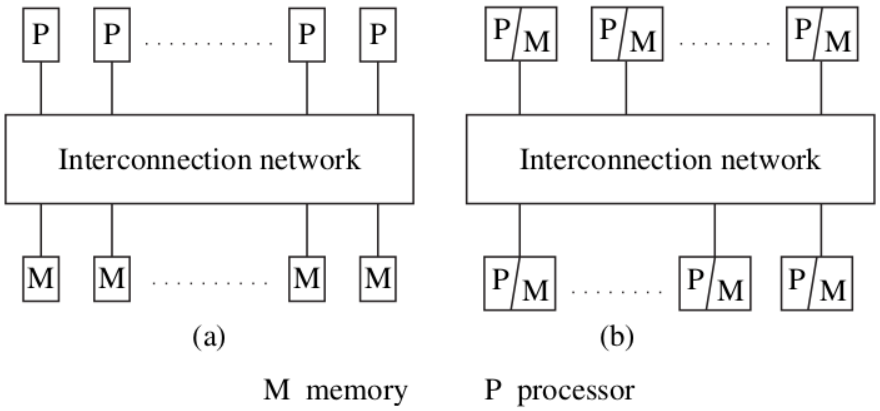


Figure 5: Parallel architecture (a) and Distributed architecture (b) (Kshemkalyani and Singhal, 2008).

It is clear that, since the two paradigms differ for some key aspects, algorithms shall be differently designed, taking into account of the architecture on which they will be running. In parallel computing context, the computational time is higher than the communication time, so it makes sense to design algorithms that are as much as possible computationally efficient, while ensuring that message passing operations among processors do not slow down overall the solution process. Rather, in distributed computing context, as the interconnection among processes may take place through many layers and

it is usually remote, the communication time overcome the computational time. In this regard, it takes only a little sense to optimise the model from a computational point of view in the case it requires many messages to be exchanged among the nodes. The need to carry out distributed computations usually may regard different reasons: data may be dislocated (e.g. local measurements), there might be the need to keep separated memory access from different nodes involved in the computation (e.g. peer-to-peer systems) or simply there might be the possibility to exploit distributed resources more effectively.

2.5 Iterative Methods, Distributed Power Flow & Current Practices

As anticipated, this work employs an iterative linear solver for the solution of the sequence of Jacobian linear systems obtained in Section 2.3. Rather than using an iterative method for solving the power flow equations faster, we exploit the Krylov-Schwarz approach to carry out a distributed computation of power flow equations. A remote computation that only involves a minimal amount of data-sharing have the double advantage to not compromise the overall simulation time while effectively keep confidentiality of input data for parties involved in the computation.

Thus far, we introduced concepts on linear solvers in order to provide a comprehensive overview of the topic. Here, the goal is somewhat to put together all the concepts introduced in the previous sections in the specific context of power flow. We will start from a brief comparison of direct methods and iterative methods and review in general the potentialities of iterative methods in such a context.

2.5.1 Current Practices and Iterative Methods in Power Flow

From a scientific point of view, power flow problems cannot presently be considered as 'hardly complex' problems to solve. For instance, non-linear systems of equations arising from the discretisation of Partial Differential Equations (PDE) in Computational Fluid Dynamics (CFD) (but in many other fields as well), are often much bigger in terms of size and present more difficulties to handle the solution process when compared with power flow problems. In contexts of this kind, where the size of the problem may even reach $n = 10^9$, direct solvers are impractical as the computational cost of them ($\mathcal{O}(n^3)$) can't be handled by supercomputing platforms. In power flow context this has not been historically the case, mainly for two reasons. First, TSOs solve currently power flow problems with 10^3 unknowns, which can be still efficiently handled by direct methods. The other reason being that the robustness and the predictable behaviour of direct solvers make them better candidates for commercial software packages. However, there are at least two important reasons at present that may see iterative solvers to overcome the use of direct methods in the future at least for some specific purposes. As the integration of DERs increases in power grids, the need of grid operators to perform more frequent, more detailed and closer to real-time power system analysis increases at the same pace. Today, the close-to-real-time power flow simulations are carried out through the Fast Decoupled Load Flow (FDLF) approach, that allows to simplify the power flow problem by losing a little of precision on the final solution. Despite its speed performances, a drawback of the FDLF approach is the lack of robustness, some failure event when the Jacobian is ill-conditioned. Iterative methods offer the possibility to speed-up the solution of problems that are particularly computational demanding and prone to parallelisation (e.g. $N - k$ contingency analysis). Furthermore, as present grids are being increasingly interconnected at both national and distribution level, the problem size grows; market operations are more integrated with grid operations, more computational expensive models could be required (e.g. nodal pricing implemented as an optimal power flow problem in AC); sector coupling (or even the transmission-distribution coupling itself) may require to carry out coordinated and heterogeneous modelling among different grids (e.g. gas and electricity networks). These factors may contribute essentially to increase the computational complexity of the power flow problem, while also requiring

solvers with different features, able to handle the heterogeneous modelling, able to efficiently run in parallel, etc. This changing paradigm motivates further research in iterative methods for the power system context. Relevant insights on performance comparison between iterative methods and direct methods for solving power flow equations are provided in (Idema, 2012), where direct methods and mixed direct-iterative approaches are tested and compared in terms of computational time. (Idema, 2012) shows how direct methods for power flow equations stall when the number of buses go over $4 * 10^5$, while proving the robustness of Newton-Krylov methods for different sets of generation/load profiles. [20] Among the multitude of applications that iterative solvers may have in power system to decrease computational time, a rather different approach is used here. The next sections detail the main idea.

2.5.2 Distributed Approaches

In the past, energy utilities were organized as vertical integrated companies. No competitive auctions were in place and there was no asymmetry of information among different stages of the supply chain, since the complete process was managed by a single company. For this reason, the main concern of power flow solver developers in these years was to provide grid operators with fast and above all *reliable* softwares, mainly based on direct solvers as discussed in the previous sections. Over the last years, the economic and operational framework of the electricity sector has undergone a complete restructuring. Transmission and distribution networks at European level are managed by different entities, which often share boundaries and inter-connect through lines. However, power system analysis tools have not changed: power grids planning and scheduling is still done by solving power flow and optimal power flow simulations at the transmission level only. In this new configuration of the power system, where the network management is being more and more distributed and unbundled, we foresee the need of new generation power flow solvers. In the emerging scenario, distributed solvers shall respect at least three main requirements: They need to be *fast*, *reliable* and *information/data protective*. In literature some distributed solvers for power flow equations have been already developed over the years, mainly with reference to the DC OPF problem (e.g. (Mohammadi et al., 2014), (Kargarian et al., 2016)) which represents a considerable simplification of the AC version. It is worth mentioning that nomenclature can sometimes be a little misleading when referring to *distributed* solvers. For instance, (Kargarian et al., 2016) distinguishes between *distributed* and *delocalised* approaches. In distributed approaches there is the need of a coordinating entity (i.e. master) that sets the list of operations to execute for coordinated entities (in a master/slave fashion). This does not imply the definition of a global problem at the master machine; information may be in fact still inputed from different locations. On the contrary, delocalised approaches provide an autonomous framework for entities, where each one can collaborate at same level. From a computer science perspective, this last case coincides with the *peer-to-peer* framework.

Both approaches (distributed/delocalised) share a similar working methodology composed by two main parts. First, it is needed to re-formulate the global problem into an *equivalent*[21] set of sub-problems. Then, a tailored distributed/delocalised algorithm must be chosen to solve the set of sub-problems in an organized way. The issue of *information* protection has not been deserved much attention in the literature so far. This is mainly due to the fact that traditional application scenarios do not face the requirements that are arising in the power sector between different utilities. In this work we propose a methodology that is highly scalable and only involves a small amount of data exchange between the parties involved in the power flow computation. The next sections explain our approach in detail.

[20] Note that the simulations carried out in the mentioned work were performed on a single-core platform (i.e. not in parallel).

[21] The re-formulation must change only in its form. Equivalent means the mathematical problem is not changed and hence must deliver the same solution.

2.5.3 Krylov-Schwarz Methods & Distributed Computation of Power Flow

In this work we propose the use of iterative preconditioned solvers, that have mainly been used in parallel computing platforms and in contexts where the computational complexity is the main factor of attention. We use here such methods in a different perspective. We propose the use of block preconditioned Krylov methods to solve the power flow equations in a distributed computing environment, focusing mainly on the limitation of information sharing (i.e. through message passing) rather than on actual computational performances.

As mentioned in 2.4.4, distributed computing platforms introduce slow-downs due to inter-process communications. As a matter of fact, we should not expect better computational performances compared to local (potentially parallel) simulations. The goal is to validate a methodology that allows to exchange a limited amount of information on input data (generation and load profiles, topology) as well as on the solution (state variables), by renouncing only to a bit of speed performance.

To achieve this goal, the solver used requires a precise set of features:

- It must be prone to parallelism;
- It shall solve the linear problem in a reasonable number of iterations;
- It shall minimise the number of collective operations, hence of communications;
- It shall be scalable, in order to deal with grid-couplings that may lead to a very large number of nodes;
- It shall be robust and provide convergence on solution at interface nodes;

Note that keeping the overall number of linear iterations low is an important factor since this slows down the process. A good candidate to meet the criteria listed above is a solver which is based on the Block Preconditioned Krylov method, in particular on the Krylov-Schwarz method.

Block Preconditioners

Block preconditioners are often used in order to better resemble the properties of the coefficient matrix, in practice $M \approx A$. As mentioned over sections in 2.4, more complex preconditioners make more computationally expensive the solution of the preconditioned linear system over each iteration k while leading to overall less iterations k. Block preconditioners represent a class of preconditioners quite wide and realize a good compromise between expenditure per iteration and iteration number reduction. Let us consider a simple example to deepen the idea behind this concept.

Jacobi preconditioning is based on selecting $M = D$ where $D = diag(A)$ is a matrix formed by only the diagonal entries of A. As for any preconditioned method, we need to solve the easier preconditioned linear system:

$$D\mathbf{z} = \mathbf{r} \tag{87}$$

That is a set of equations as:

$$\begin{cases} a_{11}z_1 = r_1 \\ a_{22}z_2 = r_2 \\ \dots \\ a_{nn}z_n = r_n \end{cases} \tag{88}$$

In the regular Jacobi scheme thus, at each step k each unknown $\mathbf{x}_i$ is updated regardless of the other. This intuitively works very good on parallel machines. We can divide the set of equations (we call this set $\mathcal{S}$) in equal (or almost equal) $\mathcal{S}_P$ subsets ($\mathcal{S} = \bigcup \mathcal{S}_P$), assign them to P nodes of a parallel machine and solve the problem. Eventually, we can gather the results to build the solution vector $\mathbf{x}$. As the computation of each x_i is completely decoupled from the others, the Jacobi approach requires many iterations to converge. The Jacobi scheme has been one of the first iterative methods proposed in literature for solution of linear systems and it is thus not really suitable for real applications. An improvement of the Jacobi scheme is based on introducing more dependencies among the x_i variables at each step k, in order to have more accurate solutions. The Gauss-Seidel approach uses this principle by setting $M = L + D$, where L is the strict lower triangular matrix of A. This means we solve, at each step k, a set of equations in the preconditioned residuals z_i as[22]:

$$\begin{cases} a_{11}z_1 = r_1 \\ a_{11}z_1 + a_{22}z_2 = r_2 \\ \dots \\ a_{11}z_1 + a_{22}z_2 + \dots + a_{nn}z_n = r_n \end{cases} \tag{89}$$

These equations can be solved in one step by forward substitution. It is possible to show that Gauss-Seidel converges faster than Jacobi by analysis on the spectral radius of its iteration matrix B. Although Gauss-Seidel improves the convergence properties of the succession $\mathbf{x}^{(k)}$, it complicates its parallel implementation. Let us suppose we use the same approach as before. We call again the set of equations to solve $\mathcal{S}$ and divide it into $\mathcal{S}_P$ subsets ($\mathcal{S} = \bigcup \mathcal{S}_P$). After that we assign them to P nodes. The generic node $p \in \{1, \dots, P\}$ will need to wait until all the $1, \dots, P-1$ other nodes have completed the solution of the assigned subset $\mathcal{S}_p$ of equations, since the equations in $\mathcal{S}_p$ are all dependent on unknowns contained in $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_{P-1}$.

Since the parallelism is a fundamental requirement in order not to share information and keep reasonable time of computation, Gauss-Seidel or similar approaches are not really suited for the work at issue. At the same time we need more effective methods than classical Jacobi scheme, able to deal with big linear systems without stalling in a huge number of iterations. In this sense, block preconditioners come in handy. For instance, let consider the extension of the Jacobi preconditioner to its block-form. Instead of taking only the diagonal entries of A, the block-Jacobi method takes diagonal blocks extracted out of A. In other way around we construct a preconditioner $M = diag(A_{11}, A_{22}, \dots, A_{PP})$ made up of subsequent diagonal blocks (we call them A_{pp}) from A of potential difference size. We can either have as many blocks as the processors, or give out multiple blocks to single nodes of the parallel system. Here we consider for sake of simplicity that each block is ideally assigned to a processor p. The set of preconditioned equations to solve are this time:

$$\begin{cases} A_{11}z_1 = r_1 \\ A_{22}z_2 = r_2 \\ \dots \\ A_{PP}z_P = r_P \end{cases} \tag{90}$$

With P the number of total blocks and number of total processors available. Each equation above is actually a small linear system to solve, which is as much small as the selected number of total blocks into which we divide A. Notice that as the dimension of blocks P gets to n, block-Jacobi turns back to the regular Jacobi scheme, as the dimension of the block would be exactly of one diagonal entry. On the other hand, as $P \to 1$, we are preconditioning the linear system with A itself, hence the iterative procedure is actually a direct method ending up in exactly one step.

[22] Here, we are simply assuming A is dense.

As for the regular Jacobi scheme, block Jacobi preconditioning works really good in parallel. In both cases we can solve the set of linear systems associated with the generic node P independently (i.e. locally). As the number of equations in a generic block $q \in \{1, \ldots, Q\}$ is usually much lower than that of A, we can nicely use factorisation techniques (e.g. LU) to solve the blocks, according to the simple computational complexity analysis presented in Section 2.4.4.

A formal mathematical description of block preconditioners can be derived in many ways. One way is to do it in terms of *projectors*[23]. Here we limit to introduce two operators W_p^T (*restriction operator*) and V_u (*prolongation operator*), which applied together allow to define:

$$A_{pu} = W_p^T A V_u \tag{91}$$

Where A_{pu} is a rectangular matrix of dimension $p \times u$, representing the block pu extracted from A. Dimensionally speaking W^T is a $q \times n$ matrix and V is a $n \times u$ matrix. Note that W_p^T, V_u only get the components from A without applying any transformation, hence those are matrices made up of only zeros and ones (projectors). In the case of considering the blocks squared, $W = V$ and the equation Eq.(91) becomes:

$$A_{pp} = V_p^T A V_p \tag{92}$$

Each block is then allocated along the main diagonal of M, overlapping the portion of blocks corresponding to variables in common. This brings to a situation as the one depicted in Fig. 6.

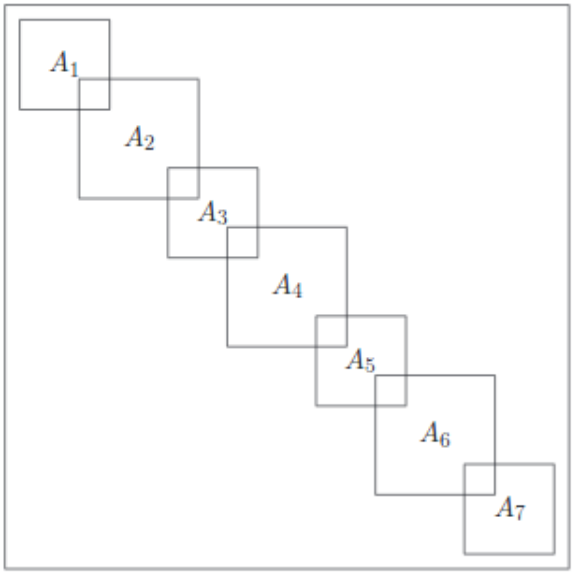


Figure 6: Overlapping Block-Jacobi Preconditioner.

At this stage we have a block diagonal preconditioner M made up of P blocks. A prototype algorithm for block Jacobi could be the one described in Algorithm 2 below.

Notice that the preconditioning matrix (of dimension $n \times n$) in this case would be:

$$M = \sum_{p=1}^{P} W_p^T A_{pp} V_p = \sum_{p=1}^{P} W_p^T (V_p A W_p^T) V_p \tag{93}$$

[23] See Chapter 5 of (Saad, 2003) for the whole procedure.

Algorithm 2 Prototype algorithm for block Jacobi solver.

Begin
  initialize $\mathbf{x}_0, \mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$
  for $k = 0, 1, \ldots,$ until convergence **do**
    set $A_{qq} = W_q^T A V_q$
    solve $A_{qq}\mathbf{z}^{(k)} = W_q^T \mathbf{r}^{(k)}$
    update the solution $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + V_q \mathbf{z}^{(k)}$
    update the residual $\mathbf{r}^{(k+1)} = \mathbf{r} - A\mathbf{z}^{(k)}$
    compute $\|\mathbf{r}\|$ and check convergence
  end for
End

In practice, this matrix is almost never built up as it would add unnecessarily further cost, both in terms of memory storage and computational time. In our application this is even more important as building M implies to collect all the information on the coefficients of matrix A (i.e. the topology, as it is embedded in the sparsity pattern of the Jacobian).

The Algorithm 2 actually give us an insight on the needed communications in a parallel environment. The computation of the residual norm is a required step to check convergence for any iterative algorithm. This requires at least a global communication over processes. Remember the norm-2 (i.e. Euclidean norm) of a generic vector $\mathbf{w}$ is defined as:

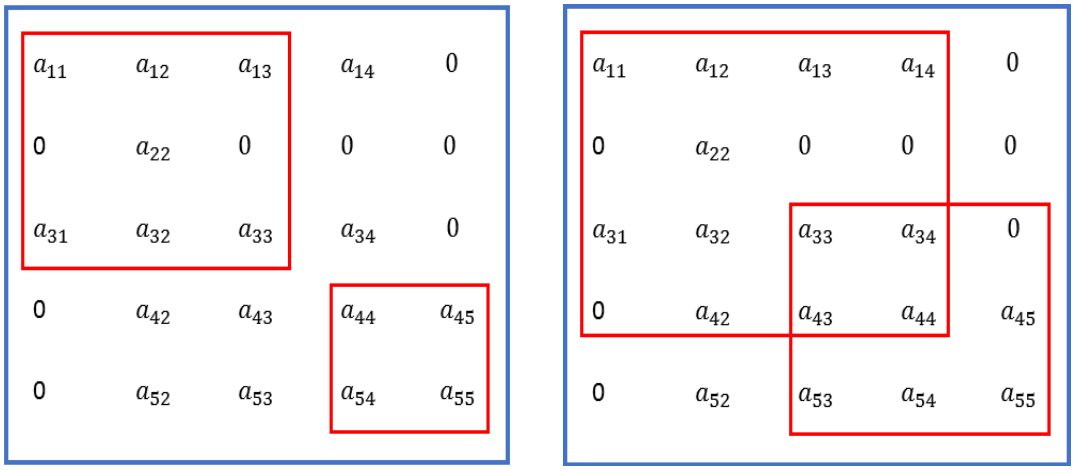
$$\|\mathbf{w}\| = \sqrt{w_1^2 + w_2^2 + \ldots w_n^2} \tag{94}$$

So that the square root operation must be done eventually on one processor, which is usually the master. A further comment on parallel communications can be done regarding matrices A and M. One may think that we need to store the matrix A explicitly in order to divide the problem into blocks. Actually, as it will be explained through Section 4, by using a parallel assembling procedure we do not need to build explicitly neither A nor M. In other words, the input data for the power flow problem (i.e. topologies, generation and load profiles) remain local on the separated machines. Assuming the simple case with only two machines (or *nodes*, in a parallel nomenclature), hence we would have two blocks. This can then be extended to a generic number of machines. In practice, each block acts as a local preconditioner.

From a power system perspective, using such an approach may result counter-intuitive. One may ask: if we solve the two problems separately, how can we manage to obtain convergence on interface variables? Solving the linear systems separately does not mean we are not taking into account what happens on neighbouring domains. This is the power of iterative methods. At each linear sweep we evaluate a new Jacobian, which is in turn function of the new solution vector. Notice that even if we introduce an accelerator in Algorithm 2, the procedure may still converge slowly, requiring a remarkable amount of linear iterations to converge. In the next subsection we introduce the concept of blocks *overlap*. The idea of overlapping is to share only a small amount of interfacing variables, gaining on the other hand a considerable speed-up. More strictly, this produces an exponential speed-up in the solution process, as for instance shown in (Gander, 1996). From the point of view of shared information, this amounts to share only information about topology (connectivity and grid parameters), demand and generation profiles about such interfacing nodes. Given the current resistence to share data for security or strategic reasons, the promise to share only a little amount of information at the interface between two (or more) utilities gives indeed more chances to our method to be used from distributed power flow problems.

Overlapping Block Preconditioners

Another improvement to block preconditioning techniques can be done by allowing the blocks to *overlap*. By including overlap (thus more dependences) we can improve the convergence of the algorithm. In other words, the idea of overlap consists in 'sharing' some variables (and the related constraining equations i.e. the power balances at the related nodes) among processes involved in the computation, solve these equations and eventually weight the solution on the shared variables. The procedure is then restarted until convergence. The Fig. 7 shows the same (generic) coefficient matrix A, in the two cases: preconditioned by block Jacobi and overlapping block Jacobi. The preconditioner M is constructed in the two cases by extracting only elements inside the red blocks.



(a) Block Jacobi. **(b)** Overlapping block Jacobi.

Figure 7: Block Jacobi preconditioner and overlapping block Jacobi. Only entries inside the red boxes take up into the preconditioning matrix M.

Note that $M \neq A$, which means when we solve the preconditioned system we do not get the exact solution and hence it motivates the need to iterate the procedure. In the overlapping case we include more dependencies among variables, making M resembling more A. In this case the overlapping preconditioner involves common solution on the preconditioned residuals z_3, z_4 (hence x_3, x_4). Let us clarify this aspect. Let us write explicitly the equations of the above linear system and the preconditioned equations with reference to Fig. 7:

$$\textbf{a)} \begin{cases} a_{11}x_1 + a_{12}x_1 + a_{13}x_3 + a_{14}x_4 = b_1 \\ a_{22}x_2 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 = b_3 \end{cases} \begin{cases} a_{11}z_1 + a_{12}z_2 + a_{13}z_3 = r_1 \\ a_{22}z_2 = r_2 \\ a_{31}z_1 + a_{22}z_2 + a_{33}z_3 = r_3 \end{cases} = M_{upperblock}$$

$$\begin{cases} a_{42}a_2 + a_{43}x_3 + a_{44}x_4 + a_{45}x_5 = b_4 \\ a_{52}x_2 + a_{53}x_3 + a_{54}x_4 + a_{55}x_5 = b_5 \end{cases} \begin{cases} a_{44}z_4 + a_{45}z_5 = r_4 \\ a_{52}z_2 + a_{53}z_3 + a_{54}z_4 + a_{55}z_5 = r_5 \end{cases} = M_{bottomblock} \tag{95}$$

$$\textbf{b)}\begin{cases} a_{11}x_1 + a_{12}x_1 + a_{13}x_3 + a_{14}x_4 = b_1 \\ a_{22}x_2 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 = b_3 \end{cases} \begin{cases} a_{11}z_1 + a_{12}z_2 + a_{13}z_3 + a_{14}z_4 = r_1 \\ a_{22}z_2 = r_2 \\ a_{31}z_1 + a_{32}z_2 + a_{33}z_3 + a_{34}z_4 = r_3 \\ a_{42}z_2 + a_{43}z_3 + a_{44}z_4 = r_4 \end{cases} = M_{upperblock}$$

$$\begin{cases} a_{42}a_2 + a_{43}x_3 + a_{44}x_4 + a_{45}x_5 = b_4 \\ a_{52}x_2 + a_{53}x_3 + a_{54}x_4 + a_{55}x_5 = b_5 \end{cases} \begin{cases} a_{33}z_3 + a_{34}z_4 = r_3 \\ a_{43}z_3 + a_{44}z_4 + a_{45}z_5 = r_4 \\ a_{53}z_3 + z_{54}z_4 + a_{55}z_5 = r_5 \end{cases} = M_{bottomblock} \tag{96}$$

In case a) the equations of upper and bottom block do not share any unknowns. They can be solved regardless of each other. In a parallel environment, usually one machine (i.e. the *master* machine) assign sub-problems to other processors (i.e. the *slaves*). For instance, in our case we can think of a master solving the upper block and distribute the bottom block to a second processor. Hence, each processor solves locally the owned blocks, in the a) case without exchanging any information (at least during the solution process of blocks). On the other hand, to complete the linear sweep we need to update the residual and solution vectors. If we want to avoid inter-communication processes we shall have such coefficients stored locally from the beginning. Using a parallel input can avoid the exchange of this information as well.

In case b) overlap is introduced. We introduce one more equation per block to solve. This means, the upper block now solves with respect z_4 as well, whereas the bottom block also gets the value of z_3. As the system of equations differ from upper block to bottom block, we expect different values on z_3, z_4 out of each block. How do we update the values of z_3, z_4 at the end of the linear iteration? A common choice is to weight the results got out of each block, that means according to:

$$\begin{aligned} z_3 &= \xi_3 z_{3,ub} + (1-\xi_3) z_{3,bb} \\ z_4 &= \xi_4 z_{4,ub} + (1-\xi_4) z_{4,bb} \end{aligned} \tag{97}$$

Where ξ is the weight parameter and the subscripts ub,bb stand for *upper block* and *bottom block* respectively. Let us now analyze what changes in terms of parallel communications. Let us assume a parallel input of data is in place, so that row $1, 2, 3$ and $4, 5$ of the coefficient matrix are respectively local on two different processors, which we call proc1 and proc2 respectively. In order to solve the two blocks on each processor, we need to collect the coefficients a_{43}, a_{44} from proc2 to proc1, and a_33, a_34 from proc1 to proc2. Once the blocks are solved, the average on interfacing variables is carried out according to Eq.(97). This can be done either on proc1 or proc2. z_3, z_4 are then distributed back to the other processor so that the local solution and residual vectors can be updated. A last step that involve parallel communication is that of residual norm check, similarly to what seen in Example 1[24].

The brief parallel communication analysis provided here shows how is possible to solve a linear system by means of block-preconditioners and by only exchanging a limited amount of information (i.e. 1- Coefficients relative to overlapping variables; 2- Solution on interfacing variables; 3- Local summation of residuals). In the next section we introduce domain decomposition methods. This will allow to introduce a preconditioner (i.e. the Additive Schwarz Method (ASM)), an important piece of our distributed approach for power flow equations.

[24] Remember that $\langle \cdot, \cdot \rangle = \|\cdot\|_2^2$

Domain Decomposition Methods

In science and engineering it is common to formulate modelling equations with reference to a physical domain of interest. These domains are often continuum and involve complex differential equations that cannot be analytically solved. For this reason, the common practice consists in defining a *mesh*, that is a discrete domain where differential equations are in some sense *averaged* from point to point. Many techniques exist in literature one of them being the *finite difference* approach. This process allows to turn the origin differential problem into a set of algebraic equations that can be solved more easily. Although discretisation techniques allow to greatly simplify the problem, typical algebraic systems may include up to billions of unknowns, remaining a considerable challenging task to be solved. *Domain Decomposition Methods* are used to further simplify the problem. The use of DDM demonstrated very effectively in handling these kind of huge problems. The principle is that of divide-and-conquer, which implies a domain division into pieces, with each sub-domain (set of equations) solved separately. This technique is iterated until convergence at the sub-domain interface variables is obtained. In practice, DDMs are used to build preconditioners that can effectively decouple problems and help in the solution of complex problems.

It is worth to point out that domain decomposition process can follow different criteria. We distinguish these criteria in terms of type of *partitioning*, that is according to one of the following case:

- **Vertex Partitioning** - vertices are collected into sub-sets that in turn set the decomposition into sub-domains;
- **Branch Partitioning** - branches are collected into sub-sets that in turn set the decomposition into sub-domains;
- **Elements Partitioning** - elements are collected into sub-sets that in turn set the decomposition into sub-domains;

For sake of clarity this is depicted in Fig. 8 as well.

In the power grids context the above approach is to some extent inverted. We already have a discrete domain: the graph representing the topology of the grid, on which modelling equations (i.e. power flow equations) are defined are valid bus by bus. In this context we can still use domain decomposition techniques to further sub-divide the power grid graph into sub-problems. Notice that usually DDMs are used to decrease the computational complexity of the problems. Again, here we propose to use them specifically to implement a distributed approach. In the previous sections we presented some preconditioning approaches that exploit a block partitioning of the coefficient matrix A to break the starting linear system down into linear problems that are easier to solve and parallelise. We then extended the approach introducing the possibility of inputing the data in parallel, hence to avoid the complete share of input information. This has been presented without making any reference to the graph domain. We will see a specific DDM approach, the ASM procedure, that allow to obtain a preconditioner that is practically equivalent to the overlapping block Jacobi procedure seen through Section 2.5.3. The only difference of ASM compared to Overlapping Block Jacobi is that we provide a mathematical background to decompose the problem starting from the graph-domain and building up the preconditioning matrix as a result.

Additive Schwarz Preconditioning

The first Schwarz method proposed an *alternating* technique to solve Boundary Value Problems (BVPs). A BVP problem is a mathematical problem where the unknown quantities are expressed in terms of their derivatives through a differential equation, with reference to a domain of interest.

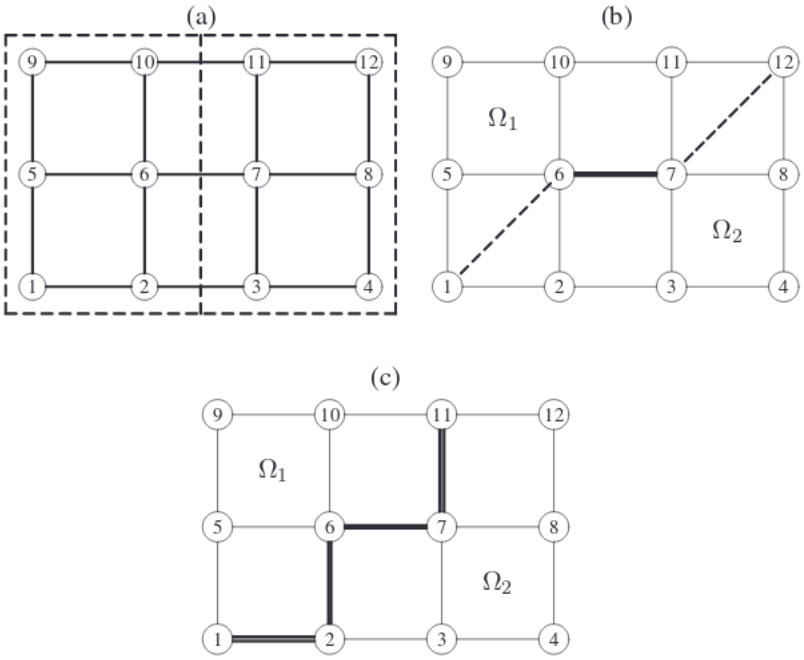
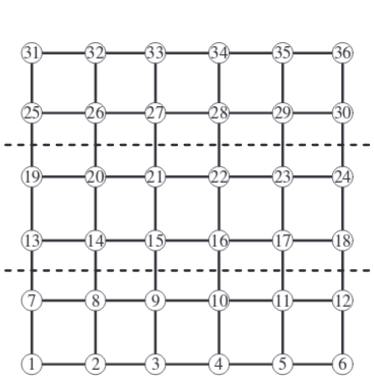
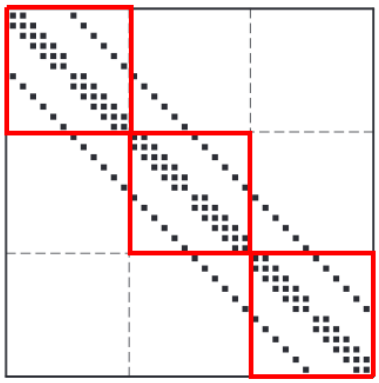


Figure 8: Type of partitionings. a) Vertex-based, b) Branch-based c) Element-based. (Saad, 2003)

Differently from algebraic problems, where it is sufficient to have same number of (independent) equations and unknowns, to find a unique solution of a BVP some *boundary conditions* are needed as well. Schwarz demonstrated how is possible to solve a BVP on a generic domain, by sub-dividing it into sub-BVP and iterating over solutions obtained on such sub-domains. This technique is nowadays often used as a preconditioner to decrease complexity of harsh computational problems. A nice wrap up of Schwarz techniques can be found in (J. GANDER, 2008). Here we limit ourselves to a simple example. Let us consider Fig. 9:



(a) Domain decomposition of a structured grid.



(b) Matrix representation and Schwarz preconditioning.

Figure 9: Example of a domain decomposition and its respective matrix representation ((Saad, 2003)).

In Fig. 9a an example of vertex-partitioning of a structured[25] domain into three sub-domains is reported. In Fig. 9b the matrix (we call it A) representing the graph is depicted, where non-zero entries are represented by the black dots and the dashed lines corresponding to sub-domain decomposition give a 3×3 block structure. We can use this decomposition as a preconditioner by simply setting $M = blockdiag(A)$, where *blockdiag* is a proper operator extracting the red surrounded blocks. We have already seen this operator in the context of Block Jacobi iteration, when we introduced the prolongation and restriction operator and as a result we defined the preconditioner in Eq.(93). In DDM we define these operators with reference to the domain decomposition and according to the partition criteria chosen (Fig. 8). Let us present it more formally.

Let consider a DDM that divides the starting domain Ω into S subdomains Ω_i with $i = 1, \ldots, S$, where the Ω_i are not necessarily disjoint (i.e. they can overlap). Then, we define proper *Index Sets* (IS) that gather indexes of either vertices, branches or elements (depending upon the partitioning criteria) to define which objects belong to a sub-domain Ω_i and which not. Using proper nomenclature, the IS S_i with reference to sub-domain Ω_i can be defined as:

$$S_i = \{j_1, j_2, \ldots, j_{n_i}\} \tag{98}$$

Where the j_k (with $k = 1, \ldots, n_i$) are the indexes of elements belonging to subdomain Ω_i. ISs can be used to define the prolongation R_i and restriction R_i^T operators. Let R_i^T an operator from Ω to Ω_i defined as follows: if the generic vector $\mathbf{v} \in \Omega$, then the vector $R_i^T\mathbf{v} \in \Omega_i$. Similarly we define R_i: if the generic vector $\mathbf{v} \in \Omega_i$, then the vector $R_i\mathbf{v} \in \Omega$. We can use these operators to define the matrix representing the sub-graph A_i relative to Ω_i as follows:

$$A_i = R_i A R_i^T \tag{99}$$

And define the residual vector $\mathbf{r}_i$, the unknown vector $\mathbf{x}_i$ and right-hand side $\mathbf{b}_i$ relative to Ω_i as:

$$\begin{aligned} \mathbf{r}_i &= R_i^T \mathbf{r} \\ \mathbf{x}_i &= R_i^T \mathbf{x} \\ \mathbf{b}_i &= R_i^T \mathbf{b} \end{aligned} \tag{100}$$

The problem at the generic sub-domain Ω_i becomes:

$$A_i \mathbf{x}_i = \mathbf{b}_i \tag{101}$$

By solving these sub-problems we can get the solution vector components of $\mathbf{x}$:

$$\mathbf{x} = \sum_{i=1}^{s} R_i \mathbf{x}_i = \sum_{i=1}^{s} R_i A_i^{-1} \mathbf{b}_i = \sum_{i=1}^{s} R_i A_i^{-1} R^T \mathbf{b} = \sum_{i=1}^{s} R_i (R_i A R_i^T)^{-1} R^T \mathbf{b} \tag{102}$$

As the sub-problems are decoupled (or a little coupled through overlapping variables) the Eq.(102) does not deliver the exact solution. We can thus define a fixed-point linear iteration as:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \sum_{i=1}^{S} R_i^T A_i^{-1} R_i (\mathbf{b} - A\mathbf{x}) \tag{103}$$

[25] A *structured* domain (or grid) is a graph where the position of each element follows some regularity principle. In other words we can find the position of a node (and its connectivities) by means of a mathematical expression. In contrast, an *unstructured* grid is a graph where the position of a generic node must be given through a precise connectivity relationship (e.g. a graph incidence matrix). Electrical grids, which are represented through graphs, are typically unstructured grids.

Which compared Eq.(103) with Eq.(70) gives the expression for the ASM preconditioner:

$$M_{ASM}^{-1} = \sum_{i=1}^{S} R_i^T A_i^{-1} R_i \tag{104}$$

As for the Block-Jacobi case, one do not need to explicitly build up the preconditioner matrix. It is possible to set up a procedure to input the sub-domains on each node of a parallel machine and define locally the sub-problems. The ASM procedure allows to keep domains decoupled and to build up local problems based on the domain owned. In case of overlapping variables, message passing are needed to add up spaces to **b** and as many rows as is the number of overlapping variables. Another message passing operation will be needed to check the global norm of the residual vector. The ASM procedure is simple and provides a basis for minimum exchange of communications among nodes (sub-domains). In practice, accelerators are needed to enlarge the area of convergence and to obtain reasonable time of computation. In the next section we provide an example of the distributed methodology applied to a simple case.

2.5.4 Distributed Power Flow

In this section we provide an in-dept explanation, with a concrete example, of the different steps of the methodology to be followed in order to build a distributed power flow. Wet consider an example of partitioning applied to a simple grid case: case13 (see Fig. 10).

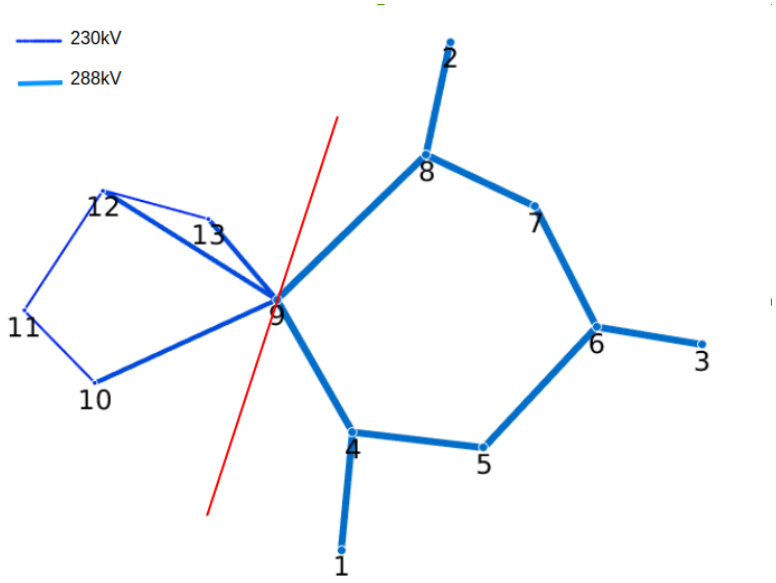


Figure 10: Topology of case13, obtained by merging case9.m and case5.m from Matpower libraries (Zimmerman and Murillo-Sanchez, 2016).

In the classic application of a DDM the domain is divided into sub-domains that are possibly assigned to different processors of a parallel machine. For instance, one can apply the branch-partitioning depicted in Fig. 10 with reference to the case13, and divide it into case5 and case9. The case5 and case9 are then assigned to different processors according to the partitioning criteria chosen. As the aim is to protect grid operators from information sharing about their input data, we shall adopt a different strategy here. We consider again only two machines. The parallel solution of power flow problem follows an approach that can be conceptually divided in two stages:

1. **Set-up Phase**: In this phase the data are locally read and stored in each of the two machines. The network layout is set up and the two networks are connected through a shared bus (or multiple) and additional lines are also considered.

2. **Solving Phase**: In this phase the Power Flow problem is defined and a suited solver is called up.

In this section we focus on the solving phase. The **Set-up Phase** will be described in detail in the chapter (Section 4) after that the structure and functioning of PETSc has been provided. Before entering into that, let us recall some key concepts from the previous sections that are important for this phase:

- The power flow problem is a non-linear problem, hence we need a solver that can deal effectively with non-linearity;
- Proper use of preconditioning eases the solution of linear systems and attenuates numerical errors due to floating-point operations;
- Proper use of accelerators eases the solution of linear systems by allowing convergence in few iterations;
- If the combined use of preconditioner-accelerator involves only operations that can be carried out in parallel (MVM, norm computation), we can compute the majority of them locally and exchange only few information (minimising the inter-process communication);

In (Rinaldo et al., 2018) we show that the Newton-Krylov-Schwarz (NKS) approach is a good candidate to meet the conditions mention above. The NKS uses a Newton approach to initiate an iterative procedure that gives out a succession of linear systems to solve in the Jacobian coefficient matrix, as shown in Section 2.3. Then, each linear system is passed to an inner linear iterative procedure, the Krylov-Schwarz. Krylov works as an accelerator for the iterative procedure, according to the concepts presented in Section 2.4, whereas Schwarz carries out the blocks and acts as a preconditioner. When run in parallel on two machines, assuming parallel input of data, the solving phase with reference to Fig. 10, proceeds by initiating a non-linear procedure (Newton with Line-Search) and subsequently an inner linear procedure (Krylov-Schwarz) on each machine with reference to the owned problem (case9 and case5 respectively in this example). Depending on whether the left or right preconditioning formulation is used, the preconditioner is applied outside or inside the accelerator procedure, respectively. For instance, let us suppose that we choose as Krylov solver the right preconditioned GMREs (see Algorithm 4) by Schwarz and overlap=1 (i.e. we share only one bus between the two parties, bus 9 in this case). The GMREs algorithm starts by building a Krylov Subspace into which the search for an approximated solution of the linear problem is carried out. Schwarz is applied inside the Arnoldi process and at solution update (step 3 and step 11 of Algorithm 4, which means, we form locally the block of the linear problem, again with only reference to the topology owned. What happens at interfacing bus 9? Each processor computes its state variables, based on its owned topology and input production and demand profiles. In other words, at the end of the linear sweep, the bus 9 has different values for the state variables defined. At this stage, the update on the solution of bus 9 is exchanged among both the processors and the solution averaged according to Eq.(97). The iteration starts again taking this value into account.

What happens if we set the overlap to be equal to 2? A different bus (other than 9) is shared from both sides. In our case13 example, setting the overlap=2 would make the bus 8 and the bus 10 (as well as the connecting branches and the electrical quantities related to them) to be shared among processors. This means that the power flow problem is solved on both sides including bus 10

and 8 and by following the same procedure explained for the case with only one bus overlap. Notice that the DDM aspect of Schwarz is a bit hidden in the latter description. In practice, this takes partially place through the **Set-Up Phase**, where the graphs of sub-networks are built locally on each processor, rather than on a master machine and then distributed to the slave machines. Based on this step, the solver *understands* that the local network is actually part of a bigger network shared with another entity and builds the solution procedure accordingly. In this sense, the Jacobi blocks are built up starting from the local graph, but knowing that each block is mathematically part of a bigger problem, exactly as it would be done after a proper local 'decomposition' on a master machine and send through to a slave machine. Another mention goes on parallel communication analysis. This has been done in this work strictly with reference to stationary block-preconditioned methods 2.5.3. In practice adding up accelerators (i.e. Krylov solvers) causes the procedure to be more complex and structured in more steps (see Annex 2 for GMREs full algorithm). A precise analysis of parallel communications of the Krylov-Schwarz solver shall be employed to understand exactly the information passed through machines. On the other hand, a Krylov accelerator involves mainly MVM and norm computations which can be easily carried out in parallel. This means the computation can be carried out by performing only few collective operations. Further analysis on parallel communications of Krylov-Schwarz solvers are let to future works.

3 PETSc

Almost anything that relates to natural phenomena is described through complex time-varying, and sometimes non linear, PDEs. Their solution is obtained in scientific computing through accurate discretisation techniques, that approximate the PDE models locally, turning the problem into a set of algebraic equations. Even though systems of algebraic equations are much easier to solve when compared with differential ones, the design of effective numerical techniques is not a simple task. Such issues can be of the following type:

1. Problems should be addressed with the best tailored techniques, in order to decrease the computational complexity and to obtain good levels of method's robustness;

2. parallelisation is almost always needed due to the computational complexity of the problems, which often makes the implementation of certain numerical techniques non-trivial.

Tackling both these issues is still a big challenge for many scientific problems. Numerical scientists struggle nowadays for developing scientific computing tools that address those problems. Among the many libraries developed in the world there is the Portable, Extensible Toolkit for Scientific Computation (PETSc). PETSc is "a suite of data structures and routines for parallel solution of complex scientific problems" according to the definition that its developers at Argonne National laboratory give. This set of libraries has been developed purposely for parallel computing applications. PETSc is meant to be used both as a solver and as an API to program with. This allowed many libraries to be developed starting from PETSc, such as: Hypre, SuperLU, Trilinos and many others. PETSc has been used for simulations in a broad range of subjects: fluid dynamics, material science, rocket science and many others. It can be used for application codes in both C, C++, Fortran and Python. The libraries enable easy customisation and extension of both algorithms and implementations. This approach promotes code reuse and flexibility, and separates the issues of parallelism from the choice of algorithms. Moreover, PETSc successfully handles the complexities introduced by MPI (message passing interface) by managing completely on itself the message passing operations.

3.1 PETSc Structure

PETSc consists of a variety of libraries and each library manipulates a certain family of objects. The organisation is hierarchical as shown in Figure 11. The most simple objects (Matrices, Vectors and Index Sets) are built on top of BLAS (Basic Linear Algebra Subprograms) and MPI. BLAS is a portable library of routines for performing matrix-vector operations while MPI is the interface that manages inter process communications. For more information about MPI see (Pacheco, 1998).
On top of Matrices, Vectors and Index Sets, user can find KSP (Kryolv solvers) and PC (Preconditioners) family objects. KSP consists of over thirty Krylov subspace methods, for iterative solution of linear systems. PC objects are used in conjunction with KSP to perform preconditioned linear iterations. KSP is integrated with underlying direct solvers (LU factorisation, QR factorisation) that are used to solve subproblems generated by either the KSP iteration scheme or by domain decomposition provided by PC.
The most complex objects built in PETSc are represented by SNES (Scalable Nonlinear Equations Solvers) and TS (time-steppers) solvers, that are built on top of KSP and PC. SNES is a collection of nonlinear solvers and TS is used for solving time-dependent PDEs.
All the solvers available can be simply selected by the user at runtime. A more detailed explanation of PETSc routines can be found in the Users Manual (Balay et al., 2018a). In the next sections further information on PETSc application codes and solvers will be given and on how the program copes with large networks.

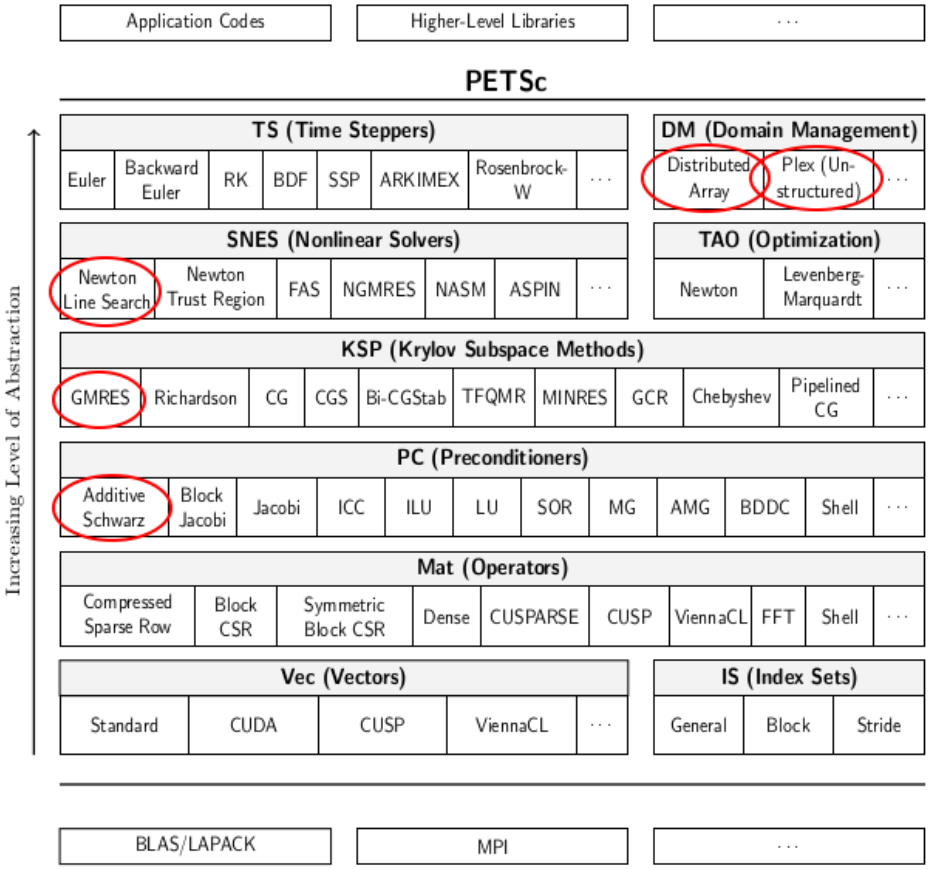


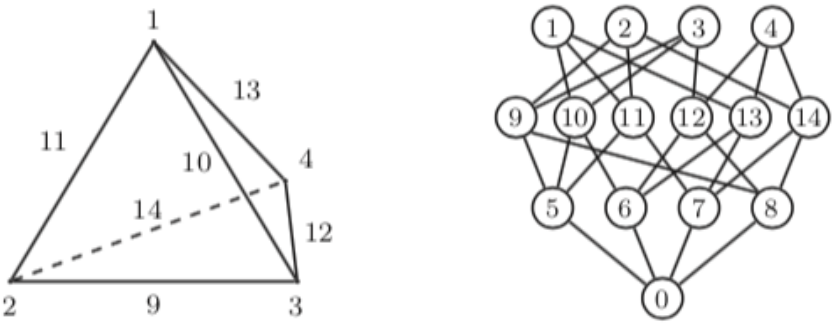
Figure 11: PETSc hierarchy. The red encircled solvers are those relevant for the Power Flow problem and dealt in this work.

3.2 PETSc representation of networks: DMNetwork and DMPlex

This section shows how PETSc manages structured and unstructured grids. Networks are used commonly to represent power distribution systems, water and gas distribution, communication, transportation or electric circuits. DM (Domain Management) are libraries used for managing interactions between mesh data structures and mathematical objects like vectors and matrices. DMNetwork is a class of functions in PETSc that is used for network representation. It allows the user to manage the topology and the physics of large scale networks and the coupling with linear, non linear or time variant solvers. In (Maldonado et al.,) different application frameworks are presented. DMNetwork data structure in PETSc provides efficient and convenient ways for network decomposition. It is possible to split the domain in n sub-domains (through a domain decomposition) or divide a multi-physics system into different single physical subsystems (field-split).
In order to build a network object, all the components have to be recorded. In power system, the network components are mainly of four types: buses, branches, generators and loads (protection devices are not taken into account at this stage). Once those information are recorded DMNetwork builds a mathematical model that could be a set of linear, non linear or time-dependent equations. The system of equations is then solved using KSP, SNES or TS libraries.
Some features of this framework include:

- the possibility of assigning different degrees of freedom to nodes (buses) or edges (branches);
- the possibility of assigning network components (e.g. loads or generators) to each bus;
- the support on network partitioning among different processors;
- the creation of the Jacobian operator (associated to the network) as well as the vectors for residual evaluation;
- the storing of information on ghost nodes that need to perform communication with the other processors;
- the full compatibility with KSP, SNES and TS solvers.

DMNetwork is actually built on top of DMPlex, a data management object that embeds the topology of unstructured grids to provide a range of functionalities common to many scientific applications (Lange et al., 2015b). The topological connectivities between the components in DMPlex are expressed as a Directed Acyclic Graph (DAG). A DAG is a graph where vertices are connected through edges, each edge is directed from a vertex to another so that there is no way to start from a vertex and loop back to it again.



(a) Edge and vertex numbering **(b)** Connectivity in a DAG

Figure 12: Example of a tetrahedron and its respective DAG representation (Lange et al., 2015a).

Fig. 12 illustrates a Directed Acyclic Graph representation of a simple tetrahedron. Component's enumeration starts from the entire figure (0), vertices are enumerated from 1 to 4, faces from 5 to 8 and edges from 9 to 14. The DAG is divided in layers and each component belongs to a certain layer. Once the DAG is built there are several operations inside DMPlex that allow the user to extract elements of a grid, see connectivities, mapping data layout and distribute data among parallel processors.
Moreover DMPlex provides other functionalities for grid partitioning and domain decomposition. There are graph partitioning methods already implemented in PETSc while many others can be found in external libraries such as Chaco (Hendrickson and Leland, 1993) or ParMetis (Karypis et al., 1997). Six different partitioners can be chosen. They allow the user to decompose the problem in the most suitable way for a certain application code; by default PETSc decomposes the grid in such a way that all the processors own more or less the same number of elements.
Once that creation and distribution steps are carried out, a network object contains:

- a partitioned graph representation of the problem

- a memory space allocation for the problem's unknowns in each vertex and edge
- a set of physical data in each bus
- an implicit nonzero structure determination of the Jacobian matrix

3.3 Solving Power Flow Equations using PETSc

In Section 2 the power flow problem has been presented and the most suitable numerical techniques used over the course of time have been briefly discussed. It has been shown how the power flow problem needs a proper numerical method to handle the non-linearity and an inner linear solver. In PETSc, it is done by setting an outer SNES solver and an inner KSP solver, possibly preconditioned through a PC object. An example written in C (power.c) to solve power flow problems in PETSc can be found in the following folder:

```
$PETSC_DIR/src/snes/examples/tutrtials/network/power/
```

The C code can be divided in three main stages:

1. Reading stage
2. Set up stage
3. Solving stage

Reading Phase

The input file is a Matpower test case. In this example the reference grid is case9.m which can be found in the Matpower libraries.
`MATPOWER` is a package of free, open-source Matlab-language M-files for solving steady-state power system simulation and optimisation problems such as power flow (AC and DC) or Optimal Power Flow (OPF).
Matpower includes also a set of test cases, ranging from small to large size power grids, up to close approximations of the European High Voltage Transmission Grid. Details about entries of Matpower caseformat can be accessed from `MATLAB` by typing `help caseformat`. Cases are made of 6 fields, 4 of which are structs. Those structs include data for buses, generators, branches and generator costs. For more details on `MATPOWER` test cases the reader can consult (Zimmerman and Murillo-Sanchez, 2016). In the following the categories of each matrix in data struct are shown for sake of clarity. Bus struct is a $n \times 13$ matrix (where n is the number of buses) containing the following columns:

```
bus_i type Pd Qd Gs Bs area Vm Va baseKV zone Vmax Vmin
```

Generator data matrix is a $n_{gen} \times 21$ matrix, where n_{gen} is the number of generators, divided as follow:

```
bus Pg Qg Qmax Qmin Vg mBase status Pmax Pmin Pc1 Pc2 Qc1min Qc1max Qc2min
    Qc2max ramp_agc ramp_10 ramp_30 ramp_q apf
```

Branch data matrix is a $m \times 13$ matrix where m is the number of branches

```
fbus tbus r x b rateA rateB rateC ratio angle status angmin angmax
```

Finally generator costs data matrix has the following structure:

```
1 startup shutdown n x1 y1 ... xn yn
2 startup shutdown n c(n-1) ... c0
```

Additionally MATPOWER libraries include since 2015 a list of cases which correspond to real world cases. In (Josz et al.,) different test cases are presented: RTE test cases represent the French very-high voltage grid, while PEGASE test cases give a snapshot of the European high-voltage grid. Those data can be read by PETSc application code power.c, previously mentioned. Note that they should be provided as an input at runtime.
From an implementation point of view the first part of the code is the following:

```
/* READ THE DATA */
if (!crank) {
/* READ DATA */
/* Only rank 0 reads the data */
ierr = PetscOptionsGetString(NULL,NULL,"-pfdata",pfdata_file,PETSC_MAX_PATH_LEN
    -1,NULL);CHKERRQ(ierr);
ierr = PetscNew(&pfdata);CHKERRQ(ierr);
ierr = PFReadMatPowerData(pfdata,pfdata_file);CHKERRQ(ierr);
User.Sbase = pfdata->sbase;
numEdges = pfdata->nbranch;
numVertices = pfdata->nbus;
ierr = PetscMalloc1(2*numEdges,&edges);CHKERRQ(ierr);
ierr = GetListofEdges_Power(pfdata,edges);CHKERRQ(ierr);
}
```

The conditional operator if(!crank) implies that only rank 0 processor is in charge of reading data. Using PetscOptionsGetString() it is possible to provide a string for a particular option in the database. In this case the string "-pfdata" allows to select the test case that is going to be used at runtime. An empty PETSc object is then created using PetscNew() function. The funtion PFReadMatPowerData() takes as an input the datafile provided from Matpower library and fills all the fields in pfdata struct, which are:

```
typedef struct{
PetscScalar sbase; /* System base MVA */
PetscInt nbus,ngen,nbranch,nload; /* # of buses,gens,branches, and loads */
VERTEX_Power bus;
LOAD load;
GEN gen;
EDGE_Power branch;
} PFDATA
```

The fields VERTEX_Power, LOAD, GEN and EDGE_Power are structs as well and they are declared in power.h header file. When all the fields are correctly filled the function GetListofEdges_Power() creates an array edges of size $2m$ where m is the number of branches. It is created in this way:

```
for (i=0;i<m;i++)
{
edges[2*i]= from_bus; /* the first column values in mpc.branch struct */
edges[2*i+1]= to_bus; /* the second column values in mpc.branch struct */
}
```

After this procedure the reading phase can be considered complete.

Set Up Phase

Independently from the application code and from the type of mathematical problem to solve, the Set Up phase has a standard procedure in PETSc when dealing with networks.

```
/* Create an empty network object */
ierr = DMNetworkCreate(PETSC_COMM_WORLD,&networkdm);CHKERRQ(ierr);

/* Register the components in the network */
ierr = DMNetworkRegisterComponent(networkdm,"branchstruct",sizeof(struct
    _p_EDGE_Power),&User.compkey_branch);CHKERRQ(ierr);
ierr = DMNetworkRegisterComponent(networkdm,"busstruct",sizeof(struct
    _p_VERTEX_Power),&User.compkey_bus);CHKERRQ(ierr);
ierr = DMNetworkRegisterComponent(networkdm,"genstruct",sizeof(struct _p_GEN),&
    User.compkey_gen);CHKERRQ(ierr);
ierr = DMNetworkRegisterComponent(networkdm,"loadstruct",sizeof(struct _p_LOAD)
    ,&User.compkey_load);CHKERRQ(ierr);
```

An empty network object and a component library have to be created first. This is done through the function `DMNetworkCreate()` and `DMNetworkRegisterComponent()`. A "component" is a specific data at any node/edge of the network required for its residual evaluation. For example, components could be resistor, inductor data for circuit applications or generator/transmission line data for power grids. In this case components are buses, branches, generators and loads. The function `DMNetworkRegisterComponent()` stores component keys in the struct `User` which has the following fields:

```
struct _p_UserCtx_Power{
PetscScalar Sbase;
PetscBool jac_error; /* introduce error in the jacobian */
PetscInt compkey_branch;
PetscInt compkey_bus;
PetscInt compkey_gen;
PetscInt compkey_load;
}
```

`compkey` is an integer key that can be used for setting/getting the component at a node or an edge. The set up phase continues setting network size and edge connectivity:

```
/* Set number of nodes/edges */
ierr = DMNetworkSetSizes(networkdm,1,0,&numVertices,&numEdges,&NumVertices,&
    NumEdges);CHKERRQ(ierr);
/* Add edge connectivity */
ierr = DMNetworkSetEdgeList(networkdm,&edges,NULL);CHKERRQ(ierr);
```

`DMNetworkSetSizes()` sets the number of subnetworks, local and global vertices and edges for each subnetwork. The second and third arguments of the function are the number of subnetworks and the number of coupling network respectively. In this case we have only one network owned by rank 0 processor.
The function `DMNetworkSetEdgeList()` takes as an input edges array and copies it in: `networkdm->data->subnet->edgelist`.
The layout of the network is then set calling:

```
/* Set up the network layout */
ierr = DMNetworkLayoutSetUp(networkdm);CHKERRQ(ierr);
```

Once that a bare layout is defined components and number of variables can be set in each node/branch. The process in charge of adding components and variables is rank 0 process again.

```
if (!crank) {
genj=0; loadj=0;
ierr = DMNetworkGetEdgeRange(networkdm,&eStart,&eEnd);CHKERRQ(ierr);
for (i = eStart; i < eEnd; i++) {
ierr = DMNetworkAddComponent(networkdm,i,User.compkey_branch,&pfdata0->branch[i
    -eStart]);CHKERRQ(ierr);
}
ierr = DMNetworkGetVertexRange(networkdm,&vStart,&vEnd);CHKERRQ(ierr);
for (i = vStart; i < vEnd; i++) {
ierr = DMNetworkAddComponent(networkdm,i,User.compkey_bus,&pfdata->bus[i-vStart
    ]);CHKERRQ(ierr);
if (pfdata->bus[i-vStart].ngen) {
for (j = 0; j < pfdata->bus[i-vStart].ngen; j++) {
ierr = DMNetworkAddComponent(networkdm,i,User.compkey_gen,&pfdata->gen[genj++])
    ;CHKERRQ(ierr);
}
}
if (pfdata->bus[i-vStart].nload) {
for (j=0; j < pfdata0->bus[i-vStart].nload; j++) {
ierr = DMNetworkAddComponent(networkdm,i,User.compkey_load,&pfdata->load[loadj
    ++]);CHKERRQ(ierr);
}
}
/* Add number of variables */
ierr = DMNetworkAddNumVariables(networkdm,i,2);CHKERRQ(ierr);
}
}
```

After that all the components have been successfully added, the network is ready to be distributed among different processors.

```
ierr = DMSetUp(networkdm);CHKERRQ(ierr);
```

This function has to be called every time as it provides a signal that the network is ready to be distributed. The distribution is made by:

```
ierr = DMNetworkDistribute(&networkdm,0);CHKERRQ(ierr);
```

By default PETSc distributes the network object in such a way that each processor owns more or less the same number of elements. The distribution process can be modified changing the partitioner used by `DMNetworkDistribute()`.

Solving Phase

Once that each processor has received its sub-problem, the solving phase starts. The master processor communicates the solving procedure to use and a local solver is built up on each processor.

The PETSc library provides three layers of solvers: KSP, SNES, and TS to solve these problems, respectively.

Table 2: PETSc Solvers

Solver Name	Description	Mathematical Form
KSP and PC	Krylov Subspace Methods and Preconditioners	$Ax - b = 0$
SNES	Non Linear Solver	$F(x) = 0$
TS	Time Stepping Solver	$F(t, x, \dot{x}) = 0$

As already mentioned, in power flow context according to steady state approximation a non-linear system of equation of the type $F(x) = 0$ has to be solved. Using Newton's method the problem can be approximated to a set of linear equation:

$$\begin{aligned} J(x_k)\Delta x_k &= -F(x_k) \\ x_{k+1} &= x_k + \Delta x \end{aligned} \tag{105}$$

The function $F(x_k)$ and the Jacobian matrix $J(x_k)$ have to be evaluated at each iteration and Eq.(105) has to be solved iteratively to obtain the approximated solution x_k.
PETSc takes care of parallel distribution, preallocation, partitioning and setting up data structures. The user needs to provide two functions, one for $F(x)$ evaluation and one for the Jacobian $J(x)$.
The application flow starts creating a non linear solver context and adapting DMNetwork object to it. After that both residual function and Jacobian have to be created. The non linear problem is then ready to be solved.

```
/* HOOK UP SOLVER */
ierr = SNESCreate(PETSC_COMM_WORLD,&snes);CHKERRQ(ierr);
ierr = SNESSetDM(snes,networkdm);CHKERRQ(ierr);
ierr = SNESSetFunction(snes,F,FormFunction,&User);CHKERRQ(ierr);
ierr = SNESSetJacobian(snes,J,J,FormJacobian_Power,&User);CHKERRQ(ierr);
ierr = SNESSetFromOptions(snes);CHKERRQ(ierr);
ierr = SNESSolve(snes,NULL,X);CHKERRQ(ierr);
```

With the function `SNESSetFromOptions()` it is possible to set parameters for non linear solver at runtime. For example it is possible to set the maximum number of iterations, absolute and relative tolerances, type of solver (Newton, Richardson, GMREs) and many others. Once that `SNESSetFromOptions()` is invoked all the underlying layers' options are enabled.
To summarize, `rank 0` processor takes as an input a `Matpower` test case and saves it into a struct. It builds a network object and distributes it among the processors involved in the computation. Power flow equations are then solved using Newton Raphson method coupled with a direct preconditioned solver. Both linear and non linear methods can be chosen between all the ones implemented in PETSc libraries.

4 Distributed Power Flow in PETSc

In the previous section a broad overview of the PETSc functioning principles was given. We presented how PETSc manages power grid structures and sets up solvers once all network related objects have been correctly defined. This section is devoted to a comprehensive overview of the distributed power flow solver in PETSc. We will see in this section how the Set Up process needs to be modified in order to keep input data locality. Solvers are still created locally on each processor and work out the solution with reference to the owned network and coupling with other processors' sub-problems through overlapping variables.

4.1 Introduction

Newton coupled with a preconditioned Krylov method that allows domain decomposition is particularly interesting in power flow framework. As described in the previous section, `power.c` example allows to solve power flow problem using different techniques suitable for parallel or distributed computer environments. Those kind of methodologies (Idema et al., 2012) can provide a better scalability and performances compared to classical approaches when problem size increases.
As mentioned, the common goal these techniques is that of being able to solve large and complex grids into a reasonable computational time. `DMNetwork` class provides an interface to cope with large scale graphs that usually arise in power system. Despite numerical methods needed in our framework are already implemented, `DMNetwork` routines have some limitation which needs to be tackled in order to run according to our novel/different approach.
The aim of this section is thus not that of explaining how to better perform in terms of computational speed in PETSc, but rather that of validating the methodology presented in Section 2 by discussing its implementation in PETSc.

4.2 Parallel communication analysis of set up phase in PETSc

The general flow of an application code using `DMNetwork` can be summarised in Figure 13.

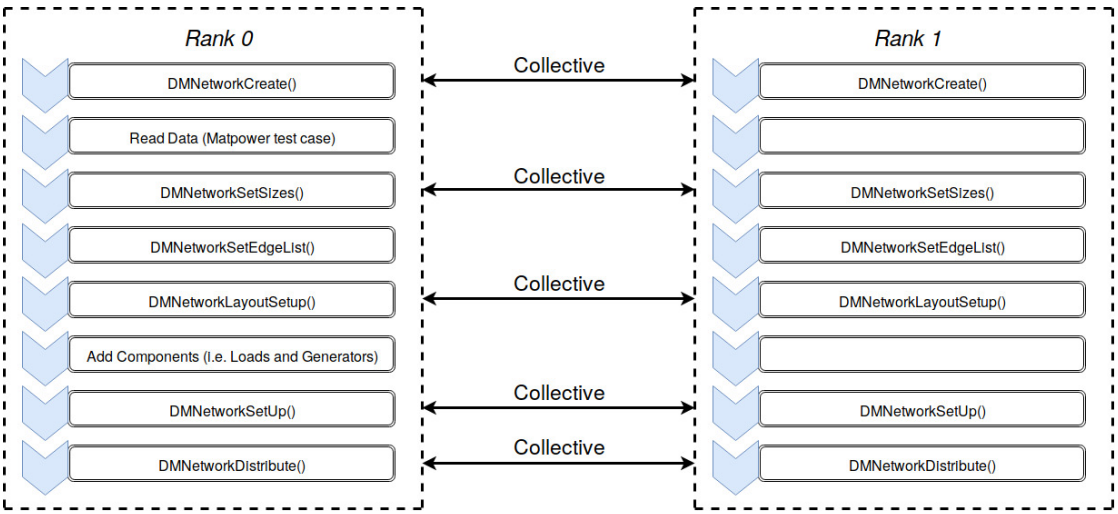
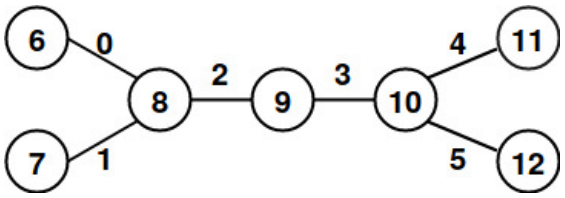


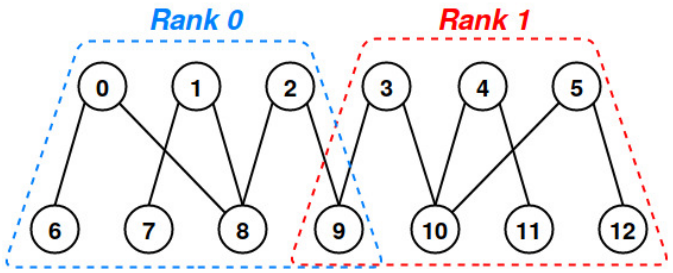
Figure 13: Parallel code flow steps of DMNetwork.

It shows how two processors cooperate to create and build a network object. During the building process it is possible to find collective and non-collective functions. The first ones have need to be executed simultaneously by both processors while the second ones do not require any information

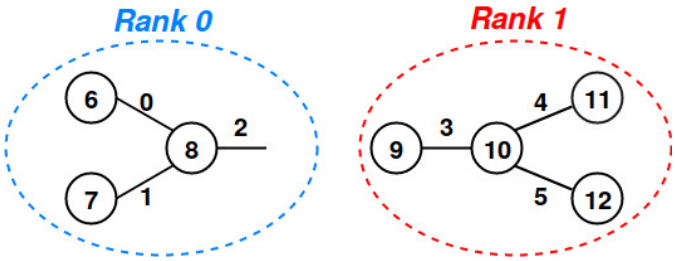
exchange and can be executed separately. "Read Data" and "Add Components" steps are carried out by `rank 0` process which is usually considered as the master node (in a master-client paradigm). In this case only the latter has access to the network input data and until the `DMNetworkDistribute()` function is not called all grid information is kept by the `rank 0` node. Successively (after the call) graph elements and associated data are distributed among the nodes. Fig. 14 shows how PETSc manages and distributes a simple seven buses network.



(a) Simple 7 buses network. PETSc enumeration



(b) Directed Acyclic Graph (DAG) for the network above



(c) Splitted network on two processors. Shared elements are moved to higher rank processor

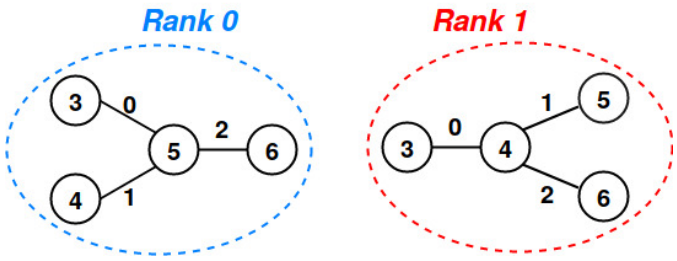
Figure 14: Network management in PETSc

Network components enumeration starts from edges and it continues with buses. All components are registered in a Direct Acyclic Graph (DAG). Branches are stored in the first layer (first row in Figure 14b) while buses in the layer below. Network decomposition is done through edges. According to Figure 14b, branches $\{0, 1, 2\}$ are assigned to `rank 0` and $\{3, 4, 5\}$ to `rank 1`, buses are allocated accordingly to edge partition. By default shared elements (bus 9 in this case) are moved to the higher rank processor.

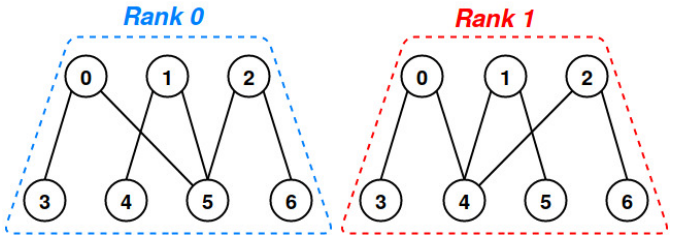
The two nodes involved can be considered as two different entities at national or international level. Within this framework the asymmetry of information is evident as one entity owns all the data regarding topology, generation and demand profiles.

4.3 Parallel Assembling procedure

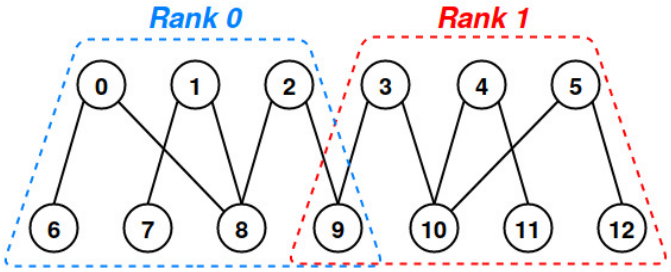
In order to obtain a distributed input of data as well as a parallel assembling of the network some modifications are indeed necessary. Fig. 15 illustrates the idea behind a parallel assembling procedure. Input data are located remotely on two different workstations. Each processor takes as an input a case data file (its own data network). In reality, bus 6 on `rank 0` and bus 3 on `rank 1` in Figure 15a refer to the same bus (bus 9 in Figure 14a).



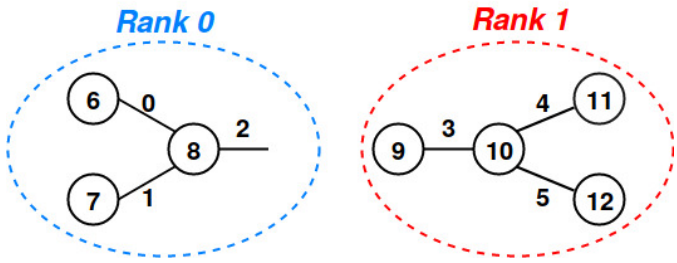
(a) Simple 7 buses network where input data are on two different nodes



(b) Directed Acyclic Graph (DAG) for the previous network: in the default set up procedure DAGs are not connected



(c) Directed Acyclic Graph (DAG) for the previous network using a parallel assembling procedure



(d)

Figure 15: Network management in PETSc using a parallel assembling procedure

If the Read Data function is not a collective operation, the two networks are considered two separate grids instead of being two portions of the same network. In other words DAG in Figure 15b does not have any connectivity between ranks. This has to be modified in order to obtain a graph like the one in Figure 15c.
To make this, edge layer identifiers need to be contiguous. In this sense the number of branches on `rank 0` has to be added to every branch identifier on `rank 1`. Thus Reading phase needs collectivity in order to exchange information regarding the total number of branches. `Rank 0` sends to `rank 1` an integer corresponding to the total number of branches on master node, `rank 1` changes branch enumeration accordingly.
Bus layer identifiers change their enumeration during the layout set up step. Among other tasks, the function `DMNetworkLayoutSetUp()` overwrites bus indexes in order to obtain a consecutive enumeration right after for branches. The modified flow of the application code using parallel assembling procedure is shown in Fig. 16.

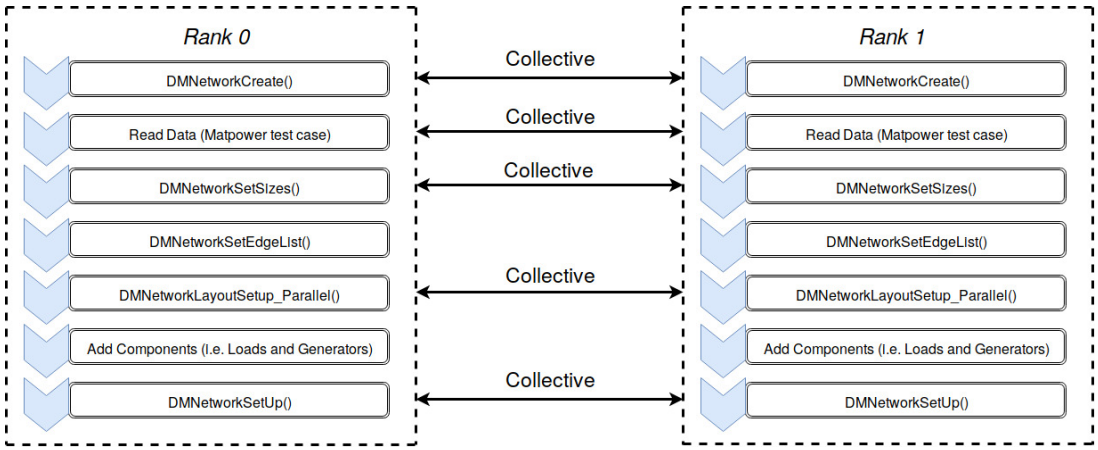


Figure 16: Modified flow

The main differences that can be highlighted are the following:

1. Both nodes execute all the functions
2. The Read data phase is collective
3. The Layout set up of the network allows a parallel assembling procedure

In a wider context this application flow can be seen as two entities running a coordinated power flow. Each entity has its own remote control centre where information regarding topology, demand and generation profiles are stored. The two grids are connected through a certain number of interconnection branches thus the solution of one sub-network is dependent on the characteristics of the neighbouring grid.

4.4 Solving Phase

After that the previous procedure has been completed, the network elements and corresponding DAG are correctly set up among the processes involved in the computations. It is not necessary to invoke `DMNetworkDistribute()` since the network is already correctly split between the processors. Power flow equations are then solved iteratively, voltage magnitudes and phase angles are found at each node of the network. It is important to stress that the solution obtained in this way is the same that it will be obtained by one entity running a global power flow on the two sub-networks together.

In order to prove the well-functioning of this concept a test case was set. Two small test cases in MATPOWER libraries have been selected: `case9.m` and `case5.m` respectively. They have been merged together considering that the first bus in `case5.m` corresponds to the ninth bus in `case9.m`. The resulting network is illustrated in Figure 17. Bus 1 is the slack node, buses $2, 3, 11, 12, 13$ are PV buses while the others are PQ buses. Input data were distributed among two different processors in the configuration shown in Figure 18.

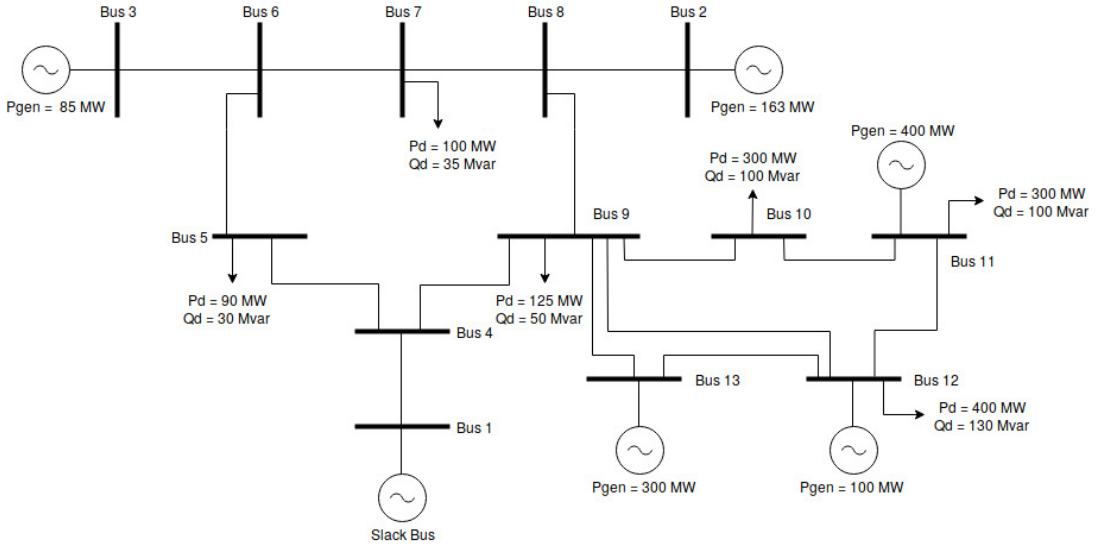


Figure 17: case13.m

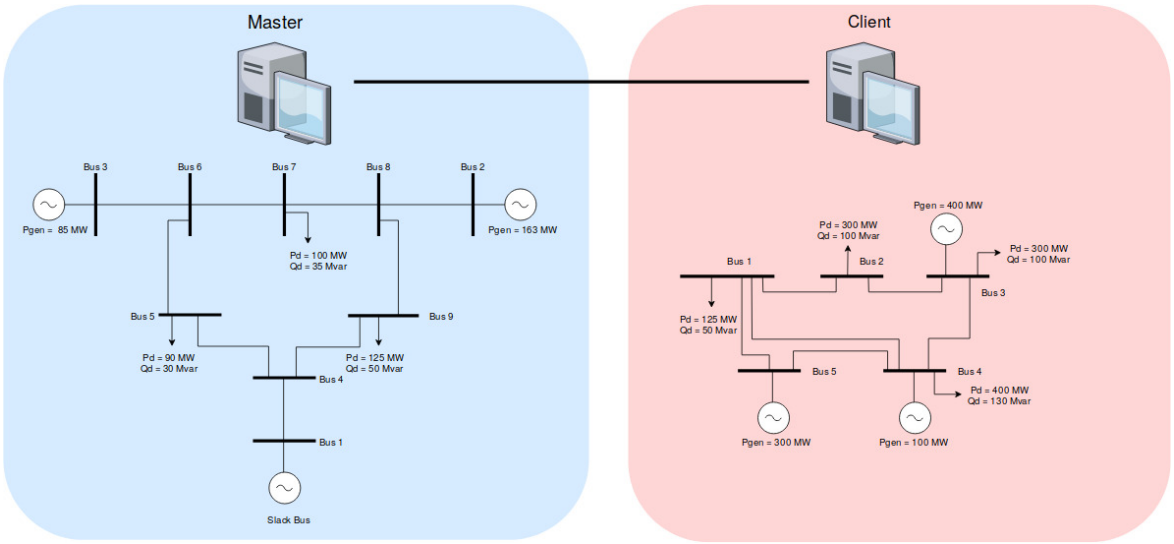


Figure 18: case13.m divided in two workstations

`case9.m` was assigned to master node and `case5.m` to client node. Reading and set up procedures follow the steps previously described. Once the network object has been correctly set up, power flow equations are solved by SNES library in PETSc. The Jacobian linear system is solved using GMREs preconditioned with overlapping Additive Schwarz.

The results in terms of voltage magnitude and phase angles are reported for both processors in Table 3. Bus ownership reflects input data locality. Once the voltage profiles are found as a solution of the power flow problem, active and reactive power flow in each branch can be calculated. Table 4 illustrates the amount of power flows in each branches subdomain. Those data can be easily saved remotely on each workstation depending on the network ownerships.

Table 3: Results: Voltage Magnitude and Phase Angle per node

Bus Number	`rank 0`		`rank 1`	
	Voltage Magnitude	Phase Angle	Voltage Magnitude	Phase Angle
1	1	0		
2	1	-0.07103		
3	1	-0.1086		
4	0.9957	-0.163		
5	0.9839	-0.2153		
6	1.0063	-0.1582		
7	0.9922	-0.2078		
8	1.0047	-0.1726		
9	0.9943	-0.3571	0.9943	-0.3571
10			0.9878	-0.4126
11			1	-0.4021
12			1	-0.4001
13			1	-0.3492

Table 4: Results: Active and Reactive Power in each branch

`Rank`	`Branch`		`From bus injection`		`To bus injection`	
	From	To	P [MW]	Q [MVar]	P [MW]	Q [MVar]
0	1	4	280.613	30.314	-280.613	15.572
0	4	5	56.325	-4.020	-55.779	-8.504
0	5	6	-34.221	-21.496	34.700	-11.876
0	3	6	85.000	-8.736	-85.000	13.015
0	6	7	50.299	-1.139	-49.992	-17.123
0	7	8	-50.008	-17.876	50.233	4.932
0	8	2	-163.000	15.885	163.000	0.721
0	8	9	112.767	-20.817	-108.726	10.573
0	9	4	-219.214	37.259	224.288	-11.551
1	9	10	194.837	8.598	-193.755	1.515
1	9	12	137.839	-29.732	-137.228	35.186
1	9	13	-129.736	-76.696	129.881	75.046
1	10	11	-106.244	-101.515	106.481	102.055
1	11	12	-6.481	0.3175	6.483	-0.979
1	12	13	-169.254	20.953	170.119	-12.984

Note that this approach allows convergence with a tolerance of less than 10^{-8} on the absolute norm[26]. Overlapping variables converge at the same solution on both sides (machines) with the same

[26]The convergence check is done also on relative norm of the residual vector

convergence check above. Another important aspect that can be highlighted is that the solution is found keeping a reasonable convergence time ($4.39\ 10^{-2}$ seconds on two 8-node 32 GB RAM machine, each node being an Intel Xeon E3-1275 3.50 Ghz running on an Ubuntu 17.10 64-bit GNU/Linux distribution and mounting a network card of 1 Gbit/s).
From an implementation point of view, it is wort mentioning that PETSc allows to evaluate simulation performances with profiling options. In fact PETSc routines automatically log performance data and they can be printed using different profiling options. The option `-log_view` prints an overall performance summary, including times, floating-point operations, computational rates and message passing activity.
The bottleneck is usually the type of interconnection chosen between the workstations. In (Rinaldo et al., 2018) some simulation tests by means of Newton-Krylov-Schwarz on a distributed system architecture are presented. Two interconnected European high transmission voltage grids were tested using different type of network configurations. The results showed the feasibility of performing the distributed calculation remotely, keeping the overall simulation times quite low (only few times slower than locally considering two grids in the order of thousands nodes sizes).

5 Applications

This section discusses the potential application framework of distributed/delocalised approaches to power flow equations. In particular, we focus on two cross-border frameworks: 1) the TSO-TSO congestion management in Europe; 2) the TSO-DSO coordination at the transmission-distribution interface. With respect to the first case we focus mainly on the Capacity Calculation Methodology (CCM) currently in place in Europe to tackle congestion management issues. As we will see, this is actually a complex procedure made up of many steps, during which TSOs and various stakeholders collaborate in order to provide the market with the bounds for Capacity Allocation (CA) process.

For the second case the discussion will be more qualitative. As DSOs are currently facing a re-organisation in terms of roles to accommodate RES penetration at distribution level and currently do not generally carry out power flow computations yet, the methodology is presented in a 'softer' way. However, it is worth mentioning that numerous TSO-DSO coordination schemes that require the DSO to run power flow computations are being presented in research and pilot projects (e.g SmartNet, Coordinet), underpinning the importance of the topic at issue.

5.1 Congestion Management at European Level

In order to harmonise and liberalise the EU's internal energy market, measures have been adopted since 1996 to address market access, transparency and regulation, consumer protection, supporting interconnection, and adequate levels of supply. These measures aim to build a more competitive, customer-centred, flexible and non-discriminatory EU electricity market with market-based supply prices. In so doing, they strengthen and expand the rights of individual customers and energy communities, address energy poverty, clarify the roles and responsibilities of market participants and regulators and address the security of the supply of electricity, gas and oil, as well as the development of trans-European networks for transporting electricity and gas(Gouarderes, 2019). A number of big improvements has been already achieved in these years: looking at market integration, for instance, we have today a single price coupling algorithm, called EUPHEMIA (EU + Pan-European Hybrid Electricity Market Integration Algorithm) which calculates electricity prices across Europe (for 27 countries). Operated by eight NEMOs (Nominated Electricity Market Operators), the algorithm allocates simultaneously electricity and capacity between bidding zones[27], a process known as implicit allocation or market coupling. The aim of market coupling is that of maximising economic surplus by using efficiently all the available resources. Inter-connectors are hence fundamental to achieve this goal being the means to connect different European bidding zones.

In theory two exclusive situations should be observed when looking at the flows at the interconnection points between bidding zones: one, in which the inter-connectors are partially used (below their maximum capacity) and the price is the same in the two neighbouring bidding zones; the other, in which the line is congested so that there is a price differential between the two bidding zones, that is a *market splitting*. In reality, what one can see is that very often there are situations in which the inter-connector is only partially used but there is a price differential between the connected bidding zones. Even more paradoxical than that, there were cases in the past mainly due to explicit allocation of capacity between bidding zones in which electricity was flowing from zones of higher prices to those of lower prices.

In (ACER and CEER, 2018), the Agency for the Cooperation of Energy Regulators (ACER) and the Council of European Energy Regulators (CEER) argue that low level of cross-zonal transmission capacity currently observed in their monitoring might be due to congestions not properly addressed by the current bidding zone configuration in Europe. That is, structural congestions might be present

[27] Bidding zones in Europe mainly coincide with national borders, apart from some cases as Sweden and Italy which have four and six bidding market zones, respectively.

within bidding zones rather than on bidding zone borders. Moreover, another reason identified in the same report is the lack of coordination between TSOs concerning the cross-zonal capacity calculation. In order to regulate this kind of matters, four guidelines and four network codes have been adopted at European level, below the umbrella of the Third Energy Package (TEP), currently amended by the recently adopted Clean Energy Package (CEP). In particular, the Capacity Allocation and Congestion Management Guideline (CACM GL) (CAC, 2015), which belongs to the group of market codes[28], sets among other things, the basis for congestion management. At the core of congestion management there are two subsequent sub-processes: the Capacity Calculation and the Capacity Allocation. At market level (with respect to the CA) the process has gradually evolved from explicit auctions of capacity to the more economic efficient implicit auctions. Implicit auctions allow to integrate the grid constraints in the electricity trading process while explicit auctions may give rise to opportunistic behaviour of parties, that may book capacity and prevent others to its use.

In the following sections we focus on the CC step of congestion management process and we assume a MC is in place at CA level. We start by reviewing the main methodologies used in practice: Available Transfer Capacity (ATC) and Flow Based (FB). This will set the basis for a discussion on how distributed approaches might be used in this context.

5.1.1 CA and CC: History and Implementations

If CA is the phase during which cross-border capacity is 'sold' to the market and coupling of bids among zones takes place, the CC stage requires to compute the constraining parameters to pass to market coupling algorithm[29]. Going back, the first experience of MC started in 2006 in the Central West Europe (CWE) area with the trilateral MC among France, Belgium and Netherlands, for the DAM. As electricity flows according to physics and not following commercial transactions, an injection/withdrawal in a bidding zone may cause flows in neighbouring regions. This is particularly true for highly meshed networks (e.g. the CWE) that cause flows to spread all over and even get back to their originating region. The MC mechanism allows to coordinate commercial exchanges among different zones and by checking that these transactions are feasible in terms of compatibility with grid constraints. As the markets are coupled, this also allows to obtain price signals that reflect demand/supply conditions (i.e. need/excess of electricity). The improvement brought by MC solutions increased the participation of more EU countries. In 2007 National Regulatory Authorities (NRAs), TSOs, PXs, Market Parties Platform (MPP) and Ministers of the Penta-Lateral Energy Forum (PLEF) signed a Memorandum of Understanding (MoU) to improve the cooperation in the field of cross-border exchanges. The main objective was the analysis, design and implementation of a FB MC approach within the five countries of the CWE (PLEF, 2007). As an intermediate step towards FB, ATC MC became active starting from 2010. The ATC MC clearing procedure were made through an optimisation algorithm called COSMOS, based on the maximisation of total surplus of consumers and suppliers. Starting from 2013 the CWE eventually moved to an FB approach. In 2015 EUPHEMIA replaced COSMOS, which since then is currently the algorithm in place for DAM clearing. Nowadays as mentioned the DAM is solved for all Europe. At present, some borders are modelled through FB approach, others through ATC and some others are moving from ATC to FB. In the following sections we review the framework and the methodology of CC and CA approaches, agreed by all stakeholders through a long policy process and currently in place in Europe. Several organisations took part to this complex process, so we consider important to provide an overview of the process before going through the technical details of ATC and FB CC approaches.

[28] Network codes and guidelines are divided into three main families: 1) Connection, 2) System Operation and 3) Market.

[29] When we mention the market, we implicitly refer to DAM only.

5.1.2 Relevant CACM articles

Moving back to the CACM, its content can be divided into the following categories:

- Optimal definition of bidding zones;
- Calculation of capacities between bidding zones;
- Allocation of cross-zonal capacities with MC;
- Management of residual congestions with remedial actions (e.g. re-dispatch, counter-trading).

Each one includes sub-sections providing detailed procedures to follow. Member States can then implement them only with little freedom. For major amendments, TSOs must present modifications proposal of the CACM to NRAs and ACER for final approval.

The CACM details how the CC process must be carried out in Europe. The main steps of the regulation includes:

- Requirements for definition of Capacity Calculation Regions (**CCRs**);
- Requirements for definition of a Common Grid Model (**CGM**);
- Generation and Load Data Provision (**CGM**) methodology;
- Capacity Calculation Methodology (**CCM**);
- Coordinated Capacity Calculation (**CCC**) process.

The CCRs are those EU regions where the CC process is meant to be carried out by TSOs. Fig. 19 shows the first coordinated proposal made by all TSOs on CCRs on the. Following to the decision of ACER on the 17th November 2016, the CWE CCR and the Central East Europe (CEE) CCR have been merged into a unique CCR, namely the CORE CCR. Some follow-ups as public consultations among TSOs have been launched to discuss small amendments on CCRs. In general, ACER expects the number of CCRs to decrease over time. Merging CCRs give the advantage of reducing uncertainties of CC process due to inter-dependency (only slightly present) of some CCRs.

In simple wording, the CCRs are nothing but borders among bidding zones[30]. This implies that each TSO acting on the borders must take part to the CC process. The CCC process starts by asking each bordering TSO to prepare an Individual Grid Model (IGM), according to Article 19 of CACM, part of the section dedicated to CGM (Articles 14-19). The IGM is an approximated model of the each TSO's transmission system, relevant for the purpose of CCC. IGMs are then merged to form the CGM. Notice that Article 19 provides several requirements to the IGMs. In practice, an IGM must include all the ingredients to allow load flow computations on it (i.e. generation and load profiles, topology; all updated according to the best possible forecasts). Article 16 develops requirements of the GLDP procedure and requires "TSO to jointly develop a proposal for the delivery of the generation and load data required to establish the CGM". Articles 20-26 provides the CCM, in short what parameters (Critical Network Elements (CNE) and Contingencies (CNEC), Generation Shift Keys (GSKs), Power Transfer Distribution Factors (PTDFs), Reliability Margins (RMs), Remedial Actions (RAs)) to compute and how. Articles 27-30 provide more details on the CC based on ATC and FB approaches.

Currently, the process to collect the information from TSOs, to merge the IGMs into the CGM and to carry out the capacity calculations is under the responsibility of the Regional Security Coordinators (RSCs). These are companies owned or controlled by TSOs. RSC are also in charge of proposing

[30] Note that a bidding zone border may have more than one TSO operating its own control area (e.g. Germany).

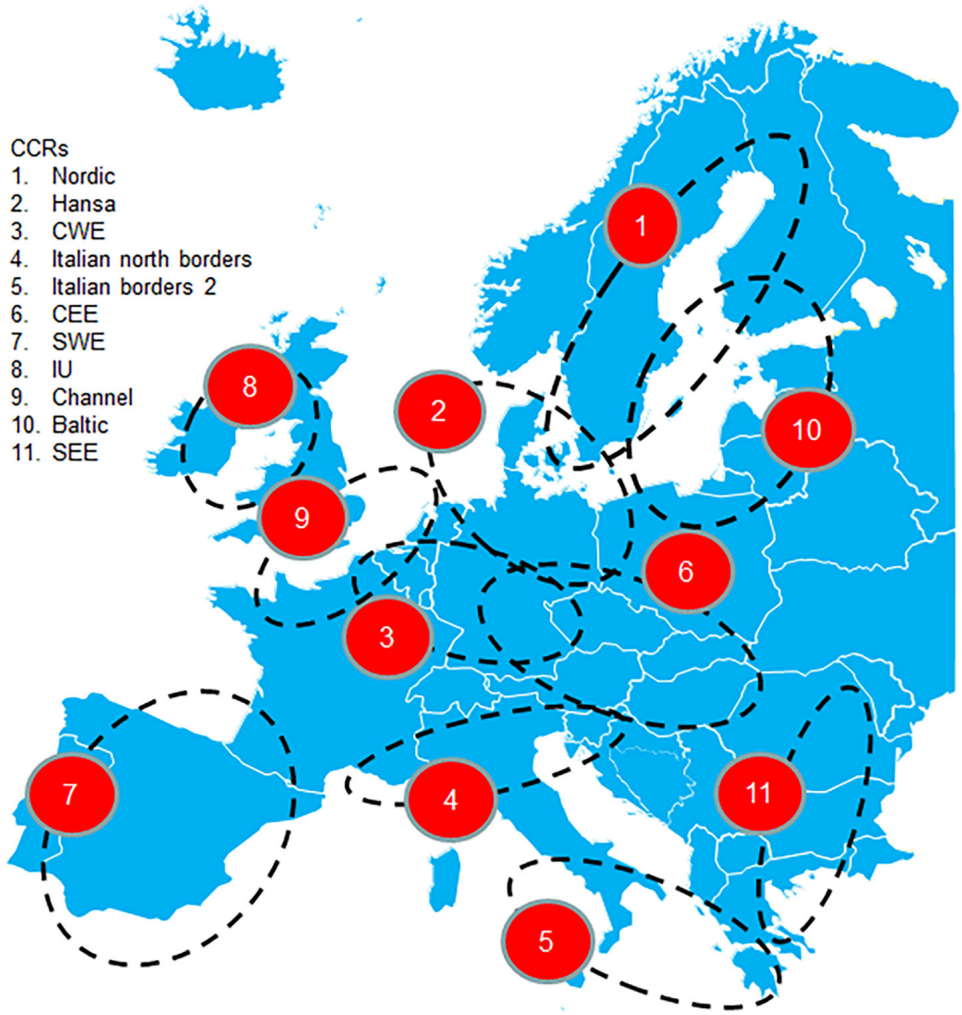


Figure 19: Capacity Calculation Regions firstly proposed by all TSOs according to Article 15 of 1222/2015 (CACM).

improvements to the quality of computation. As the process of cross-border capacity ends up, the results obtained by RSC are passed back to TSOs. TSOs start a validation phase of the results with reference to their CNEs and contingencies. The capacity validation and allocation constraints are shared with other relevant TSOs of the CCR and with RSC. Eventually, the RSC provides the NEMOs with validated capacities and allocation constraints for the market coupling stage.

Note that the actual CCM and CCC may differ among the CCRs as the CACM provides indications and requirements on methodology proposal, but not strict rules on its implementation. In other words, the CACM requires TSOs to formulate their own CCM and CCC procedure according to their operating needs, scenarios and cost-benefit analysis. An example of proposed CCC by the Nordic CCR is reported in Fig. 20.

Of course, each specific CCM and CCC adopted for each CCR must be formally prepared and ratified by NRAs and ACER. As mentioned, in European Transmission Network (ETN) both ATC and FB CC approaches are used, depending on the 'meshed' extent of interconnections. The CACM provides strong incentives onto using FB approach for meshed interconnections. In the next sections ATC and FB CC procedures are presented. The discussion about ATC approach is dealt quite generally,

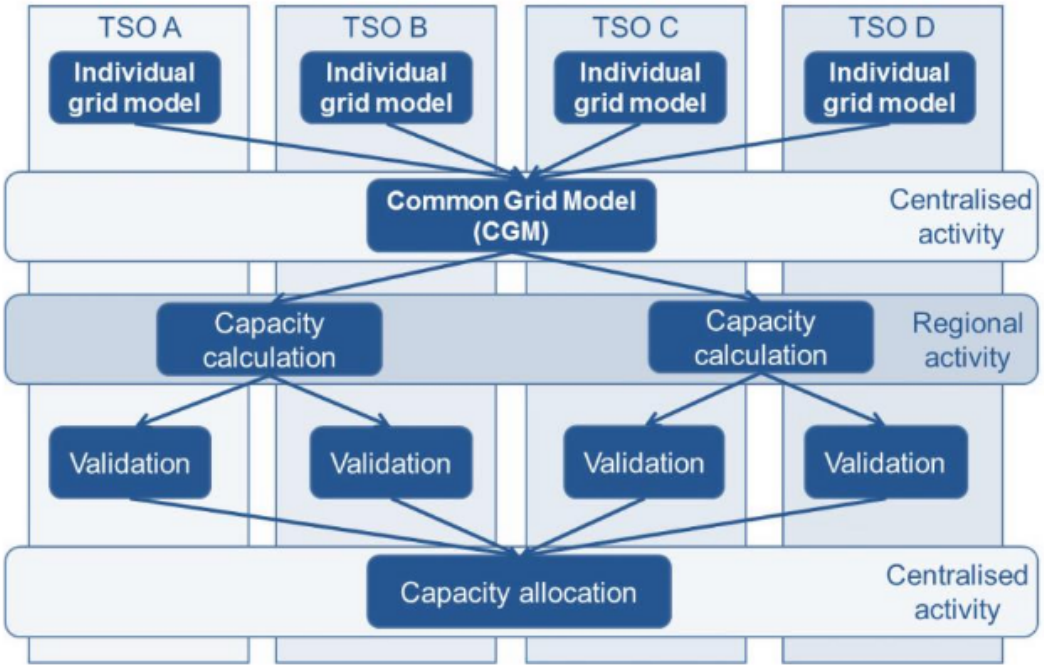


Figure 20: CCC process proposal by TSOs member of the Nordic CCR (Nordic-TSOs, 2018).

with no specific CCR reference. The FB CC is discussed with reference to the CC approach used by the CORE CCR ((CORE-TSOs, 2018)).

5.1.3 Capacity Calculation Mechanisms: ATC

The ATC procedure aims to calculate the *Available Transfer Capacity* between two neighbouring regions (in either directions) that can be sold in the Day Ahead and Intraday markets without endangering the security of each (CCR) system. In order to define the *Available Transfer Capacity* for successive trading over market stages, RSC and TSOs must coordinate to compute three fundamental parameters: the Total Transfer Capacity (TTC), the Transmission Reliability Margin (TRM) and the Net Transfer Capacity (NTC). Before this, let us introduce the necessary inputs required for their computation.

1. **Critical Network Element Contingency (CNEC)**:
   Generally, grid elements (branches, tie-lines, transformers, etc.) present physical limits that must be respected at any time during their operation. TSOs usually associate to each of these elements a maximum stationary allowed current (I_{max}) and optionally a maximum temporary current. The physical limits to respected during operation are of three main types:

   - Thermal limits
   - Voltage limits
   - Stability limits

   A correct estimation of I_{max} is not a simple task to deal with. For instance, weather conditions need to be taken into account, for instance, during summer the cables are cooled less effectively

than on other seasons. Starting from the definition of an I_{max}, it is possible to calculate a maximum allowed power flow F_{max} on a component by the following relation:

$$F_{max} = \sqrt{3}\ I_{max,cont}\ V \cos\phi \tag{106}$$

Where V and $\cos\phi$ are the reference voltage and the power factor of the tie-line, respectively. In the past, ATC methodologies required TSOs to identify branches that could be sensitive to cross-border exchanges, in the sense that operated in real-time could result too close to their power flow limits (F_{max}). For each CB TSOs had to assess also the impact of possible Critical Outages (COs) of other components. The set of all CB and COs formed the so-called CBCOs set. Following to the introduction of CACM, CBCOs have been replaced by the CNECs - Critical Network Elements Contingencies. In practice, CNECs are very similar to CBCOs with the only distinction being that they are more broadly defined, thus include a wider range of contingencies and of possibly critical components.

2. **Generation Shift Keys (GSKs):**
   The GSKs define how changes in the net position of the concerned bidding zone is mapped on generators. For instance, let us consider a generator G_i that has associated a $GSK_i = 10\%$ and let us consider a change in the net position amounting to $100MW$. This means that the output of G_i will change of a quantity equal to $10MW$ (with the sign defined by the sign of the net change of the zone). GSKs are usually associated to generating units that are market driven and that can provide flexibility in changing electric power output.

3. **Remedial Actions (RAs)**:
   A list of RAs is prepared by each TSO and integrated in the IGM. Each RA details the expected impact (i.e. the flow change) over each CNE under a specific contingency. This information is passed to RSC in order to let it compute the best RA to apply whenever a certain contingency is faced and to consider ex-ante the impact that this will have on the flow pattern. This is also called Remedial Action Optimisation (RAO) process.

The GSKs, RAs and CNECs list together with the IGM (individual TSO topology + GLDP) are collated in a file known as the D-2 Congestion Forecast (D2CF) that allow for congestion forecast of the individual TSO situation. The D2CF must respect precise requirements in terms of data format and quality. The Common Information Model (CIM) exchange sets the standard for information exchange regarding not only D2CF, but more generally for any data exchange involving TSOs, DSOs, NEMOs, etc. The RSC brings together all the IGMs to assemble the CGM (or base case). The base case consist of a global congestion forecast for the CCR. The RSC starts its calculation process from the base case with the aim to assess the maximum bilateral inter-connection capacity among two bidding zones.

1. **TTC Calculation**:
   The TTC value is defined as the maximum exchange program between two areas, compatible with Operational Security Policy (OSP) applicable at each system (typically N-1 security criteria) and with tie-line technical limits. This value represents the Total Capacity of the line and hence must be never surpassed. In other words, capacity calculations must only make a portion of such total capacity available to the market and anticipate possible emergency situations, unscheduled flows, uncertainties on generation and load estimations, etc.

2. **TRM Calculation**:
   In real-time cross-border power exchanges might deviate from the forecasted values mainly due to imperfect information from market players and unexpected events (e.g. weather changes). An example of such deviations is balancing power injected/withdrawn from the grid through

frequency control regulation. The evaluation of TRM can be done by the TSOs according to past experiences or using statistical methods. Usually TRM values are typically agreed and fixed for a long time period (e.g. 1 year). In other situations this is just fixed to an active power value (e.g. Elia reserve 250MW at each border) or as a percentage of TTC.

3. **NTC Calculation**:
   Given the TTC and TRM the NTC on a given cross-border interconnection is calculated as:

$$NTC = TTC - TRM \tag{107}$$

   The NTC value represents the maximum amount of power that can flow through a tie line taking into account the uncertainties in future network conditions. Two neighbouring TSOs should calculate NTC for the same border in both directions. As usually TSOs find different NTC values, they need to reach an agreement on which value to set. As always, the most conservative approach is chosen for sake of security of the system: the lower value is taken as common NTC value.

A possible flow diagram of the calculation process is depicted in Fig. 21. The methodology is straightforward. Let consider A and B as two adjacent bidding zones. The responsible RSC uses step-wise changes (e.g. $+100MW$ from A to B) and monitor at each step how changes in the forecast net position impact the power flows over critical grid elements (CNECs). In the case in which a security violation on any CNECs is observed, the last net position is recorded as the maximum allowable power flow - i.e. the $TTC_{A\to B}$. The opposite is done as well to compute $TTC_{B\to A}$ in the opposite direction. At this stage, the $NTCs$ are simply obtained by subtracting the $TRMs$:

$$\begin{aligned} NTC_{A\to B} &= TTC_{A\to B} - TRM_{A\to B} \\ NTC_{B\to A} &= TTC_{B\to A} - TRM_{B\to A} \end{aligned} \tag{108}$$

As such, the $NTCs$ are not the $ATCs$ yet. As we are considering day-ahead time frames, remind that prior to DAM, Forward Markets take place. This means that part of the NTC might be already sold through previous market sessions (in forward markets). The $ATCs$ are hence obtained as:

$$\begin{aligned} ATC_{A\to B} &= NTC_{A\to B} - LTAs_{A\to B} \\ ATC_{B\to A} &= NTC_{B\to A} - LTAs_{B\to A} \end{aligned} \tag{109}$$

The couple of values $ATC_{A\to B}, ATC_{B\to A}$ represent a *trading* quantity that can be offered to the market. A graphical representation of this concept is given in Fig. 22.

The ATC quantity is passed to the MC algorithm as an allocation constraint (or input). As a general rule, we can say that this value tends to be estimated in a more conservative way as the delivery is far in time. As the time of delivery approaches, the level of uncertainty decreases and more capacity can be allocated. This is indeed what happens in the IDM, where usually additionally ATC quantity is offered to the market.

5.1.4 Capacity Calculation Mechanisms: Flow Based

The FB procedure is considered more accurate to calculate the capacity transfer between bidding zones especially for meshed grids. Rather than constraining the capacity of tie lines as for the ATC CC approach, the FB CC approach takes directly into account the arising flows over CNEs resulting from cross-border exchanges. The FB CC procedure starts locally for each TSO, similarly as for ATC approach. Each TSO prepares a set of input data that later will be collected by the RSC for FB

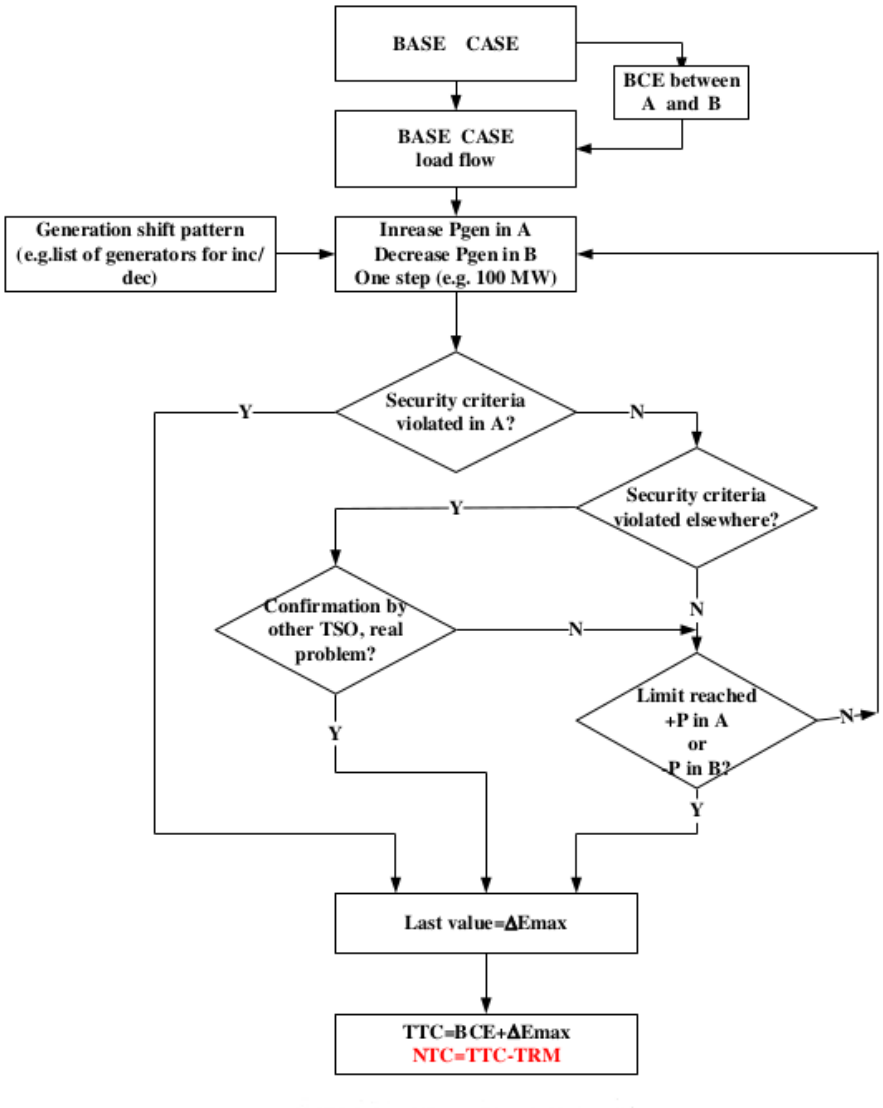


Figure 21: Flow-diagram for TTC and NTC calculation.

parameters computation. This includes again an IGM, containing an approximated representation of TSOs' grid and input data describing load and generation patterns forecast (GLDP). Other inputs are the GSKs, some (additional) External Constraints (ExtC), the Flow Reliability Margin (FRM), LTAs (Long-Term Acquisitions) and possible information about HVDC interconnections. Let us detail the computation of some input parameters for the FB CC procedure.

- **CNECs:** The selection process follow the same rules of ATC. The difference here is that each TSO explicitly associate to each CNE a maximum allowed flow F_{max}. Different contingencies are considered for each CNE according to the TSO security policy, to form the CNECs (CNE with a Contingency) elements.

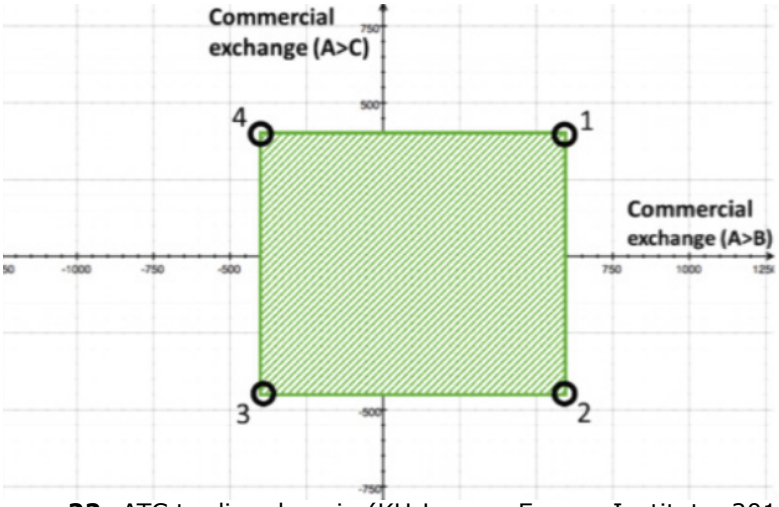


Figure 22: ATC trading domain (KU-Leaven-Energy-Institute, 2015).

FRM: Similarly to TRM, TSOs define a security margin to cover for risks associated to unexpected events and in general to forecast limitations due to D-2 time-frame of the base case model. The procedure to compute FRM is a statistical based procedure and is based on monitoring differences between D-2 CGM forecast flow and real-time flow on a given CNEC. In order to account only for non-deterministic events that the D-2 CGM forecast model is inherently subject to, the expected D-2 CGM flow is adjusted by real-time final net position and relieved of the remedial actions performed in real-time. A statistical evaluation process is done upon monitored difference and a risk level is associated. In other words, the role of FRM is to ensure under a certain risk percentage, that the difference between the forecast flow (on a given CNEC) and the real-time flow will be contained in a certain range.

All the computed parameters and the IGM are gathered into a file, the D2CF, individually by each TSO. Next, all the individual D2CF files are merged by the responsible RSC for the CCC at the CCR of interest. The result of the merging process gives the so-called *base case*, a snapshot of the European grid topology, generation and load profiles two days ahead of the delivery. Prior to the merging process the RSC thoroughly checks the quality and plausibility for the IGMs provided by TSOs. Having the CGM and all input parameters for FB, the RSC starts the 'Coordinated Capacity Calculation' (CCC) process. The goal of the FB CCC is to define remaining available margins (RAMs) on each CNEC. This in turn will define a 'feasible trading domain', identifying which trades are feasible without endangering the security of the grid and which are not possible. The CCC methodology carried out by RSCs can be outlined by calculation of the following parameters:

- **PTDF:** Through the D-2 CGM, the RSC owns all the ingredients to perform a Load Flow (LF) computation (i.e. grid topology, load and generation profiles). To compute the PTDFs (Power Transfer Distribution Factors), the LF is linearised around the working point predicted by the base case. The linearisation allows thus to define a simple relationship (based on a set of coefficients - the PTDFs) between a change in the net position of a bidding zone part of the CCR and the resulting flows over the CNECs. In other words, we can consider PTDFs as sensitivities coefficients. In general, what is needed is a set of coefficients that directly relate importing/exporting flow from zone to zone with changes of flows over CNECs. In this way it is possible to define whether the change in the net position of a bidding zone with respect another is feasible or not. In other words whether a commercial exchange is feasible or not. The calculation of PTDFs is actually done through three steps. First, the RSC computes Nodal-

to-Slack (N2S) PTDFs by varying the injection/withdrawals per each node and by registering the change in the flow per each CNEC. Then, by using the GSKs turns the N2S PTDFs into Zone-to-Slack (Z2S) PTDFs coefficients. This is done to scale the change of the node to the change in the net position of the zone. Eventually, the Z2S PTDFs are used to compute Zone-to-Zone (Z2Z) PTDFs coefficients.

- **Net Position Flow F_{NCP} :** We define the F_{NCP} as the flow of active power through a CNEC given a certain Net Commercial Position (NCP) relative to bidding zone A. This practically represent the expected power flow as the market clears, according to the final NCP. As mentioned, many other factors intervene in the real-time so that this flow is only an estimation given the output of the market. With reference to a CNEC i, the F_{NCP} can be computed by using the previous defined Z2S PTDFs as follows:

$$F_{NCP,i} = F_{ref,i} + \sum_{A=1}^{\mathcal{N}} PTDF_{A,i}(NCP_A - NCP_{A,ref}) \tag{110}$$

  Where $NCP_{A,ref}$ is the net commercial position of bidding zone A at base case NCP.

  RAMs: The RAMs define the remaining available margin per each CNEC. Available margins will result by subtracting out of the maximum possible F_{max}, all the factors that 'take up' a portion of the F_{max}. This is done according to:

$$RAMs = F_{max} - FRM - FAV - F_{NCP} \tag{111}$$

  Where the FAV term stands for 'Final Adjustment Value' and may be or may be not done by each TSO at the validation stage (i.e. after the RSC send back the CGM and the computed results to all CCR TSOs).

Once the RAMs of each CNEC are computed, the whole set of constrains can be used to build up a trading domain similar to the ATC one Fig. 22. In a plane of net commercial positions $NCP_{A\to B}$-$NCP_{B\to A}$, the set of equations Eq.(111) form a set of straight lines that provide the limits to the trading domain. The situation is shown in Fig. 23, where a comparison between the FB and the ATC trading domain is depicted.

In Fig. 23 the domain representation is limited to the exchanges among three bidding zones A, B and C. In such trilateral case, the set of constrains are simply straight lines. Notice that a CCR that involves a set of generic $\mathcal{N}$ bidding zones involve higher dimensional space of representation for the trading domain. More precisely, the trading domain of a generic bidding zone A with respect all the other $B, C, \ldots$ lies in a $\mathcal{N}$ dimensional space. Each constrain would be represented as an hyperplane of dimension $\mathcal{N}$.

From an economical and an engineering point of view, the interesting fact that can be drawn from Fig. 23 is the wider resulting domain obtained through the FB approach with respect to the ATC. A wider trading domain allows to increase the amount of cross-border exchanges, to obtain greater price convergence between zones and as a consequence an higher social welfare. Generally, methodologies that can enlarge the trading domain without decreasing the level of operational security are of great relevance in a context where RES production may be substantially higher in some countries in short-times, and hence offer convenient exchange opportunities.

5.1.5 Limitation of Current Practices

Some well-known limitations exist on ATC and FB CC approaches. In the ATC approach (Fig. 23) the trading domain is usually constrained by a set of horizontal and vertical lines that produce

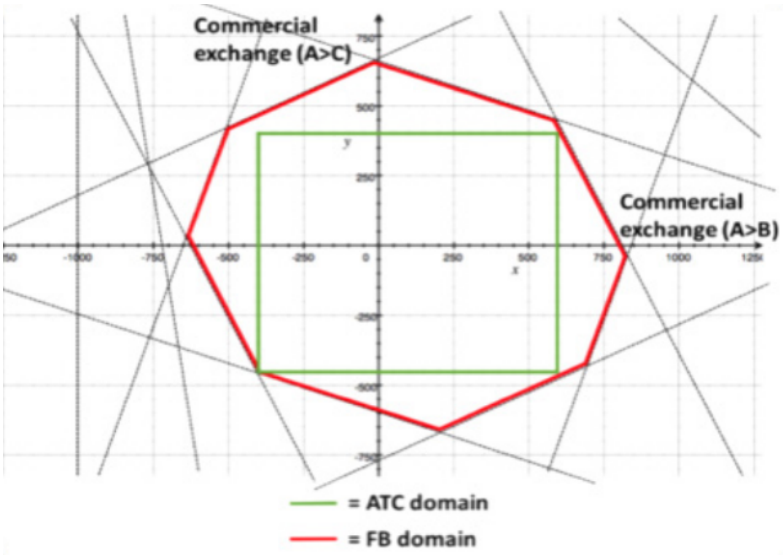


Figure 23: FB and ATC trading domains (KU-Leaven-Energy-Institute, 2015).

in general a rectangular domain[31]. This means that if we reduce the power flow exchange between, for instance, A and B, this do not result in a higher capacity between A and C. The ATC approach thus assesses the capacities bilaterally, and by taking a conservative approach on potential *loop flows* and *transit flows* (Fig. 24) which might result from neighbouring cross-border exchanges[32].

In other words, the ATC approach do not use a real mathematical model to describe the flows resulting from neighbouring zones, but rather takes conservative assessments on them based on experience. On the other hand, the FB approach incorporates the inter-dependency of zones through a physical model of power flows (although through a linearised approach). In this way, since any commercial exchange among neighbouring zones is translated into a change in the flow on each critical component (CNECs), we can get a much better modelling of the constraints. For example, if we follow a straight line constraint in Fig. 23 we see that a change of commercial exchange on one border (say A-B) has an impact on another border (A-C). Since FB CC can be considered a more precise approach to model cross-border commercial exchanges, one may ask why in some European CCRs the ATC procedure is still in place. One of the reasons is that bidding zones that are poorly meshed with their neighbours can be modelled effectively with ATC approach (e.g Italy-Greece CCR). In these cases ATC and FB approaches give in fact the same results, with the advantage for ATC of being a simpler procedure which involve less parameters to compute (and indeed less coordination). On the contrary, Central Europe is characterised by highly meshed interconnections and flows through different bidding zones show a strong interdependency with each other. Hence, considering the ATC approach as 'limited' with respect to the FB makes sense only for those regions that are sufficiently meshed. On this matter, the CACM states (in Article 20, point 7) that:"*TSOs may jointly request the competent regulatory authorities to apply the coordinated net transmission capacity approach if the TSOs concerned are able to demonstrate that the application of the capacity calculation methodology using the flow based approach would not yet be more efficient compared to the net transmission capacity approach and assuming the same level of operational security in the concerned region*" which shows how the CACM strongly stimulates for implementation of FB CCC where significant benefits can be obtained at same level of operational security. An example

[31]The domain is squared when the cross-border capacities are symmetric in either direction.

[32]Loop flows are those flows that start off and get back into its originating bidding zone. Transit flows are flows that before reaching the destination pass through neighbouring bidding zones.

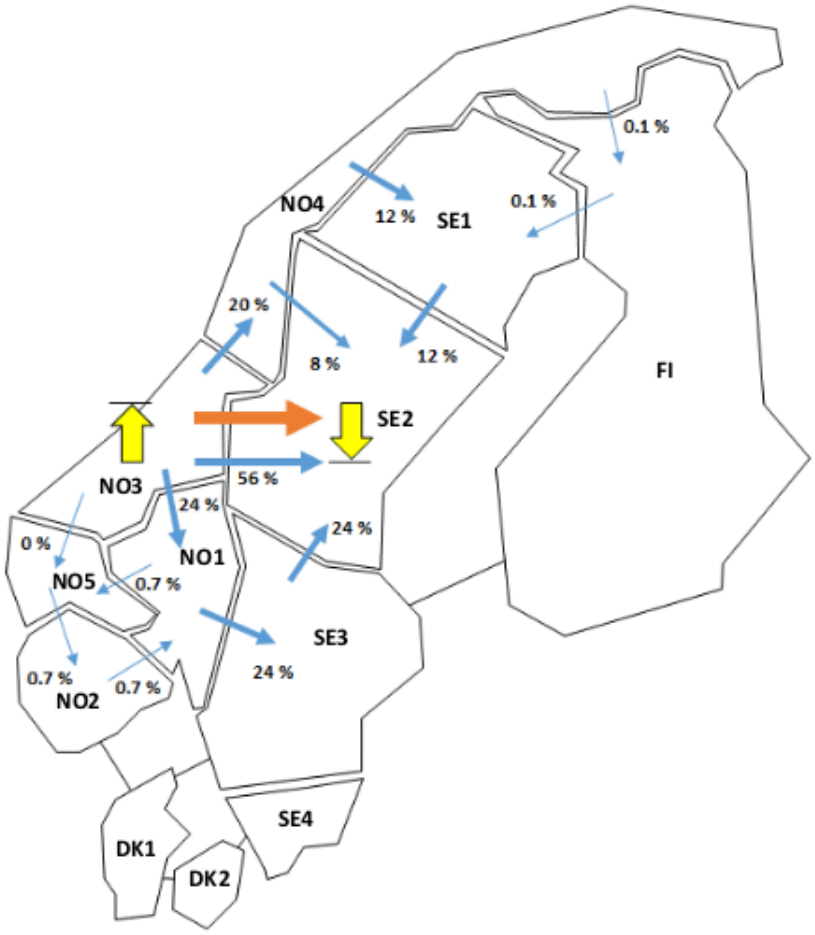


Figure 24: Example of bilateral trade (yellow arrows) and resulting physical flows over the grid (blue arrows). Commercial flows do not translate in physical flows, but rather create several loop and transit flows according to Kirchhoff's Law.

of ongoing implementation of FB in place of ATC is that one concerning the Nordic CCR, which is expected to end in 2022 (Nordic-TSOs, 2018). After a cost-benefit Nordic TSOs have indeed identified FB as a solution that can bring benefits to CCC process. As ATC and FB CCC shall be differently applied depending on the topology of CCR, possible improvements to both procedures can be foreseen. In previous sections we discussed the methodologies in a generic framework. We can conceptually divide a CCM in three steps:

1. ATC/FB inputs preparation (IGM) at TSOs control centres;
2. RSC collects input information from TSOs through a dedicated digital platform according to the Common Information Exchange Model and builds the CGM;
3. RSC sends back to TSOs the results of the ATC/FB computation. The TSOs validate the results and the trading domain is passed to the market.

Currently these steps are only partially coordinated among TSOs though it is known that by increasing coordination at various stages may yield a better estimation of ATC and FB parameters and deliver

a more precise estimation of the available cross-border capacity. For instance, during the first step each TSOs builds its own IGM and makes an estimation of generation and demand within its control area. At this stage, the internal distribution of flows is obviously not known. Cross-border flows are only estimated based on historical data. Inputs are asked to neighbouring TSOs, but the extent of coordination is though low. This means that the actual preparation of data concerning RAs, CNECs, GSKs, GLDP and forecast on topology state at day of delivery can be imprecise thus leading to imprecise load flow computations performed at this initial stage.
Over the second step the RSC builds up the CGM and carries out the CCC process. A similar discussion can be carried out for the third step, where the validation process is only left to the TSO directly interested without involving neighbouring TSOs which are 'indirectly' involved.
With respect to the CCC of meshed CCRs, another source of inaccuracy is given by the linearisation of power flow problem for the computation of the PTDFs coefficients. One may think to perform a more accurate estimation of the PTDFs coefficients by carrying out an AC load flow computation. As a result, the trading domain would be more accurately defined, at the cost of not having a linear problem to solve. This actually involves a complex and broad discussion involving DAM market design and the Price Coupling Algorithm (PCA) which is not the scope of this work.

5.1.6 Advantages of Distributed Capacity Calculation

In the previous sections we highlighted some limitation of the current approaches to compute the available capacity trading domain. Proposing more accurate calculations for processing parameters, either in the context of ATC or FB CC, may allow to decrease the level of uncertainty on future flows distribution and grid conditions, hence enlarging the trading capacity domain offered to the market. In practical terms, this means to be able to exchange more power as needed, and in turn obtaining the benefit of a more efficient allocation of generating power. A distributed approach to compute power flow equations at the initial stage of the CCC may allow to plan coordinately the inputs, to foresee the impact of contingencies, to cope with solar and wind uncertainty and possible outages, on each other assets. This could be done with the advantage of not sharing detailed information about owned assets, e.g. the graph topology of the transmission system, generation and demand profiles. The distributed power flow methodology can be intended as a mean to bridge the gaps due to asymmetry of information among TSOs, without entailing them to share sensitive information. By introducing more detailed information gives in turn the benefits of having as a result a more reliable computation. The same reasoning can be applied as mentioned for the validation stage, after the RSC CCC process ends up.
Another important factor concerns the time-frame. The problem of correctly calculating and allocating cross-border capacity can be seen as a chicken-and-egg problem. At the real-time the uncertainty on production, demand and possible contingencies is null, as production and demand profiles are well-defined. On the other hand, the definition of a schedule for suppliers and consumers is only defined once capacities are defined and allocated to the market. The distributed power flow methodology presented here allows for a fully automated approach that only requires the TSOs to produce topology, generation and load inputs for the coordinated calculation. The time required for the simulation is close to that of local ones provided that a fast interconnection among control centres is available (at least $1Gbit/s$). With automation and fast simulation times, the approach may allow TSOs to bring their analysis closer to the real-time, or to evaluate more scenarios at the same time. Another remark is that by decomposing the load flow problem into sub-problems, for instance by means of the methodology presented here, the mathematical formulation of the problem does not change. This means that the solution that is obtained is exactly the same that would be obtained in the case a unique entity owned all the input data for the load flow and solved it locally. As the possibility to have a unique TSO managing the whole ETN does not seem realistic at the moment, a concrete solution to align

TSOs positions could be that of relying on distributed/delocalised approaches. A final consideration goes to the possibility of scaling up the methodology. As the grid becomes more active at lower voltage levels, TSOs may require to include in their power flow computation information about underlying distribution networks. This makes the computation more complex and more computational expensive and may require TSOs to face new investments in dedicated computational infrastructures. The costs of such can be shared among TSOs if they adopt a distributed approach. As the computational burden is shared among grid operators, the computational power required locally is reduced. In the next final section we briefly discuss possible applications of distributed approaches to the mentioned case of TSO-DSO interface.

5.2 TSO-DSO Coordination

This section aims at giving a short overview on how new policy provisions might change the role of TSOs and DSOs and especially their interaction. In the coming years the higher penetration of wind and solar distributed power production, the stochastic recharging connection behaviour of EVs, the appearance of new market actors (prosumers, energy communities), the possibility to have storage system integration, are expected to remarkably have a direct impact on distribution grids management and indirectly on transmission system operation. The old role of the DSO as passive *network operator* moved by the 'fit and forget' approach, is expected to change into a new active role more similar to that of TSOs in transmission networks management. First generation network codes and guidelines (those under the Third Energy package) have already started to address the change, by stimulating TSO and DSO to cooperate and to define coordination schemes on some operational tasks. Under the CEP new Network Codes (NCs) are expected to be developed from ENTSO-e and the new EU DSO entity in the next years. In the second part of this section possible applications of distributed approaches in view of the emerging changes are briefly discussed. The most natural framework of application being the transmission-distribution networks interface.

5.2.1 Operational & Policy Context in Distribution Grid Management

The entry into force of TEP in 2009 has brought important changes in the European energy sector. The establishment of a body for TSOs cooperation (ENTSO) has started a process of standardisation of European system operation, congestion management activities, market interactions, and many other TSOs related tasks, through preparation and implementation of a set of technical (and legally binding) rules (network codes (NCs) and guidelines (GLs)). The aim of Network Codes is to support the objectives set by TEP by means of appropriate legal instruments (GLs and NCs) that require Member States and all energy stakeholders to implement and develop a set of required actions. These actions are aimed at enhancing competition in energy markets, operate the system in the most secure and cost efficient way, while managing hurdles introduced by RES and ease their introduction in the European energy framework. In this role, ENTSO-e has a clear representative role for TSOs; as a matter of fact TSOs can be seen as *market facilitators*, meaning that TSO grid management principles shall be aligned with the objective of maximising market efficiency, through least cost interventions and always ensuring security of supply. TSOs are *active* system operators, which need to continuously take actions considering the real-time conditions of the grid. In the current context of DER and new market actors introduction, DSOs are expected to assume a role of market facilitators at distribution level, in analogy to that of TSOs at transmission level. So far, the distribution grid over-sizing allowed to introduce RES without too much concern and only requiring DSOs to collaborate on specific actions. As the rise of RES continues, to meet decarbonisation objective set for the next decades the dualism market-system operation may need to undergo another big re-settlement. DSOs need to actively manage their networks in a way that could resemble that of TSOs: plan investments, monitor grid flows and state variables profiles, provide non-discriminatory network access to their

users, solve local congestions and even carry out balancing actions on their system. This requires DSOs to invest properly to improve their ability of monitoring and control of distribution grids. In order to accelerate this process the CEP has required to establish an EU DSO entity in charge of developing NCs and GLs for distribution system operation, similarly to what ENTSO-e already does in the transmission context. Coordination between the two entities is indeed a must. The new legislation will help clarify which actions shall be taken by TSOs, which by DSOs and which needs coordination. This is a matter of great attention as, for instance, TSOs and DSOs may have the need to access flexibility resources (e.g. load shift, storage facilities) located at distribution level for different tasks (e.g. TSOs for balancing purposes and DSOs for solving local congestions). Note that, in the current context, DSOs are still preparing to the on-going change.

5.2.2 TSO-DSO Current Cooperation

Currently, TSOs ask on pro-rata basis to DSOs for aggregate estimation of load and production at distribution level, with particular attention to RES production forecast[33]. This enable TSOs to run more accurate power flow analysis and check whether the forecasted operational conditions meet established operational security policies. This is an example of TSO and DSO cooperation currently in place. Current shared responsibilities and interactions between TSO and DSOs are regulated through two GLs and one NCs: the System Operation GL, Electricity Balancing GL and Emergency and Restoration NC (CEER, 2016). The GLs and NC provide also indications on data format, media and details to be exchanged depending on which coordinated action is planned to be performed. Insights on future collaboration schemes between TSO and DSOs can be found in (CEER, 2016) itself. The need of establishing TSO-DSO cooperation schemes that can be efficient and can reflect the TSO and DSO needs to access reciprocal resources is thus crucial in the current evolution of distribution grids. Some key projects have been launched in the last years aimed at clarify which TSO-DSO scheme can offer higher value to the power system as a whole. SmartNet project, for instance, focuses on ancillary service provision from distribution grid and coordination schemes between TSO and DSOs (Gerard et al., 2016). One of the aim of such project is to assess the efficiency of different coordination schemes under different paradigms (e.g. DSO are in charge of managing local congestions but do not perform balancing; DSOs can carry out both balancing and congestion management activities within their systems; etc.).

The changing role of DSOs will demand them more accurate predictions on load and generation profiles over the distribution grid. This will enable them in turn to perform power flow analysis and monitor the grid flows and state variables to ensure proper operational security. This is a key fact as we assume in the following that DSOs are in possess of input data for performing power flow computations on their system. In this view, it is interesting the ENTSO-e position for future regional coordination at transmission-distribution level: "TSOs need to share their individual grid models with the other TSOs and the RSCs. RSCs are responsible for merging the different grid models of the TSOs and issue common grid models. These are then shared with the TSOs in order for them to adapt their operational planning with this new regional information. *The same process can be used in the future for next TSO-DSO coordination*."(ENTSO-E, 2017). In other words, ENTSO-e foresees the implementation of a regional coordination service at transmission-distribution interfaces, with DSOs preparing their own IGMs, sharing them with TSOs, etc. similarly to what seen in the previous sections with respect to CCC process.

[33] Both industry and research is well active in developing methodologies that allow to obtain more accurate estimations of RES production. An example of research in this context can be found in (Falabretti et al., 2018).

5.2.3 TSO-DSO interface with distributed approach

The 'transmission-distribution interface' can be seen as the boundary between a system operated by a TSO and a system operated by a DSO. This usually consist in the HV-MV interface, the 'substation', that brings down the voltage level and supply customers at medium voltage. Differences exist at European level when talking about voltage levels. For a more in depth discussion on this topic, please refer to (Prettico et al., 2019). This interface can be radial or meshed, depending on its geographical location. For instance, in rural zones, the interface is usually radial whereas in urban areas is meshed. If we imagine a RSC entity responsible for calculating cross-border capacities at transmission-distribution interfaces, DSOs shall prepare to have in place a GLDP methodology, an IGM and all the input parameters to perform power flow computations and estimate limits on its CNECs. Eventually, they will pass the inputs to the RSC to perform the CCC on transmission-distribution interface. For meshed topologies, a CCC based on an FB procedure can be imagined, while an ATC based capacity calculation process might be done when the interconnection is radial. As for the transmission-transmission interface, the inputs to pass and the exchange of IGM could be better analysed concurrently among TSO and DSOs if a coordinated distributed power flow tool is available. By using a methodology that allow delocalised simulation the sharing of input data for power flow computation is not needed as widely mentioned throughout this report.

6 Conclusions & Future Works

This report presented a new methodology for distributed solution of power flow equations and a brief discussion on its potential uses in the power sector. The key-facts drawn by this work cam be summarised as follows:

- The methodology presented here is highly scalable and robust, which makes it appealing for future power grids, that are envisioned to be larger and more interconnected. This has been extensively proved in the literature through the Krylov-Schwarz approaches;
- The information sharing between the two (or more) parties involved is limited and most of the solution process can be carried out in an autonomous fashion in the computation. This gives the advantage to grid operators to input more detailed information about their topology state, generation and load profiles, at the benefit of obtaining more accurate load flow calculations (and hence a 'better' snapshot of grid conditions);
- We can think of distributed/delocalised approaches in a large number of situations for grid operation scopes but not only. These approaches finds an application whenever the need of aligning power flows among different grid operators is demanded and the local input information cannot be shared between the parties. In the current and emerging European framework, we identified two specific areas of application: 1) the Capacity Calculation process between TSOs at European level; 2) the calculation of power flows at the TSO-DSO interface.
- In the context of Capacity Calculation some stages are performed regionally by TSOs. A coordinated distributed approach in load flow computation might allow to better coordinate some steps of the established procedures and get more accurate estimation of ATC/FB parameters. As a consequence of that wider capacity trading domains to offer to the market might be obtained increasing price convergence between bidding zones without decreasing the level of operational security requested to TSOs.
- In the context of TSO-DSO cooperation, distributed approaches to power flow equations can be envisaged with reference to transmission-distribution interface management. If a Coordinated Capacity Calculation mechanism is in place at distribution level as well, similar to that applied to Capacity Calculation Regions at national level, same considerations done on TSO-TSO coordination might be easily translated to the TSO-DSO local coordination.

Several questions and ideas have raised during the completion of this project, which might be tackled in future works on this topic. A non-exhaustive list is reported below:

- A more accurate analysis on parallel communications of Inexact Newton methods based on Krylov-Schwarz inner linear solvers;
- An improvement of the methodology into a full delocalised approach, so that each grid entity can act as a *peer* in a peer-to-peer scenario;
- An assessment of the possibility to use Distributed/Delocalised approaches for AC Optimal Power Flows implementations;
- A more in-depth analysis and possibly use case of TSO-TSO and TSO-DSO schemes to perform coordinated power flow computations.

References

(2015). *Commission Regulation (EU) 2015/1222 of 24 July 2015 establishing a guideline on capacity allocation and congestion management (2015)*. Official Journal of the European Union, OJ L197 (CACM GL).

(2019a). *Directive (EU) 2019/944 of the European Parliament and of the Council of 5 June 2019 on common rules for the internal market for electricity and amending Directive 2012/27/EU*. Official Journal of the European Union.

(2019b). *Regulation (EU) 2019/943 of the European Parliament and of the Council of 5 June 2019 on the internal market for electricity*. Official Journal of the European Union.

ACER and CEER (2018). *Annual Report on the Results of Monitoring the Internal Electricity and Natural Gas Markets in 2017*. doi:10.2851/525797.

Alfio Quarteroni, Fausto Saleri, P. G. (2012). *Calcolo Scientifico*. Springer.

Anzt, H., Dongarra, J., Flegar, G., and Quintana-Orti, E. S. (2017). Batched gauss-jordan elimination for block-jacobi preconditioner generation on gpus. pages 1–10.

Bagchi, A. (1996). Conflicting nationalisms: the voice of the subaltern in Mahasweta Devi's Bashai Tudu. *Tulsa Studies in Women's Literature*, 15(1):41–50.

Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Dener, A., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., May, D. A., McInnes, L. C., Mills, R. T., Munson, T., Rupp, K., Sanan, P., Smith, B. F., Zampini, S., Zhang, H., and Zhang, H. (2018a). PETSc users manual. Technical Report ANL-95/11 - Revision 3.10, Argonne National Laboratory.

Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Dener, A., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., May, D. A., McInnes, L. C., Mills, R. T., Munson, T., Rupp, K., Sanan, P., Smith, B. F., Zampini, S., Zhang, H., and Zhang, H. (2018b). PETSc Web page. `http://www.mcs.anl.gov/petsc`.

Benzi, M. (2002). Preconditioning techniques for large linear systems : A survey.

CEER (2016). Ceer position paper on the future dso and tso relationship.

Chen, K. (2005). *Matrix Preconditioning Techniques and Applications*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press.

CORE-TSOs (2018). "explanatory note on the day-ahead and intraday common capacity calculation methodologiesfor the core ccr".

ENTSO-E (2017). Where the digital transformation of the european electricity system starts: Entso-e's common grid model.

Falabretti, D., Delfanti, M., and Merlo, M. (2018). Distribution networks observability: A novel approach and its experimental test. *Sustainable Energy, Grids and Networks*, 13:56–65.

Farebrother, R. W. (1988). *Linear Least Squares Computations*. Marcel Dekker, Inc., New York, NY, USA.

Ferronato, M. (2012). Preconditioning for sparse linear systems at the dawn of the 21st century: History, current developments, and future perspectives. *ISRN Applied Mathematics*.

Gander, M. (1996). Overlapping schwarz for parabolic problems. pages 4–7.

Gerard, H., Rivero, E., and Six, D. (2016). Basic schemes for tso-dso coordination and ancillary services provision. *SMARTNET Deliv. D*, 1.

Gouarderes, F. (2019). *Fact Sheets on the European Union: Internal energy market*. European Parliament website, https://www.europarl.europa.eu/factsheets/en/sheet/45/internal-energy-market.

Gutknecht, M. H. (2007). A brief introduction to krylov space methods for solving linear systems. pages 53–62.

Hanon, P., Ramet, P., and Roman, J. (2002). Pastix: a high-performance parallel direct solver for sparse symmetric positive definite systems. *Parallel Computing*, 28(2):301 – 321.

Hendrickson, B. and Leland, R. (1993). The chaco user' s guide. version 1.0.

Idema, R. (2012). *Newton-Krylov Methods in Power Flow and Contingency Analysis*.

Idema, R. and Lahaye, D. (2014). *Computational Methods in Power System Analysis*. Atlantis Studies in Scientific Computing in Electromagnetics. Atlantis Press.

Idema, R., Lahaye, D. J. P., Vuik, C., and van der Sluis, L. (2012). Scalable newton-krylov solver for very large power flow problems. *IEEE Transactions on Power Systems*, 27(1):390–396.

Ipsen, I. and Meyer, C. (1997). The idea behind krylov methods. *The American Mathematical Monthly*, 105.

J. GANDER, M. (2008). Schwarz methods over the course of time. *ETNA. Electronic Transactions on Numerical Analysis [electronic only]*, 31.

Josz, C., S.Fliscounakis, Maeght, J., and Panciatici, P. AC Power Flow Data in MATPOWER and QCQP Format: iTesla, RTE Snapshot, and PEGASE.

Kargarian, A., Mohammadi, J., Guo, J., Chakrabarti, S., Barati, M., Hug, G., Kar, S., and Baldick, R. (2016). Toward distributed/decentralized dc optimal power flow implementation in future electric power systems. *IEEE Transactions on Smart Grid*, PP:1–1.

Karypis, G., Schloegel, K., and Kumar, V. (1997). Parmetis: Parallel graph partitioning and sparse matrix ordering library.

Kshemkalyani, A. D. and Singhal, M. (2008). *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, New York, NY, USA, 1 edition.

KU-Leaven-Energy-Institute (2015). "cross-border electricity trading: towards flow-based market coupling".

Lange, M., Mitchell, L., Knepley, M. G., and Gorman, G. J. (2015a). Efficient mesh management in firedrake using PETSc-DMPlex. *CoRR*, abs/1506.07749.

Lange, M., Mitchell, L., Knepley, M. G., and Gorman, G. J. (April, 2015b). Flexible, scalable mesh and data management using PETSc DMPlex.

Maldonado, D. A., Abhyankar, S., Smith, B., and Zhang, H. Scalable Multiphysics Network Simulation Using PETSc DMNetwork.

Mohammadi, J., Kar, S., and Hug, G. (2014). Distributed approach for dc optimal power flow calculations.

Nordic-TSOs (2018). "supporting document for the nordic capacity calculation regionâĂŹs proposal for capacity calculation methodology in accordance with article 20(2) of commission regulation (eu) 2015/1222 of 24 july 2015 establishing a guideline on capacity allocation and congestion management".

Pacheco, P. S. (March 30, 1998). *A User's Guide to MPI*. Department of Mathematics,University of San Francisco.

PLEF (2007). Memorandum of understanding of the pentalateral energy forum on market coupling and security of supply in central western europe.

Pollan, M. (2006). *The omnivore's dilemma*. Penguin Group, New York.

Poston, T. (2000). *A draft of history*. University of Georgia Press, Athens.

Prettico, G., Flammini, M. G., Andreadou, A., Vitiello, S., Fulli, G., and Masera, M. (2019). Distribution system operator observatory 2018 - overview of the electricity distribution system in europe. Technical report, Joint Research Centre, Via Enrico Fermi 2749, Ispra, VA, Italy.

Rinaldo, S. G., Ceresoli, A., Lahaye, D. J. P., Merlo, M., Cvetkovic, M., Vitiello, S., and Fulli, G. (2018). Distributing load flow computations across system operators boundaries using the newton-krylov-schwarz algorithm implemented in petsc. *Energies*, 11(11).

Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition.

Saad, Y. and Schultz, M. (1986). Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869.

Surname, F. (2015). JRC technical reports template. Technical report, Joint Research Centre, Via Enrico Fermi 2749, Ispra, VA, Italy.

Waldrop, M. M. (2016). The chips are down for moore's law. *Nature*, 530:144–147.

Ziavras, S. G. (2003). Parallel direct solution of linear equations on fpga-based machines. In *Proceedings International Parallel and Distributed Processing Symposium*, pages 8 pp.–.

Zimmerman, R. D. and Murillo-Sanchez, C. E. (December 16, 2016). Matpower 6.0 user's manual.

List of abbreviations and definitions

AC Alterning Current

API Application Programming Interface

ASM Additive Schwarz Method

ATC Available Transfer Capacity

BLAS Basic Linear Algebra Subprograms

BVP Boundary Value Problem

CA Capacity Allocation

CC Capacity Calculation

CEP Clean Energy Package

CFD Computational Fluid Dynamics

CG Conjugate Gradient

CPU Central Processing Unit

DAG Directed Acyclic Graph

DAM Day-Ahead Market

DC Direct Current

DDM Domain Decomposition Method

DERs Distributed Energy Resources

DSO Distribution System Operator

EC European Commission

ENTSO-E European Network of Transmission System Operators for Electricity

EUPHEMIA Pan-European Hybrid Electricity Market Integration Algorithm

EV Electric Vehicle

FDLF Fast Decoupled Load Flow

FLOPS Floating Point Operation Per Second

GMREs Generalized Minimal RESidual method

GNU GNU is Not Linux

HV High Voltage

IDM Intra-Day Market

IEM Internal Energy Market

IPC Inter-process Communication Protocol

KCL Kirchhoff's Current Law

KSP Krylov Subspace linear solver of PETSc

LV Low Voltage

MC Market Coupling

MPI Message Passing Interface

MV Medium voltage

MVM Matrix Vector Multiplication

NEMO Nominated Electricity Market Operator

NKS Newton-Krylov-Schwarz

NR Newton-Raphson

NTC Net Transfer Capacity

OPF Optimal Power Flow

PCR Price Coupling of Regions

PDE Partial Differential Equation

PETSc Portable Extensible Toolkit for Scientific Computation

PMF Power Mismatch Function

PX Power Exchange

R-GMREs Restarted GMREs

RES Renewable Energy Sources

SNES Scalable Non-linear Equations Solver of PETSc

SPD Symmetric Positive Definite

TEP Third Energy Package

TRM Transmission Reliability Margin

TS Time Stepper solvers of PETSc

TSO Transmission System Operator

TTC Total Transfer Capacity

WAN Wide-Area Network

List of figures

List of tables

Annexes

Annex 1. Ybus

The *nodal* admittance matrix Y_{bus} can be derived in many ways. Here we report only its definition. With reference to a grid of N buses, the Y_{bus} is an NxN matrix having entries in position i, j defined as:

$$Y_{bus} = \begin{cases} \sum_{j=1}^{N} y_{ij}, & i = j \\ -y_{ij}, & i \neq j \end{cases} \tag{112}$$

With y_{ij} the *branch* admittance of line connecting bus i with j. If no line connects bus i with j, then $Y_{bus,ij} = 0$.

Annex 2. GMREs

Here, we report the full Generalized Minimal Residual method algorithm under its left and right preconditioned form ((Saad, 2003)). Notice the Arnoldi's orthogonalization process from step 2 to 10 to build the basis for the Krylov Subspace.

Algorithm 3 Left-preconditioned GMRES for iterative solution of linear systems.

1: **Compute** $r_0 = M^{-1}(b - Ax_0), \beta := ||r_0||_2$, **and** $v_1 := r_0/\beta$
2: **For** $j = 1, ..., m$ **Do**:
3: **Compute** $w := M^{-1}Av_j$
4: **For** $i = 1, ..., j$ **Do**:
5: $h_{ij} := (w_j, v_i)$
6: $w_j := w_j - h_{ij}v_i$
7: **end Do**
8: **Compute** $h_{j+1,j} = ||w_j||_2$ **and** $v_{j+1} = w_j/hj + 1, j$
9: **end Do**
10: **Define** $V_m := [v_1, \ldots, v_m]$, **the Hessenberg matrix** $\hat{H}_m = \{h_{ij}\}_{1 \leq i \leq m+1, 1 \leq j \leq m}$.
11: **Compute** $y_m = arg\min_y ||\beta e_1 - \hat{H}_m y||_2$ **and** $x_m = x_0 + V_m y_m$
12: **If satisfied Stop, else set** $x_0 := x_m$ **and GoTo 1**

Annex 3. Installing the Software

In this section the procedure for installing the software will be explained. All the steps as well as configuration options and FAQ can be found at (Balay et al., 2018b).
The first step is to download the last released version. It can be done using Git with:

```
git clone -b maint https://bitbucket.org/petsc/petsc petsc
```

New patches can be obtained anytime using

```
git pull
```

in the petsc directory.
Another option is to download the compact tarball from PETSc webpage:

Algorithm 4 Right-preconditioned GMRES method for iterative solution of linear systems.

1: **Compute** $r_0 = (b - Ax_0), \beta := ||r_0||_2$, **and** $v_1 := r_0/\beta$
2: **For** $j = 1, ..., m$ **Do**:
3: **Compute** $w := AM^{-1}v_j$
4: **For** $i = 1, ..., j$ **Do**:
5: $h_{ij} := (w_j, v_i)$
6: $w_j := w_j - h_{ij}v_i$
7: **end Do**
8: **Compute** $h_{j+1,j} = ||w_j||_2$ **and** $v_{j+1} = w_j/hj + 1, j$
9: **end Do**
10: **Define** $V_m := [v_1, \ldots, v_m]$, **the Hessenberg matrix** $\hat{H}_m = \{h_{ij}\}_{1 \leq i \leq m+1, 1 \leq j \leq m}$.
11: **Compute** $y_m = arg\min_y ||\beta e_1 - \hat{H}_m y||_2$ **and** $x_m = x_0 + M^{-1}V_m y_m$
12: **If satisfied Stop, else set** $x_0 := x_m$ **and GoTo 1**

```
petsc-3.10.3.tar.gz
```

And then untar from terminal to create the PETSc folder:

```
tar -xzf petsc-3.10.3.tar.gz
```

Once that PETSc directory has been created the installation procedure can begin. If MPI and BLAS/LAPACK are already installed it is possible to configure the software from petsc directory with:

```
./configure
make all test
```

In case MPI and BLAS/LAPACK are not present in the machine (most of the cases) user can specify to download and install them with the options:

```
./configure --download-mpich --download-fblaslapack
make all test
```

It is possible to specify the compilers that the user is willing to use:

```
./configure --with-cc=gcc --with-cxx=g++ --with-fc=gfortran --download-mpich --
    download-fblaslapack
make all test
```

In this case PETSc will be configured with C, C++ and Fortran compilers. All the configuration options can be found in PETSc webpage.
Once that all tests have been run successfully, PETSc is ready to be used. The subfolders are divided according to Figure 11. Every family object folder contains a list of examples already implemented so the user can have a better understanding of the libraries. In this way the application code can be written starting from a draft.

Annex 4. Setting up a small cluster computing environment

The configuration of machine's cluster have been done through GNU bash built-in commands of UNIX systems, that means from terminals of each machine. All the commands that will follow are

then executed in a bash shell. First of all, we create a brand new profile into each machine that we call MPIUSER. This is important to get a neat configuration as well as because executable file must have same path. The new user can be create as follows:

```
sudo adduser mpiuser sudo
```

Where the SUDO at the end gives admin privileges to the user MPIUSER. Then, we make PETSc working on each machine, specifically configured with same MPICH version for both machines. Notice that path of installations must be the same in both machines. The procedure is the same explained in Paragraph 6.
Before continuing is good to check that the remote connection between machines work in the proper way. To do such a thing, we need IP numbers. The IP numbers can be found through the IFCONFIG command that lists all the active connections of the machine (included the one with itself). Notice that the command must be executed from each machine. Filtering the output of IFCONFIG for the WLAN which we want to use as local network, in our case we get the two IPs, that in our case resulted in:

```
10.0.0.102 (machine1)
10.0.0.101 (machine2)
```

The next step is host file configuration. This step is not explicitly needed but allows to get a neater configuration. Host files allow to give a name to each machine so that the machines are identified with their names rather than the IPs. Every time an application asks for the IP, we just put the host name and then the DNS servers activate and get the correspond IP for the application. The host file can be found in /ETC/HOSTS directory of each machine. Thus we access the file as follows:

```
sudo vim /etc/hosts
```

And make for both the modification:

```
10.0.0.102 master
10.0.0.101 client
```

At this stage we check that our machines are able to ping with each other and that our configuration of host file properly worked as follows:

```
ping master
ping client
```

Respectively from client machine and master machine. The next necessary condition for our MPI cluster to work is that each machine must be able to remotely log into the other one. So we check that remote log in from master to client and from client works properly. The bash built-in SSH command is a command exactly for these purposes. Its use is straighforward: for instance, to log into client machine from master:

```
ssh client
```

The command above asks for password though. This is not wanted when making up a MPI cluster: message communications must be exchanged freely. To enable passwordless login we generate encrypted keys. Let just type:

```
ssh-keygen -t rsa
```

To create the key. Then, let's share the keys with all the machines that take part to the cluster, that means, in our case only with the client machine. Hence we type:

```
ssh-copy-id client
```

To copy the access key to client machine. At this stage, we're able to passwordless log into client machine. Notice that firewall must be disabled so as to be able to log remotely through ssh. Firewall disabling is done as:

```
sudo ufw disable
```

The configuration is complete. In the next section we will how to run the pf simulation.

Annex 5. Running the Distributed Power Flow Simulations

Accomplished the configuration of the cluster computing, the code is ready to be compiled and run. Hence, the compiling process is made as usual through:

```
make pf
```

That calls the MAKEFILE and generate an executable file called "pf". To carry out the parallel execution onto machines of the code an hostfile must be created. This is quite simple. Create a file named "hostfile" in the same folder of the master node by typing in the terminal:

```
sudo vi hostfile
```

And write into it the number of processes that one wants to perform on each of the machine that take part into the distributed simulation. Namely, write into this file:

```
master:n1
client:n2
```

Where $n1$ and $n2$ are two integers representing the number of processes that are wanted to be run onto each machine.

At this stage the last thing to do is to run from master the simulation by typing in the folder of the executable file:

```
mpiexec -n <np> -f /path/to/hostfile ./power -pfdata <case.m>
```

Where np is the number of total processes, namely the sum of $n1$ and $n2$.

GETTING IN TOUCH WITH THE EU

In person

All over the European Union there are hundreds of Europe Direct information centres. You can find the address of the centre nearest you at: https://europa.eu/european-union/contact_en

On the phone or by email

Europe Direct is a service that answers your questions about the European Union. You can contact this service:

- by freephone: 00 800 6 7 8 9 10 11 (certain operators may charge for these calls),

- at the following standard number: +32 22999696, or

- by electronic mail via: https://europa.eu/european-union/contact_en

FINDING INFORMATION ABOUT THE EU

Online

Information about the European Union in all the official languages of the EU is available on the Europa website at: https://europa.eu/european-union/index_en

EU publications
You can download or order free and priced EU publications from EU Bookshop at: https://publications.europa.eu/en/publications. Multiple copies of free publications may be obtained by contacting Europe Direct or your local information centre (see https://europa.eu/european-union/contact_en).

KJ-NA-29998-EN-N

The European Commission's science and knowledge service

Joint Research Centre

JRC Mission

As the science and knowledge service of the European Commission, the Joint Research Centre's mission is to support EU policies with independent evidence throughout the whole policy cycle.

EU Science Hub

ec.europa.eu/jrc

@EU_ScienceHub

EU Science Hub - Joint Research Centre

Joint Research Centre

EU Science Hub

Publications Office of the European Union

doi:10.2760/869640

ISBN 978-92-76-11252-5