Universidad de Jaén

Escuela de Doctorado

TESIS DOCTORAL

ADVANCED ALGORITHMS FOR POWER FLOW ANALYSIS

PRESENTADA POR:
MARCOS TOSTADO VÉLIZ

DIRIGIDA POR:
FRANCISCO JURADO MELGUIZO

JAÉN, septiembre 2020
ISBN…….

Universidad de Jaén

Escuela de Doctorado

TESIS DOCTORAL

ADVANCED ALGORITHMS FOR POWER FLOW ANALYSIS

Marcos Tostado Véliz

El acto de defensa y lectura de Tesis se celebra el día……de 2020 en la Universidad de Jaén, ante el siguiente Tribunal evaluador, quien decide otorgar la calificación de:

.....

El Presidente **El Secretario**

El vocal

Universidad de Jaén

Escuela de Doctorado

TESIS DOCTORAL

ADVANCED ALGORITHMS FOR POWER FLOW ANALYSIS

Director de Tesis:

Francisco Jurado Melguizo *Catedrático de Universidad (Universidad de Jaén)*

TRIBUNAL EVALUADOR

Presidente:
Secretario:
Vocal:
Suplente:
Suplente:

Jaén, septiembre 2020

Acknowledgement

At the end of this work, I remember many people who have participated somehow to make it possible. Let me express with brief words my deep acknowledgement.

Firstly, to my pioneer professors from University of Seville Dr. Juan Carlos del Pino, Dr. Pedro Luis Cruz Romero and Dr. Jose Luis Martínez Ramos. They gave me the very first encouragement to undertake This path.

Secondly, to my director Dr. Francisco Jurado Melguizo and Dr. Salah Mohamed Kamel Mohamed Hassan. All the words cannot express my feelings towards them. Undoubtedly, this work could not be possible without their effort and confidence.

My appreciation also goes to my colleagues of Electrical Engineering Department of University of Jaén.

And finally, my warmest thanks is dedicated to the people who supported me from the distance. To my mother, my brother and my father and, of course, special thanks to Fany (with one n).

Marcos Tostado Véliz

ABSTRACT

The Power Flow is a very well-known computational tool in Power System Analysis. It finds multiple applications, like control and planning of power systems, security analysis, dynamic simulations or optimization among others. Due to its importance for power systems operators, the Power Flow has attracted huge research efforts since mid 50's. Even now, it is object of many works and papers.

Mathematically, the Power Flow is a nonlinear problem which is typically solved using iterative techniques. Usually, the Newton Raphson technique is used for this purpose in industry tools. Although, plenty of solution techniques have been developed, they have not found widespread usage in industry tools.

Despite the plethora of works available in the literature, many gaps have been on this field have been detected. The main issues which I aim at filling by this work can be summarized as follows:

- In ill-conditioned systems, the performance of the standard Newton Raphson method is totally unsatisfactory. In order to overcome this drawback, many robust techniques have been proposed in the literature. However, this kind of methods are totally inefficient to be used in industry tools.
- Despite the huge amount of literature available about solution of nonlinear equations, most of available solution techniques have not been explored for Power Flow solution yet. This is surprising since some of the available techniques may outperform the conventional

Newton Raphson method and, consequently, replace the traditional technique as the most standard Power Flow solver.

This work aims at filling this gap by developing or analysing novel solution techniques for the Power Flow problem. Some of the studied techniques have been developed in the context of this work, while others are already available in the literature but they have not been considered as Power Flow solvers yet.

Various numerical results in several large-scale realistic systems are provided in order to validate the analysed techniques.

Contents

List of Figures v

List of Tables ix

Chapter 1: Introduction 11

1.1.- Context and Motivation 11

1.2.- Literature Review 12

1.3.- Detected Gaps 14

1.4.- Thesis Objectives 15

1.5.- Thesis Layout 15

Chapter 2: Conventional Power Flow solvers 17

2.1.- Power Flow Equations 17

2.2.- Classification of Buses 19

2.3.- Newton Raphson 19

2.4.- Fast Decoupled Load Flow 21

2.5.- DC Power Flow 23

2.6.- A Logic-Switching Strategy for handling with Equipment Limits 25

2.7.- Conclusions 25

Chapter 3: Various Power Flow solvers based on the Continuous Newton's method 27

3.1.- Outlines of the Continuous Newton's method 27

3.2.- Various Power Flow Solvers based on the Runge-Kutta Formulas 29

3.2.1.- Studied Runge-Kutta methods.....30

3.2.2.- Step size control.....32

3.3.- Adams Bashforth's methods 33

3.3.1.- Step size control.....35

3.4.- Bulirsch Stoer Algorithm 35

3.4.1.- The Reverse-Bulirsch-Stoer algorithm 38

3.4.2.- Step size control.....39

3.5.- The Continuous Newton-Broyden paradigm 39

3.5.1.- Step size control.....41

3.6.- A Framework based on a Multistep Continuous Newton's paradigm 42

3.6.1.- Step size control.....43

3.6.2.- Two Power Flow solvers based on the developed multistep Continuous Newton's paradigm.....44

3.7.- Conclusions 45

Chapter 4: A Power Flow solver based on Gauss-Newton method 47

4.1.- Gauss-Newton method for Power Flow analysis 47

4.2.- Conclusions 49

Chapter 5: Robust and Efficient Power Flow solvers.... 51

5.1.- A robust and efficient Power Flow Solution paradigm based on a Combined approach 51

5.1.1.- Basic principles of Homotopy 52

5.1.2.- Solution of the Power Flow equations using the homotopy principle . 52

5.2.- A Robust Power Flow Solution technique based on Richardson Extrapolation 55

5.3.- A three-stage Power Flow solver based on a Semi-Implicit approach 57

5.3.1.- Solving nonlinear equations using a Semi-Implicit approach 58

5.3.2.- A three-stage Power Flow solver based on a Semi-Implicit approach 59

5.4.- A Robust Power Flow solver based on the Composite Newton-Cotes formula 62

5.4.1.- Newton-Cotes formulas 63

5.4.2.- Motivation on the usage of the Newton-Cotes formulas 64

5.4.3.- Application of the Newton-Cotes formulas for solving Power Flow ... 64

5.5.- A Power Flow approach based on the S-Iteration process 68

5.6.- A robust and efficient PF solver based on Heun and King-Werner's methods 70

5.6.1.- Natural suitability for both well and ill-conditioned cases 73

5.6.2.- Natural evolution towards quadratic convergence 74

5.7.- Conclusions 75

Chapter 6: High Order Newton-like methods 77

6.1.- The Newton-Raphson Predictor Corrector 77

6.2.- An efficient multistep method 79

6.2.1.- Convergence analysis *81*

6.3.- Comparative analysis 83

6.4.- Conclusions 84

Chapter 7: Numerical Results 85

7.1.- Simulation conditions 85

7.2.- Studied systems 85

7.3.- Results for those methods based on the Continuous Newton's paradigm 87

7.4.- Results for Regularization-based methods 91

7.5.- Results for robust and efficient Power Flow solvers 94

7.6.- Results for high order Newton-like solvers 97

7.7.- Conclusions 100

Chapter 8: Conclusions and Future works 103

8.1.- Conclusions 103

8.2.- Future Works 105

Appendix A: Linear Systems solution in Matlab 107

Appendix B: Curriculum Vitae 113

Appendix C: Thesis Publications 115

Bibliography 119

List of Figures

Fig. 1.1 - Requirements for a PF solver 12

Fig 1.2 - Thesis Layout 16

Fig 2.1 - Pictorial representation of the Jacobian matrix for the IEEE 30-bus test system. The lighter the area the higher the absolute value of the elements of the Jacobian matrix 22

Fig 2.2 - Maximum phase angle error introduced by the DC power flow as a function of the loading level for the IEEE 30-bus system 24

Fig. 3.1 - Structure of the Butcher's Tableau for the Runge-Kutta methods 30

Fig 3.2 - Influence of the step size for solving the ODE $\dot{y} = yt$, with initial condition $\dot{y}(0) = 2$ 32

Fig 3.3 - Sketch of BS 37

Fig. 3.4 - A comparison between BS and RBS. 38

Fig 3.5 - Accuracy of the Explicit Euler and 4th order Runge Kutta formulas for solving the ODE $\dot{y} = yt$, with initial condition $\dot{y}(0) = 2$ and $h = 0{,}2$ ... 44

Fig. 5.1 - Sketch of the homotopy principle 52

Fig. 5.2 - Solution paths and the progression of the PF state vector in 3-bus tutorial example [73], using the presented solution paradigm 54

Fig. 5.3.- Conceptual idea of the developed 3S-SIA. 60

Fig. 5.4.- Sketch of Composite Integration techniques using 2 (left), 4 (middle) and 8 (right) subintervals 63

Fig 5.5.- Behavior of proposed CNC-PF for large values of the intermediate (a) and extreme slope coefficients (b) 67

Fig 5.6.- Behavior of slope coefficients depending on the size of main interval 67

Fig 5.7.- Behavior of proposed CNC-PF when the size of main interval is small enough (Note as the point calculated for a large value of ρ is closer to the solution).....67

Fig. 5.8.- Behavior of developed HKW in case of ill-conditioned cases 74

Fig. 5.9.- Behavior of developed HKW in case of well-conditioned cases...74

Fig. 6.1.- Sketch of NRPC 78

Fig. 6.2.- Efficiency index (6.19) for various PF solution methods. 84

Fig. 7.1.- Convergence profiles for base cases in the case3012wp (a), case3375wp (b) and case13659pegase (c). 93

Fig. 7.2.- Error in voltage angles for the limit load scenario in the case13659pegase 94

Fig. 7.3.- Convergence profiles of various high order Newton-like methods for the case300 for base cases 99

Fig. A.1 - Examples of a sparse (left) and a dense (right) matrix. As observed, although both matrixes have the same dimension, the dense matrix has much more nonzero elements (nz) than the sparse matrix 108

Fig. A.2 - Code for converting the matrix **A** into sparse format in Matlab .108

Fig. A.3 - Code for solving the linear system (A.1) using backward substitution in Matlab.....109

Fig. A.4 - Code for solving the linear system (A.1) using LDU factorization in Matlab 110

Fig. A.5 - Code for solving the linear system (A.1) using Cholesky factorization in Matlab. If the matrix **A** is not positive-definite, the command chol(.) will fail 111

List of Tables

Table 2.1 - Main characteristics of the PF buses 19

Table 2.2 - Description of the most conventional PF solvers 26

Table 3.1 - Parameters for ARK3 31

Table 7.1.- Main characteristics of the studied systems 87

Table 7.2.- Loading levels considered in simulations 87

Table 7.3.- Parameters for the Continuous Newton-based solvers considered in simulations 88

Table 7.4.- Results of the Continuous Newton-based methods for base cases 89

Table 7.5.- Results of the Continuous Newton-based methods with reactive limits 90

Table 7.6.- Results of the Continuous Newton-based methods for limit load cases 91

Table 7.7.- Results of some Regularization-like methods for base cases 92

Table 7.8.- Results of some Regularization-like methods with reactive limits 93

Table 7.9.- Results of some Regularization-like methods for limit load cases 93

Table 7.10.- Parameters for the robust and efficient solvers considered in simulations 95

Table 7.11.- Results of the robust and efficient methods for base cases 96

Table 7.12.- Results of the robust and efficient methods with reactive limits 96

Table 7.13.- Results of the robust and efficient methods for limit load cases 97

Table 7.14.- Results of various high order Newton-like methods for base cases 98

Table 7.15.- Total LU factorizations required by various high order Newton-like methods for base cases 99

Table 7.16.- Results of various high order Newton-like methods with reactive limits.....100

Table 7.17.- Results of various high order Newton-like methods for limit load cases 100

Chapter 1

Introduction

1.1.- Context and Motivation

Power Flow (PF) is probably the most important computational tool in power system analysis. It finds innumerable such that power system planning, control and operation or security and dynamic analysis, among others [1].

Mathematically, PF is a nonlinear problem which is customary solved using a nonlinear iterative solver. From a PF solution method one mainly expects computational efficiency (avoid heavy calculations), reliability (result obtained has to be accurate enough) and robustness (a PF solver should not be affected by external factors like the starting guess considered for initializing the iterative procedure) (see Fig. 1.1). Traditionally, the Newton-Raphson technique (NR) has been widespread using in industry tools.

Despite that NR is frequently accepted in power system industry, this solver still suffers some important drawbacks such that:

- NR presents local convergence. It means that its convergence features may be lost if the starting guess considered for initializing the iterative procedure lies too far from the PF solution. In fact, if this starting point lies outside of the so-

called Region of Attraction, NR will fail for finding the PF solution. This behaviour is normally due to divergence or an oscillatory pattern.

NR has quadratic convergence, which leads this mapping to reaches the solution in 3-10 iterations. Although this amount of iterations may look acceptable, important computational savings may be achieved with higher convergence orders.

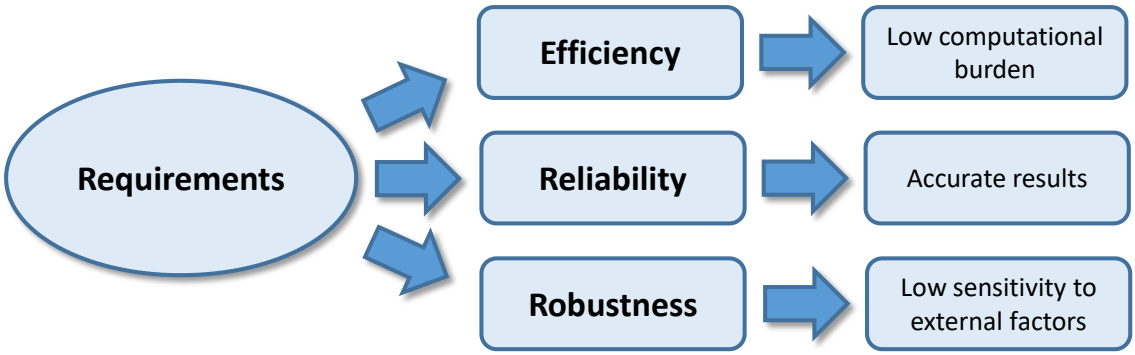


Fig. 1.1 - Requirements for a PF solver

Due to the importance of PF for power system operators, the development and analysis of novel solvers, which aim at outperforming NR in some of the aforementioned aspects is always well-justified and attractive.

1.2.- Literature Review

Different PF solvers have been developed during decades since the pioneer works at mid 50's [2]. Conventional solvers were developed on the basis of Newton-like techniques [3], decoupled approaches [4-6], second order methods [7], [8], direct solutions [9], [10] and inexact solvers [11], among others [12]. Apart from the conventional rectangular and polar PF formulations based on power injections, other alternative forms have been explored such as those based on current injections [13], [14], hybrid approaches [15], *dq* reference frame [16], [17], multiphase framework [18-20] and complex notation [21].

PF solution of ill-conditioned systems became to be studied using minimization and optimal step size-based techniques. The first steps in this direction were made in [22, 23], and posteriorly embellished in other references [24-26]. This kind of techniques are typically called optimal step-size solvers, since they are based on calculating an optimal step, which aims at modifying the increment vector (5) so that the PF state vector is

conducted on a direction which the residuals are reduced. Although this kind of techniques are quite robust, they are unaffordable slow as the solution is approached [27].

Continuation or homotopy methods have been profusely studied for avoiding the singularity of the Jacobian matrix in the vicinity of the maximum loadability point. The Continuation Power-Flow [28] has been extensively used for voltage stability analysis. Alternative approaches have been also considered using decoupled techniques [29] or in distribution systems [30]. Homotopic approaches have been also used for calculating all the possible PF solutions [31]. More recently, various works have been conducted on studying the applicability of the Holomorphic Embedding technique for the PF equations [32-34]. These techniques may be very robust and allow to calculate all the PF solutions, however, they are not competitive with other conventional solvers due to they typically require many PF calculations.

Alternative approaches have been conducted on solving the PF equations as an optimization problem. Thus, some metaheuristic techniques can be exploited as PF solvers [35-37]. These solution approaches are globally convergent (they may reach the PF solution independently on the starting guess taken) and versatile enough to efficiently incorporating some topics like the equipment limits. However, they result very inefficient in comparison with conventional iterative solvers.

The Continuous Newton's method [38], establishes that a set of algebraic nonlinear equations can be conceived as an autonomous set of ordinary differential equations. On the basis of this analogy, any numerical integration routine can be used for solving this kind of systems. This idea was exploited for PF analysis by Milano [39]. Here, a formal analogy between NR and the Forward Euler method was established. In this same reference, a robust PF solver based on the 4^{th} order Runge-Kutta method was developed. Reference [39] only explored the explicit formulation of the Continuous Newton's method, pointing out that its implicit counterpart may be not efficient due to the requirement of factorizing the inverse of the Jacobian matrix. This issue was tackled by the same author in [40], proposing a simple alternative for avoiding this calculation. Those PF solution methods based on the Continuous Newton's principle are typically more efficient than the optimal step size-based techniques, however, these methods require as many factorizations as stages of the numerical arrangement, which occasionally results on an unaffordable computational

burden. In addition, although generally wider convergent than NR, their convergence features are not global.

The set of equations (1) can be also raised as an artificial dynamic system. Thus, any numerical integration routine (like ode solvers in Matlab) can be used for solving them. This idea has been tackled in some recent papers [41-43]. The resulting solution paradigm is barely affected by the starting guess, however, it is computationally not competitive and totally prohibitive in large-scale systems.

The Regularization methods are devoted on avoiding the singularity of the Jacobian matrix by adding some proper elements on its diagonal. These techniques have been studied for PF considering some Levenberg-like methods [44-48]. These approaches are quite robust, however, they present some important drawbacks. For example, this kind of methods may offer very inaccurate results. On the other hand, they tend to be very slow when the starting guess lies far away to the actual PF solution.

On the other hand, the high order Newton-like methods are those techniques with higher convergence orders in comparison with NR (i.e. higher than 2). To the best of my knowledge, only the reference [49] has tackled the applicability of this kind of techniques for PF analysis. The authors of this reference considered a 3^{rd}, 4^{th} and 5^{th} order techniques. Among the tested methods, only the 3^{rd} order approach may be considered competitive with NR as just one factorization is required each iteration.

1.3.- Detected Gaps

After a deep analysis of the available literature reported before, the following gaps have been detected:

- Most of robust PF solvers are totally inefficient, which has limited their widespread application in industry tools. In addition, some of them are not easily coded to be considered in standard computational tools.
- Although ill-conditioned systems are becoming more frequent [41], the well-conditioned cases are by far the most common situation. Since most of the available robust techniques are not competitive with NR, its utilization in industry tools has been very limited.

- Although on the basis of the Continuous Newton's method any numerical arrangement can be applied as PF solver, only the 4^{th} order Runge-Kutta has been studied so far [39].

Despite the huge amount of high-order Newton-like methods available in the literature (see e.g. [50] and references therein), only the reference [49] has been devoted on analyzing this kind of techniques for PF analysis.

1.4.- Thesis Objectives

This Thesis has as main objective exploring a variety of alternative PF solvers, which aim at filling the gaps showed by NR. For example, it is aimed to develop and validate a variety of robust and efficient solvers, which should be suitable to be widespread used in industry applications. To achieve that, the developed techniques should be computationally competitive with NR and, of course, outperform other available robust methodologies. On the other hand, although NR is quite efficient, the high order Newton-like methods may easily outperform it as this kind of techniques usually require less iterations for converging, surprisingly, these family of methods has been barely studied for PF yet.

In this context, a huge variety of different robust and efficient techniques will be presented through this Thesis. Some of the studied techniques are already available in the literature, but they have not been explored for PF yet, while other methods have been developed in the context of this work. The analyzed techniques have been mainly considered with the aim at filling the detected gaps in the available literature.

Various numerical results on either well or ill-conditioned systems are provided in order to show the performance of the analyzed methodologies. In this regard, this work has especially focused on realistic large-scale systems, in order to validate the tested approaches for industry tools.

1.5.- Thesis Layout

This Thesis is organized in several chapters. A brief explanation about the content and purpose of each chapter is provided below while the Thesis layout is showed in Fig 1.2.

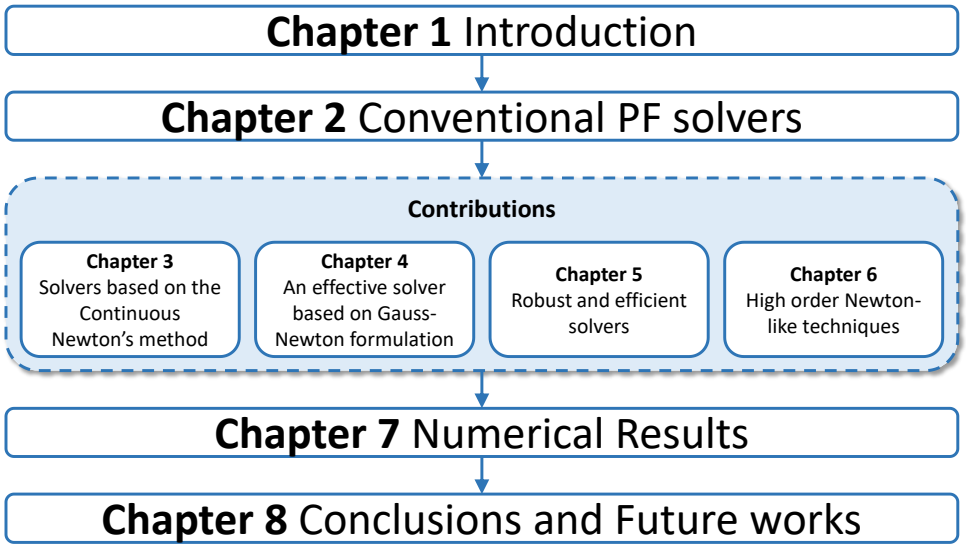


Fig 1.2 - Thesis Layout

- Chapter 2 - Conventional PF solvers: in this chapter, the most conventional solvers are described.
- Chapter 3 - Solvers based on the Continuous Newton's method: this section presents some methodologies which aims at further analyzing the Continuous Newton's method as applied for PF analysis. In addition, different alternative solution paradigms are also presented.
- Chapter 4 - An effective solver based on Gauss-Newton formulation: this chapter presents an effective PF solver based on Gauss-Newton formulation.
- Chapter 5 - Robust and efficient solvers: several robust and efficient PF solvers are developed in this chapter.
- Chapter 6 - High order Newton-like techniques: in this chapter, various high order Newton-like methods are studied for PF analysis. Some of them are already available in the literature but they have not been studied for PF calculation yet, while other have been developed in the context of this work.
- Chapter 7 - Numerical results: various numerical results on either well or ill-conditioned systems are provided in order to show the performance of the analyzed methodologies.
- Chapter 8 - Conclusions and Future works: the main conclusions and potential future lines are duly drawn in this chapter.

Chapter 2

Conventional Power Flow Solvers

Many PF solvers have been developed or studied during decades. However, very few have found widespread usage in industry tools. In this section, the PF equations in their most standard form are presented. Secondly, the most conventional PF solvers are described. Finally, a logic-switch sequence for incorporating some functional restrictions within a PF algorithm is described. In addition, the main advantages and drawbacks of the most conventional methods are commented. It is worth noting that most references include the Gauss-Seidel technique as a conventional PF solver. However, it has not been included in this work since this Thesis is mainly devoted on industry applications. The Gauss-Seidel method, at least recently, has found a marginal usage in industry applications. This technique is, however, frequently used for academy purposes. The interested reader may find a deep explanation about this method and others in [27].

2.1.- Power Flow Equations

In their most general form in polar coordinates, the PF equations are defined as a set of n nonlinear algebraic equations as follows:

$$\mathbf{g}(\boldsymbol{x}) = \begin{cases} \mathbf{g_P}, \text{For all buses} \\ \mathbf{g_Q}, \text{For PQ buses} \end{cases} \tag{2.1}$$

$$g_{P_i} = P_i^{sch} - \sum_{j=1}^{n} |V_i| \left|V_j\right| \left|Y_{ij}\right| \cos\left(\theta_{ij} - \delta_i + \delta_j\right) \tag{2.2}$$

$$g_{Q_i} = Q_i^{sch} - \sum_{j=1}^{n} |V_i| \left|V_j\right| \left|Y_{ij}\right| \sin\left(\theta_{ij} - \delta_i + \delta_j\right) \tag{2.3}$$

where, $P_i^{sch} \in \mathbb{R}$ and $Q_i^{sch} \in \mathbb{R}$ are the active and reactive power injected at i^{th} bus, respectively. $V_i \angle \delta_i \in \mathbb{C}$ is the complex voltage at i^{th} bus. $Y_{ij} \angle \theta_{ij} \in \mathbb{C}$ is the ij^{th} element of the admittance matrix, and $n \in \mathbb{N}$ is the size of the state vector $\boldsymbol{x} \in \mathbb{R}^n$, which is given by:

$$\boldsymbol{x} = \left[\boldsymbol{\delta_{PV}} | \boldsymbol{\delta_{PQ}} | \boldsymbol{V_{PQ}}\right]^T \tag{2.4}$$

where, $\boldsymbol{\delta_{PV}} \in \mathbb{R}^{n_g}$ is the vector of voltage angles at PV buses, $\boldsymbol{\delta_{PQ}} \in \mathbb{R}^{n_l}$ is the vector of voltage angles at PQ buses and $\boldsymbol{V_{PQ}} \in \mathbb{R}^{n_l}$ is the vector of voltage angles at PQ buses. On the other hand, $n_g \in \mathbb{N}$ and $n_l \in \mathbb{N}$ represent the total number of PV and PQ buses, respectively.

When (2.1) are well-conditioned, its solution is easily reachable, however, conventional solvers may present convergence difficulties when the PF equations are ill-conditioned (or ill-posed). A power system network may be ill-conditioned due to some of the following reasons:

- Large R/X ratios.
- Heavy loading profile (i.e. close to the maximum loadability point).
- Incidence at the same bus of large and short branches.
- Cascading failures.
- Difficult initialization of the iterative procedure.

During this Thesis, the definition of ill-conditioned system given below is followed [39].

Definition 2.1. *Ill-conditioned systems*: a power system network will be denoted as ill-conditioned if, despite its solution does exists, it is not reachable using NR and a flat start (e.g. all voltage magnitudes equal to 1 pu and all voltage angles at PQ buses equal to 0 deg).

2.2.- Classification of Buses

In PF analysis, the power system buses are typically classified according their known and unknown specifications, as follows:

- *Slack bus(es)*: at a slack bus, the voltage angle and magnitude are known, while the injected active and reactive powers are unknown. In a network may be one or more slack bus(es).
- *PV buses*: at a PV bus, the injected active power and the voltage magnitude are known, while the injected reactive power and the voltage angle are unknown. This kind of buses provides a variable to the state vector (voltage angle). The injected reactive power is normally calculated so that the nodal voltage magnitude is kept at its specified value. These buses normally correspond with generator nodes.
- *PQ buses*: at a PQ bus, the injected active and reactive power are known, while the voltage angle and magnitude are unknown. So that, this kind of buses provide two variables to the state vector. These buses normally correspond with load nodes.

Table 2.1 summarizes the main characteristics of the different type of buses. Although the aforementioned buses are the most commonly classification in conventional PF analysis, other types have been considered for some specific situations such as the inclusion of FACTS (see e.g. [51]).

Table 2.1 - Main characteristics of the PF buses

Bus type	Known	Unknown
Slack	V, δ	P, Q
PV	V, P	δ, Q
PQ	P, Q	V, δ

2.3.- Newton Raphson

One can observe that (2.1) are strongly nonlinear. Hence, they cannot be directly solved. Among all available nonlinear solution techniques, NR has been by far the most widely used in PF analysis. Solution of the set of equations (2.1) using NR is iteratively defined by:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \left[\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right)\right]^{-1}\mathbf{g}\left(\boldsymbol{x}^{(k)}\right) \tag{2.5}$$

where, $\mathbf{g}' = \nabla_{\boldsymbol{x}}\mathbf{g} \in \mathbb{R}^{n\times n}$ is the Jacobian matrix and the superindexes indicate the iteration counter. An iteration of the mapping (2.5) involves a function and a matrix evaluation, a linear system solution and a LU factorization. This latter is considered as the heaviest part of NR [39]. The elements of the PF Jacobian matrix are calculated as follows:

$$\mathbf{g}'(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial P_1}{\partial \delta_1}^{(k)} & \cdots & \frac{\partial P_1}{\partial \delta_n}^{(k)} & \frac{\partial P_1}{\partial |V_1|}^{(k)} & \cdots & \frac{\partial P_1}{\partial |V_n|}^{(k)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial P_n}{\partial \delta_1}^{(k)} & \cdots & \frac{\partial P_n}{\partial \delta_n}^{(k)} & \frac{\partial P_1}{\partial |V_1|}^{(k)} & \cdots & \frac{\partial P_1}{\partial |V_n|}^{(k)} \\ \frac{\partial Q_1}{\partial \delta_1}^{(k)} & \cdots & \frac{\partial Q_1}{\partial \delta_n}^{(k)} & \frac{\partial Q_1}{\partial |V_1|}^{(k)} & \cdots & \frac{\partial Q_1}{\partial |V_n|}^{(k)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial Q_n}{\partial \delta_1}^{(k)} & \cdots & \frac{\partial Q_n}{\partial \delta_n}^{(k)} & \frac{\partial Q_n}{\partial |V_1|}^{(k)} & \cdots & \frac{\partial Q_n}{\partial |V_n|}^{(k)} \end{bmatrix} = \begin{bmatrix} [\mathbf{J_1}] & [\mathbf{J_2}] \\ [\mathbf{J_3}] & [\mathbf{J_4}] \end{bmatrix} \tag{2.6}$$

The diagonal elements of the Jacobian matrix are given by:

$$\frac{\partial P_i}{\partial \delta_i} = \sum_{j\neq i}|V_i||V_j||Y_{ij}|\sin\left(\theta_{ij} - \delta_i + \delta_j\right) \tag{2.7}$$

$$\frac{\partial P_i}{\partial |V_i|} = 2|V_i||Y_{ii}|\cos\theta_{ii} + \sum_{j\neq i}|V_j||Y_{ij}|\cos\left(\theta_{ij} - \delta_i + \delta_j\right) \tag{2.8}$$

$$\frac{\partial Q_i}{\partial \delta_i} = \sum_{j\neq i}|V_i||V_j||Y_{ij}|\cos\left(\theta_{ij} - \delta_i + \delta_j\right) \tag{2.9}$$

$$\frac{\partial Q_i}{\partial |V_i|} = 2|V_i||Y_{ii}|\sin\theta_{ii} + \sum_{j\neq i}|V_j||Y_{ij}|\sin\left(\theta_{ij} - \delta_i + \delta_j\right) \tag{2.10}$$

While the off-diagonal elements are calculated as follows:

$$\frac{\partial P_i}{\partial \delta_j} = -|V_i||V_j||Y_{ij}|\sin\left(\theta_{ij} - \delta_i + \delta_j\right) \;\; \forall\, j \neq i \tag{2.11}$$

$$\frac{\partial P_i}{\partial |V_j|} = |V_i||Y_{ij}|\cos\left(\theta_{ij} - \delta_i + \delta_j\right) \;\; \forall\, j \neq i \tag{2.12}$$

$$\frac{\partial Q_i}{\partial \delta_j} = -|V_i||V_j||Y_{ij}|\cos\left(\theta_{ij} - \delta_i + \delta_j\right) \;\; \forall\, j \neq i \tag{2.13}$$

$$\frac{\partial Q_i}{\partial |V_j|} = -|V_i||Y_{ij}|\sin\left(\theta_{ij} - \delta_i + \delta_j\right) \;\; \forall\, j \neq i \tag{2.14}$$

This Section is completed by summarizing the main steps of PF solution by NR using pseudocode in Algorithm 2.1. During this work, $\|\mathbf{g}\|_\infty$ has been taken as convergence

criteria. In addition, the algorithm is considered failed if the iteration counter surpasses a predefined threshold namely $k_{\max}$

Algorithm 2.1: PF solution using NR

1: Let $\boldsymbol{x}^{(0)}$, ε, $k_{\max}$ be given
2: Initialize the iteration counter $k \leftarrow 0$
3: **while** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k)}\right)\right\|_{\infty} > \varepsilon$ **do**
4: Solve (2.5)
5: **if** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k+1)}\right)\right\|_{\infty} < \varepsilon$ **then**
6: **break** *# convergence*
7: **else**
8: $k \leftarrow k + 1$
9: **if** $k > k_{\max}$ **then**
10: **break** *# failure*
11: **end if**
12: **end if**
13: **end do**
14: **return** solution $\boldsymbol{x}^{(k)}$

2.4.- Fast Decoupled Load Flow

The Fast Decoupled Load Flow (FDLF) was proposed at mid 70's [4], with the aim at overcoming the heaviest computational part of NR, i.e. the factorization of the Jacobian matrix. It is based on a clear decoupling between the active and reactive problems in transmission networks. This principle is clearly observed in the structure of the Jacobian matrix. Fig. 2.1 shows the pictorial representation of the Jacobian matrix in polar coordinates for the IEEE-30 bus test system. As observed, submatrices $\mathbf{J_2}$ and $\mathbf{J_3}$ are almost null.

FDLF assumes the following simplifications:

$$\begin{cases} \mathbf{J_1} = \boldsymbol{H}' \\ \mathbf{J_2} = \mathbf{0} \\ \mathbf{J_3} = \mathbf{0} \\ \mathbf{J_4} = \boldsymbol{H}'' \end{cases} \tag{2.15}$$

where $\boldsymbol{H}' \in \mathbb{R}^{(n_b-1)\times(n_b-1)}$, $\boldsymbol{H}'' \in \mathbb{R}^{n_l \times n_l}$ and $n_b \in \mathbb{N}$ is the total number of buses. The matrices $\boldsymbol{H}'$ and $\boldsymbol{H}''$ are simplified admittance matrices, as follows:

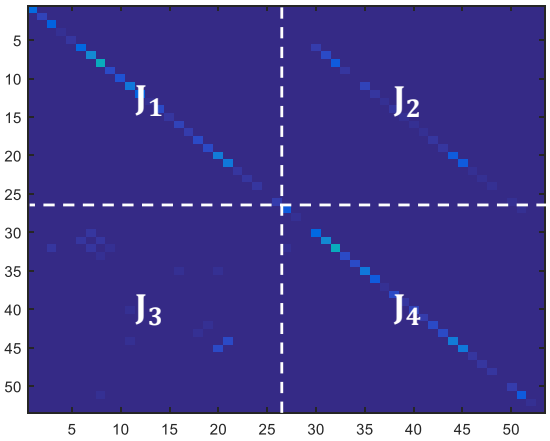


Fig 2.1 - Pictorial representation of the Jacobian matrix for the IEEE 30-bus test system. The lighter the area the higher the absolute value of the elements of the Jacobian matrix

- Line charging, shunts and transformer tap ratios are neglected when computing $\boldsymbol{H}'$.
- Phase shifters are neglected and line charging and shunts are doubled when computing $\boldsymbol{H}''$.

Aforementioned assumptions are reasonable in transmission systems, where the line resistances are typically one order of magnitude lower than reactances. However, these ideas are not generally applicable in distribution systems. In addition, the FDLF formulation only allows to incorporate the voltage magnitudes and angles in polar coordinates as variables.

PF solution using FDLF is achieved by iteratively solving the following two linear systems:

$$\left[\boldsymbol{\delta}_{\boldsymbol{PV}}^{(k+1)}, \boldsymbol{\delta}_{\boldsymbol{PQ}}^{(k+1)}\right] = \left[\boldsymbol{\delta}_{\boldsymbol{PV}}^{(k)}, \boldsymbol{\delta}_{\boldsymbol{PQ}}^{(k)}\right] - [\boldsymbol{H}']^{-1}\left[\boldsymbol{V}^{(k)}\right]\mathbf{g}_P(\boldsymbol{x}) \tag{2.16}$$

$$\left[\boldsymbol{V}_{\boldsymbol{PQ}}^{(k+1)}\right] = \left[\boldsymbol{V}_{\boldsymbol{PQ}}^{(k)}\right] - [\boldsymbol{H}'']^{-1}\left[\boldsymbol{V}^{(k)}\right]\mathbf{g}_Q\left(\boldsymbol{\delta}_{\boldsymbol{PV}}^{(k+1)}, \boldsymbol{\delta}_{\boldsymbol{PQ}}^{(k+1)}, \boldsymbol{V}_{\boldsymbol{PQ}}^{(k)}\right) \tag{2.17}$$

where $[\boldsymbol{V}] = \text{diag}\left(V_1, V_2, \ldots, V_{n_b}\right)$. As observed, the matrices $\boldsymbol{H}'$ and $\boldsymbol{H}''$ are constant during the whole iterative procedure and, therefore, only two LU factorizations have to be computed, which supposes the main merit of FDLF compared with NR. The main steps of FDLF are summarized in Algorithm 2.2 using pseudocode.

Algorithm 2.2: PF solution using FDLF

1: Let $\boldsymbol{x}^{(0)}$, ε, $k_{\max}$ be given
2: Initialize the iteration counter $k \leftarrow 0$
3: Build and factorize $\boldsymbol{B}'$ and $\boldsymbol{B}''$
4: **while** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k)}\right)\right\|_{\infty} > \varepsilon$ **do**
5: Solve (2.16)
6: Solve (2.17)
7: **if** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k+1)}\right)\right\|_{\infty} < \varepsilon$ **then**
8: **break** *# convergence*
9: **else**
10: $k \leftarrow k + 1$
11: **if** $k > k_{\max}$ **then**
12: **break** *# failure*
13: **end if**
14: **end if**
15: **end do**
16: **return** solution $\boldsymbol{x}^{(k)}$

2.5.- DC Power Flow

All methodologies discussed so far manage with the exact form of the PF equations, therefore it is expected that the studied solvers provided the exact solution of (2.1). However, these conventional solvers typically face several problems related with the nonlinearity of the PF equations [27]:

- Difficulties in finding the solution (divergence).
- Multiple solutions.
- In case of no convergence, it is not possible to determine if the problem is unsolvable of its solution cannot be reached.

These issues may be bypassed by simplifying **g** in order to get it linear. In addition, it does allow to obtain an acceptable PF solution quickly. With the aim at achieving it, the following simplifications are introduced:

- All voltage magnitudes are assumed to be equal to 1.0 pu and reactive powers are neglected.
- Line resistance and charging are neglected when computing the simplified admittance matrix $\boldsymbol{H}$
- Bus voltage phases are considered small enough to assert that $\delta_{ij} \approx \delta_{ji}$

The resulting system of equations is:

$$\boldsymbol{P} = \boldsymbol{H}\boldsymbol{\delta} + \boldsymbol{P}^{\phi} \tag{2.18}$$

where $\boldsymbol{P} \in \mathbb{R}^{n_b}$ are the injected active powers, $\boldsymbol{H} \in \mathbb{R}^{n_b \times n_b}$ is the simplified admittance matrix, $\boldsymbol{\delta} \in \mathbb{R}^{n_b}$ is the vector of voltage angles and $\boldsymbol{P}^{\phi} \in \mathbb{R}^{n_b}$ are the power shifts introduced by phase shifting transformers.

As observed, equation (2.18) totally neglects the reactive power problem, which may be arguable in some situations such as voltage stability analysis. Nevertheless, several research efforts have been recently conducted in order to overcome this issue [9, 10]. In addition, accuracy of the results obtained may be compromised especially for high loading levels as observed in Fig 2.2. As main advantage of the DC power flow it is worth commenting that equation (2.18) is totally linear and can be directly solved as follows:

$$\boldsymbol{\delta} = [\boldsymbol{H}]^{-1}\left(\boldsymbol{P} - \boldsymbol{P}^{\phi}\right) \tag{2.19}$$

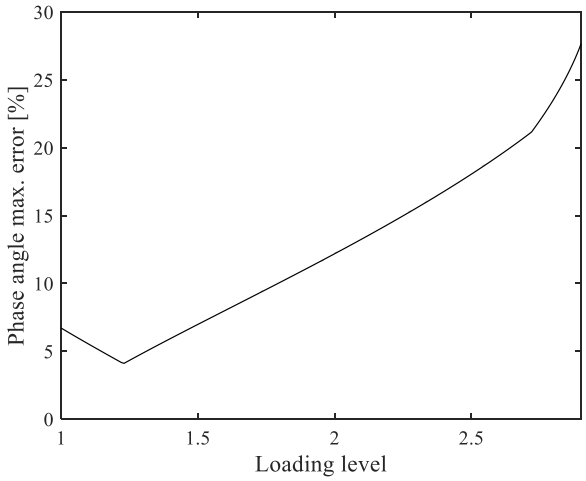


Fig 2.2 - Maximum phase angle error introduced by the DC power flow as a function of the loading level for the IEEE 30-bus system

2.6.- A Logic-Switching Strategy for handling with Equipment Limits

It is important including some restrictions within the PF calculation. One of most important is related with the practical equipment limits. This kind of restrictions can be easily incorporated within the described algorithms using a logic sequence. As example, the following steps are devoted on including the generators' reactive power limits in PF calculation [39, 52]:

Step 1: Solve the PF problem. If there are not PV buses and no solution has been found, the problem is declared unfeasible. Otherwise, take $\boldsymbol{x}^{(k)}$ and go to Step 2.

Step 2: Check if any reactive limit is violated. If yes, go to Step 3, otherwise, return $\boldsymbol{x}^{(k)}$ as solution.

Step 3: Create a $\mathbb{Q}^{+}$ set with those generators' indices whose higher limit is violated and go to Step 4.

Step 4: Create a $\mathbb{Q}^{-}$ set with those generators' indices whose lower limit is violated and go to Step 5.

Step 5: Convert those buses $\in [\mathbb{Q}^{+}, \mathbb{Q}^{-}]$ to PQ buses and go to Step 6.

Step 6: For each $j \in \mathbb{Q}^{+}$ set $Q_j = Q_j^{\max}$ (where $Q_j^{\max}$ is the upper bound on the injected reactive power at bus j) and go to Step 7.

Step 7: For each $j \in \mathbb{Q}^{-}$ set $Q_j = Q_j^{\min}$ (where $Q_j^{\min}$ is the lower bound on the injected reactive power at bus j) and go to Step 8.

Step 8: Take $\boldsymbol{x}^{(0)} = \boldsymbol{x}^{(k)}$ and return to Step 1.

It is worth mentioning that similar strategies can be used for incorporating other equipment limits or controls like tap changers or phase shifts.

2.7.- Conclusions

In this section we have described the most conventional PF solvers. In addition, a logic sequence for incorporating the generators' reactive and other equipment limits and

controls has been explained. Table 2.2 summarized the main advantages and disadvantages of the commented methods.

Table 2.2 - Description of the most conventional PF solvers

Solver	Advantages	Disadvantages
NR	• Simple and efficient. • Quadratic convergence order.	• Fail in ill-conditioned systems (local convergence).
FDLF	• Very efficient.	• Very sensitive to R/X ratio and loading level. • Not suitable for distribution systems. • Only voltage magnitudes and angles can be taken as variables. • Linear convergence.
DC Power Flow	• Reachability of the solution is ensured if $\boldsymbol{H}$ is invertible. • Very fast (solution is directly obtained)	• Reactive problem is neglected (voltage magnitudes are not calculated). • Low accuracy.

Chapter 3

Various Power Flow solvers based on the Continuous Newton's method

On the basis of the Continuous Newton's method [38], any numerical arrangement can be considered for developing a nonlinear solver. The resulting technique usually presents acceptable robustness and efficiency. Reference [39] was the pioneer on considering this topic for PF analysis, however it was limited to study a PF solver based on the 4^{th} order Runge-Kutta, while other numerical approaches were not considered. This Section aims at filling this gap by firstly developing a common framework for considering any member of the Runge-Kutta family [53] for PF analysis. On the other hand, other numerical arrangements like the Adams-Bashforth's family [54] and the Bulirsch-Stoer algorithm (BS) [55] are considered. Finally, alternative further Continuous Newton-based paradigms are developed. Thus, a Continuous Newton-Broyden and a multistep Continuous Newton's paradigm are presented.

3.1.- Outlines of the Continuous Newton's method

Let us consider a set of n autonomous differential equations (ODEs) as follows:

$$\dot{\boldsymbol{x}} = \mathbf{f}(\boldsymbol{x}) \tag{3.1}$$

For a specified time instant t and an arbitrary time step Δt, the simplest method for integrating (3.1) is the Explicit-Euler technique, as follows:

$$\Delta \boldsymbol{x}^{(t)} = \Delta t \mathbf{f}\left(\boldsymbol{x}^{(t)}\right) \tag{3.2}$$

$$\boldsymbol{x}^{(t+\Delta t)} = \boldsymbol{x}^{(t)} + \Delta \boldsymbol{x}^{(t)} \tag{3.3}$$

As pointed out in [39], an analogy between (2.5) and (3.2)-(3.3) can be easily established if one defines:

$$\mathbf{f}(\boldsymbol{x}) = -[\mathbf{g}'(\boldsymbol{x})]^{-1}\mathbf{g}(\boldsymbol{x}) \tag{3.4}$$

In other words, the traditional NR method for solving nonlinear equations can be viewed as the Explicit-Euler formula for solving ODEs.

This paradigm is coherent with the Newton's theorem. As observed in [56], NR solves a nonlinear equation by approximation the area under the function by a rectangle. Keeping this in mind, the Continuous Newton's method can be viewed as a generalization of the Newton's theorem to any other integration form.

In order to study the stability of the Continuous Newton's method, it may be suitable to introduce the following definitions.

Definition 3.1. *Hyperbolic point:* let $\mathbf{G} = \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}$ be the dynamic system associated with an iterative procedure. Then, $\boldsymbol{x}^*$ such that $\mathbf{g}(\boldsymbol{x}^*) = \mathbf{0}$ is an equilibrium point of this system. In addition, $\boldsymbol{x}^*$ is called hyperbolic point if and only if all the eigenvalues associated with the Jacobian of $\mathbf{G}$ at the equilibrium point have a nonzero real part.

Definition 3.2. *Hyperbolic point asymptotically stable:* let $\mathbf{G} = \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}$ be the dynamic system associated with an iterative procedure. If $\boldsymbol{x}^*$ such that $\mathbf{g}(\boldsymbol{x}^*) = \mathbf{0}$ is a hyperbolic point of this system, it is said to be asymptotically stable (or sink), if all the eigenvalues associated with the Jacobian of $\mathbf{G}$ at $\boldsymbol{x}^*$ have a negative real part.

Definition 3.3. *Type-m equilibrium point:* let $\boldsymbol{x}^*$ be hyperbolic equilibrium point of the system $\mathbf{G} = \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}$ associated with an iterative procedure. Then, $\boldsymbol{x}^*$ is said to be type-m if the Jacobian associated with $\mathbf{G}$ at $\boldsymbol{x}^*$ has exactly m eigenvalues with a positive real part.

From the Definitions 3.1-3.3 one expects from the PF solution $\boldsymbol{x}^*$ to be a type-0 equilibrium point. The following Theorem demonstrates that $\boldsymbol{x}^*$ such that $\mathbf{g}(\boldsymbol{x}^*) = \mathbf{0}$ is a type-0 solution for the Continuous Newton's method.

Theorem 3.1. *Stability of the Continuous Newton's method:* let $\boldsymbol{x}^*$ be a PF solution such that $\mathbf{g}(\boldsymbol{x}^*) = \mathbf{0}$, then, it is a type-0 equilibrium point for the Continuous Newton's method.

Proof. The function $\mathbf{G}$ related to the Continuous Newton's method is:

$$\mathbf{G} = -[\mathbf{g}'(\boldsymbol{x})]^{-1}\mathbf{g}(\boldsymbol{x}) \tag{3.5}$$

By differentiating (3.5) with respect $\boldsymbol{x}$ one obtains:

$$\nabla_{\boldsymbol{x}}\mathbf{G} = -\nabla_{\boldsymbol{x}}[\mathbf{g}'(\boldsymbol{x})]^{-1}\mathbf{g}(\boldsymbol{x}) - [\mathbf{g}'(\boldsymbol{x})]^{-1}\mathbf{g}'(\boldsymbol{x}) \tag{3.6}$$

By evaluating (3.6) at the PF solution:

$$\nabla_{\boldsymbol{x}}\mathbf{G}|_{\boldsymbol{x}^*} = -\boldsymbol{I} \tag{3.7}$$

where $\boldsymbol{I} \in \mathbb{R}^{n\times n}$ is the identity matrix. Equation (3.7) says that all the eigenvalues of $\nabla_{\boldsymbol{x}}\mathbf{G}$ at $\boldsymbol{x}^*$ are equal to -1, which completes the proof. □

3.2.- Various Power Flow Solvers based on the Runge-Kutta Formulas

The most general formulation of a Runge-Kutta technique for PF analysis is given by:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + h\sum_{i=1}^{p} b_i \boldsymbol{v}_i^{(k)} \tag{3.8}$$

where, $h \in \mathbb{R}^+$ is the step size, b's $\in \mathbb{R}$ are the weights, $p \in \mathbb{N}$ is the number of stages and $\boldsymbol{v}$'s $\in \mathbb{R}^n$ are calculated as follows:

$$\begin{cases} \boldsymbol{v}_1^{(k)} = \mathbf{f}\left(\boldsymbol{x}^{(k)}\right) \\ \boldsymbol{v}_2^{(k)} = \mathbf{f}\left(\boldsymbol{x}^{(k)} + ha_{21}\boldsymbol{v}_1^{(k)}\right) \\ \boldsymbol{v}_3^{(k)} = \mathbf{f}\left(\boldsymbol{x}^{(k)} + h\left(a_{31}\boldsymbol{v}_1^{(k)} + a_{32}\boldsymbol{v}_2^{(k)}\right)\right) \\ \vdots \\ \boldsymbol{v}_p^{(k)} = \mathbf{f}\left(\boldsymbol{x}^{(k)} + h\left(a_{p,1}\boldsymbol{v}_1^{(k)} + a_{p,2}\boldsymbol{v}_2^{(k)} \ldots + a_{p,p}\boldsymbol{v}_{p-1}^{(k)}\right)\right) \end{cases} \tag{3.9}$$

where, a's $\in \mathbb{R}$ are the elements of the Runge-Kutta matrix. As observed, a Runge-Kutta technique requires p LU factorizations each iteration. This supposes the main computational burden of this kind of methods.

3.2.1.- Studied Runge-Kutta methods

Any member of the family of Runge-Kutta formulas, can be defined using the well-known Butcher's Tableau. The structure of this mnemonic device is shown in Fig. 3.1.

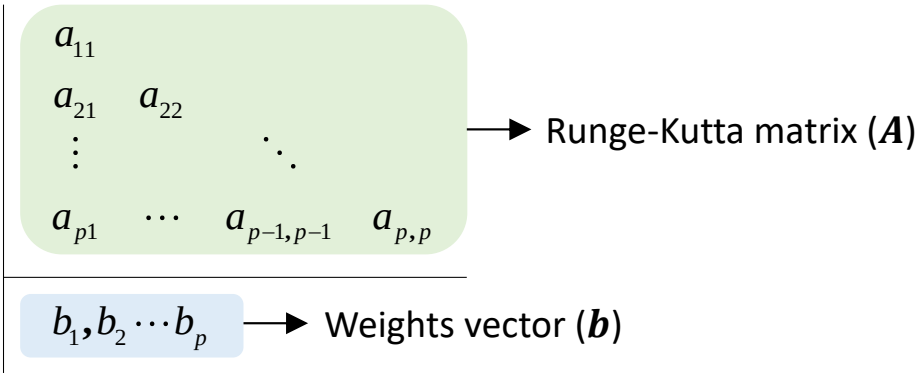


Fig. 3.1 - Structure of the Butcher's Tableau for the Runge-Kutta methods

The Runge-Kutta matrices and weights vectors of the considered PF solvers based on Runge-Kutta techniques are given below.

Midpoint's Rule (MP)

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 \\ 1/2 & 0 \end{bmatrix}, \boldsymbol{b} = [0 \quad 1] \tag{3.10}$$

Heun's 3rd order method (3OH)

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 \\ 1/3 & 0 & 0 \\ 0 & 2/3 & 0 \end{bmatrix}, \boldsymbol{b} = [1/4 \quad 0 \quad 3/4] \tag{3.11}$$

4th order Runge-Kutta method (RK4)

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \boldsymbol{b} = [1/6 \quad 1/3 \quad 1/3 \quad 1/6] \tag{3.12}$$

Simpson's 3/8 rule (S38)

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 0 \\ -1/3 & 1 & 0 & 0 \\ 1 & -1 & 1 & 0 \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} 1/8 & 3/8 & 3/8 & 1/8 \end{bmatrix} \tag{3.13}$$

The Runge-Kutta formulas described by equations (3.10)-(3.13), are well-known and collect a wide variety of this kind of techniques. It is worth mentioning that higher order Runge-Kutta formulas are available in the literature, however, as they require more than for factorizations each iterations, presumably their computational burden would be totally not competitive. Therefore, they have not been included in this work, nevertheless, they may be easily validated using the common framework (3.8), (3.9). Nevertheless, one should not ignore other further developments made in this area. For example, in reference [57], an accelerated 3rd order Runge-Kutta method (ARK3) was developed. This technique can be adapted for PF calculation as follows:

At $k = 0$

$$\begin{cases} \boldsymbol{v}_1^{(0)} = \mathbf{f}\left(\boldsymbol{x}^{(0)}\right) \\ \boldsymbol{v}_2^{(0)} = \mathbf{f}\left(\boldsymbol{x}^{(0)} + h\beta\boldsymbol{v}_1^{(0)}\right) \\ \boldsymbol{x}^{(1)} = \boldsymbol{x}^{(0)} + h\boldsymbol{v}_1^{(0)} \end{cases} \tag{3.14}$$

For $k > 0$

$$\begin{cases} \boldsymbol{v}_1^{(k)} = \mathbf{f}\left(\boldsymbol{x}^{(k)}\right) \\ \boldsymbol{v}_2^{(k)} = \left(\boldsymbol{x}^{(k)} + h\beta\boldsymbol{v}_1^{(0)}\right) \\ \boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + h\left[a_1\boldsymbol{v}_1^{(k)} + \widetilde{a_1}\boldsymbol{v}_1^{(k-1)} + b\left(\boldsymbol{v}_2^{(k)} - \boldsymbol{v}_2^{(k-1)}\right)\right] \end{cases} \tag{3.15}$$

where, $\beta \in \mathbb{R}$, $b \in \mathbb{R}$, $a_1 \in \mathbb{R}$ and $\widetilde{a_1} \in \mathbb{R}$ are defined following the guidelines facilitated in [57] and reported in Table 3.1.

Table 3.1 - Parameters for ARK3

β	b	a_1	$\widetilde{a_1}$
$1/3$	$5/4$	$1/4$	$3/4$

It is worth noting that ARK3 achieves third order by just requiring two LU factorizations each iteration. As comparison, the other third order Runge-Kutta formula considered (3OH), requires three matrix factorizations.

3.2.2.- Step size control

The control of the step size is a critical aspect for solving ODEs. Fig. 3.2 shows a comparison for the Forward Euler methodology with different step sizes. Clearly, the smaller step size the higher accuracy. However, higher computational burden is obtained as counterpart since more intermediate points have to be calculated. From this analysis it is deduced that a proper adaptive step size mechanism is fundamental for obtaining a good performance from those PF solvers based on the Runge-Kutta formulas.

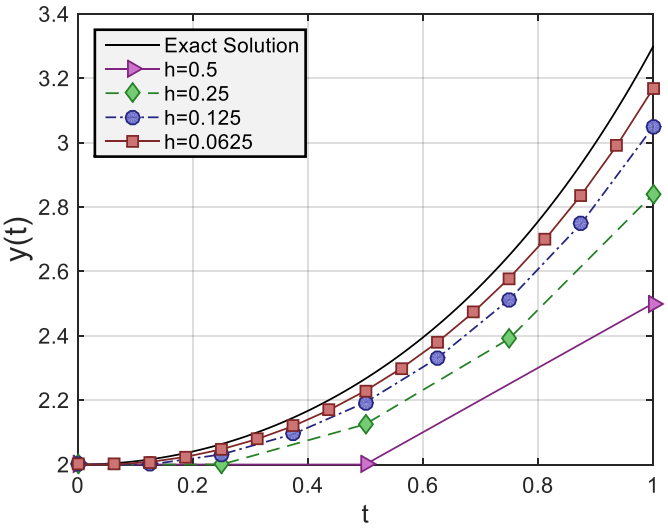


Fig 3.2 - Influence of the step size for solving the ODE $\dot{y} = yt$, with initial condition $\dot{y}(0) = 2$

For the considered PF solvers based on the Runge-Kutta techniques, we can use the half-step method for adapting h as follows [39]:

$$\mu = \left\| \boldsymbol{v}_{p/2}^{(k)} - \boldsymbol{x}^{(k+1)} \right\|_{\infty} \tag{3.16}$$

$$\begin{cases} \textbf{if } \mu > \text{SF } \textbf{then } h \leftarrow \max(\sigma_1 h, h_{\min}) \\ \textbf{if } \mu \leq \text{SF } \textbf{then } h \leftarrow \min(\sigma_2 h, h_{\max}) \end{cases} \tag{3.17}$$

where SF $\in \mathbb{R}$ is a security factor, σ's > 0 are damping factors while $h_{\min} \in \mathbb{R}^+$ and $h_{\max} \in \mathbb{R}^+$ are the minimum and maximum step size, respectively. Algorithms 3.1 and 3.2 summarize the main steps involved in the PF calculation using the considered PF solvers based on the Runge-Kutta formulas. As observed, the step size is initialized equal to $h_{\min}$.

This approach aims at improving the robustness features of the proposed algorithm, when it evolves far away to the solution. Under this situation, poorer performance is expected due to the local convergence of the Newton's technique.

3.3.- Adams Bashforth's methods

The Adams-Bashforth's methods form a family of explicit numerical integration techniques [54]. They belong to the family of linear multistep numerical techniques, since they need the information of several previous points to compute the following step. The generic formulation of a p^{th} order Adams-Bashforth method as applied to the PF problem is given by:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + h\left[\alpha_1 \mathbf{f}\left(\boldsymbol{x}^{(k)}\right) + \alpha_2 \mathbf{f}\left(\boldsymbol{x}^{(k-1)}\right) + \cdots + \alpha_s \mathbf{f}\left(\boldsymbol{x}^{(k-p-1)}\right)\right] \tag{3.18}$$

Algorithm 3.1: PF solution using solvers based on the Runge-Kutta methods

1: Let $\boldsymbol{x}^{(0)}$, ε, $k_{\max}$, SF, σ_1, σ_2, $h_{\min}$ and $h_{\max}$ be given
2: Initialize the iteration counter $k \leftarrow 0$
3: Initialize the step size $h \leftarrow h_{\min}$
4: **while** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k)}\right)\right\|_{\infty} > \varepsilon$ **do**
5: Solve (3.8)
6: **if** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k+1)}\right)\right\|_{\infty} < \varepsilon$ **then**
7: **break** *# convergence*
8: **else**
9: $\mu \leftarrow$ solve (3.16)
10: $h \leftarrow$ solve (3.17)
11: $k \leftarrow k + 1$
12: **if** $k > k_{\max}$ **then**
13: **break** *# failure*
14: **end if**
15: **end if**
16: **end do**
17: **return** solution $\boldsymbol{x}^{(k)}$

Algorithm 3.2: PF solution using ARK3

1: Let $\boldsymbol{x}^{(0)}$, ε, $k_{\max}$, SF, σ_1, σ_2, $h_{\min}$ and $h_{\max}$ be given

2: Initialize the iteration counter $k \leftarrow 0$

3: Initialize the step size $h \leftarrow h_{\min}$

4: **while** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k)}\right)\right\|_{\infty} > \varepsilon$ **do**

5: **if** $k = 0$ **then**

6: Solve (3.14)

7: **else**

8: Solve (3.15)

9: **end if**

10: **if** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k+1)}\right)\right\|_{\infty} < \varepsilon$ **then**

11: **break** *# convergence*

12: **else**

13: $\mu \leftarrow$ solve (3.16)

14: $h \leftarrow$ solve (3.17)

15: $k \leftarrow k + 1$

16: **if** $k > k_{\max}$ **then**

17: **break** *# failure*

18: **end if**

19: **end if**

20: **end do**

21: **return** solution $\boldsymbol{x}^{(k)}$

where α's $\in \mathbb{R}$. It is worth noting that equation (3.18) requires information of $p-1$ previous points. Therefore, this kind of techniques requires to be initialized. It is customary to use a $(p-1)^{th}$ order Adams-Bahsforth technique for this labour. As example, let the 2nd order Adams-Bashforth (AB2) procedure for solving the PF problem be described as follows:

At $k = 0$

$$\boldsymbol{x}^{(1)} = \boldsymbol{x}^{(0)} + h\mathbf{f}\left(\boldsymbol{x}^{(0)}\right) \tag{3.19}$$

For $k > 0$

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + h\left[{}^{-1}/_{2}\,\mathbf{f}\left(\boldsymbol{x}^{(k-1)}\right) + {}^{3}/_{2}\,\mathbf{f}\left(\boldsymbol{x}^{(k)}\right)\right] \tag{3.20}$$

As observed, the developed PF solver based on AB2 uses a 1st order Adams-Bashforth formula (Explicit Euler) at first iteration. If one desires to use a 3rd order expression, the Explicit Euler and the equation (3.20) would be used at $k = 0$, and $k = 1$, respectively. Following this scheme, one can applied any Adams-Bashforth method to the PF problem in the same manner.

3.3.1.- Step size control

A novel methodology to update the step size of the Adams-Bashforth methods is used. It is based on the Richardson Extrapolation [58]. To achieve that, two guess values of the unknown vector $\widehat{\boldsymbol{x}}_1^{(k+1)}$ and $\widehat{\boldsymbol{x}}_2^{(k+1)}$, are computed each iteration using the explicit Euler method with $h/2$ and $h/4$ as follows:

$$\widehat{\boldsymbol{x}}_1^{(k+1)} = \boldsymbol{x}^{(k+1)} + \frac{h}{2}\mathbf{f}\left(\boldsymbol{x}^{(k+1)}\right) \tag{3.21}$$

$$\widehat{\boldsymbol{x}}_2^{(k+1)} = \boldsymbol{x}^{(k+1)} + \frac{h}{4}\mathbf{f}\left(\boldsymbol{x}^{(k+1)}\right) \tag{3.22}$$

Then, this information is used to calculate μ:

$$\mu = \left\| \frac{\widehat{\boldsymbol{x}}_1^{(k+1)} - \widehat{\boldsymbol{x}}_2^{(k+1)}}{h} \right\|_{\infty} \tag{3.23}$$

Then, this information is used for updating the step size using (3.17). In this case, the step size is initialized using the following rule:

$$\begin{cases} \textbf{if } \left\|\mathbf{g}(\boldsymbol{x}^{(0)})\right\|_{\infty} > \rho \textbf{ then } h \leftarrow h_{\min} \\ \textbf{if } \left\|\mathbf{g}(\boldsymbol{x}^{(0)})\right\|_{\infty} \leq \rho \textbf{ then } h \leftarrow 1 \end{cases} \tag{3.24}$$

where $\rho \in \mathbb{R}$. The full explanation of the Adams-Bashforth-based PF solvers, is concluded by describing the main steps of the PF solver based on AB2 raised by the mapping (3.19), (3.20) in Algorithm 3.3 using pseudocode. One should note that any other Adams-Bashforth formula may be applied for PF analysis using the same scheme.

3.4.- Bulirsch Stoer Algorithm

Now, a more sophisticated numerical integration method called Bulirsch-Stoer (BS) [59] (also called the Gragg-Bulirsch-Stoer algorithm [60]) is explored. BS is a powerful

method for solving set of ODEs. In fact, it is said that BS supposes the best-known way to obtain high accuracy solutions to ODEs [61]. BS collects the advantages of various numerical arrangements like the Explicit Euler method or the Modified Midpoint rule. This approach starts taking an arbitrary "big" step size $H \in \mathbb{R}^+$, which is splitted into $N \in \mathbb{N}$ equal smaller steps as follows:

$$h = {}^{H}/_{N} \tag{3.25}$$

Algorithm 3.3: PF solution using AB2

1: Let $\boldsymbol{x}^{(0)}$, ε, $k_{\max}$, SF, σ_1, σ_2, $h_{\min}$, $h_{\max}$ and ρ be given
2: Initialize the iteration counter $k \leftarrow 0$
3: Initialize the step size $h \leftarrow$ solve (3.24)
4: **while** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k)}\right)\right\|_{\infty} > \varepsilon$ **do**
5: **if** $k = 0$ **then**
6: Solve (3.19)
7: **else**
8: Solve (3.20)
9: **end if**
10: **if** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k+1)}\right)\right\|_{\infty} < \varepsilon$ **then**
11: **break** *# convergence*
12: **else**
13: $\mu \leftarrow$ solve (3.23)
14: $h \leftarrow$ solve (3.17)
15: $k \leftarrow k + 1$
16: **if** $k > k_{\max}$ **then**
17: **break** *# failure*
18: **end if**
19: **end if**
20: **end do**
21: **return** solution $\boldsymbol{x}^{(k)}$

The normal BS procedure starts with a small value of n and increases it if the calculated state value is not accuracy enough. Theoretically, the zero error is reached when

$N = \infty$. However, it is just an idealization and this value cannot be reached in practical applications. Instead, two series have been proposed in the literature [59], [61], [62]:

$$\boldsymbol{N} = 2, 4, 6, 8, 12, 16, 24, 32, 48, 64 \ldots \tag{3.26}$$

$$\boldsymbol{N} = 2, 4, 6, 8, 10, 12, 14 \ldots \tag{3.27}$$

Theses series are endless. Therefore, it is necessary to truncate them at a specified value. It is worth noting that the highest N implies the highest robustness. Nevertheless, the highest N also induces the heaviest computational effort. Fig 3.3 plots the sketch of BS for solving the ODE (3.1). As observed, more accuracy results are achieved by increasing N, however, more intermediate points need to be calculated.

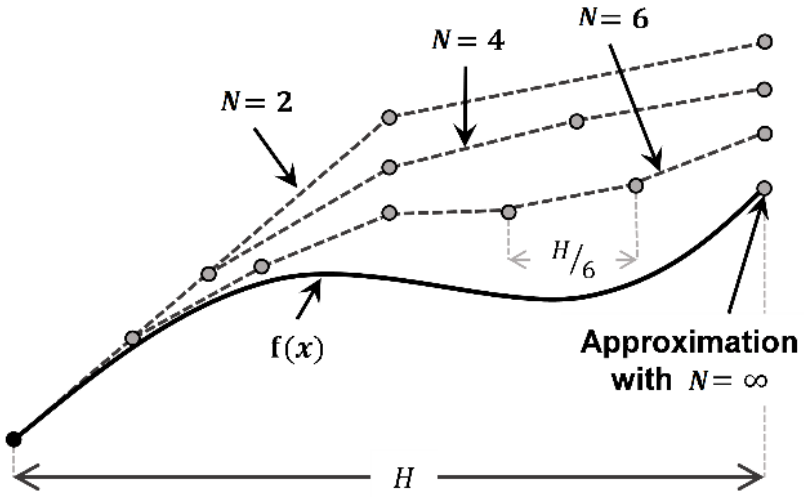


Fig 3.3 - Sketch of BS

BS can be adapted for PF calculation applying the principles described in [39]. To do that, it is suitable firstly define:

$$\boldsymbol{x}^{(k)} = \boldsymbol{z}_0^{(k)} \tag{3.28}$$

The following step consists on calculating $\boldsymbol{z}_1$ using the Forward Euler method as follows:

$$\boldsymbol{z}_1^{(k)} = \boldsymbol{z}_0^{(k)} - h\mathbf{f}\left(\boldsymbol{z}_0^{(k)}\right) \tag{3.29}$$

Next, up to p intermediate points are calculated using the Modified Midpoint Rule as follows:

$$\boldsymbol{z}_{j+1}^{(k)} = \boldsymbol{z}_j^{(k)} - 2h\mathbf{f}\left(\boldsymbol{z}_j^{(k)}\right), \text{ for } j = 2,3, \ldots, N \tag{3.30}$$

Finally, the state vector is updated using the trick proposed by Gragg:

$$\boldsymbol{x}^{(k+1)} = {}^{1}/_{2}\left[\boldsymbol{z}_N^{(k)} + \boldsymbol{z}_N^{(k)} + h\mathbf{f}\left(\boldsymbol{z}_N^{(k)}\right)\right] \tag{3.31}$$

3.4.1.- The Reverse-Bulirsch-Stoer algorithm

BS was originally conceived for set of ODEs, hence, in this case it is suitable to start with the smallest possible value of N and increase it until the desirable accuracy is reached. Thus, expensive steps are not computed if they are not necessary. However, in PF analysis, this approach is not desirable. Instead, it would be convenient starting the iterative procedure with the maximum value of $\boldsymbol{N}$ (i.e. $N_{\max}$), then, it may be reduced until its minimum value of $\boldsymbol{N}$ (i.e. $N_{\min}$). The main idea behind this approach is founded on the traditional behavior of a PF solver: when an iterative procedure is susceptible to diverge, the most difficult part is the computation of the first steps. Hence, the main idea would be using the most exact steps (i.e. the biggest N) at first iterations and then would be convenient to change to bigger steps in order to accelerate the convergence. This philosophy, applied to BS, has blossomed in the proposed Reverse-Bulirsch-Stoer method (RBS). For the sake of exemplary, Fig. 3.4 shows the procedure of the BS and the RBS, for a truncation at $N = 6$, are compared. As observed, while BS increases the value of N each iteration, RBS proposes the opposite scheme.

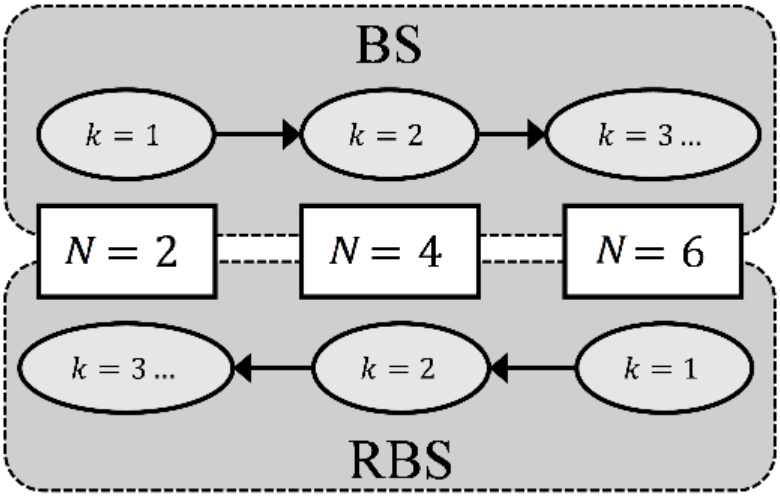


Fig. 3.4 - A comparison between BS and RBS.

The main idea behind RBS lies on reducing the total amount of factorizations required. For example, let us consider the case of BS, we start with $N = 2$ and it would be increased up to the biggest N. Then, iterations are computed repeatedly with $N = 6$ until

the convergence is reached. Therefore, BS would require to compute up to $6 + 6(K - 2)$ factorizations (where K is the total number of iterations). On the other hand, let us consider the same case for RBS, starting with $N = 6$ and reduce it until $N = 2$. Then, remainder iterations would be computed repeatedly with $N = 2$ instead of $N = 6$. Thus, RBS would need to compute $10 + 2(K - 2)$ factorizations. Considering a reasonable $K = 10$, BS would compute 54 LU decompositions, while RBS would require 28 less factorizations.

3.4.2.- Step size control

For controlling the "big" step size, the following updating rule is proposed:

$$\mu = \left\| \boldsymbol{z}_N^{(k)} - \boldsymbol{z}_{N-1}^{(k)} \right\|_\infty \tag{3.32}$$

$$\begin{cases} \textbf{if } \mu > \text{SF } \textbf{then } H \leftarrow \max(\sigma_1 H, H_{\min}) \\ \textbf{if } \mu \leq \text{SF } \textbf{then } H \leftarrow \min(\sigma_2 H, H_{\max}) \end{cases} \tag{3.33}$$

where $H_{\min} \in \mathbb{R}^+$ and $H_{\max} \in \mathbb{R}^+$ are the minimum and maximum "big" step size. The main steps of BS and RBS as applied to PF problem are summarized in Algorithm 3.4 using pseudocode.

3.5.- The Continuous Newton-Broyden paradigm

The main disadvantage of the Continuous Newton's methods is its high computational cost. As commented, when for example a Runge-Kutta formula is adapted as PF solver, it requires as many LU factorizations as stages of the numerical approach considered. With the aim at alleviating this computational effort, the Broyden's method is used in combination with the Continuous Newton's framework. To do that, the function (3.4) is modified as follows:

$$\tilde{\mathbf{f}}_s^{(k)}(\boldsymbol{x}) = -\boldsymbol{D}_s^{(k)} \mathbf{g}(\boldsymbol{x}) \tag{3.34}$$

The matrix $\boldsymbol{D} \in \mathbb{R}^{n \times n}$ varies with the iteration k and the stage s of the Runge-Kutta method considered. Thus, this matrix is given by:

Algorithm 3.4: PF solution using BS and RBS

1: Let $\boldsymbol{x}^{(0)}$, ε, $k_{\max}$, SF, σ_1, σ_2, $H_{\min}$, $H_{\max}$, $\boldsymbol{N}$, $N_{\min}$ and N_{max} be given

2: Initialize the iteration counter $k \leftarrow 0$

3: Initialize the step size H

3: Initialize $N = N_{\min}$ *# In the case of RBS take* $N = N_{\max}$

4: **while** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k)}\right)\right\|_{\infty} > \varepsilon$ **do**

5: $h \leftarrow$ solve (3.25)

6: Solve (3.29)

7: $j = 2$

8: **for** $j \leq N$ **do**

9: Solve (3.30)

10: **end do**

11: $\boldsymbol{x}^{(k+1)} \leftarrow$ **s**olve (3.31)

12: **if** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k+1)}\right)\right\|_{\infty} < \varepsilon$ **then**

13: **break** *# convergence*

14: **else**

15: $\mu \leftarrow$ solve (3.32)

16: $H \leftarrow$ solve (3.33)

17: $N \leftarrow$ take the immediately higher/lower value of $\boldsymbol{N}$

18: **if** $N > N_{\max}$ **do** *# In the case of RBS it would be* $N < N_{\min}$

19: $N = N_{\max}$ *# In the case of RBS it would be* $N = N_{\min}$

20: **end do**

21: $k \leftarrow k + 1$

22: **if** $k > k_{\max}$ **then**

23: **break** *# failure*

24: **end if**

25: **end if**

26: **end do**

27: **return** solution $\boldsymbol{x}^{(k)}$

$$\boldsymbol{D}_1^{(0)} = \left[\mathbf{g}\left(\boldsymbol{x}^{(0)}\right)\right]^{-1} \tag{3.35}$$

$$\boldsymbol{D}_1^{(k)} = \boldsymbol{D}_p^{(k-1)} + \frac{\left(\boldsymbol{s}_1^{(k)} - \boldsymbol{D}_p^{(k-1)}\boldsymbol{y}_1^{(k)}\right)\boldsymbol{s}_1^{(k)^T}\boldsymbol{D}_p^{(k-1)}}{\boldsymbol{s}_1^{(k)^T}\boldsymbol{D}_p^{(k-1)}\boldsymbol{y}_1^{(k)}}, \text{ for } k > 0 \tag{3.36}$$

$$\boldsymbol{D}_s^{(k)} = \boldsymbol{D}_{s-1}^{(k)} + \frac{\left(\boldsymbol{s}_s^{(k)} - \boldsymbol{D}_{s-1}^{(k)}\boldsymbol{y}_s^{(k)}\right)\boldsymbol{s}_s^{(k)^T}\boldsymbol{D}_{s-1}^{(k)}}{\boldsymbol{s}_s^{(k)^T}\boldsymbol{D}_{s-1}^{(k)}\boldsymbol{y}_s^{(k)}}, \text{ for } s \neq 1 \tag{3.37}$$

where:

$$\boldsymbol{s}_1^{(k)} = \boldsymbol{x}_1^{(k)} - \boldsymbol{x}_p^{(k-1)} \tag{3.38}$$

$$\boldsymbol{y}_1^{(k)} = \mathbf{g}\left(\boldsymbol{x}_1^{(k)}\right) - \mathbf{g}\left(\boldsymbol{x}_p^{(k-1)}\right) \tag{3.39}$$

$$\boldsymbol{s}_s^{(k)} = \boldsymbol{x}_s^{(k)} - \boldsymbol{x}_{s-1}^{(k)}, \text{ for } s \neq 1 \tag{3.40}$$

$$\boldsymbol{y}_s^{(k)} = \mathbf{g}\left(\boldsymbol{x}_s^{(k)}\right) - \mathbf{g}\left(\boldsymbol{x}_{s-1}^{(k)}\right), \text{ for } s \neq 1 \tag{3.41}$$

The different $\boldsymbol{x}_s$'s depend on the Runge-Kutta formula used, in a generic form, they respond to the following expression.

$$\begin{cases} \widetilde{\boldsymbol{v}}_1^{(k)} = \tilde{\mathbf{f}}\left(\boldsymbol{x}^{(k)}\right) = \tilde{\mathbf{f}}\left(\boldsymbol{x}_1^{(k)}\right) \\ \widetilde{\boldsymbol{v}}_2^{(k)} = \tilde{\mathbf{f}}\left(\boldsymbol{x}^{(k)} + ha_{21}\widetilde{\boldsymbol{v}}_1^{(k)}\right) = \tilde{\mathbf{f}}\left(\boldsymbol{x}_2^{(k)}\right) \\ \widetilde{\boldsymbol{v}}_3^{(k)} = \tilde{\mathbf{f}}\left(\boldsymbol{x}^{(k)} + h\left(a_{31}\widetilde{\boldsymbol{v}}_1^{(k)} + a_{32}\widetilde{\boldsymbol{v}}_2^{(k)}\right)\right) = \tilde{\mathbf{f}}\left(\boldsymbol{x}_3^{(k)}\right) \\ \vdots \\ \widetilde{\boldsymbol{v}}_p^{(k)} = \tilde{\mathbf{f}}\left(\boldsymbol{x}^{(k)} + h\left(a_{p,1}\widetilde{\boldsymbol{v}}_1^{(k)} + a_{p,2}\widetilde{\boldsymbol{v}}_2^{(k)} \ldots + a_{p,p}\widetilde{\boldsymbol{v}}_{p-1}^{(k)}\right)\right) = \tilde{\mathbf{f}}\left(\boldsymbol{x}_p^{(k)}\right) \end{cases} \tag{3.42}$$

As observed, the proposed solution paradigm (3.42) just requires a matrix inversion at $s = 1$ and $k = 0$. However, this calculation has to be computed explicitly, which supposed a heavier computation than a LU decomposition. Definitely, equation (3.42) is normally more competitive than (3.9) in small-scale systems, while it normally presents a prohibitive computational burden in large-scale networks since the computations involved in (3.35)-(3.37) suppose an important computational burden.

Finally, the state vector is updated similarly to (3.8):

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + h\sum_{i=1}^{p} b_i\widetilde{\boldsymbol{v}}_i^{(k)} \tag{3.43}$$

3.5.1.- Step size control

For the step size control of the proposed paradigm (3.42), we can use the half-step approach as for (3.9). In this case, the expression (3.16) is accordingly modified:

$$\mu = \left\| \widetilde{\boldsymbol{v}}_{p/2}^{(k)} - \boldsymbol{x}^{(k+1)} \right\|_{\infty} \tag{3.44}$$

The main steps of the proposed paradigm (3.42) are summarized in Algorithm 3.5 using pseudocode.

Algorithm 3.5: PF solution using the proposed Continuous Newton-Broyden paradigm

Let $\boldsymbol{x}^{(0)}$, ε, $k_{\max}$, SF, σ_1, σ_2, $h_{\min}$ and $h_{\max}$ be given
Initialize the iteration counter $k \leftarrow 0$
Initialize the step size $h \leftarrow h_{\min}$
while $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k)}\right)\right\|_{\infty} > \varepsilon$ **do**
    Solve (3.43)
    if $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k+1)}\right)\right\|_{\infty} < \varepsilon$ **then**
        break *# convergence*
    else
        $\mu \leftarrow$ solve (3.44)
        $h \leftarrow$ solve (3.17)
        $k \leftarrow k + 1$
        if $k > k_{\max}$ **then**
            break *# failure*
        end if
    end if
end do
return solution $\boldsymbol{x}^{(k)}$

3.6.- A Framework based on a Multistep Continuous Newton's paradigm

Now, a modification of the standard Continuous Newton's method is proposed. It is mainly based on modifying the equation (3.8) into a multistep paradigm as follows:

$$\begin{cases} \hat{\boldsymbol{x}}_1^{(k)} = \boldsymbol{x}^{(k)} + h\sum_{i=1}^{p} b_{1i}\boldsymbol{v}_i^{(k)} \\ \hat{\boldsymbol{x}}_2^{(k)} = \hat{\boldsymbol{x}}_1^{(k)} + h\sum_{i=1}^{p} b_{2i}\boldsymbol{v}_i^{(k)} \\ \vdots \\ \boldsymbol{x}^{(k+1)} = \hat{\boldsymbol{x}}_{K-1}^{(k)} + h\sum_{i=1}^{p} b_{Ki}\boldsymbol{v}_i^{(k)} \end{cases} \tag{3.45}$$

where $\boldsymbol{v}$'s are calculated as in (3.9). As observed, the main difference of (3.45) compared with (3.8) lies in the fact that the updated vector is recursively refined in order to obtain a more accurate value. Thus, up to K steps are computed. In addition, the weights b's are now organized in a matrix form as follows:

$$\boldsymbol{B} = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & & \ddots & \vdots \\ b_{K1} & b_{K2} & \cdots & b_{Kp} \end{bmatrix} \in \mathbb{R}^{K\times p} \tag{3.46}$$

This matrix arrangement allows to concatenate different Runge-Kutta formulas. This introduced solution paradigm can be conceived as the application of the Embedded Runge-Kutta formulas [63] for solving the PF. However, the introduced solution paradigm shows several important differences with respect the standard Embedded Runge-Kutta methods:

- The introduced solution paradigm is more versatile since it allows to use K numerical methods with the same order or any other combination.
- In calculation of ODEs, typically higher order implies higher accuracy as showed in Fig. 3.5. However, it is important noting that higher order supposes higher computational burden, since as many factorizations as the order of the Runge-Kutta method considered have to be computed. In this case, rather than the order, superior computational performance is achieved for higher K.

3.6.1.- Step size control

As commented, methods based on the introduced solution paradigm and the Embedded Runge-Kutta formulas have similar structure. These techniques allow an effective step size control based on the estimation of the local truncation error. Thus, we can take advantage of this characteristic within the introduced solution framework. Specifically, in reference [64], the step size is updated each iteration according to the following rule:

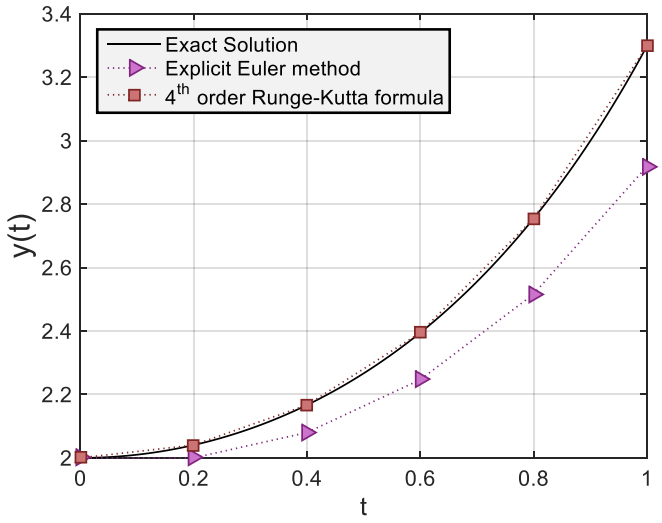


Fig 3.5 - Accuracy of the Explicit Euler and 4th order Runge Kutta formulas for solving the ODE $\dot{y} = yt$, *with initial condition* $\dot{y}(0) = 2$ *and* $h = 0{,}2$

$$h = \min\left(h_{\max}, \max\left(h_{\min}, 0.9h\left(\frac{1}{\epsilon^{(k)}}\right)^{p+1}\right)\right) \tag{3.47}$$

where ϵ is the local truncation error which can be calculated as follows:

$$\epsilon^{(k)} = \left\| \boldsymbol{x}^{(k+1)} - \hat{\boldsymbol{x}}_{K-1}^{(k)} \right\|_{\infty} \tag{3.48}$$

Nevertheless, the step size has to be initialized. To do that, the following expression is used:

$$h = \min\left(h_{\max}, \max\left(h_{\min}, \frac{1}{W\left(\boldsymbol{x}^{(0)}\right)^{0.06}}\right)\right) \tag{3.49}$$

where W is the sum of square of residuals given by:

$$W\left(\boldsymbol{x}^{(k)}\right) = \frac{1}{2}\left[\mathbf{g}\left(\boldsymbol{x}^{(k)}\right)\right]^{T}\mathbf{g}\left(\boldsymbol{x}^{(k)}\right) \tag{3.50}$$

3.6.2.- Two Power Flow solvers based on the developed multistep Continuous Newton's paradigm

On the basis of the developed solution paradigm, two PF solution techniques have been developed. Their matrix ***A*** and ***B*** are given below:

Midpoint-Euler (MPE)

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 \\ 1/2 & 0 \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} 0 & 1 \\ 1/2 & 0 \end{bmatrix} \tag{3.51}$$

Trapezoidal-Midpoint-Euler (TMPE)

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} 1/2 & 1/2 \\ 0 & 1/2 \\ 1/2 & 0 \end{bmatrix} \tag{3.52}$$

The main steps of the developed MPE and TMPE as applied for PF analysis are summarized in Algorithm 3.6 using pseudocode.

Algorithm 3.6: PF solution using the proposed Multistep Continuous Newton paradigm

Let $\boldsymbol{x}^{(0)}$, ε, $k_{\max}$, $h_{\min}$ and $h_{\max}$ be given
Initialize the iteration counter $k \leftarrow 0$
Initialize the step size $h \leftarrow$ solve (3.49)
while $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k)}\right)\right\|_{\infty} > \varepsilon$ **do**
    Solve (3.9) *# using (3.51) or (3.52)*
    $\boldsymbol{x}^{(k+1)} \leftarrow$ solve (3.45) *# using (3.51) or (3.52)*
    if $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k+1)}\right)\right\|_{\infty} < \varepsilon$ **then**
        break *# convergence*
    else
        $h \leftarrow$ solve (3.47)
        $k \leftarrow k + 1$
        if $k > k_{\max}$ **then**
            break *# failure*
        end if
    end if
end do
return solution $\boldsymbol{x}^{(k)}$

3.7.- Conclusions

The Continuous Newton's method as applied for PF analysis was firstly studied by Milano in [39]. However, this reference is limited to study the 4th order Runge-Kutta

formula. In this Section, a common framework to incorporate any other Runge-Kutta formula for PF analysis has been described. This proposal has been used for developing a variety of novel PF solvers based on this kind of integration techniques.

On the other hand, other families of numerical integration schemes have been considered. Thus, the family of Adams-Bahsforth methods along the Bulirsch-Stoer algorithm have been also considered for developing various PF solvers.

Those techniques based on the Continuous Newton's method may present a high computational burden, due to the numerous LU factorizations required. With the aim at overcoming this issue, a combined Continuous Newton-Broyden paradigm has been introduced. It replaces a LU factorization by vectors and matrix computations. However, the Jacobian matrix has to be explicitly inverted at first iteration, which may suppose an unaffordable effort especially in large-scale cases.

Finally, a Multistep Continuous Newton's paradigm has been developed. It aims at improving the convergence properties of the standard Continuous Newton's framework by refining the updated state vector within a multistep paradigm.

Chapter 4

A Power Flow solver based on Gauss-Newton method

Regularization methods have been extensively studied for ill-posed nonlinear problems [65]. For PF analysis, several solution techniques based on regularization methods have been proposed [45]-[47]. This kind of techniques bring an effective mechanism for avoiding the singularity of the Jacobian matrix at turning points. This is achieved by adding some elements to the Jacobian matrix so that its invertibility is ensured. A so-called regularization matrix is typically used for this purpose. In this section, the application of a regularization technique based on Gauss-Newton formulation for PF analysis is explored.

4.1.- Gauss-Newton method for Power Flow analysis

As commented, the solution of the PF equations is typically found using iterative solvers. This procedure is properly equivalent to solve the following nonlinear gradient equation:

$$\nabla_{\boldsymbol{x}} \boldsymbol{W}(\boldsymbol{x}) \coloneqq [\mathbf{g}'(\boldsymbol{x})]^{*}\mathbf{g}'(\boldsymbol{x}) = 0 \tag{4.1}$$

where $[\cdot]^*: \mathbb{R}^{m\times n} \to \mathbb{R}^{n\times m}$ is the adjoint operator. As observed, the Newton's method can be used for solving (4.1), which leads to the following mapping:

$$\nabla^2_{xx}\boldsymbol{W}(\boldsymbol{x}) \coloneqq [\mathbf{g}'(\boldsymbol{x})]^*\mathbf{g}'(\boldsymbol{x}) + R \tag{4.2}$$

where $\nabla^2_{xx}\boldsymbol{W}$ is the Hessian matrix of (4.1) and R the second order term. In this case, the Newton's technique may be not tractable due to the calculation of the Hessian matrix. Alternatively, the gradient equation (4.1) may be solved using the Gauss-Newton's method as follows:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \left[\boldsymbol{A}^{(k)}\right]^{-1}\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right)\mathbf{g}\left(\boldsymbol{x}^{(k)}\right) \tag{4.3}$$

The main merit of this alternative approach lies on the fact that the matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ can be calculated in some efficient way. Different alternatives exist. For example, the matrix $\boldsymbol{A}$ may be Moore-Penrose pseudoinverse of the Hessian matrix [66]. On the other hand, this matrix can be equal to $\left[\mathbf{g}'\left(\boldsymbol{x}^{(0)}\right)\right]^*\mathbf{g}'\left(\boldsymbol{x}^{(0)}\right)$ [67] or $\left[\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right)\right]^*\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right)$. In this case, this latter option is preferred due to the following reasons:

- The main advantage of using $\boldsymbol{A}^{(k)} = \left[\mathbf{g}'\left(\boldsymbol{x}^{(0)}\right)\right]^*\mathbf{g}'\left(\boldsymbol{x}^{(0)}\right)$ is avoiding the factorization of Jacobian matrix each iteration. However, if a regularization scheme is introduced, this advantage will be lost.
- The use of $\boldsymbol{A}^{(k)} = \left[\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right)\right]^*\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right)$ always provides a better a better approximation for $\boldsymbol{x}^{(k)}$ since the most updating Jacobian matrix is used. Thus, the convergence is normally speed up.

Therefore, replacing $\boldsymbol{A}$ in (4.3) by the option preferred one obtains:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \left[\left[\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right)\right]^*\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right)\right]^{-1}\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right)\mathbf{g}\left(\boldsymbol{x}^{(k)}\right) \tag{4.4}$$

If the operator $\left[\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right)\right]^*\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right)$ is no boundedly invertible, it may be necessary to introduce a regularization mechanism. It basically consists on adding some elements on the diagonal of the introduced operator, in order to avoid the singularity of the Jacobian matrix at turning points. This is achieved by using a so-called regularization matrix. Multiple approaches have been proposed in the literature [68], however the identity matrix is normally preferred due to it contributes to retain the sparsity pattern of the system [69]. In addition, a sequence of positive numbers $\{\alpha\}_{k=0}^{\infty} \to 0$ is also introduced in order to control the influence of the regularization matrix. This latter point is crucial since the accuracy of

the results obtained depends on the progression of this sequence [70]. After introducing the regularization mechanism, equation (4.4) becomes to:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \left[\left[\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right)\right]^{*}\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right) + \alpha \boldsymbol{I}\right]^{-1}\left(\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right)\mathbf{g}\left(\boldsymbol{x}^{(k)}\right) + \alpha\left(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(0)}\right)\right) \quad (4.5)$$

The damping factor α can be updated using the following formula:

$$\alpha = \alpha e^{-2k} \quad (4.6)$$

where *e* represents the natural exponential. The main steps of the developed PF solver based on Gauss Newton formulation are summarized in Algorithm 4.1 using pseudocode.

Algorithm 4.1: PF solution using Gauss-Newton

1: Let $\boldsymbol{x}^{(0)}$, ε, $k_{\max}$, and α be given
2: Initialize the iteration counter $k \leftarrow 0$
3: **while** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k)}\right)\right\|_{\infty} > \varepsilon$ **do**
4: Solve (4.5)
5: **if** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k+1)}\right)\right\|_{\infty} < \varepsilon$ **then**
6: **break** *# convergence*
7: **else**
8: $k \leftarrow k + 1$
9: $\alpha \leftarrow$ solve (4.6)
10: **if** $k > k_{\max}$ **then**
11: **break** *# failure*
12: **end if**
13: **end if**
14: **end do**
15: **return** solution $\boldsymbol{x}^{(k)}$

4.2.- Conclusions

In this chapter, a PF solver based on Gauss-Newton formulation has been presented. It aims at avoiding the singularity of the Jacobian matrix using a regularization scheme while the resulting iterative mapping is efficient yet.

Chapter 5

Robust and Efficient Power Flow solvers

As explained in Chapter 1, most of available robust PF solvers are, in fact, very inefficient to be used in realistic large-scale systems. This reason has limited the widespread application of this kind of solvers in industry applications. This chapter is devoted on presenting various efficient and reasonably robust PF solution methods. They aim at being comparable with NR regarding computational efficient, while outperform this conventional method in the resolution of ill-conditioned cases. The presented solvers have been developed on the basis of some available numerical methodologies like the Richardson extrapolation method, the homotopy principle or the Newton-Cotes formulas, among others.

5.1.- A robust and efficient Power Flow Solution paradigm based on a Combined approach

This Section presents a PF solution paradigm based on combining three powerful numerical methods. On the one hand, it uses the Newton's method and a numerical integration technique (e.g. the Runge-Kutta formulas) in an original combination. On the

other hand, these two numerical arrangements are framed using a homotopic-based approach.

5.1.1.- Basic principles of Homotopy

Homotopy-based techniques are global convergence continuation-like techniques widespread used for solving nonlinear equations [71]. Let us assume that **f** is, in this case, difficult to solve. However, it is available other version of the PF equations $\bar{\mathbf{g}}$ which is easier to be solved. Then, a homotopy-based method solves **f** by building a chain of "intermediate" points from $\bar{\mathbf{g}}$ to **f**. This chain is constructed by introducing an external parameter $\xi \in [0,1]$. This parameter is linked with the original set of nonlinear equations by a so-called homotopy function $\boldsymbol{h}(\boldsymbol{x},\xi)\colon \mathbb{R}^{n+1} \rightarrow \mathbb{R}^{n+1}$. The main idea of the homotopy principle is to construct this homotopy function so that $\boldsymbol{h}(\boldsymbol{x},0) = \bar{\mathbf{g}}$ and $\boldsymbol{h}(\boldsymbol{x},1) = \mathbf{f}$. As observed, the original set of equations is solved for $\xi = 1$. The basic principle of homotopy is sketched in Fig. 5.1.

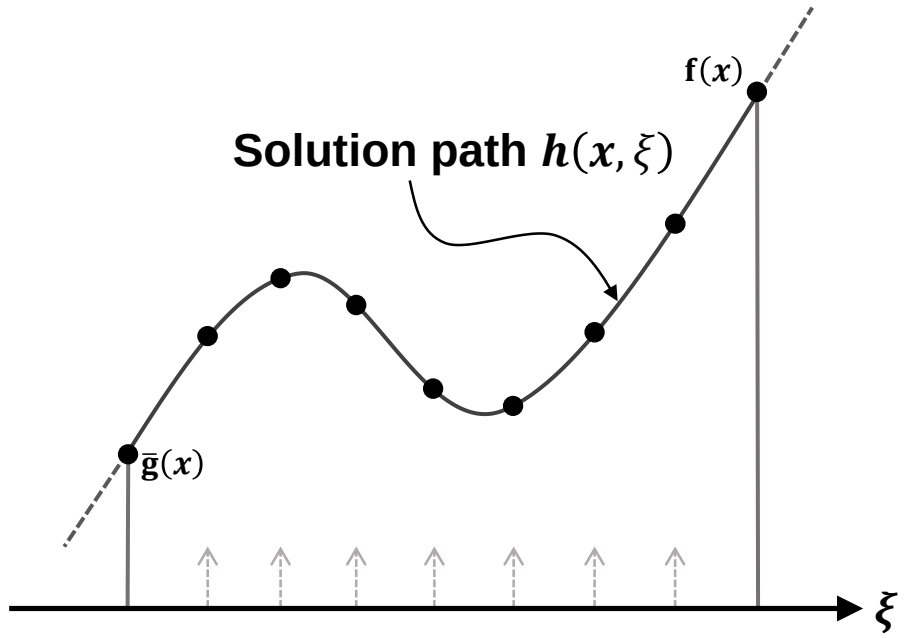


Fig. 5.1 - Sketch of the homotopy principle

5.1.2.- Solution of the Power Flow equations using the homotopy principle

The very first ingredient for solving the PF equations using the homotopy principle is building a homotopy function. This function has to meet two requirements. On the one hand, at $\xi = 0$ the homotopy function has to correspond with the "easy" system, i.e.

$\boldsymbol{h}(\boldsymbol{x}, 0) = \bar{\mathbf{g}}$. On the other hand, the homotopy function has to be equal to the objective system at $\xi = 1$, i.e. $\boldsymbol{h}(\boldsymbol{x}, 1) = \mathbf{f}$. Taking into account these points, the following homotopy function is used in this case [72]:

$$\boldsymbol{h}(\boldsymbol{x}, \xi) = \xi\mathbf{f} + (1 - \xi)\bar{\mathbf{g}} \tag{5.1}$$

In a second term, we need to define the "easy" system. In this case we have used the fixed point function [72], which may be given as follows:

$$\bar{\mathbf{g}} = \boldsymbol{x}_{\xi}^{(k)} - \boldsymbol{x}_{1}^{(k-1)} \tag{5.2}$$

As observed, in this case the PF state vector is updated within two loops. On the one hand the reader can observe the standard iteration loop defined by the counter k. On the other hand, a secondary loop is defined by the evolution of the parameter ξ. The main idea of the presented paradigm is to force the state vector to lie on the homotopy path defined by (5.1) through the defined secondary loop. To do that, the vector $\boldsymbol{x}$ is updated using the Forward-Euler technique as follows:

$$\boldsymbol{x}_{\xi+\Delta\xi}^{(k)} = \boldsymbol{x}_{\xi}^{(k)} + h \times \boldsymbol{h}\left(\boldsymbol{x}_{\xi}^{(k)}, \xi\right), \text{ for } \xi = 0 \text{ to } \xi = 1 \tag{5.3}$$

where $\Delta\xi \in \mathbb{R}^{+}$ is the increment of the parameter ξ. Instead of the Forward-Euler method, the Ralston formula could be used as follows:

$$\begin{cases} \boldsymbol{k}_1 = \boldsymbol{h}\left(\boldsymbol{x}_{\xi}^{(k)}, \xi\right) \\ \boldsymbol{y} = \boldsymbol{x}_{\xi}^{(k)} + \frac{2}{3}h\boldsymbol{k}_1 \\ \boldsymbol{k}_2 = \left(\lambda + \frac{2}{3}\Delta\xi\right)\boldsymbol{k}_1 + \left(1 - \left(\lambda + \frac{2}{3}\Delta\xi\right)\right)\left(\boldsymbol{y} - \boldsymbol{x}_{1}^{(k-1)}\right) \\ \boldsymbol{x}_{\xi+\Delta\xi}^{(k)} = \boldsymbol{x}_{\xi}^{(k)} + h\frac{\boldsymbol{k_1}+3\boldsymbol{k_2}}{4} \end{cases}, \text{ for } \xi = 0 \text{ to } \xi = 1 \tag{5.4}$$

It is worth mentioning that any other Runge-Kutta formula may be used in the same way. The main effect of introducing the homotopy function is showed in Fig. 5.2 for a generic 2-bus system, which has been taken from the system described in [73] (pp. 337-338) eliminating the bus #3. As observed, the PF state vector evolves according along the homotopy path.

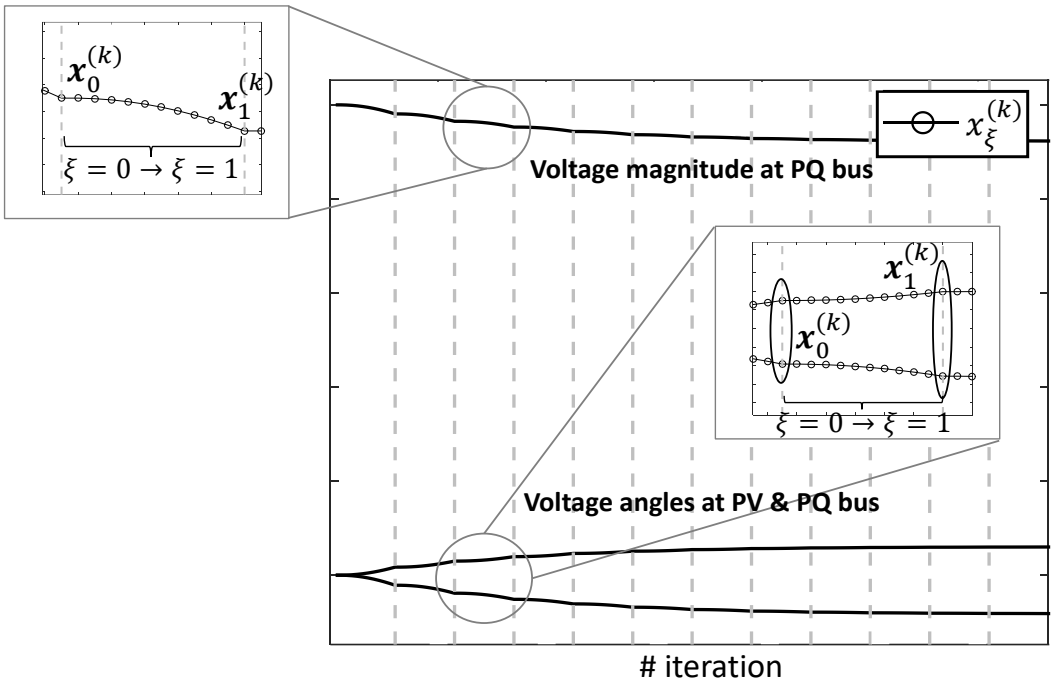


Fig. 5.2 - Solution paths and the progression of the PF state vector in 3-bus tutorial example [73], using the presented solution paradigm

For determining the step size, two guess values for $\boldsymbol{x}$ are considered in a similar way to (3.21) and (3.22) as follows:

$$\hat{\boldsymbol{x}}_{(1/2)} = \boldsymbol{x}_0^{(k)} + \frac{1}{2}\mathbf{f}\left(\boldsymbol{x}_0^{(k)}\right) \tag{5.5}$$

$$\hat{\boldsymbol{x}}_{(1/4)} = \boldsymbol{x}_0^{(k)} + \frac{1}{4}\mathbf{f}\left(\boldsymbol{x}_0^{(k)}\right) \tag{5.6}$$

Then, this information is used to calculate μ:

$$\mu = \left\| \frac{\hat{\boldsymbol{x}}_{(1/2)} - \hat{\boldsymbol{x}}_{(1/4)}}{h} \right\|_\infty \tag{5.7}$$

Finally, this information is used for updating the step size using (3.17). The main steps of the presented PF solution paradigm are summarized in the Algorithm 5.1 using pseudocode.

Algorithm 5.1: PF solution using a combined approach

1: Let $\boldsymbol{x}^{(0)}$, ε, $k_{\max}$, SF, σ_1, σ_2, $h_{\min}$, $h_{\max}$ and $\Delta\xi$ be given

2: Initialize the iteration counter $k \leftarrow 0$

3: Initialize the step size $h \leftarrow h_{\min}$

4: **while** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k)}\right)\right\|_{\infty} > \varepsilon$ **do**

5: $\quad \boldsymbol{x}_0^{(k)} \leftarrow \boldsymbol{x}^{(k)}$

6: $\quad \xi \leftarrow 0$

7: $\quad$ **for** $\xi \leq 1$ **do**

8: $\quad\quad$ Solve (5.3) *# or (5.4)*

9: $\quad\quad \xi = \xi + \Delta\xi$

10: $\quad$ **end do**

11: $\quad \boldsymbol{x}^{(k+1)} \leftarrow \boldsymbol{x}_1^{(k)}$

12: $\quad$ **if** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k+1)}\right)\right\|_{\infty} < \varepsilon$ **then**

13: $\quad\quad$ **break** *# convergence*

14: $\quad$ **else**

15: $\quad\quad \mu \leftarrow$ solve (5.7)

16: $\quad\quad h \leftarrow$ solve (3.17)

17: $\quad\quad k \leftarrow k + 1$

18: $\quad\quad$ **if** $k > k_{\max}$ **then**

19: $\quad\quad\quad$ **break** *# failure*

20: $\quad\quad$ **end if**

21: $\quad$ **end if**

22: **end do**

23: **return** solution $\boldsymbol{x}^{(k)}$

5.2.- A Robust Power Flow Solution technique based on Richardson Extrapolation

The Richardson Extrapolation is a very popular technique, widespread used for obtaining very high order integration by just using a series of values from trapezoidal rule [53]. Let us consider the problem of numerically integrating a scalar function $f: \mathbb{R} \rightarrow \mathbb{R}$.

This considered function can be properly approximated by an auxiliary function $\mathcal{F}: \mathbb{R} \to \mathbb{R}$, which depends on an arbitrary step size $h \in \mathbb{R}^+$. Then, the quadratic behaviour of the errors can be given as follows [74]:

$$\mathcal{F} = f + \phi_2 h^2 + \phi_4 h^4 + \cdots \tag{5.8}$$

where, the functions ϕ_n are normally unknown. If h is small enough, ϕ_4 and superior may be neglected. The main ingredient of the Richardson extrapolation is to combine two separate discrete solutions f_1 (coarse) and f_2 (fine), on two different step sizes h_1 and h_2 ($h_1 > h_2$). This approach allows to eliminate ϕ_2. Then, the value of f can be estimated as:

$$f \approx f_2 + \frac{f_2 - f_1}{r^{\psi} - 1} \tag{5.9}$$

where, r is the ratio $\frac{h_1}{h_2}$ and ψ is a real coefficient. Equation (5.9) corresponds with the h^2-*extrapolation*, it is worth mentioning that f may be more refined approximated by evaluating more than two points, however, it requires the solution of a system of simultaneous linear equations, which implies an arithmetic burden.

The main idea of the Richardson extrapolation can be easily translated to the PF problem. To do that, two guess values of the PF state vector are calculated using the Explicit-Euler method, then, the value of $\boldsymbol{x}$ can be properly refined using (5.9). This arises in the following mapping for solving the PF equations:

$$\begin{cases} \widehat{\boldsymbol{x}}_1^{(k)} = \boldsymbol{x}^{(k)} + h\mathbf{f}\left(\boldsymbol{x}^{(k)}\right) \\ \widehat{\boldsymbol{x}}_2^{(k)} = \boldsymbol{x}^{(k)} + \frac{h}{2}\mathbf{f}\left(\boldsymbol{x}^{(k)}\right) \\ \boldsymbol{x}^{(k+1)} = \frac{2^{\psi}\widehat{\boldsymbol{x}}_2^{(k)} - \widehat{\boldsymbol{x}}_1^{(k)}}{2^{\psi} - 1} \end{cases} \tag{5.10}$$

The information provided by $\widehat{\boldsymbol{x}}_1$ and $\widehat{\boldsymbol{x}}_2$ can be used for calculating the parameter μ as follows:

$$\mu = \left\| \widehat{\boldsymbol{x}}_1^{(k)} - \widehat{\boldsymbol{x}}_2^{(k)} \right\|_{\infty} \tag{5.11}$$

Thus, the value of (5.11) can be used for updating the step size using (3.17). The developed PF solver based on the Richardson extrapolation is summarized in Algorithm 5.2 using pseudocode.

Algorithm 5.2: PF solver based on Richardson extrapolation

1: Let $\boldsymbol{x}^{(0)}$, ε, $k_{\max}$, SF, σ_1, σ_2, $h_{\min}$, $h_{\max}$ and ψ be given

2: Initialize the iteration counter $k \leftarrow 0$

3: Initialize the step size $h = 1$

4: **while** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k)}\right)\right\|_{\infty} > \varepsilon$ **do**

8: Solve (5.10)

9: **if** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k+1)}\right)\right\|_{\infty} < \varepsilon$ **then**

10: **break** *# convergence*

11: **else**

12: $\mu \leftarrow$ solve (5.11)

13: $h \leftarrow$ solve (3.17)

14: $k \leftarrow k + 1$

15: **if** $k > k_{\max}$ **then**

16: **break** *# failure*

17: **end if**

18: **end if**

19: **end do**

20: **return** solution $\boldsymbol{x}^{(k)}$

5.3.- A three-stage Power Flow solver based on a Semi-Implicit approach

In this Section a novel PF solver based on a Semi-Implicit approach (SIA) [75] is presented, but combining the following three numerical methods in an original way:

- Lavrentiev's regularization [76], for enhancing the robustness characteristics at turning points.
- Chebyshev-like method with cubic convergence [77], for speeding up the convergence features.
- Heun's method [53], for obtaining a good balance between robustness and efficiency.

The resulting algorithm aims at gathering the features of these three numerical techniques, arising in a robust and efficient PF solver.

5.3.1.- Solving nonlinear equations using a Semi-Implicit approach

Firstly, let me briefly outline how SIA [75] is used for solving nonlinear problems. Let me begin with the simplest 1-dimensional case given by:

$$f(y) \equiv y - \varphi(y) = 0 \tag{5.12}$$

where, $y \in \mathbb{R}$ and $\varphi: \mathbb{R} \mapsto \mathbb{R}$ is a primitive iterative method as follows:

$$y^{(k+1)} = \varphi\left(y^{(k)}\right) \tag{5.13}$$

Eq. (5.13) can be modified to become a new equation with the same roots as:

$$y^{(k+1)} = \rho\left(y^{(k)} - \varphi\left(y^{(k)}\right)\right) + \varphi\left(y^{(k)}\right) \tag{5.14}$$

where, $\rho \in \mathbb{R}$ introduces a new degree of freedom introduced to optimize the convergence rate.

Now, the more general n-dimensional case is considered. Thus, equation (5.12) is extended to its multivariable counterpart as follows:

$$\boldsymbol{y}^{(k+1)} = \left(\boldsymbol{A}^{(k)} - \boldsymbol{I}\right)\boldsymbol{\psi}\left(\boldsymbol{y}^{(k)}\right) + \boldsymbol{y}^{(k)} \tag{5.15}$$

where $\boldsymbol{y} \in \mathbb{R}^n$, $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{\psi}: \mathbb{R}^n \longmapsto \mathbb{R}^n$ is given by:

$$\boldsymbol{\psi}(\boldsymbol{x}) = \boldsymbol{y} - \boldsymbol{\varphi}(\boldsymbol{y}) \tag{5.16}$$

where, $\boldsymbol{\varphi}: \mathbb{R}^n \longmapsto \mathbb{R}^n$ is an old iteration method, which is defined as follows:

$$\boldsymbol{\varphi}(\boldsymbol{y}) = \boldsymbol{y} - \mathbf{f}(\boldsymbol{y}) \tag{5.17}$$

Elements of matrix $\boldsymbol{A}$ are computed by differentiating (5.16):

$$\boldsymbol{A} = \boldsymbol{I} + (\boldsymbol{R} - \boldsymbol{I})[\mathbf{f}'(\boldsymbol{y})]^{-1} \tag{5.18}$$

where $\mathbf{f}' = \nabla_{\boldsymbol{y}}^{T}\mathbf{f}(\boldsymbol{y}) \in \mathbb{R}^{n\times n}$ is the Jacobian matrix formed by the first partial derivatives of $\mathbf{f}$ with respect $\boldsymbol{y}$ and:

$$\boldsymbol{R} = \begin{bmatrix} \rho_{11} & \cdots & \rho_{1n} \\ \vdots & \ddots & \\ \rho_{n1} & \cdots & \rho_{nn} \end{bmatrix} \in \mathbb{R}^{n\times n} \tag{5.19}$$

Elements of the matrix $\boldsymbol{R}$ are taken as free parameters which should be adapted each iteration. One should note that the matrix $\boldsymbol{A}$ is only defined if $\mathbf{f}'$ is invertible. Grouping (5.16)-(5.19), a generic k^{th} iteration of SIA for solving systems of nonlinear equations is given by the following map:

$$\boldsymbol{y}^{(k+1)} = \left(\boldsymbol{R}^{(k)} - \boldsymbol{I}\right)\left[\mathbf{f}'\left(\boldsymbol{y}^{(k)}\right)\right]^{-1}\boldsymbol{\psi}\left(\boldsymbol{y}^{(k)}\right) + \boldsymbol{y}^{(k)} \tag{5.20}$$

5.3.2.- A three-stage Power Flow solver based on a Semi-Implicit approach

Now, the process described in the preceding section is properly adapted for developing a novel PF solver. As commented, the developed algorithm is based on combining three powerful numerical methodologies. It is worth noting that the mapping (5.20) is only stable if the Jacobian matrix is continuously invertible. However, this point may not be true at the vicinity of a turning point. To overcome this drawback, a Regularization scheme may be used (see Chapter 4). Two basic regularization mechanisms are available in the literature. On the one hand, the Tikhonov's regularization [78] is quite stable and produces better condition numbers, on the other hand, the Lavrentiev's regularization scheme [76] is more efficient. In this work, the latter scheme has been preferred due to its efficiency.

Most of available robust PF solvers are very inefficient, at least, in comparison with NR. A reason of this low degree of efficiency is their linear convergence rate, which is reflected in a huge amount of iterations for achieving a solution. Alternatively, other nonlinear solvers with high convergence rate can be considered. This kind of techniques present a convergence order higher than two (NR has quadratic convergence order). However, this high-order methodology should be efficient. In that sense, those techniques that require more than a matrix factorization per iteration should be avoided. This motivates us to use the Chebyshev-like method with cubic convergence defined in [77].

High-order methodologies are typically weaker than NR [79]. In order to preserve an acceptable robustness but maintaining the resulting algorithm efficient enough, it is compulsory to establish some mechanism for balancing the efficient and robust properties

of the considered numerical arrangements. The Heun's method [53] present a particular structure which allows to easily achieve this target.

Result of combining the aforementioned techniques is a three stage algorithm. Thus, the Lavrentiev's regularization is introduced in a first step. On the other hand, the Chebyshev-like [77] and Heun's methods are respectively used at second and third stage. Finally, the described SIA has been found as an elegant framework for combining the considered numerical arrangements. The developed algorithm has been called 3S-SIA, and its conceptual idea is sketched in Fig. 5.3.

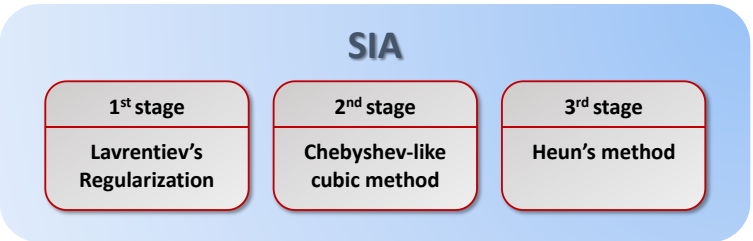


Fig. 5.3.- Conceptual idea of the developed solver.

Now, the formulation of the developed 3S-SIA is presented. Firstly, let me define the following increment vector based on the Lavrentiev's regularization scheme:

$$\boldsymbol{\phi}^{(k)} = \left[\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right) + \alpha \boldsymbol{I}\right]^{-1} \mathbf{h}\left(\boldsymbol{x}^{(k)}\right) \tag{5.21}$$

where $\mathbf{h}: \mathbb{R}^n \longmapsto \mathbb{R}^n$ is given by:

$$\mathbf{h}(\boldsymbol{x}) = \boldsymbol{x} - \boldsymbol{\mathcal{H}}(\boldsymbol{x}) \tag{5.22}$$

where $\boldsymbol{\mathcal{H}}: \mathbb{R}^n \longmapsto \mathbb{R}^n$ is defined as:

$$\boldsymbol{\mathcal{H}}(\boldsymbol{x}) = \boldsymbol{x} - \boldsymbol{\epsilon} \tag{5.23}$$

where $\boldsymbol{\epsilon} = \mathbf{g}(\boldsymbol{x})$. In this case, (5.23) is used for calculating an intermediate value of the PF state vector as follows:

$$\hat{\boldsymbol{x}}^{(k)} = \left(\boldsymbol{R}^{(k)} - \boldsymbol{I}\right)\boldsymbol{\phi}^{(k)} + \boldsymbol{x}^{(k)} \tag{5.24}$$

Now, in order to accelerate the convergence of the developed algorithm, the structure of the considered cubic Chebyshev-like method can be used for calculating a second increment vector as follows:

$$\hat{\boldsymbol{\phi}}^{(k)} = \left[\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right) + \alpha^{(k)} \boldsymbol{I}\right]^{-1} \mathbf{g}\left(\hat{\boldsymbol{x}}^{(k)}\right) \tag{5.25}$$

It is worth noting that (5.21) represents a robust evolution of $\boldsymbol{x}$ in the sense of it uses the Lavrentiev's regularization. On the other hand, (5.25) aims at speeding up the convergence rate in the sense of it shares structure with the cubic methodology [77]. Higher convergence rate is in this case provided by evaluating the PF state vector in two different points namely $\boldsymbol{x}^{(k)}$ and $\widehat{\boldsymbol{x}}^{(k)}$. This is the basis of the multipoint iterative methods [80] which achieves higher convergence rate without extra matrix factorizations. Since both (5.21) and (5.25) bring different features on the evolution of $\boldsymbol{x}$, it makes sense on providing some mechanism in order to properly balance the influence of each element. This can be elegantly achieved using the Heun's method as follows:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \left(\boldsymbol{R}^{(k)} - \boldsymbol{I}\right)\left(\boldsymbol{\phi}^{(k)} + \widehat{\boldsymbol{\phi}}^{(k)}\right) \tag{5.26}$$

The developed 3S-SIA have two degrees of freedom namely $\boldsymbol{R}$ and α. In order to get a simple implementation, these parameters should be updated each iteration using adaptive mechanisms. In this case, SIA provides easy updating schemes as it will be showed during this section. Firstly, matrix $\boldsymbol{R}$ can be initialized as follows:

$$\rho_{ij}^{(0)} = \delta_{ij}\rho_0 \tag{5.27}$$

where δ is the Kroenecker delta and $\rho_0 \in (0,1)$. A logic strategy is fixed ρ_0 close to 1 and reduces it as solution is successfully approached [75]. In that sense, $\|\mathbf{g}\|_\infty$ provides a good indicator for determining if the solution is successfully reaching. Thus, $\boldsymbol{R}$ is updated for $k \geq 1$ as follows:

$$\boldsymbol{R}^{(k)} = \left(\alpha^{(k-1)} e^{\frac{-\left\|\mathbf{g}\left(x^{(0)}\right)\right\|_\infty}{\left\|\mathbf{g}\left(x^{(k)}\right)\right\|_\infty}}\right)\boldsymbol{R}^{(k-1)} \tag{5.28}$$

where e is the natural exponential.

It is worth mentioning that $\boldsymbol{R}$ evolves according to the value of α, this is because both parameters should follow similar patterns. The reader can observe that equation (5.21), does not lead to the actual solution of PF equations for $\alpha \neq 0$ [44]. However, if α is fixed too short, robustness properties of the Lavrentiev's scheme are lost due to the nonsingularity of the Jacobian matrix is not ensured. Thus, it is easily deduce that α, as elements of $\boldsymbol{R}$, should be large at first iterations while they should be progressively reduce

as the iterative procedure progresses Following this guideline, the damping factor α may be updated as follows:

$$\alpha = \max\left(\text{diag}\left(\boldsymbol{R}^{(k)}\right)\right) \tag{5.29}$$

The main steps of the developed 3S-SIA are summarized in Algorithm 5.3 using pseudocode.

Algorithm 5.3: PF solution using 3S-SIA

1: Let $\boldsymbol{x}^{(0)}$, ε, $k_{\max}$ and ρ_0 be given
2: Initialize the iteration counter $k \leftarrow 0$
3: $\boldsymbol{R}^{(0)} \leftarrow$ Solve (5.27)
4: **while** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k)}\right)\right\|_{\infty} > \varepsilon$ **do**
5: $\alpha \leftarrow$ Solve (5.29)
6: Solve (5.21) *# Stage 1*
7: Solve (5.24)
8: Solve (5.25) *# Stage 2*
9: Solve (5.26) *# Stage 3*
10: **if** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k+1)}\right)\right\|_{\infty} < \varepsilon$ **then**
11: **break** *# convergence*
12: **else**
13: **if** $k > k_{\max}$ **then**
14: **break** *# failure*
15: **end if**
16: $\boldsymbol{R}^{(k)} \leftarrow$ solve (5.29)
17: **end if**
18: **end do**
19: **return** solution $\boldsymbol{x}^{(k)}$

5.4.- A Robust Power Flow solver based on the Composite Newton-Cotes formula

This Section is devoted on presenting a novel PF solver based on the Composite Newton-Cotes formulas [81]. Despite that the Composite Newton-Cotes formulas are a

family of well-known numerical integration techniques, the reader should not confuse the introduced technique with those solvers described in the Chapter 3, since the developed solver is not devoted in integrating the function (3.4).

5.4.1.- Newton-Cotes formulas

Firstly, the solution of an integral problem using the Newton-Cotes formulas is briefly explained. To do that, consider the following continuous function f over the interval $[a, b]$:

$$\int_a^b f(y)dy \tag{5.30}$$

Solution of the integral problem (5.30) can be solved using the well-known Trapezoidal rule [53]. This technique approximates (5.30) by evaluating f at two different points as follows:

$$\int_a^b f(y)dy \approx \frac{b-a}{2}(f(a)+f(b)) \tag{5.31}$$

Since the Trapezoidal rule uses two points on the function f for solving (5.30), it is known as a second order numerical integration approach. However, solution of (5.30) can be also obtained using third order techniques like the Simpson's method or even fourth order approaches like the Boole's rule.

However, as reader can detect, this approach is limited to the number of points considered. Alternatively, more accurate approximations of (5.30) may be achieved by dividing $[a, b]$ into smaller subintervals. This idea is sketched in Fig. 5.4, where it is easily checked that accuracy in the calculation is improved as more subintervals are considered. However, computational efficient of the method is affected since more calculations are required.

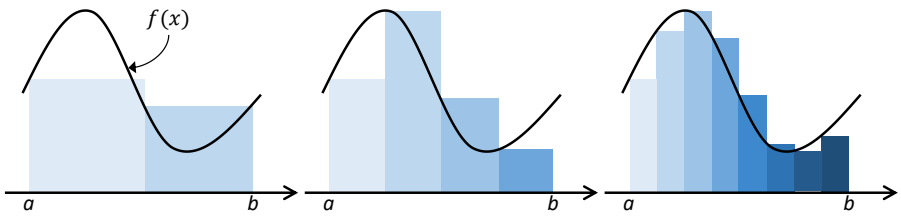


Fig. 5.4.- Sketch of Composite Integration techniques using 2 (left), 4 (middle) and 8 (right) subintervals

This is the main ingredient of the Newton-Cotes formula. For example, the Trapezoidal rule can be extended to its composite counterpart as follows:

$$\int_a^b f(x)dx \approx \frac{b-a}{n_{si}}\left(\frac{f(a)}{2} + \sum_{i=1}^{n_{si}-1} f\left(a + i\frac{b-a}{n_{si}}\right) + \frac{f(b)}{2}\right) \tag{5.32}$$

where, $n_{si} \in \mathbb{N}$ is the number of subintervals. It is worth mentioning that other numerical strategies (e.g. the Simpson or Boole's rules) can be used instead of the Trapezoidal rule in the same way.

5.4.2.- Motivation on the usage of the Newton-Cotes formulas

As it is well-known, PF is in fact a nonlinear problem. Solution of this kind of problems is an extensive in applied mathematics. Some of the available nonlinear solvers are conceived on the basis of the Newton's theorem, which founds that the well-known Newton's method can be derived as the approximation of the area under the studied function by a rectangle. On the basis of this deduction, one can immediately deduce that other geometric figures instead of a rectangle could be used. Indeed, this has been studied in other papers like [56]. Alternatively, the area under a function (or a curve), can be also approximated using quadrature formulas [82, 83]. Keeping this in mind, one can suspect that the Newton-Cotes formulas can be properly conceived as a power foundation for developing robust PF solvers.

5.4.3.- Application of the Newton-Cotes formulas for solving Power Flow

For considering the Newton-Cotes formulas for solving the PF problem, it is suitable to firstly define the following function:

$$\tilde{\mathbf{f}}(\boldsymbol{x}, \boldsymbol{y}) = -[\mathbf{g}'(\boldsymbol{x})]^{-1}\mathbf{g}(\boldsymbol{y}) \tag{5.33}$$

One can observe the main difference between (3.4) and (5.33), since the latter allows to evaluate the function $\mathbf{g}$ and its Jacobian at different points (i.e. $\boldsymbol{y}$ and $\boldsymbol{x}$, respectively). One of the main topics on integration using the Newton-Cotes formulas is to establish the main interval namely $[\boldsymbol{x}_a, \boldsymbol{x}_b]$. While the lower bound can be fixed in this case $\boldsymbol{x}_a^{(k)} = \boldsymbol{x}^{(k)}$, the upper bound can be calculated as follows:

$$\boldsymbol{x}_b^{(k)} = \boldsymbol{x}_a^{(k)} + h\tilde{\mathbf{f}}\left(\boldsymbol{x}_a^{(k)}, \boldsymbol{x}_a^{(k)}\right) = \boldsymbol{x}_a^{(k)} + h\mathbf{f}\left(\boldsymbol{x}_a^{(k)}\right) \tag{5.34}$$

As observed, equation (5.34) corresponds with the application of the Forward Euler method to the function (3.4). The step size can be updated each iteration by the following rule:

$$h = \min\begin{cases}\max\begin{cases}h_{\min}\\ h_{\max}/\left\|\tilde{\mathbf{f}}\left(\boldsymbol{x}_a^{(k)},\boldsymbol{x}_a^{(k)}\right)\right\|_\infty\end{cases}\\ h_{\max}\end{cases} \tag{5.35}$$

The following step should be devoted on splitting the main interval $[\boldsymbol{x}_a, \boldsymbol{x}_b]$. To do that, up to $m-1$ intermediate points (where $m \in \mathbb{N}$) are given by:

$$\boldsymbol{x}_i^{(k)} = \boldsymbol{x}_a^{(k)} + \frac{h}{i}\tilde{\mathbf{f}}\left(\boldsymbol{x}_a^{(k)},\boldsymbol{x}_a^{(k)}\right), \text{for } i = 2,3,\dots,m \tag{5.36}$$

Finally, using the information of the main interval and the intermediate points, the state vector can be updated as follows:

$$\boldsymbol{\phi}^{(k)} = \psi\tilde{\mathbf{f}}\left(\boldsymbol{x}_a^{(k)},\boldsymbol{x}_a^{(k)}\right) + \psi\tilde{\mathbf{f}}\left(\boldsymbol{x}_a^{(k)},\boldsymbol{x}_b^{(k)}\right) + \rho\sum_{i=2}^{m^{(k)}}\tilde{\mathbf{f}}\left(\boldsymbol{x}_a^{(k)},\boldsymbol{x}_i^{(k)}\right) \tag{5.37}$$

$$\boldsymbol{x}_a^{(k+1)} = \boldsymbol{x}_a^{(k)} + \frac{h}{\beta}\boldsymbol{\phi}^{(k)} \tag{5.38}$$

where, $\psi \in \mathbb{R}^+$ and $\rho \in \mathbb{R}^+$ are the extreme and intermediate slope coefficients, respectively and $\beta \in \mathbb{R}^+$ is the damping factor. The developed solver has some degrees of freedom defined by the parameters involved. The following expressions are devoted on providing some updating rules for these parameters.

$$\rho = \frac{1}{\left\|\boldsymbol{x}_a^{(k)} - \boldsymbol{x}_b^{(k)}\right\|_\infty^{\zeta}} \tag{5.39}$$

$$\gamma = \frac{m}{\rho} \tag{5.40}$$

$$\psi = \min[\gamma, \psi_{\max}] \tag{5.41}$$

$$m = \max\begin{cases}\min\begin{cases}\text{round}(\gamma)\\ m_{\max}\end{cases}\\ m_{\min}\end{cases} \tag{5.42}$$

$$\beta = \max[\gamma, \beta_{\min}] \tag{5.43}$$

where; $\mu \in \mathbb{R}$, $m_{\min} \in \mathbb{N} \geq 2$, $m_{\max} \in \mathbb{N}$, $\beta_{\min} \in \mathbb{R}$ and $\psi_{\max} \in \mathbb{R}$ define the minimum and maximum values of different parameters. On the other hand, $\text{round}(\cdot)$ rounds to the nearest integer.

The performance of the developed solver based on the Composite Newton-Cotes is strongly influenced by the values of the involved parameters. The value of these parameters along the updating rules (5.39)-(5.43) have been built according to heuristic principles a logic ideas. The following points aim at explaining the main ideas behind this topic:

- **Step size (h):** the self-adapted mechanism (5.35) aims at reducing the step size when the evolution of the algorithm is not satisfactory, thus, it aims at ensuring the convergence. This issue can be intuitively measured by the size of $\tilde{\mathbf{f}}(\boldsymbol{x}_a, \boldsymbol{x}_a)$. The idea behind this mechanism is as; if the size of this increment vector is too large, the step size should be reduced in order to compensate it and reduce the risk of divergence. Oppositely, if the increment vector is reasonably short, the step size can be increased with the aim to accelerate the convergence. The bounds related with the step size may be fixed attending to get an optimal balance between efficiency and robustness.

- **Parameter ζ:** this parameter is devoted to properly scale the size of the main interval considered. A reasonable value for this parameter may be found on the interval $0.01 \leq \zeta \leq 0.1$.

- **Slope coefficients (ρ and ψ):** these parameters can be viewed as the weights attributed for each intermediate point. Basically, as showed in Fig. 5.5, the developed solver performs as a truncated version of NR when ρ is high. Oppositely, when the extreme slope coefficient ψ is large, the developed technique performs similarly to a high order NR technique. One can expect from the main interval to be large at first iterations. Oppositely, when the convergence is ensured, the size of the main interval tends to drastically decrease. In this situation, ρ is should be tuned high while ψ tends to vanish (see Fig. 5.6). This is explained in the fact that, frequently the solution at last iterations lies presumably close to the intermediate points. Increasing the value of ρ, one gives more importance to intermediate than extreme points, so that, convergence is normally accelerated as showed in Fig. 5.7. Regarding the bound of these slope coefficients, it is found that $1 \leq \psi^{max} \leq 2$ works quite well in most cases.

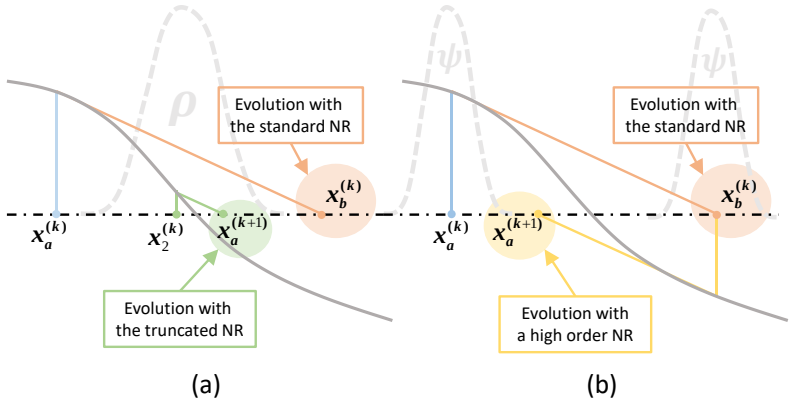


Fig 5.5.- Behavior of proposed CNC-PF for large values of the intermediate (a) and extreme slope coefficients (b)

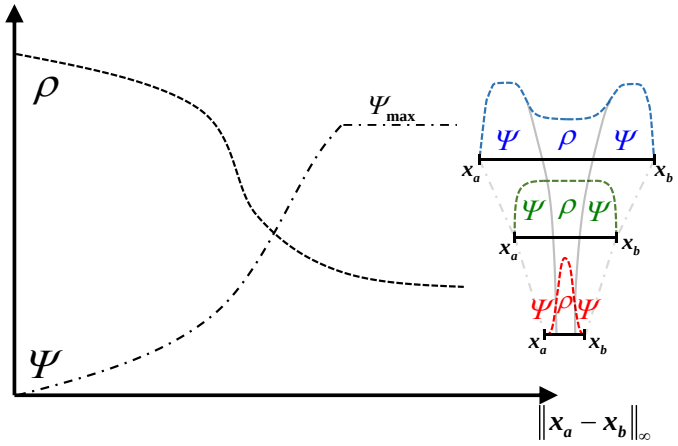


Fig 5.6.- Behavior of slope coefficients depending on the size of main interval

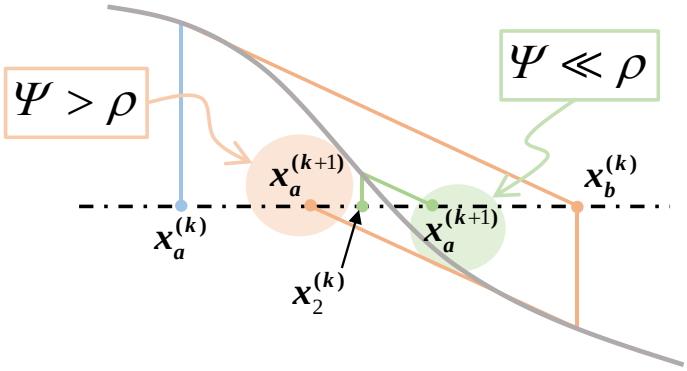


Fig 5.7.- Behavior of proposed CNC-PF when the size of main interval is small enough (Note as the point calculated for a large value of ρ is closer to the solution)

- **Number of intermediate points (m):** In this case, it is worth mentioning that one cannot take less than 1 sub-intervals. Therefore, at least one intermediate point should be taken. Consequently, a good option is fixing $m_{\min} \geq 2$. Intuitively,

taking more intermediate points should imply more robustness since more information is extracted from the increment vector, nevertheless, too many intermediate points should not be considered since they would imply many computations. Therefore, $m_{\max} \geq 3$ looks a logic option. The self-adapted mechanism of this parameter is focused on reducing it as soon as possible in order to save computational effort without affecting the robustness.

- **Damping factor ($\boldsymbol{\beta}$):** this factor has been included in order to mimetic the standard formulation of Newton-Cotes formula. It acts as a compensation for the step size. Thus, it is recommended to reduce it as the algorithm evolves, following the opposite pattern of the step size. In this regard, it should be lower bounded in order avoid instability, in practice it is founded that $2 \leq \beta_{\min} \leq 4$ works quite well in many cases.

The main steps of the developed PF solver based on the Newton-Cotes formulas are summarized in Algorithm 5.4 using pseudocode.

5.5.- A Power Flow approach based on the S-Iteration process

The Picard Iteration was introduced for finding the fixed points of nonexpansive mappings. In a general way, the Picard Iteration can be established as follows:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{T}\left(\boldsymbol{x}^{(k)}\right) \tag{5.44}$$

where $\boldsymbol{T}: \mathbb{R}^n \to \mathbb{R}^n$ is a mapping. This map can be defined in multiple ways, the most common way is maybe the Newton's method given by:

$$\boldsymbol{T}\left(\boldsymbol{x}^{(k)}\right) = \boldsymbol{x}^{(k)} - \left[\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right)\right]^{-1}\mathbf{g}\left(\boldsymbol{x}^{(k)}\right) \tag{5.45}$$

In this case, the fixed point iteration (5.44) is usually called Picard-Newton Iteration. Due to the issues experimented by (5.44) in some nonexpansive mappings (even in Banach spaces), alternative fixed point iterations were developed. The most popular are maybe the Krasnoselskii [84], Mann [85] and Ishikawa [86] fixed point iterations. More recently, a fixed point iteration called S-iteration process was introduced in [87], which is given by:

Algorithm 5.4: PF solver based on the Composite Newton-Cotes formulas

1: Let $\boldsymbol{x}^{(0)}$, ε, $k_{\max}$, ζ, $\psi_{\max}$, $\beta_{\min}$, $h_{\min}$, $h_{\max}$, m, $m_{\min}$ and $m_{\max}$ be given

2: Initialize the iteration counter $k \leftarrow 0$

4: **while** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k)}\right)\right\|_{\infty} > \varepsilon$ **do**

5: $h \leftarrow$ Solve (5.35)

6: Solve (5.34)

7: $\psi \leftarrow$ Solve (5.41)

8: $m \leftarrow$ Solve (5.42)

9: $\beta \leftarrow$ Solve (5.43)

10: $j = 2$

11: **for** $j \leq m$ **do**

12: Solve (5.36)

13: **end do**

14: Solve (5.38)

15: **if** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k+1)}\right)\right\|_{\infty} < \varepsilon$ **then**

16: **break** *# convergence*

17: **else**

18: **if** $k > k_{\max}$ **then**

19: **break** *# failure*

20: **end if**

21: **end if**

22: **end do**

23: **return** solution $\boldsymbol{x}^{(k)}$

$$\begin{cases} \boldsymbol{y}^{(k)} = \left(1-\beta^{(k)}\right)\boldsymbol{x}^{(k)} + \beta^{(k)}\boldsymbol{T}\left(\boldsymbol{x}^{(k)}\right) \\ \boldsymbol{x}^{(k+1)} = \left(1-\alpha^{(k)}\right)\boldsymbol{T}\left(\boldsymbol{x}^{(k)}\right) + \alpha^{(k)}\boldsymbol{T}\left(\boldsymbol{y}^{(k)}\right) \end{cases} \tag{5.46}$$

where, $\left\{\alpha^{(k)}\right\}$ and $\left\{\beta^{(k)}\right\}$ are real sequences in $(0,1)$ which satisfy the following condition:

$$\sum_{k=1}^{\infty} \alpha^{(k)}\beta^{(k)}\left(1-\beta^{(k)}\right) = \infty \tag{5.47}$$

It is demonstrated that the S-iteration process has similar convergence rate to the Picard iteration, but it is faster that the Mann and Ishikiawa's fixed points methods [87]. The S-iteration process has been also used for solving constrained minimization and split

feasibility problems [88]. Indeed, if the mapping $\boldsymbol{T}$ is defined as (5.45), the process (5.46) is called the S-iteration-Newton process. Nevertheless, an alternative approach was proposed in [89]. In this reference, a Newton's mapping with fixed Jacobian was used in combination with the S-iteration process. Thus, the following mapping was obtained:

$$\begin{cases} \boldsymbol{z}^{(k)} = \boldsymbol{x}^{(k)} - \left[\mathbf{g}'\left(\boldsymbol{x}^{(0)}\right)\right]^{-1}\mathbf{g}\left(\boldsymbol{x}^{(k)}\right) \\ \boldsymbol{y}^{(k)} = (1-\alpha)\boldsymbol{x}^{(k)} + \alpha\boldsymbol{z}^{(k)} \\ \boldsymbol{x}^{(k+1)} = \boldsymbol{y}^{(k)} - \left[\mathbf{g}'\left(\boldsymbol{x}^{(0)}\right)\right]^{-1}\mathbf{g}\left(\boldsymbol{y}^{(k)}\right) \end{cases} \tag{5.48}$$

In this case, $\alpha \in \mathbb{R}^{+}$ is a parameter rather than a serie. As observed, the mapping (5.48) presents a very low computational burden since the Jacobian matrix is only factorized once. However, this advantage provokes linear convergence rate, which may be problematic especially in heavy loading systems. In order to overcome this drawback, we could update the Jacobian matrix (and factorize it), only when the convergence turns very slow. To do that, the following indicator can be used:

$$\mu = \left| \left\| \left[\mathbf{g}'\left(\boldsymbol{x}^{(0)}\right)\right]^{-1}\mathbf{g}\left(\boldsymbol{x}^{(k)}\right) \right\|_{\infty} - \left\| \left[\mathbf{g}'\left(\boldsymbol{x}^{(0)}\right)\right]^{-1}\mathbf{g}\left(\boldsymbol{x}^{(k-1)}\right) \right\|_{\infty} \right| \tag{5.49}$$

If (5.49) is lower than a predefined threshold ϵ, the Jacobian matrix should be updated in order to accelerate the convergence.

The mapping (5.48) defines itself a PF solver which has been called S-iteration-Newton's method (SIP-NR). In order to avoid confusions, when the indicator (5.49) is used within SIP-NR, the resulting PF solver has been called the Modified SIP-NR (MSIP-NR). These two techniques are summarized in Algorithms 5.5 and 5.6, respectively, using pseudocode.

5.6.- A robust and efficient PF solver based on Heun and King-Werner's methods

The King-Werner's methods [66], form a family of high order Newton-like methods for nonlinear problems. They are usually quite efficient and present convergence orders higher than two. Examples of this kind of techniques can be found in [90-92]. This kind of techniques are generally less robust than NR, however, combining them with some well-known robust approach, an efficient and reasonably robust solver may be obtained. In this case, a King-Werner-like method has been combined with the Heun's method [53] for

obtaining a competitive algorithm. The developed PF solver may be summarized in the following steps:

Algorithm 5.5: PF solution using SIP-NR

1: Let $\boldsymbol{x}^{(0)}$, ε, $k_{\max}$ and α be given
2: Initialize the iteration counter $k \leftarrow 0$
4: **while** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k)}\right)\right\|_{\infty} > \varepsilon$ **do**
5: Solve (5.48)
6: **if** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k+1)}\right)\right\|_{\infty} < \varepsilon$ **then**
7: **break** *# convergence*
8: **else**
9: **if** $k > k_{\max}$ **then**
10: **break** *# failure*
11: **end if**
12: **end if**
13: **end do**
14: **return** solution $\boldsymbol{x}^{(k)}$

Step 1: calculate (3.4) at $\boldsymbol{x}^{(k)}$.

Step 2-Euler's update: the Heun's method is a 2-stage Runge-Kutta formula which uses, as first stage, the Forward Euler method as follows:

$$\boldsymbol{y}^{(k)} = \boldsymbol{x}^{(k)} + h\mathbf{f}\left(\boldsymbol{x}^{(k)}\right) \tag{5.50}$$

In this case, step size has to be initialized. To do that, the following rule is considered:

$$h = \max\left\{h_{\min}, \min\left\{h_{\max}, {}^{1}/_{(SSR^{(0)})^{\zeta}}\right\}\right\} \tag{5.51}$$

where, $\zeta \in \mathbb{R}^{+}$ while SSR is given by:

$$SSR^{(k)} = \frac{1}{2}\left[\mathbf{g}\left(\boldsymbol{x}^{(k)}\right)\right]^{T}\mathbf{g}\left(\boldsymbol{x}^{(k)}\right) \tag{5.52}$$

Step 3-The King-Werner's update: as in [92], the state vector is now updated by calculating the midpoint of the interval defined by $\boldsymbol{x}$ and $\boldsymbol{y}$.

$$\tilde{\boldsymbol{x}}^{(k)} = \frac{1}{2}\left(\boldsymbol{x}^{(k)} + \boldsymbol{y}^{(k)}\right) \tag{5.53}$$

Step 4-Evaluation of the midpoint: in this step, the midpoint calculated in the previous step is used for evaluating within the function (3.4). The reader can check that this step supposes an extra factorization, which implies an extra computational burden. However, this step provides to the resulting algorithm of good properties as explained in Section 5.6.1.

Algorithm 5.6: PF solution using MSIP-NR

1: Let $\boldsymbol{x}^{(0)}$, ε, $k_{\max}$, α and ϵ be given

2: Initialize the iteration counter $k \leftarrow 0$

4: **while** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k)}\right)\right\|_{\infty} > \varepsilon$ **do**

5: Solve (5.48) *# using the most updated Jacobian matrix*

6: **if** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k+1)}\right)\right\|_{\infty} < \varepsilon$ **then**

7: **break** *# convergence*

8: **else**

9: **if** $k > k_{\max}$ **then**

10: **break** *# failure*

11: **end if**

12: **end if**

13: $\mu \leftarrow$ Solve (5.49)

14: **if** $\mu < \epsilon$ *# update the Jacobian matrix and factorize it*

15: **end if**

16: **end do**

17: **return** solution $\boldsymbol{x}^{(k)}$

Step 5-Heun's update: finally, state vector is updated using the Heun's method as follows:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \frac{h}{2}\left(\psi\mathbf{f}\left(\boldsymbol{x}^{(k)}\right) + (2-\psi)\mathbf{f}\left(\tilde{\boldsymbol{x}}^{(k)}\right)\right) \tag{5.54}$$

where $\psi \in (0,2)$. This parameter is here introduced for naturally leading the developed approach to achieve quadratic convergence. One should note that the developed method corresponds with NR when $\psi = 2$.

Step 6: if the convergence criteria is not satisfied, increase the iteration counter. If $k > k_{\max}$ or the method has converged at $\boldsymbol{x}^{(k+1)}$ then, stop, otherwise go to Step 7.

Step 7-Update parameters: the developed iterative procedure has two degrees of freedom namely h and ψ. These parameters are updated each iteration according to the following rules:

$$\epsilon = \left\| \boldsymbol{x}^{(k)} - \boldsymbol{y}^{(k-1)} \right\|_{\infty} \tag{5.55}$$

$$h = \begin{cases} \max\{0{,}9h, h_{\min}\} & \text{if } \epsilon > \alpha \\ \min\{1{,}1h, h_{\max}\} & \text{otherwise} \end{cases} \tag{5.56}$$

$$\psi = 2 \left| \frac{SSR^{(k)} - SSR^{(0)}}{SSR^{(0)}} \right| \tag{5.57}$$

Once the parameters have been properly updated, return to Step #1.

The main steps of the developed Heun-King-Werner's method are summarized in Algorithm 5.7 using pseudocode. It is worth noting that equation (2.5) is used when $\psi \geq \psi_{\max}$ (where $\psi_{\max}$) in order to alleviate the total computational cost of the algorithm. It has sense since, as commented, the developed Heun-King-Werner's method correspond with NR when $\psi = 2$. Consequently, setting $\psi_{\max} \approx 2$ is recommended.

The developed Heun-King-Werner's approach shows some interesting features, which are commented in the following Sections.

5.6.1.- Natural suitability for both well and ill-conditioned cases

Steps 1-7 described in Section 5.6, describe a naturally robust algorithm. This can be clearly observed in Fig 5.8. In this figure, the process for solving an ill-conditioned case is sketched. In this case, scenario with $h = h_{\max} = 1$ is showed since it clearly supposes the most unfavorable situation. As seen, naturally evolution of the developed technique leads it to reach the correct solution, while NR diverges.

Since $h = 1$ in the considered scenario, the effect of this parameter can be considered null. Consequently, only the developed iterative procedure is itself analyzed. Hence, it can be concluded that the developed Heun-King-Werner algorithm constitutes a robust technique, independently of the value of the step size.

The performance of the developed technique in well-conditioned cases is sketched in Fig. 5.9. As observed, the developed method performs like a high order Newton-like technique, employing less iterations than NR for reaching the solution.

From previous examples, one can conclude that the developed Heun-King-Werner's method present acceptable characteristics for both well and ill-conditioned systems. These universal features are not easy to find in available solvers. Indeed, while the robust Pf solvers are usually very inefficient to be used in well-conditioned systems, the conventional methods perform poorly in ill-conditioned cases.

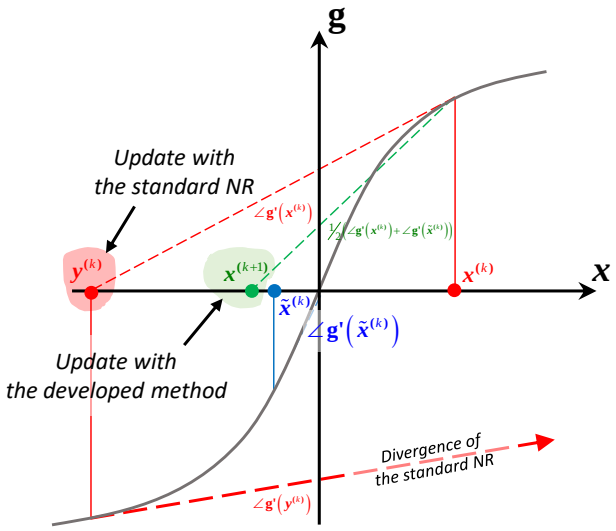


Fig. 5.8.- Behavior of developed HKW in case of ill-conditioned cases

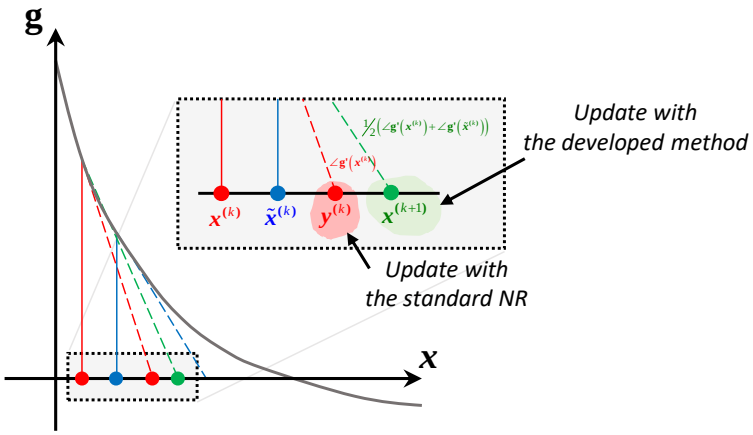


Fig. 5.9.- Behavior of developed HKW in case of well-conditioned cases

5.6.2.- Natural evolution towards quadratic convergence

As previously commented, the developed method has quadratic convergence when $\psi = 2$ and $h = 1$. Adaptive mechanisms (5.56) and (5.57) have been developed so that both ψ and h, naturally evolve until achieve quadratic convergence characteristics (as NR).

Algorithm 5.7: PF solution using the Heun-King-Werner's method

1: Let $\boldsymbol{x}^{(0)}$, ε, $k_{\max}$, $h_{\min}$, $h_{\max}$, ζ and ψ be given

2: Initialize the iteration counter $k \leftarrow 0$

4: **while** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k)}\right)\right\|_{\infty} > \varepsilon$ **do**

5: **if** $\psi \geq \psi_{\max}$

6: Solve (2.5) *# Newton-Raphson*

7: **else**

8: **if** $k = 0$

9: $h \leftarrow$ Solve (5.51)

10: **end if**

11: Solve (5.50)

12: Solve (5.53)

13: Solve (5.54)

14: **end if**

15: **if** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k+1)}\right)\right\|_{\infty} < \varepsilon$ **then**

16: **break** *# convergence*

17: **else**

18: **if** $k > k_{\max}$ **then**

19: **break** *# failure*

20: **end if**

21: **end if**

22: $h \leftarrow$ Solve (5.56)

23: $\psi \leftarrow$ Solve (5.57)

24: **end do**

25: **return** solution $\boldsymbol{x}^{(k)}$

5.7.- Conclusions

One of the main objectives of this work, is to develop PF solvers which present acceptable robustness and efficiency to be suitable to be widespread used in industry tools. Various solvers which aim at covering this point has been presented in this Chapter. All the

presented techniques have been developed in the context of this work, with the exception of SIP-NR which was already developed in reference [89].

Most of introduced PF solvers present a competitive computational burden (comparable to NR), which supposes an inescapable characteristic for any industrial or commercial solver. However, it is difficult to know *a priori* which of the developed techniques is more competitive since the convergence rate here supposes a critical aspect. This will be discussed in Chapter 7, where various numerical results are provided.

Chapter 6

High Order Newton-like methods

The most conventional PF solver, i.e. NR, has been widely used due to some interesting features. On the one hand, its robustness is acceptable in many cases. On the other hand, it presents a very competitive computational burden since just a matrix factorization is computed each iteration. Nevertheless, NR may require many iterations in some cases like heavy loading systems. This is due to its quadratic convergence. Although this feature supposes an advantage of NR with respect other conventional solvers, it may be insufficient under some conditions. Unlike to NR, the high order Newton-like methods present a convergence order higher than two, which is usually reflected in a lower number of iterations. Despite the huge amount of high order Newton-like techniques available in the literature (see e.g. [50]), only the reference [49] has considered this kind of methods for PF analysis. This chapter presents two novel PF solvers based on high order Newton-like methods. While the methodology described in Section 6.1 was already developed for nonlinear equations in [92], the method introduced in Section 6.2 has been developed in the context of this work.

6.1.- The Newton-Raphson Predictor Corrector

In reference [92], McDougall and Wotherspoon introduced an efficient high order Newton-like method. It belongs to the family of King-Werner's method and achieves a convergence order of $1+\sqrt{2}$. For a scalar function $f(x)$, the method described in [92] proceeds as follows:

At $k=0$

$$\begin{cases} \hat{x}^{(0)} = x^{(0)} \\ x^{(1)} = x^{(0)} - \dfrac{f\left(x^{(0)}\right)}{f'\left(\frac{1}{2}\left(x^{(0)}+\hat{x}^{(0)}\right)\right)} \end{cases} \tag{6.1}$$

For $k \geq 1$

$$\begin{cases} \hat{x}^{(k)} = x^{(k)} - \dfrac{f\left(x^{(k)}\right)}{f'\left(\frac{1}{2}\left(x^{(k)}+\hat{x}^{(k-1)}\right)\right)} \\ x^{(k+1)} = x^{(k)} - \dfrac{f\left(x^{(k)}\right)}{f'\left(\frac{1}{2}\left(x^{(k)}+\hat{x}^{(k)}\right)\right)} \end{cases} \tag{6.2}$$

where f' is the first Fréchet derivative of f. As observed, this method performs as NR at $k=0$. The authors of [92] identified the first step of (6.2) as a predictor while the second step is called corrector. Due to that, I have called this technique the Newton-Raphson Predictor Corrector (NRPC). Fig. 6.1 sketches the performance of NRPC.

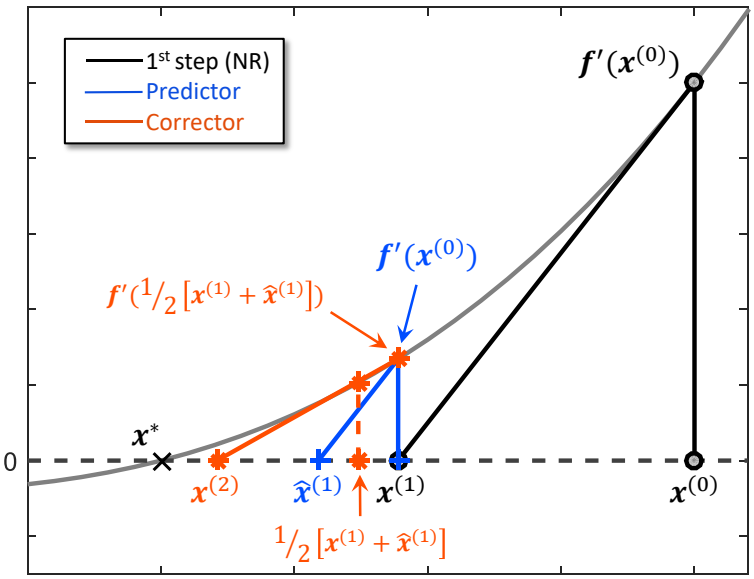


Fig. 6.1.- Sketch of NRPC

NRPC can be easily uses for PF analysis by simply extending its formulation to the n-dimensional case. This leads to the following mapping:

At $k = 0$

$$\begin{cases} \widehat{\boldsymbol{x}}^{(0)} = \boldsymbol{x}^{(0)} \\ \boldsymbol{x}^{(1)} = \boldsymbol{x}^{(0)} - \left[\mathbf{g}'\left(\frac{1}{2}\left(\boldsymbol{x}^{(0)} + \widehat{\boldsymbol{x}}^{(0)}\right)\right)\right]^{-1} \mathbf{g}\left(\boldsymbol{x}^{(0)}\right) \end{cases} \tag{6.3}$$

For $k \geq 1$

$$\begin{cases} \widehat{\boldsymbol{x}}^{(k)} = \boldsymbol{x}^{(k)} - \left[\mathbf{g}'\left(\frac{1}{2}\left(\boldsymbol{x}^{(k)} + \widehat{\boldsymbol{x}}^{(k-1)}\right)\right)\right]^{-1} \mathbf{g}\left(\boldsymbol{x}^{(k)}\right) \\ \boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \left[\mathbf{g}'\left(\frac{1}{2}\left(\boldsymbol{x}^{(k)} + \widehat{\boldsymbol{x}}^{(k)}\right)\right)\right]^{-1} \mathbf{g}\left(\boldsymbol{x}^{(k)}\right) \end{cases} \tag{6.4}$$

It is worth noting that the mapping (6.3)-(6.4) only requires a matrix factorization each iteration, which results in a computational burden comparable with NR. However, the convergence order achieve by NRPC is $1 + \sqrt{2} \approx 2.14$. The main steps of NRPC for PF analysis are summarized in Algorithm 6.1 using pseudocode.

6.2.- An efficient multistep method

Basically there are two ways for achieving high convergence order. On the one hand, one can recur to multiple Jacobian evaluations and factorizations, this approach has been used for a wide family of nonlinear solvers [82, 93-96]. On the other hand, instead of multiple Jacobian evaluations, one can recur to evaluate the function in several points. The latter option is undoubtedly more attractive for PF analysis as heavy calculations are avoided. Those nonlinear solvers which achieve high convergence rate by evaluating the function in different points are called multistep (or multipoint) techniques [80] and they are based on the following theorem [97].

Theorem 6.1. Let r be a simple zero of a function $f(x)$ *and let* ϕ define an iterative method of order p. Then, a composite method $\psi(x)$ introduced by Newton's method

$$\psi(x) = \phi(x) - \frac{f(\phi(x))}{f'(x)} \tag{6.5}$$

defines an iterative method of order $p + 1$

On the basis of this Theorem, the following mapping for PF analysis is developed.

$$\begin{cases} \boldsymbol{x}^{(k)} = \boldsymbol{y}_1^{(k)} \\ \boldsymbol{y}_i^{(k)} = \frac{1}{i-1}\sum_{j=1}^{i-1}\boldsymbol{y}_j^{(k)} - i\left[\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right)\right]^{-1}\mathbf{g}\left(\frac{1}{i-1}\sum_{j=1}^{i-1}\boldsymbol{y}_j^{(k)}\right), \text{for } i = 2,3 \dots, N \\ \boldsymbol{x}^{(k+1)} = \frac{1}{N}\sum_{j=1}^{N}\boldsymbol{y}_j^{(k)} \end{cases} \tag{6.6}$$

Algorithm 6.1: PF solution using NRPC

1: Let $\boldsymbol{x}^{(0)}$, ε and $k_{\max}$

2: Initialize the iteration counter $k \leftarrow 0$

3: **while** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k)}\right)\right\|_\infty > \varepsilon$ **do**

4: **if** $k = 0$

5: Solve (6.3)

6: **else**

7: Solve (6.4)

8: **end if**

9: **if** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k+1)}\right)\right\|_\infty < \varepsilon$ **then**

10: **break** *# convergence*

11: **else**

12: $k \leftarrow k + 1$

13: **if** $k > k_{\max}$ **then**

14: **break** *# failure*

15: **end if**

16: **end if**

17: **end do**

18: **return** solution $\boldsymbol{x}^{(k)}$

where $N \in \mathbb{N}$ is the total number of steps carried out within the mapping (6.6). The convergence rate of the developed PF solver strongly depends on the value of N as demonstrated in the following Section. The PF solution using the paradigm (6.6) is described in Algorithm 6.2 using pseudocode.

Algorithm 6.2: PF solution using the multistep method (6.6)

1: Let $\boldsymbol{x}^{(0)}$, ε, $k_{\max}$ and N

2: Initialize the iteration counter $k \leftarrow 0$

3: **while** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k)}\right)\right\|_{\infty} > \varepsilon$ **do**

4: Solve (6.6)

5: **if** $\left\|\mathbf{g}\left(\boldsymbol{x}^{(k+1)}\right)\right\|_{\infty} < \varepsilon$ **then**

6: **break** *# convergence*

7: **else**

8: $k \leftarrow k + 1$

9: **if** $k > k_{\max}$ **then**

10: **break** *# failure*

11: **end if**

12: **end if**

13: **end do**

14: **return** solution $\boldsymbol{x}^{(k)}$

6.2.1.- Convergence analysis

This Section is devoted on demonstrating the order of convergence of the developed multistep method (6.6).

Theorem 6.2. Let $\mathbf{g}$: D $\subseteq \mathbb{R}^n \mapsto \mathbb{R}^n$ be sufficiently differentiable at each point of an open neighborhood D of $\boldsymbol{r} \in \mathbb{R}^n$, this is a solution of the system $\mathbf{g}(\boldsymbol{x}) = \mathbf{0}$. Let us suppose that $\mathbf{g}(\boldsymbol{x})$ is continuous and nonsingular in $\boldsymbol{x}$. Then, the sequence $\left\{\boldsymbol{x}^{(k)}\right\}_{k\geq 0}$ obtained using the developed PF solution algorithm (4) converges to $\boldsymbol{r}$ with order N.

Proof. From Taylor expansion of $\mathbf{g}\left(\boldsymbol{y}_1^{(k)}\right)$ and $\mathbf{g}'\left(\boldsymbol{y}_1^{(k)}\right)$ about $\boldsymbol{r}$ we have:

$$\mathbf{g}\left(\boldsymbol{y}_1^{(k)}\right) \approx \mathbf{g}'(\boldsymbol{r})\left[\boldsymbol{e}_1^{(k)} + \boldsymbol{C}_2\boldsymbol{e}_1^{(k)^2} + \boldsymbol{C}_3\boldsymbol{e}_1^{(k)^3} + \boldsymbol{C}_4\boldsymbol{e}_1^{(k)^4}\right] + \mathbf{O}\left(\boldsymbol{e}_1^{(k)^5}\right) \tag{6.7}$$

$$\mathbf{g}'\left(\boldsymbol{y}_1^{(k)}\right) = \mathbf{g}'(\boldsymbol{r})\left[\boldsymbol{I} + 2\boldsymbol{C}_2\boldsymbol{e}_1^{(k)} + 3\boldsymbol{C}_3\boldsymbol{e}_1^{(k)^2} + 4\boldsymbol{C}_4\boldsymbol{e}_1^{(k)^3}\right] + \boldsymbol{O}\left(\boldsymbol{e}_1^{(k)^4}\right) \tag{6.8}$$

where, $\boldsymbol{C}_j = (1/j!)[\mathbf{g}'(\boldsymbol{r})]^{-1}\mathbf{g}^{(j)}(\boldsymbol{r}) \in \mathbb{R}^{n\times n}$, $j = 2,3,\ldots$, and $\boldsymbol{e}_1 = \boldsymbol{y}_1 - \boldsymbol{r} \in \mathbb{R}^n$

Considering the following inverse definition:

$$\left[\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right)\right]^{-1}\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right) = \mathbf{g}'\left(\boldsymbol{x}^{(k)}\right)\left[\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right)\right]^{-1} = \boldsymbol{I} \tag{6.9}$$

And solving the resulting linear system one obtains:

$$\left[\mathbf{g}'\left(\boldsymbol{y}_{\boldsymbol{1}}^{(k)}\right)\right]^{-1} = \left[\boldsymbol{I} - 2\boldsymbol{C}_{\boldsymbol{2}}\boldsymbol{e}_{\boldsymbol{1}}^{(k)} + (4\boldsymbol{C}_{\boldsymbol{2}} - 3\boldsymbol{C}_{\boldsymbol{3}})\boldsymbol{e}_{\boldsymbol{1}}^{(k)^2}\right][\mathbf{g}'(\boldsymbol{r})]^{-1} + \boldsymbol{O}\left(\boldsymbol{e}_{\boldsymbol{1}}^{(k)^3}\right) \tag{6.10}$$

Now, taking $i = 2$ one obtains:

$$\boldsymbol{y}_{\boldsymbol{2}}^{(k)} - \boldsymbol{r} = \boldsymbol{y}_{\boldsymbol{1}}^{(k)} - 2\left[\mathbf{g}'\left(\boldsymbol{y}_{\boldsymbol{1}}^{(k)}\right)\right]^{-1}\mathbf{g}\left(\boldsymbol{y}_{\boldsymbol{1}}^{(k)}\right) - \boldsymbol{r} = -\boldsymbol{e}_{\boldsymbol{1}}^{(k)} + 2\boldsymbol{C}_{\boldsymbol{2}}\boldsymbol{e}_{\boldsymbol{1}}^{(k)^2} - 4(\boldsymbol{C}_{\boldsymbol{2}}^{\boldsymbol{2}} - \boldsymbol{C}_{\boldsymbol{3}})\boldsymbol{e}_{\boldsymbol{1}}^{(k)^3} -$$
$$2(-4\boldsymbol{C}_{\boldsymbol{2}}^{\boldsymbol{3}} + 7\boldsymbol{C}_{\boldsymbol{2}}\boldsymbol{C}_{\boldsymbol{3}} - 3\boldsymbol{C}_{\boldsymbol{4}})\boldsymbol{e}_{\boldsymbol{1}}^{(k)^4} - 2(-8\boldsymbol{C}_{\boldsymbol{2}}^{\boldsymbol{4}} + 16\boldsymbol{C}_{\boldsymbol{2}}^{\boldsymbol{2}}\boldsymbol{C}_{\boldsymbol{3}} - 6\boldsymbol{C}_{\boldsymbol{2}}\boldsymbol{C}_{\boldsymbol{4}} - 3\boldsymbol{C}_{\boldsymbol{3}}^{\boldsymbol{2}})\boldsymbol{e}_{\boldsymbol{1}}^{(k)^5} +$$
$$\mathbf{O}\left(\boldsymbol{e}_{\boldsymbol{1}}^{(k)^6}\right) \tag{6.11}$$

At this point one has:

$$\underbrace{\tfrac{1}{2}\textstyle\sum_{j=1}^{2}\boldsymbol{y}_{j}^{(k)} - \boldsymbol{r}}_{\boldsymbol{e}_{\boldsymbol{2}}^{(k)}} = \boldsymbol{C}_{\boldsymbol{2}}\boldsymbol{e}_{\boldsymbol{1}}^{(k)^2} - 2(\boldsymbol{C}_{\boldsymbol{2}}^{\boldsymbol{2}} - \boldsymbol{C}_{\boldsymbol{3}})\boldsymbol{e}_{\boldsymbol{1}}^{(k)^3} - (-4\boldsymbol{C}_{\boldsymbol{2}}^{\boldsymbol{3}} + 7\boldsymbol{C}_{\boldsymbol{2}}\boldsymbol{C}_{\boldsymbol{3}} - 3\boldsymbol{C}_{\boldsymbol{4}})\boldsymbol{e}_{\boldsymbol{1}}^{(k)^4} -$$
$$(-8\boldsymbol{C}_{\boldsymbol{2}}^{\boldsymbol{4}} + 16\boldsymbol{C}_{\boldsymbol{2}}^{\boldsymbol{2}}\boldsymbol{C}_{\boldsymbol{3}} - 6\boldsymbol{C}_{\boldsymbol{2}}\boldsymbol{C}_{\boldsymbol{4}} - 3\boldsymbol{C}_{\boldsymbol{3}}^{\boldsymbol{2}})\boldsymbol{e}_{\boldsymbol{1}}^{(k)^5} + \mathbf{O}\left(\boldsymbol{e}_{\boldsymbol{1}}^{(k)^6}\right) \tag{6.12}$$

$$\mathbf{g}\left(\tfrac{1}{2}\textstyle\sum_{j=1}^{2}\boldsymbol{y}_{j}^{(k)}\right) \approx \mathbf{g}'(\boldsymbol{r})\left[\boldsymbol{e}_{\boldsymbol{2}}^{(k)} + \boldsymbol{C}_{\boldsymbol{2}}\boldsymbol{e}_{\boldsymbol{2}}^{(k)^2} + \boldsymbol{C}_{\boldsymbol{3}}\boldsymbol{e}_{\boldsymbol{2}}^{(k)^3} + \boldsymbol{C}_{\boldsymbol{4}}\boldsymbol{e}_{\boldsymbol{2}}^{(k)^4}\right] + \mathbf{O}\left(\boldsymbol{e}_{\boldsymbol{2}}^{(k)^5}\right) \tag{6.13}$$

Now, for $i = 3$ one has:

$$\boldsymbol{y}_{\boldsymbol{3}}^{(k)} - \boldsymbol{r} = \tfrac{1}{2}\textstyle\sum_{j=1}^{2}\boldsymbol{y}_{j}^{(k)} - 3\left[\mathbf{g}'\left(\boldsymbol{y}_{\boldsymbol{1}}^{(k)}\right)\right]^{-1}\mathbf{g}\left(\tfrac{1}{2}\textstyle\sum_{j=1}^{2}\boldsymbol{y}_{j}^{(k)}\right) - \boldsymbol{r} = -2\boldsymbol{C}_{\boldsymbol{2}}\boldsymbol{e}_{\boldsymbol{1}}^{(k)^2} + 2(5\boldsymbol{C}_{\boldsymbol{2}}^{\boldsymbol{2}} -$$
$$2\boldsymbol{C}_{\boldsymbol{3}})\boldsymbol{e}_{\boldsymbol{1}}^{(k)^3} - (-35\boldsymbol{C}_{\boldsymbol{2}}^{\boldsymbol{3}} + 35\boldsymbol{C}_{\boldsymbol{2}}\boldsymbol{C}_{\boldsymbol{3}} - 6\boldsymbol{C}_{\boldsymbol{4}})\boldsymbol{e}_{\boldsymbol{1}}^{(k)^4} + (74\boldsymbol{C}_{\boldsymbol{2}}^{\boldsymbol{4}} - 100\boldsymbol{C}_{\boldsymbol{2}}^{\boldsymbol{2}}\boldsymbol{C}_{\boldsymbol{3}} + 18\boldsymbol{C}_{\boldsymbol{2}}\boldsymbol{C}_{\boldsymbol{4}} +$$
$$12\boldsymbol{C}_{\boldsymbol{3}}^{\boldsymbol{2}})\boldsymbol{e}_{\boldsymbol{1}}^{(k)^5} + \mathbf{O}\left(\boldsymbol{e}_{\boldsymbol{1}}^{(k)^6}\right) \tag{6.14}$$

$$\underbrace{\tfrac{1}{3}\textstyle\sum_{j=1}^{3}\boldsymbol{y}_{j}^{(k)} - \boldsymbol{r}}_{\boldsymbol{e}_{\boldsymbol{3}}^{(k)}} = 2\boldsymbol{C}_{\boldsymbol{2}}^{\boldsymbol{2}}\boldsymbol{e}_{\boldsymbol{1}}^{(k)^3} + (-9\boldsymbol{C}_{\boldsymbol{2}}^{\boldsymbol{3}} + 7\boldsymbol{C}_{\boldsymbol{2}}\boldsymbol{C}_{\boldsymbol{3}})\boldsymbol{e}_{\boldsymbol{1}}^{(k)^4} + (30\boldsymbol{C}_{\boldsymbol{2}}^{\boldsymbol{4}} - 44\boldsymbol{C}_{\boldsymbol{2}}^{\boldsymbol{2}}\boldsymbol{C}_{\boldsymbol{3}} + 10\boldsymbol{C}_{\boldsymbol{2}}\boldsymbol{C}_{\boldsymbol{4}} +$$
$$6\boldsymbol{C}_{\boldsymbol{3}}^{\boldsymbol{2}})\boldsymbol{e}_{\boldsymbol{1}}^{(k)^5} + \mathbf{O}\left(\boldsymbol{e}_{\boldsymbol{1}}^{(k)^6}\right) \tag{6.15}$$

Repeating for $i = 4$ and $i = 5$, one can check that:

$$\boldsymbol{e}_{\boldsymbol{4}}^{(k)} = 4\boldsymbol{C}_{\boldsymbol{2}}^{\boldsymbol{3}}\boldsymbol{e}_{\boldsymbol{1}}^{(k)^4} + \mathbf{O}\left(\boldsymbol{e}_{\boldsymbol{1}}^{(k)^5}\right) \tag{6.16}$$

$$\boldsymbol{e}_{\boldsymbol{5}}^{(k)} = 8\boldsymbol{C}_{\boldsymbol{2}}^{\boldsymbol{4}}\boldsymbol{e}_{\boldsymbol{1}}^{(k)^5} + \mathbf{O}\left(\boldsymbol{e}_{\boldsymbol{1}}^{(k)^6}\right) \tag{6.17}$$

Thus, it is deduced for a generic *N*

$$\boldsymbol{e}_{\boldsymbol{N}}^{(k)} = \boldsymbol{e}_{\mathbf{1}}^{(k+1)} = 2^{N-2}\boldsymbol{C}_{\mathbf{2}}^{N-1}\boldsymbol{e}_{\mathbf{1}}^{(k)^N} + \mathbf{O}\left(\boldsymbol{e}_{\mathbf{1}}^{(k)^{N+1}}\right) \tag{6.18}$$

Therefore, from (6.18) one can observe that the developed PF solution paradigm (6.6) converges with order N, and the proof is completed. □

6.3.- Comparative analysis

In this section, a comparison of various high order Newton-like methods methodologies is addressed. To do that, the following efficiency index has been used [98]:

$$FEI = p^{1/CO} \tag{6.19}$$

where, $p \in \mathbb{R}^+$ is the order of convergence and CO stands for the total computational cost of an iteration in terms of flops. In this regard, the following theorem is generally used to estimate the cost of a LU decomposition [99].

Theorem 6.3. The number of products and quotients required for solving q linear systems of equations with the same matrix of coefficients, using LU factorization, is:

$$o(n,q) = \frac{1}{3}n^3 + qn^2 - \frac{1}{3}n \tag{6.20}$$

The total cost of each function evaluation $o(\mathbf{g})$ and Jacobian evaluation $o(\mathbf{g}')$, are also relevant calculations which can be considered as follows:

$$CO = o(\mathbf{g}) + o(\mathbf{g}') + o(n,q) \tag{6.21}$$

Theorem 6.4. For NRPC, one has $FEI = 1 + \sqrt{2}^{\,1/\frac{1}{3}n^3+3n^2+\frac{2}{3}n}$

Proof. NRPC requires 2 function evaluations along a Jacobian factorization each iteration. In addition, it requires to solve 2 linear systems using the same LU decomposition, consequently: $CO = 2n + n^2 + \frac{1}{3}n^3 + 2n^2 - \frac{1}{3}n = \frac{1}{3}n^3 + 3n^2 + \frac{2}{3}n$ □

Theorem 6.5. For the developed PF solution method (6.6), one has $FEI = N^{\,1/\frac{1}{3}n^3+Nn^2+\frac{3N-4}{3}n}$

Proof. The developed PF solution paradigm requires $N-1$ function evaluations along a Jacobian factorization each iteration. In addition, it requires to solve $N-1$ linear systems

using the same LU decomposition, consequently: $CO = (N-1)n + n^2 + \frac{1}{3}n^3 + (N-1)n^2 - \frac{1}{3}n = \frac{1}{3}n^3 + Nn^2 + \frac{3N-4}{3}n$ □

Fig. 6.2 shows the value of the efficiency index for various solvers. As observed, the developed PF solver (6.6) is computationally competitive with NRPC. On the other hand, the Darvishi's 3rd order method and the developed paradigm with $N = 3$ show the same computational cost. Superiority of the developed paradigm with respect to the other methods is achieved for $N > 3$. It is worth noting, the computational efficiency is not notably improved for $N > 5$.

6.4.- Conclusions

This Chapter is motivated in the fact that very few studies have been tackled in order to apply the high order Newton-like methods for PF analysis. With the aim at filling this gap (at least partially), two high order techniques have been studied. On the one hand, the method developed in [92] has been extended to the n-dimensional case and apply to PF analysis. On the other hand, an efficient multistep methodology has been developed.

The two introduced techniques have been compared with those high order Newton-like methods studied in [49]. As concluded, these two methodologies are more robust than NR. The developed solution paradigm is more efficient than NRPC and the most efficient high order technique for $N \geq 4$.

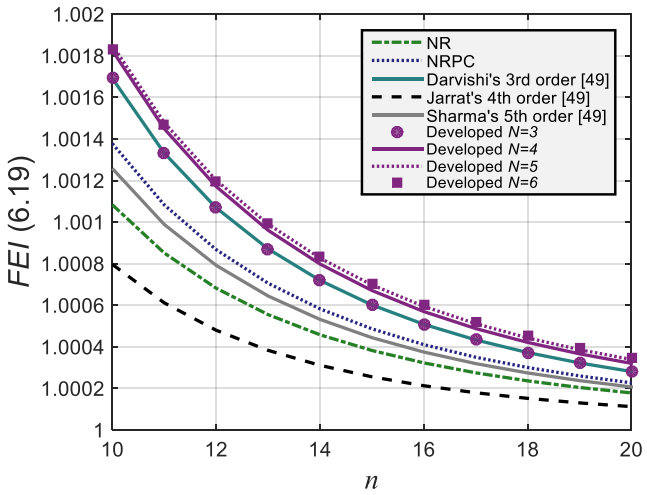


Fig. 6.2.- Efficiency index (6.19) for various PF solution methods.

Chapter 7

Numerical Results

In this Chapter, the presented PF solvers are numerically compared in various scenarios. Since one of the main objective of this work is to validate the developed methods for industry tools, realistic cases have been considered for simulations. Results are compared with the most conventional PF solver, i.e. NR.

7.1.- Simulation conditions

All tested methods have been coded in Matpower v6.0 [100] and run under Matlab R2014b on a 64-bit i5-8500F Intel Core personal computer (3 GHz, 8 GB of RAM). As it is well-known, the runtime of any computational procedure may be difficult to measure, since it strongly depends on some factors like the coding language, machine used… In order to reduce the intrinsic stochastic character of this point, the reported solution times have been calculated as the average value of 100 simulations. Finally, all the simulations have been initialized from a flat start, the required convergence tolerance is 10^{-6} and $k_{\max} = 2000$ has been taken.

7.2.- Studied systems

In order to validate the tested methodologies, the following well and ill-conditioned realistic systems have been considered:

- *case300*: the IEEE 300-bus test system [101].
- *Case2736sp*: the 2736-bus snapshot of the Polish Transmission system at summer 2004 peak [102].
- *case3012wp*: the 3012-bus snapshot of the Polish Transmission system at winter 2007-2008 evening peak [102].
- *case3375wp*: the 3374-bus snapshot of the Polish Transmission system at winter 2007-2008 evening peak [102].
- *case9241pegase*: the 9241-bus portion of the European Transmission system from the EU Pegase project [103, 104].
- *Case13659pegase*: the 13659-bus portion of the European Transmission system from the EU Pegase project [103, 104].

Some of the studied systems are well-conditioned while other may be categorized as ill-conditioned according to definition 2.1. In addition, they can be considered realistic enough to cover the targets of this work. Table 7.1 summarizes the main characteristics of the studied systems.

Along the base case scenario, the studied systems have been tested in the presence of the generators' reactive power limits and for a limit loading level. The former scenario has been considered using the strategy described in the Section 2.6. For the limit loading case, the nodal injected active and reactive powers have been modified introducing the loading level $\lambda \in \mathbb{R}^+$ as follows:

$$P_i^{sch} = \lambda P_i^{sch}, \text{for PQ and PV buses} \tag{7.1}$$

$$Q_i^{sch} = \lambda Q_i^{sch}, \text{for PQ buses} \tag{7.2}$$

Table 7.1.- Main characteristics of the studied systems

System	Buses	Branches	Generators	Load		Nº of variables (n)
				MW	MVar	
case300	300	411	69	23525.8	7788	530
case2736sp	2736	3504	420	18074.5	5339.5	5237
*case3012wp**	3012	3572	502	27169	10200	5725
*case3375wp**	3374	4161	596	48363	19527	6355
case9241pegase	9241	16049	1.445	312354.1	73581.6	17036
*case13659pegase**	13659	20467	4092	381431	98523	23225

** Ill-conditioned system*

The main aim of the limit loading scenario is to validate the considered PF solvers close to the maximum loadability point. To do that, the parameter λ is progressively increased up to the fourth decimal until all the considered methods fail. Then, the immediately lower loading level is taken for simulations. For example, $\lambda = 1.3463$ pu has been taken for the *case2383wp* since $\lambda = 1.3464$ pu gives rise divergence. Table 7.2 collects the loading levels considered in simulations.

Table 7.2.- Loading levels considered in simulations

System	λ [pu]	System	λ [pu]
case300	1.0360	*case3375wp*	1.1586
case2736sp	1.6671	*case9241pegase*	1.0767
case3012wp	1.2734	*case13659pegase*	1.0017

7.3.- Results for those methods based on the Continuous Newton's paradigm

In this Section, various results related with those solvers discussed in the Chapter 3 are presented. The following techniques are considered in this Section:

- Standard NR.
- Midpoint's rule (MP).
- Heun's 3rd order method (3OH).

- 4th order Runge-Kutta method (RK4).
- Simpson's 3/8 rule (S38).
- The accelerated 3rd order Runge-Kutta method (ARK3).
- 2nd order Adams-Bashforth's method (AB2).
- Bulirsch-Stoer algorithm (BS).
- Reverse-Bulirsch-Stoer algorithm (RBS).
- 4th order Runge-Kutta-Broyden's method (RK4B).
- Midpoint-Euler's method (MPE).
- Trapezoidal-Midpoint-Euler's method (TMPE).

The considered methods respond to the Algorithms 3.1-3.6. Table 7.3 collects the parameters employed within the considered methodologies. The tested methods are compared in the studied ill-conditioned systems.

Table 7.3.- Parameters for the Continuous Newton-based solvers considered in simulations

	MP	3OH	RK4	S38	ARK3	AB2	BS	RBS	RK4B	MPE	TMPE
Initial h	0.4	0.4	0.4	0.4	0.4	--	0.75	1	0.4	--	--
$h_{\min}$ ($H_{\min}$)	0.4	0.4	0.4	0.4	0.4	0.3	0.5	0.5	0.4	0.4	0.4
$h_{\max}$ ($H_{\max}$)	1.2	1.2	1.2	1.2	1.2	1.2	2	2	1.2	1	1
SF	10	10	10	10	10	0.003	8	8	10	--	--
σ_1	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95	--	--
σ_2	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	--	--
ρ	--	--	--	--	--	263	--	--	--	--	--
$N_{\min}$	--	--	--	--	--	--	2	2	--	--	--
$N_{\max}$	--	--	--	--	--	--	6	6	--	--	--

Table 7.4 provides the results obtained for base cases. As expected, NR failed in all studied systems. While all the studied techniques successfully solved the *case3012wp* and *case3375wp*, some of them obtained unsatisfactory results in the *case13659pegase*. Thus, RK4, AB2 and RK4B failed in the largest system, while MP, 3OH, S38 and ARK3 converged to its low voltage solution (this value is considered erroneous if one admits that the high voltage value is usually wanted). Comparing the convergence rates of the different studied methods, one can clearly check that those methods based on the multistep Continuous Newton's paradigm introduced in Section 3.6 were the most competitive. On

the other hand, AB2 usually showed the lowest convergence rate. Regarding the solution times, which measure the computational efficiency, again MPE and TMPE were the fastest methods. On the other hand, RK4B presented the lowest computational efficiency. This was expected since, as commented in Section 3.5, despite that this technique only required a LU factorization, it needs to explicitly invert the Jacobian matrix once, which supposes an important computational burden. In addition, the factorization of the Jacobian matrix is substituted by various vectors and matrix calculations which are, especially in large-scale systems, very inefficient.

Table 7.4.- Results of the Continuous Newton-based methods for base cases

Method	*case3012wp*		*case3375wp*		*case13659pegase*	
	# iter.	Time [s]	# iter.	Time [s]	# iter.	Time [s]
NR	Fail	--	Fail	--	Fail	--
MP	35	1.18	36	1.36	33*	5.21
3OH	23	1.16	23	1.28	43*	10.12
RK4	27	1.78	28	2.07	Fail	--
S38	27	1.78	28	2.07	32*	10.05
ARK3	23	0.81	23	0.90	46*	7.42
AB2	48	0.86	49	0.97	Fail	--
BS	18	1.99	19	2.35	19	10.17
RBS	14	0.82	14	0.91	15	4.13
RK4B	27	38.25	27	47.27	Fail	--
MPE	6	0.21	6	0.24	6	0.98
TMPE	5	0.18	5	0.20	5	0.82

** Low voltage solution*

Next, the ability of the tested methods with the generators' reactive limits enabled is explored. Table 7.5 collects the results obtained under this conditions. In this case, AB clearly showed worse performance, failed in all studied cases. Regarding remainder methods, although more iterations are generally required for achieving a feasible solution since various PF problems have to be solved, similar conclusions as those extracted for the base cases can be extrapolated to this scenario.

Table 7.5.- Results of the Continuous Newton-based methods with reactive limits

Method	*case3012wp*		*case3375wp*		*case13659pegase*	
	# iter.	Time [s]	# iter.	Time [s]	# iter.	Time [s]
NR	Fail	--	Fail	--	Fail	--
MP	76	2.61	87	3.33	61*	9.62
3OH	55	2.81	65	3.70	64*	15.10
RK4	63	4.23	74	5.54	Fail	--
S38	63	4.23	74	5.54	56*	17.56
ARK3	57	2.07	66	2.70	67*	10.89
AB2	Fail	--	Fail	--	Fail	--
BS	42	4.66	49	6.01	33	17.50
RBS	33	2.02	38	2.67	26	7.27
RK4B	63	95.58	73	138.55	Fail	--
MPE	10	0.36	11	0.45	8	1.30
TMPE	7	0.26	8	0.33	6	0.99

** Low voltage solution*

Finally, the tested PF solvers are analyzed in a limit load scenario, considering the loading levels collected in Table 7.2. Table 7.6 collects the results for this scenario. As in the base case, NR failed for solving all the studied systems. In this case, RKB converged to the low voltage solution in the Polish snapshots while it failed in the *case13659pegase*. RK4, AB2 and RBS also failed in the *case13659pegase* while MP and 3OH converged to the low voltage solution. On the other hand, MPE and TMPE successfully solved all the studied systems. These methods required less iterations and were faster than the remainder techniques.

Table 7.6.- Results of the Continuous Newton-based methods for limit load cases

Method	*case3012wp*		*case3375wp*		*case13659pegase*	
	# iter.	Time [s]	# iter.	Time [s]	# iter.	Time [s]
NR	Fail	--	Fail	--	Fail	--
MP	35	1.18	36	1.36	33*	5.21
3OH	23	1.16	23	1.28	24*	5.69
RK4	27	1.78	28	2.07	Fail	--
S38	27	1.78	28	2.07	48	15.08
ARK3	23	0.81	23	0.90	Fail	--
AB2	55	0.98	56	1.19	Fail	--
BS	18	1.99	19	2.35	19	10.17
RBS	14	0.82	14	0.91	Fail	--
RK4B	39*	53.95	49*	82.93	Fail	--
MPE	10	0.35	10	0.39	8	1.29
TMPE	8	0.28	8	0.31	6	0.98

** Low voltage solution*

7.4.- Results for Regularization-based methods

As commented, the regularization methods are usually quite robust, especially in the vicinity of the maximum loadability point, where these techniques avoid the singularity of the Jacobian matrix by adding some elements on its diagonal. In this section, the following conventional and Regularization-based methods are compared.

- Standard NR.
- Levenberg PF solver with adaptive damping factor [44].
- High order Levenberg-Marquardt's method (HOLM) [45].
- The developed PF solver based on Gauss-Newton's formulation (GN).

One of the most critical aspect of any regularization method, is the mechanism used for updating the damping factor. For the developed GN, the damping factor is initialized

equal to 10^3 and is updated using the equation (4.6). For the remainder Regularization-like methods this parameter is updated according the following rule [45]:

$$\alpha = \|\mathbf{g}\|_{\infty}^{1.3} \tag{7.3}$$

Since the regularization methods are usually focused on solving ill-conditioned cases, we have considered in this section the *case3012wp*, *case3375wp* and *case13659pegase*.Tables 7.7-7.9 provide results for the considered Regularization-like methods for base cases without and with reactive limits, and limit load cases, respectively. As observed, the developed GN is the most competitive method. Let me remark the results reported in Table 7.8, here, although HOLM occasionally required less iterations than GN, the former was slower due to it requires to solve two linear systems. It is remarkable the huge amount of iterations required by Levenberg and HOLM in the *case13659pegase*, which manifests the difficulties of this kind of methods when the starting guess lies far away to the solution. Fig. 7.1 shows the convergence profiles for base cases. While GN normally shows a higher a residual at first iterations, it quickly falls down to convergence. The reader should also observe the oscillatory pattern of Levenberg and HOLM in the *case13659pegase*.

Table 7.7.- Results of some Regularization-like methods for base cases

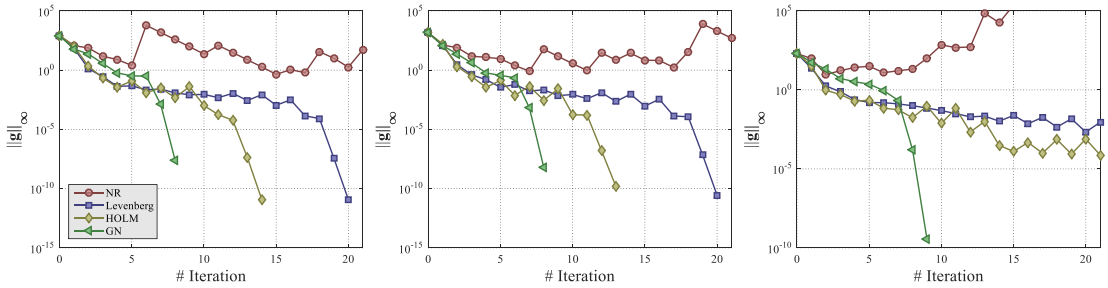
Method	*case3012wp*		*case3375wp*		*case13659pegase*	
	# iter.	Time [s]	# iter.	Time [s]	# iter.	Time [s]
NR	Fail	--	Fail	--	Fail	--
Levenberg	20	0.30	20	0.36	65	6.51
HOLM	14	0.51	13	0.52	34	6.95
GN	8	0.12	8	0.15	9	0.93

Table 7.8.- Results of some Regularization-like methods with reactive limits

Method	*case3012wp*		*case3375wp*		*case13659pegase*	
	# iter.	Time [s]	# iter.	Time [s]	# iter.	Time [s]
NR	Fail	--	Fail	--	Fail	--
Levenberg	27	0.42	28	0.52	69	6.93
HOLM	20	0.73	21	0.88	38	7.77
GN	20	0.32	25	0.45	15	1.55

Table 7.9.- Results of some Regularization-like methods for limit load cases

Method	*case3012wp*		*case3375wp*		*case13659pegase*	
	# iter.	Time [s]	# iter.	Time [s]	# iter.	Time [s]
NR	Fail	--	Fail	--	Fail	--
Levenberg	103	1.53	101	1.79	392	39.18
HOLM	48	1.67	49	1.94	177	36.10
GN	14	0.21	14	0.26	12	1.23



(a) (b) (c)

Fig. 7.1.- Convergence profiles for base cases in the case3012wp (a), case3375wp (b) and case13659pegase (c).

Other critical aspect of the Regularization-like techniques is related with the accuracy of the results. In that sense, it can be observed that GN produces better results than Levenberg and HOLM. To illustrate that, Fig. 7.2 shows the error in the angles calculated for the limit load scenario in the *case13659pegase*. In this case, the results obtained with NR using the default starting guess provided by Matpower have been considered as the correct values. As observed, both Levenberg and HOLM produced very inaccurate results.

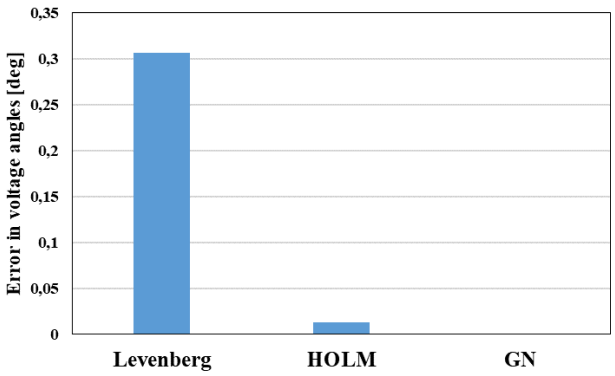


Fig. 7.2.- Error in voltage angles for the limit load scenario in the case13659pegase

7.5.- Results for robust and efficient Power Flow solvers

In Chapter 5, various robust and efficient PF solvers were presented. In this Section, these techniques are numerically compared with the conventional NR. Thus, the following methods are considered in this Section:

- Standard NR.
- Forward-Euler-homotopy method (FEH) defined by equation (5.3) and algorithm 5.1.
- Ralston-homotopy method (RH) defined by equation (5.4) and algorithm 5.1.
- Richardson Extrapolation PF solver (RE).
- Three-stage algorithm based on SIA (3S-SIA).
- A PF solver based on the Composite Newton-Cotes formulas (CNC).
- Two solvers based on the S-iteration process defined by algorithm 5.5 (SIP-NR) and algorithm 5.6 (MSIP-NR).
- Heun-King-Werner approach (HKW) for PF analysis.

The considered methods involve various parameters which are collected in Table 7.10. As robust solvers, it is suitable compared them in the considered ill-conditioned systems.

Table 7.10.- Parameters for the robust and efficient solvers considered in simulations

	FEH	RH	RE	3S-SIA	CNC	SIP-NR	MSIP-NR	HKW
Initial h	0.05	0.1	1	--	--	--	--	--
$h_{\min}$	0.05	0.1	0.75	--	0.2	--	--	0.4
$h_{\max}$	0.5	1	2	--	1.2	--	--	1
$\Delta\xi$	0.1	0.1	--	--	--	--	--	--
SF	0.3	0.3	8	--	--	--	--	--
σ_1	0.95	0.95	0.95	--	--	--	--	--
σ_2	1.05	1.05	1.05	--	--	--	--	--
ψ	--	--	8	--	--	--	--	--
ρ_0	--	--	--	0.9	--	--	--	--
ζ	--	--	--	--	0.06	--	--	0.06
$\psi_{\max}$	--	--	--	--	2	--	--	1.9
$\beta_{\min}$	--	--	--	--	2	--	--	--
Initial m	--	--	--	--	3	--	--	--
$m_{\min}$	--	--	--	--	2	--	--	--
$m_{\max}$	--	--	--	--	3	--	--	--
α	--	--	--	--	--	0.8	0.8	--
ϵ	--	--	--	--	--	--	10^{-2}	--

Tables 7.11 and 7.12 collect the results obtained with the studied robust and efficient approaches for base cases without and with reactive limits, respectively. All tested methods successfully solved all studied systems, whit the exception of FEH, which converged to the low voltage solution in the *case13659pegase*. Typically, 3S-SIA required less iterations than the remainder methods. However, those methods based on the S-iteration process (i.e. SIP-NR and MSIP-NR) were the most efficient. This is undoubtedly due to their low computational burden, one should note that SIP-NR only require a LU matrix factorization in a whole iterative procedure. However, this feature turns in its main disadvantage due to it entails linear convergence. It can be intuited in the *case13659pegase*, where SIP-NR

required many iterations. This has propitiated to MSIP-NR to be more efficient than SIP-NR.

Table 7.11.- Results of the robust and efficient methods for base cases

Method	*case3012wp*		*case3375wp*		*case13659pegase*	
	# iter.	Time [s]	# iter.	Time [s]	# iter.	Time [s]
NR	Fail	--	Fail	--	Fail	--
FEH	26	0.47	26	0.52	26*	2.13
RH	16	0.30	15	0.31	16	1.34
RE	15	0.27	15	0.31	16	1.32
3S-SIA	5	0.10	4	0.09	5	0.46
CNC	8	0.17	9	0.21	10	0.91
SIP-NR	7	0.04	7	0.04	33	0.34
MSIP-NR	6	0.06	6	0.06	13	0.27
HKW	7	0.18	7	0.20	7	0.83

** Low voltage solution*

Table 7.12.- Results of the robust and efficient methods with reactive limits

Method	*case3012wp*		*case3375wp*		*case13659pegase*	
	# iter.	Time [s]	# iter.	Time [s]	# iter.	Time [s]
NR	Fail	--	Fail	--	Fail	--
FEH	62	1.13	74	1.51	48*	3.92
RH	37	0.70	44	0.92	28	2.34
RE	37	0.68	44	0.91	29	2.38
3S-SIA	11	0.23	13	0.31	8	0.73
CNC	16	0.34	20	0.48	14	1.28
SIP-NR	11	0.09	12	0.13	36	0.44
MSIP-NR	10	0.11	11	0.15	16	0.37
HKW	13	0.37	16	0.52	10	1.25

** Low voltage solution*

Next, the performance of the studied robust methods for heavy loading conditions is explored. To do that, the studied systems for the limit loading levels reported in Table 7.2

have been considered. Results for this scenario are reported in Table 7.13. In this case, all considered approaches (except NR), successfully solved all the studied systems. 3S-SIA and CNC usually required less iteration than the other solvers. It is worth noting that SIP-NR required in this case a huge amount of iterations. This is due, as commented to its linear convergence. This feature provokes bad performance when the condition of the system is especially poor, as in the case of heavy loading conditions. Due to that, MSIP-NR was the most efficient technique.

Table 7.13.- Results of the robust and efficient methods for limit load cases

Method	*case3012wp*		*case3375wp*		*case13659pegase*	
	# iter.	Time [s]	# iter.	Time [s]	# iter.	Time [s]
NR	Fail	--	Fail	--	Fail	--
FEH	28	0.51	28	0.56	28	2.29
RH	17	0.32	17	0.35	17	1.42
RE	16	0.29	16	0.32	15	1.24
3S-SIA	8	0.16	8	0.18	7	0.63
CNC	9	0.19	9	0.21	9	0.82
SIP-NR	321	0.74	299	0.75	48	0.43
MSIP-NR	12	0.12	12	0.13	18	0.32
HKW	12	0.27	12	0.30	9	0.99

** Low voltage solution*

7.6.- Results for high order Newton-like solvers

Finally, this section is focused on the high order Newton-like methods. As commented, these kind of techniques present a convergence order higher than two, therefore, et least *a priori*, they normally require less iterations than NR for achieving a solution. The high order Newton-like methods aim at computationally outperforming NR rather than solving ill-conditioned systems. Therefore, it is suitable to validate the considered methods in the studied well-conditioned systems. Along the standard NR and the high order techniques studied in Chapter 6, those techniques considered in reference [49] are also validated. Thus, the following PF solvers have been compared in this section:

- Standard NR.

- Newton-Raphson Predictor-Corrector (NRPC) which responds to Algorithm 6.1.
- 3rd order Darvishi's method (3OD) [49].
- 4th order Jarrat's method (4OJ) [49].
- 5th order Sharma's method (5OS) [49].
- The developed multistep paradigm (6.6) with $N = 4$ (4N) and $N = 5$ (5N).

Table 7.14 shows the results obtained with the considered methods for base cases. As observed, the total iterations required is usually related with the convergence rate of the method. However, that is not true for 4OJ, which was occasionally slower than NRPC. The reason of this is observed in Fig. 7.3, where the convergence profiles for the *case300* are plotted. Here, one can observe that at first iterations 4OJ did not perform with 4th order of convergence. This is due to this technique has a strongly local convergence. Thus, its convergence properties are lost when this algorithm evolves far away to the solution. Regarding the execution time, the overall performance depends on the total factorizations required. This can be checked in Table 7.15, where the total LU factorizations required by each method is reported. As observed, 5N was the most efficient method in the *case300*, while 4N was the winner in the remainder cases. 4OJ is totally inefficient because it requires to solve a matrix system each iteration, which supposes a cost of $O(n^{3n})$. It is worth mentioning that 3OD failed in the *case2736sp* and *case9241pegase*.

Table 7.14.- Results of various high order Newton-like methods for base cases

Method	*case300*		*case2736sp*		*case9241pegase*	
	# iter.	Time [ms]	# iter.	Time [ms]	# iter.	Time [ms]
NR	5	13.44	6	102.80	6	384.73
NRPC	4	11.30	5	89.35	5	332.83
3OD	3	9.94	Fail	--	Fail	--
4OJ	4	124.67	5	18065.22	5	620744.46
5OS	2	10.84	3	104.19	3	389.62
4N	3	10.06	3	62.19	3	220.88
5N	2	8.44	3	64.49	3	223.29

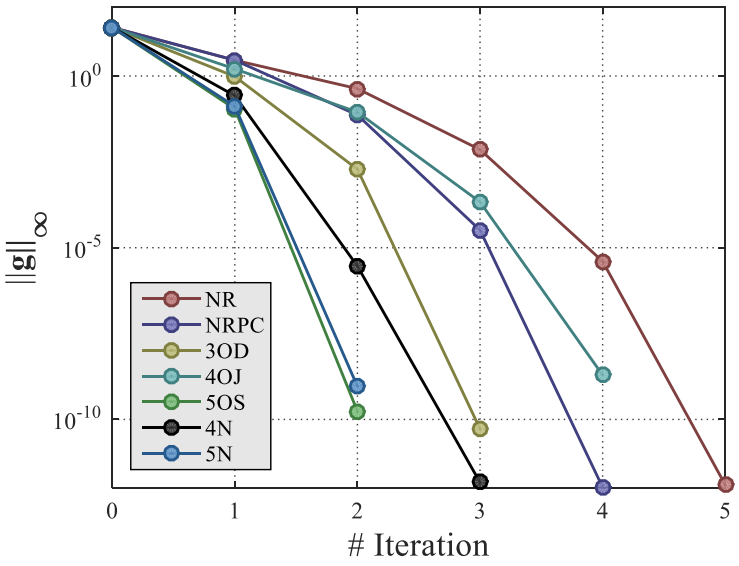


Fig. 7.3.- Convergence profiles of various high order Newton-like methods for the case300 for base cases

Table 7.15.- Total LU factorizations required by various high order Newton-like methods for base cases

Method	*case300*	*case2736sp*	*case9241pegase*
NR	5	6	6
NRPC	4	5	5
3OD	3	Fail	Fail
4OJ	8	10	10
5OS	4	6	6
4N	3	3	3
5N	2	3	3

Table 7.16 provides the results with reactive limits. In this case, 5N was the most efficient method in the *case300* while 4N outperformed the remainder methods in the *case2736sp* and *case9241pegase*. Finally, Table 7.17 collects the results for the limit load scenario. In this latter case, 3OD successfully converged in all studied systems. 4N was in this case the most efficient method.

Table 7.16.- Results of various high order Newton-like methods with reactive limits

Method	*case300*		*case2736sp*		*case9241pegase*	
	# iter.	Time [ms]	# iter.	Time [ms]	# iter.	Time [ms]
NR	11	30.04	11	203.20	13	859.54
NRPC	9	27.36	10	191.36	12	802.16
3OD	7	24.67	Fail	--	Fail	--
4OJ	10	321.16	10	37155.15	12	1453000.16
5OS	5	29.41	6	220.40	7	915.65
4N	7	24.89	6	133.59	7	524.42
5N	6	23.15	6	138.71	7	534.6

Table 7.17.- Results of various high order Newton-like methods for limit load cases

Method	*case300*		*case2736sp*		*case9241pegase*	
	# iter.	Time [ms]	# iter.	Time [ms]	# iter.	Time [ms]
NR	9	24.66	12	202.70	10	633.24
NRPC	8	19.94	9	159.14	8	522.07
3OD	7	19.09	8	148.47	7	469.99
4OJ	7	209.51	9	31817.25	8	984441.33
5OS	5	25.54	6	205.42	5	637.39
4N	5	15.61	6	118.34	5	349.62
5N	5	16.61	6	122.24	5	360.86

7.7.- Conclusions

This section has been devoted on presenting various numerical results, with the aim at validating the studied PF solvers. Several well and ill-conditioned large-scale networks have been considered for this purpose, analyzing three different scenarios. Results have been presented using the same classification followed during this Thesis. Thus, the main conclusions of this section can be summarized in the following points:

- Regarding those methodologies based on the Continuous Newton's method, the developed multistep solution paradigm clearly was the most competitive approach. On the other hand, it has been showed that some sophisticated schemes like the Bulirsch-Stoer algorithm, may occasionally show better performance than other simpler approaches like the Runge-Kutta formulas.

- Some issues related with the regularization techniques have been manifested. These kind of techniques usually showed bad performance when the starting guess is far away to the solution. In addition, the solution calculated may not be accurate enough. The developed GN seems to efficiently solve these problems.

- Various robust and efficient approaches have been studied. Among them, those techniques based on the S-iteration process look very promising. However, this kind of techniques typically suffer slow convergence due to its linear convergence rate. In this regard, some numerical combined schemes like 3S-SIA or HKW have showed promising results.

- Finally, various high order numerical methods have been compared. On the light of the results obtained, it is clearly concluded that the higher convergence order, the lower iteration number. However, one should be very careful with the local and global convergence characteristics (see results for 4OJ). In addition, a PF solver should remain efficient enough to be competitive with NR. For example, despite that 5OS exhibits very high convergence rate, there efficiency is questionable due to its high computational burden. In that sense, the developed solution paradigm (equation (6.6)), can achieve very high convergence order retaining a high degree of efficiency. This has been manifested in very competitive results.

Chapter 8

Conclusions and Future works

8.1.- Conclusions

Various PF solvers have been studied, introduced or developed in the context of this work. Some of the discussed methodologies are focused on efficiently solve ill-conditioned cases, while other methodologies are more focused on well-conditioned systems. This is important since the formers do not aim at overcoming NR from a computational point of view. The reader can check that the considered robust methods are notably less efficient than the conventional technique. This kind of methods simply aim at successfully solving ill-conditioned cases, where NR fails. On the other hand, in Chapter 6 two high-order Newton-like methods have been considered for PF analysis. These techniques should necessarily be competitive with NR, however, their performance in ill-conditioned systems is, *a priori*, irrelevant. Several numerical results in either well or ill-conditioned realistic large-scale systems have been provided, in order to show the performance of the tested methods. The main conclusions of this work can be summarized in the following points.

- It has been demonstrated that the computational performance of those methods based on Runge-Kutta formulas strongly depend on the order of the method. Thus, MP and 3OH were usually more efficient than RK4 and S38. This was expected

since this kind of techniques require as many factorizations as their order. Nevertheless, the Continuous Newton's solution paradigm has been further exploited considering other numerical arrangements like the Adams-Bashforth's methods or the Bulirsch-Stoer algorithm. In these latter cases, the developed techniques were more efficient than those methods based on the Runge-Kutta formulas. However, the best results were obtained with those methods based on the developed multistep Continuous Newton's paradigm. Finally, the Continuous Newton-Broyden's framework turned out to be very inefficient, due to the calculations involved in this developed paradigm are difficulty tractable in large-scale systems.

- The Regularization methods have manifested their ability to manage with ill-conditioned cases. However, some problems have arisen. Firstly, this kind of techniques normally suffered slow convergence when the initial guess lies far away to the solution. Secondly, the solution obtained may not be accurate enough. These drawbacks seem minimized by using the introduced PF solver based on Gauss-Newton formulation.

- With the aim at overcoming some drawbacks showed for most of available robust PF solvers in large-scale ill-conditioned systems, various robust and efficient techniques have been developed. Those methodologies based on the S-iteration process turned out to be the most competitive. However, these techniques showed very low convergence rate especially under heavy loading conditions. Oppositely, 3S-SIA showed a better trade-off between convergence rate and efficiency.

- Two high order Newton-like methods have been studied. On the one hand, a King-Werner-like method with $1+\sqrt{2}$ convergence rate has been adapted for PF analysis. On the other hand, a multistep paradigm with N^{th} order of convergence (where N is the number of steps involved) has been developed. A comparative analysis showed that both techniques are more efficient than NR. In addition, the developed multistep paradigm is more efficient than the other high order Newton-like methods for PF analysis available in the literature for $N \geq 4$. The reported results have confirmed these points.

8.2.- Future works

Several ideas and works could be tackled based on the results reported in this work. These ideas may be summarized in the following lines:

- From the results obtained, it is concluded that the standard Continuous Newton's paradigm is manifestly improvable. Thus, future works should be focused on proposing alternative solution frameworks. For example, promising lines could be introducing regularization, proposing continuous versions of different high order Newton-like methods or developing computationally cheaper schemes.

- Further developments on the regularization methods should be tackled. The adaptive mechanism for the damping factor looks a critical aspect in this kind of techniques. The accuracy and computational performance of these methods strongly depends of this mechanism. In addition, a deep analysis between the two most popular regularization methods (i.e. Lavrentiev and Tikhonov's schemes) should be made in future works. As commented, different regularization matrixes have been proposed in the literature, the impact of these matrixed in the performance of the regularization techniques should be explored. A further analysis of the vast literature available may bring novel techniques, which could show very good results in PF analysis.

- Since those techniques based on the S-iteration process showed the best performance, this kind of techniques should be further explored for PF solution in the future. For example, various methodologies based on the S-iteration process and Newton's method are available in [89, 105]. In addition, other fixed point iterations like the Mann [85] or Ishikawa's iterations [86], should be explored for PF analysis in the future.

- The studied high order Newton-like methods have reported very good results in well-conditioned systems, however, their performance in ill-conditioned cases has not been explored yet. This line should lead to develop universal solvers, able to outperform NR in well-conditioned systems and successfully solve ill-conditioned cases. For covering this point, other high order Newton-like methods available in the literature should be applied to PF analysis.

- The studied techniques should be considered for other related tools like the Continuation Power Flow [28], Optimal Power Flow or Security Analysis [106].

Appendix A

Linear Systems solution in Matlab

A.1.- Sparse matrix

Typically, a matrix is denominated sparse when the number of zero elements is proportionally much greater than the number of nonzero elements. Oppositely, a matrix is called dense when it has many nonzero elements. This concept is clearly observed in Fig. A.1 where pictorial representations of a dense and a sparse matrix are plotted. As observe, the sparse matrix has a great proportion of zero elements. It is worth mentioning that similar concept can be also applied to vectors. Sparse matrixes have several advantages with respect the dense matrixes like low memory requirements or the possibility of using cheap calculations. With the aim at getting fully advantage of their sparse structure, different particular sorting algorithms and factorizations have been developed during decades [107].

In Matlab, a sparse matrix can be built using the command *sparse(.)*. Fig. A.2 shows an example which the matrix A is converted to sparse format in Matlab.

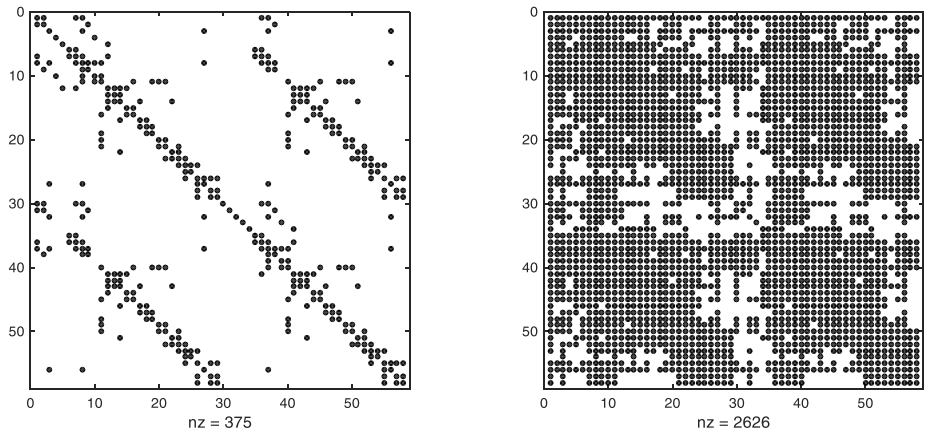


Fig. A.1 - Examples of a sparse (left) and a dense (right) matrix. As observed, although both matrixes have the same dimension, the dense matrix has much more nonzero elements (nz) than the sparse matrix

```
%% Building a sparse matrix

A = [0 0 0 0 3 0;
     3 0 0 1 0 0;
     0 0 0 0 0 4;
     0 5 0 0 0 0;
     1 1 0 0 0 0;
     0 6 0 1 0 9];

A = sparse(A);
```

*Fig. A.2 - Code for converting the matrix **A** into sparse format in Matlab*

A.2.- Sparse linear systems

Let the following linear system be considered, which the coefficient matrix $\boldsymbol{A}$ is sparse:

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{A.1}$$

The standard solution of the system (A.1) can be reached by multiplying both sides of (A.1) by the inverse of the coefficient matrix, as follows:

$$[\boldsymbol{A}]^{-1}\boldsymbol{Ax} = [\boldsymbol{A}]^{-1}\boldsymbol{b} \rightarrow \boldsymbol{x} = [\boldsymbol{A}]^{-1}\boldsymbol{b} \tag{A.2}$$

However, explicitly inverting a matrix is a very heavy calculation, which should be avoided especially in large systems. Alternatively, one can use a process called Gaussian Elimination, which the coefficient matrix is converted into a triangular matrix by elemental

modifications. These modifications, as applied also to the vector $\boldsymbol{b}$ lead to the following system:

$$\boldsymbol{U}\boldsymbol{x} = \boldsymbol{b} \tag{A.3}$$

where $\boldsymbol{U}$ is an upper triangular matrix built by elemental modifications over $\boldsymbol{A}$. The system (A.3) is much cheaper to solve than (A.1). For example, one can use the Backward Substitution technique starting by the last element of $\boldsymbol{x}$. In Matlab, the Backward Substitution can be applied using the command \, as showed in Fig. A.3.

```
%% Solving a linear system

A = [0 0 0 0 3 0;
     3 0 0 1 0 0;
     0 0 0 0 0 4;
     0 5 0 0 0 0;
     1 1 0 0 0 0;
     0 6 0 1 0 9];

b = [3 0 4 2 2 0];

A = sparse(A);

x = A\b.';
```

Fig. A.3 - Code for solving the linear system (A.1) using backward substitution in Matlab

A.3.- LU factorization

The LU factorization is simply another way to represent the Gaussian Elimination. In this case, both triangular matrixes are calculated. On the one hand, an upper triangular matrix $\boldsymbol{U}$ is computed in the same way that (A.3). On the other hand, a lower triangular matrix $\boldsymbol{L}$ is computed by storing the scalar elements used in the element transformations of $\boldsymbol{A}$ into $\boldsymbol{U}$. Formally, a third matrix $\boldsymbol{D}$ is calculated, which collects the diagonal elements of $\boldsymbol{U}$, so that the diagonal of the upper triangular matrix remain equal to 1. Due to that, the LU factorization is also known as LDU factorization. Thus, the matrix $\boldsymbol{A}$ is decomposed as follows:

$$\boldsymbol{A} = \boldsymbol{L} \cdot \boldsymbol{D} \cdot \widetilde{\boldsymbol{U}} \tag{A.4}$$

where the matrix $\widetilde{\boldsymbol{U}}$ is equal to $\boldsymbol{U}$ but with the diagonal elements equal to 1. The main merit of the LDU factorization is that it can be reused for different vectors $\boldsymbol{b}$. One should note

that this feature has important implications in some PF solvers like FDLF. By using a LDU factorization, the linear system (A.1) can be solved by computing the following steps:

1. An intermediate vector $\boldsymbol{z}$ is calculated by Forward Substitution as follows:

$$\boldsymbol{Lz} = \boldsymbol{b} \tag{A.5}$$

2. By dividing the elements of $\boldsymbol{z}$ by the diagonal elements of $\boldsymbol{D}$, the vector $\boldsymbol{y}$ is calculated as follows:

$$\boldsymbol{Dy} = \boldsymbol{z} \tag{A.6}$$

3. Finally, the unknown vector is calculated by Backward substitution as follows:

$$\boldsymbol{Ux} = \boldsymbol{y} \tag{A.7}$$

In Matlab, the LDU factorization can be computed using the command *lu(.)*. Fig. A.4 shows an example for solving the linear system (A.1) by using LDU factorization.

```
%% LDU Factorization
[L, U, P] = lu(A);
y = L\(P*b.');
x = U\y;
```

Fig. A.4 - Code for solving the linear system (A.1) using LDU factorization in Matlab

A.5.- Cholesky Factorization

The Cholesky factorization is a cheaper alternative to the LU factorization (twice as efficient as LU factorization), only applicable for Hermitian positive-definite matrixes. This kind of matrixes can be decomposed into a lower triangular matrix $\boldsymbol{L}$ and its conjugate transpose as follows:

$$\boldsymbol{A} = \boldsymbol{L} \cdot \boldsymbol{L}^{\Delta} \tag{A.8}$$

where $\boldsymbol{L}^{\Delta}$ is the conjugate transpose of $\boldsymbol{L}$. Alternatively, the LDL factorization can be established for the Cholesky factorization in an analogue manner to LDU factorization. Thus, the matrix $\boldsymbol{A}$ can be decomposed as:

$$\boldsymbol{A} = \tilde{\boldsymbol{L}} \cdot \boldsymbol{D} \cdot \tilde{\boldsymbol{L}}^{\Delta} \tag{A.9}$$

where $\tilde{\boldsymbol{L}}$ is equal to $\boldsymbol{L}$ with the diagonal elements are equal to 1, and $\boldsymbol{D}$ is a diagonal matrix with the elements of the diagonal of $\boldsymbol{L}$. By using the Cholesky factorization, the linear system (A.1) can be solved carrying out the following steps:

1. An intermediate vector $\boldsymbol{z}$ is calculated by Forward Substitution as follows:

$$\tilde{\boldsymbol{L}}\boldsymbol{z} = \boldsymbol{b} \tag{A.10}$$

2. By dividing the elements of $\boldsymbol{z}$ by the diagonal elements of $\boldsymbol{D}$, the vector $\boldsymbol{y}$ is calculated as follows:

$$\boldsymbol{D}\boldsymbol{y} = \boldsymbol{z} \tag{A.11}$$

3. Finally, the unknown vector is calculated by Backward substitution as follows:

$$\tilde{\boldsymbol{L}}^{\Delta}\boldsymbol{x} = \boldsymbol{y} \tag{A.12}$$

The Jacobian matrix of the PF problem is typically not positive-definite, so that the Cholesky factorization is generally not applicable. However, the Jacobian matrix can be converted into a positive-definite form by carrying out the product $[\mathbf{g}'(\boldsymbol{x})]^{*}\mathbf{g}'(\boldsymbol{x})$. If one shall to apply this approach to NR, the mapping (2.5) should be converted into:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \left[\left[\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right)\right]^{*}\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right)\right]^{-1}\left[\mathbf{g}'\left(\boldsymbol{x}^{(k)}\right)\right]^{*}\mathbf{g}\left(\boldsymbol{x}^{(k)}\right) \tag{A.13}$$

The computational savings offered by the Cholesky factorization on NR, are normally not attractive since the matrix $[\mathbf{g}'(\boldsymbol{x})]^{*}\mathbf{g}'(\boldsymbol{x})$ is much denser than $\mathbf{g}'(\boldsymbol{x})$. In addition, an extra matrix-matrix product and an extra matrix-vector product have to be computed. In Matlab, the Cholesky factorization can be computed using the command *chol(.)* Fig. A.5 shows an example for solving the system (A.1) using Cholesky factorization.

```
%% Cholesky Factorization
% Suppose a matrix A positive-definite
L = chol(A);
x = L\(L.'\b);
```

Fig. A.5 - Code for solving the linear system (A.1) using Cholesky factorization in Matlab. If the matrix $\boldsymbol{A}$ is not positive-definite, the command chol(.) will fail

Appendix B

Curriculum Vitae

Name: Marcos Tostado Véliz

Birth day: August 1, 1987

Nationality: Spanish

Education and qualifications:

- **Graduate Degree:** Electrical Engineering by the University of Seville in 2016 (with honors) with an average grade of 8.32.
- **Master's Degree:** Electrical Power Systems (Sistemas de Energía Eléctrica) by the University of Seville in 2016 with an average grade of 9.2.

Experience

- May - June 2019: Academic visitor, Institute for Systems and Computer Engineering, Technology and Science (INESC TEC), Porto, Portugal.
- Nov. 2017 - Until now: PhD student, Electrical Engineering Department, University of Jaén, Spain.

Honors and Awards

- 2016 Real Maestranza de Caballería de Sevilla: *Academic Award.*
- 2016 University of Seville: *Academic Award.*
- 2016 Seville Town Hall: *Academic Award.*

Appendix C

Thesis Publications

As result of the contributions of this Thesis, the following papers have been published or accepted for publication in international journals.

- M. Tostado-Véliz, S. Kamel and F. Jurado, "Development of combined Runge-Kutta Broyden's load flow approach for well and ill-conditioned power systems," *IET Gener. Transmiss. Distrib.*, vol. 12, no. 21, pp. 5723-5729, Nov. 2018.
- M. Tostado, S. Kamel and F. Jurado, "Developed Newton-Raphson based predictor-corrector load flow approach with high convergence rate," *Int. J. Elect. Power Energy Syst.*, vol. 105, pp. 785-792, Feb. 2019.
- M. Tostado-Véliz, S. Kamel and F. Jurado, "Development of different Load Flow methods for solving Large-scale ill-conditioned Systems," *Int. Trans. Elect. Energy Syst.*, Apr. 2019. DOI: 10.1002/etep.2784
- M. Tostado-Véliz, S. Kamel and F. Jurado, "A robust Power Flow Algorithm Based on Bulirsch-Stoer Method," *IEEE Trans. Power Syst.*, vol. 34, no. 4, pp. 3081-3089, Jul. 2019.

- M. Tostado-Véliz, S. Kamel and F. Jurado, "Comparison of various robust and efficient load-flow techniques based on Runge–Kutta formulas," *Elect. Power Syst. Res.*, vol. 174, Sept. 2019, 105881. DOI: 10.1016/j.epsr.2019.105881
- M. Tostado, S. Kamel and F. Jurado, "Several robust and efficient load flow techniques based on combined approach for ill-conditioned power systems," *Int. J. Elect. Power Energ. Syst.*, vol. 110, pp. 349-356, Sept. 2019.
- M. Tostado-Véliz, S. Kamel and F. Jurado, "Robust and efficient approach based on Richardson extrapolation for solving badly initialised/ill-conditioned power-flow problems," *IET Gener. Transmiss. Distrib.*, vol. 13, no. 16, pp. 3524-3533, Aug. 2019.
- M. Tostado, S. Kamel and F. Jurado, "An effective load-flow approach based on Gauss-Newton formulation," *Int. J. Elect. Power Energy Syst.*, vol. 113, pp. 573-581, Dec. 2019.
- M. Tostado-Véliz, S. Kamel and F. Jurado, "Promising framework based on multistep continuous Newton scheme for developing robust PF methods," *IET Gener. Transmiss. Distrib.*, vol. 14, no. 2, pp. 265-274, Jan. 2020.
- M. Tostado-Véliz, S. Kamel, T. Alquthami and F. Jurado, "A Three-Stage Algorithm Based on a Semi-Implicit Approach for Solving the Power-Flow in Realistic Large-Scale ill-Conditioned Systems," *IEEE Access*, vol. 8, pp. 35299-35307, 2020.
- M. Tostado-Véliz, S. Kamel and F. Jurado, "A powerful power-flow method based on Composite Newton-Cotes formula for ill-conditioned power systems," *Int. J. Elect. Power Energy Syst.*, vol. 116, Mar. 2020, 105558. DOI: 10.1016/j.ijepes.2019.105558
- M. Tostado-Véliz, S. Kamel and F. Jurado, "An efficient power-flow approach based on Heun and King-Werner's methods for solving both well and ill-conditioned cases," *Int. J. Elect. Power Energy Syst.*, vol. 119, July. 2020, 105869. DOI: doi.org/10.1016/j.ijepes.2020.105869

- M. Tostado-Véliz, S. Kamel and F. Jurado, "A novel family of efficient Power-Flow methods with high convergence rate suitable for large realistic power systems," *IEEE Syst. J.*, 2020. DOI: 10.1109/JSYST.2020.2980156.
- M. Tostado-Véliz, S. Kamel and F. Jurado, "Power Flow Approach based on the S-iteration Process," *IEEE Trans. Power Syst.*, 2020. DOI: 10.1109/TPWRS.2020.2989270.

Bibliography

[1] L. Powell, *Power System Load Flow Analysis*. New York: McGraw-Hill, 2005.

[2] J. B. Ward and H. W. Hale, “Digital computer solution of power-flow problems,” *Trans. Amer. Inst. Elect. Eng. Part III, Power App. Syst.*, vol. 75, no. 3, pp. 398-404, Jan. 1956.

[3] W. F. Tinney and C. E. Hart, “Power Flow Solution by Newton's Method,” *IEEE Trans. Power Appar. Syst.*, vol. PAS-86, no. 11, pp. 1449-1460, Nov. 1967.

[4] B. Stott and O. Alsac, “Fast Decoupled Load Flow,” *IEEE Trans. Power Appar. Syst.*, vol. PAS-93, no. 3, pp. 859-869, May 1974.

[5] R. A. M. van Amerongen, “A general-purpose version of the fast decoupled load flow,” *IEEE Trans. Power Syst.*, vol. 4, no. 2, pp. 760-770, May 1989.

[6] L. Wang and X. R. Lin, “Robust fast decoupled power flow,” *IEEE Trans. Power Syst.*, vol. 15, no. 1, pp. 208-215, Feb. 2000.

[7] M. S. Sachdev and T. K. P. Medicherla, “A second order load flow technique,” *IEEE Trans. Power Appar. Syst.*, vol. 96, no. 1, pp. 189-197, Jan. 1977.

[8] P. S. Nagendra Rao, K. S. Prakasa Rao and J. Nanda, “An Exact Fast Load Flow Method Including Second Order Terms in Rectangular Coordinates,” *IEEE Trans. Power Appar. Syst.*, vol. PAS-101, no. 9, pp. 3261-3268, Sept. 1982.

[9] S. M. Fatemi, S. Abedi, G. B. Gharehpetian, S. H. Hosseinian and M. Abedi, “Introducing a Novel DC Power Flow Method With Reactive Power Considerations,” *IEEE Trans. Power Syst.*, vol. 30, no. 6, pp. 3012-3023, Nov. 2015.

[10] J. Yang, N. Zhang, C. Kang and Q. Xia, “A State-Independent Linear Power Flow Model With Accurate Estimation of Voltage Magnitude,” *IEEE Trans. Power Syst.*, vol. 32, no. 5, pp. 3607-3617, Sept. 2017.

[11] A. Semlyen, "Fundamental concepts of a Krylov subspace power flow methodology," *IEEE Trans. Power Syst.*, vol. 11, no. 3, pp. 1528-1537, Aug. 1996.

[12] M. Karimi, A. Shahriari, M. R. Aghamohammadi, H. Marzooghi and V. Terzija, "Application of Newton-based load flow methods for determining steady- state condition of well and ill-conditioned power systems: A review," *Int. J. Elect. Power Energy Syst.*, vol. 113, pp. 298-309, Dec. 2019.

[13] V. M. da Costa, N. Martins and J. L. R. Pereira, "Developments in the Newton Raphson power flow formulation based on current injections," *IEEE Trans. Power Syst.*, vol. 14, no. 4, pp. 1320-1326, Nov. 1999.

[14] P. A. N. Garcia, J. L. R. Pereira, S. Carneiro, M. P. Vinagre and F. V. Gomes, "Improvements in the representation of PV buses on three-phase distribution power flow," *IEEE Trans. Power Delivery*, vol. 19, no. 2, pp. 894-896, April 2004.

[15] S. Kamel, M. Abdel-Akher and F. Jurado, "Improved NR current injection load flow using power mismatch representation of PV bus," *Int. J. Elect. Power Energy Syst.*, vol. 53, pp. 64-68, Dec. 2013.

[16] S. A. Saleh, "The Formulation of a Power Flow Using *d-q* Reference Frame Components-Part I: Balanced 3φ Systems," *IEEE Trans. Ind. Appl.*, vol. 52, no. 5, pp. 3682-3693, Sept.-Oct. 2016.

[17] S. A. Saleh, "The Formulation of a Power Flow Using *d-q* Reference Frame Components-Part II: Unbalanced 3φ Systems," *IEEE Trans. Ind. Appl.*, vol. 54, no. 2, pp. 1092-1107, March-April 2018.

[18] P. A. N. Garcia, J. L. R. Pereira, S. Carneiro, V. M. da Costa and N. Martins, "Three-phase power flow calculations using the current injection method," *IEEE Trans. Power Syst.*, vol. 15, no. 2, pp. 508-514, May 2000.

[19] D. R. R. Penido, L. R. de Araujo, S. Carneiro, J. L. R. Pereira and P. A. N. Garcia, "Three-Phase Power Flow Based on Four-Conductor Current Injection Method for Unbalanced Distribution Networks," *IEEE Trans. Power Syst.*, vol. 23, no. 2, pp. 494-503, May 2008.

[20] D. R. R. Penido, L. R. de Araujo, S. Carneiro and J. L. R. Pereira, "A new tool for multiphase electrical systems analysis based on current injection method," *Int. J. Elect. Power Energy Syst.*, vol. 44, no. 1, pp. 410-420, Jan. 2013.

[21] R. Pires, L. Mili and G. Chagas, "Robust complex-valued Levenberg-Marquardt algorithm as applied to power flow analysis," *Int. J. Elect. Power Energy Syst.*, vol. 113, pp. 383-392, Dec. 2019.

[22] A. M. Sasson, C. Trevino and F. Aboytes, "Improved Newton's Load Flow Through a Minimization Technique," *IEEE Trans. Power Appar. Syst.*, vol. PAS-90, no. 5, pp. 1974-1981, Sept. 1971.

[23] S. Iwamoto and Y. Tamura, "A Load Flow Calculation Method for Ill-Conditioned Power Systems," *IEEE Trans. Power Appar. Syst.*, vol. PAS-100, no. 4, pp. 1736-1743, April 1981.

[24] L. M. C. Braz, C. A. Castro, and C. A. F. Murari, "A critical evaluation of step size optimization based load flow methods," *IEEE Trans. Power Syst.*, vol. 15, no. 1, pp. 202-207, Feb. 2000.

[25] J. E. Tate and T. J. Overbye, "A comparison of the optimal multiplier in polar and rectangular coordinates," *IEEE Trans. Power Syst.*, vol. 20, no. 4, pp. 1667–1674, Nov. 2005.

[26] A. Shahriari, H. Mokhlis, M. Karimi, A.H.A. Bakar and H.A. Illias, "Quadratic Discriminant Index for Optimal Multiplier Load Flow Method in ill conditioned system," *Int. J. Elect. Power Energy Syst.*, vol. 60, pp. 378-388, Sept. 2014.

[27] F. Milano, *Power System Modelling and Scripting*. New York: Springer, 2010.

[28] V. Ajjarapu and C. Christy, "The continuation power flow: a tool for steady state voltage stability analysis," *IEEE Trans. Power Syst.*, vol. 7, no. 1, pp. 416-423, Feb. 1992.

[29] D. A. Alves, L. C. P. da Silva, C. A. Castro and V. F. da Costa, "Continuation fast decoupled power flow with secant predictor," *IEEE Trans. Power Syst.*, vol. 18, no. 3, pp. 1078-1085, Aug. 2003.

[30] H. Sheng and H. Chiang, "CDFLOW: A Practical Tool for Tracing Stationary Behaviors of General Distribution Networks," *IEEE Trans. Power Syst.*, vol. 29, no. 3, pp. 1365-1371, May 2014.

[31] X. Yang and X. Zhou, "Application of asymptotic numerical method with homotopy techniques to power flow problems," *Int. J. Elect. Power Energy Syst.*, vol. 57, pp. 375-383, May 2014.

[32] S. Rao, Y. Feng, D. J. Tylavsky and M. K. Subramanian, "The Holomorphic Embedding Method Applied to the Power-Flow Problem," *IEEE Trans. Power Syst.*, vol. 31, no. 5, pp. 3816-3828, Sept. 2016.

[33] H. Chiang, T. Wang and H. Sheng, "A Novel Fast and Flexible Holomorphic Embedding Power Flow Method," *IEEE Trans. Power Syst.*, vol. 33, no. 3, pp. 2551-2562, May 2018.

[34] D. Wu and B. Wang, "Holomorphic Embedding Based Continuation Method for Identifying Multiple Power Flow Solutions," *IEEE Access*, vol. 7, pp. 86843-86853, 2019.

[35] T. O. Ting, K. P. Wong and C. Y. Chung, "Hybrid constrained genetic algorithm/particle swarm optimisation load flow algorithm," *IET Gener. Transmiss. Distrib.*, vol. 2, no. 6, pp. 800-812, Nov. 2008.

[36] K. Gnanambal, N.S. Marimuthu and C.K. Babulal, "Three-phase power flow analysis in sequence component frame using Hybrid Particle Swarm Optimization," *Appl. Soft Comput.*, vol. 11, no. 2, pp. 1727-1734, March 2011.

[37] E. Davoodi, M.T. Hagh and S.G. Zadeh, "A hybrid Improved Quantum-behaved Particle Swarm Optimization–Simplex method (IQPSOS) to solve power system load flow problems," *Appl. Soft Comput.*, vol. 21, pp. 171-179, Aug. 2014.

[38] J.W. Neuberger, "Continuous Newton's Method," in *Sobolev Gradients and Differential Equations. Lecture Notes in Mathematics*, vol. 1670, J.W. Neuberger Ed. Berlin, Heidelberg: Springer, 2010.

[39] F. Milano, "Continuous Newton's Method for Power Flow Analysis," *IEEE Trans. Power Syst.*, vol. 24, no. 1, pp. 50-57, Feb. 2009.

[40] F. Milano, "Implicit Continuous Newton Method for Power Flow Analysis," *IEEE Trans. Power Syst.*, vol. 34, no. 4, pp. 3309-3311, July 2019.

[41] N. Xie, F. Torelli, E. Bompard and A. Vaccaro, "Dynamic computing paradigm for comprehensive power flow analysis," *IET Gener. Transmiss. Distrib.*, vol. 7, no. 8, pp. 832-842, Aug. 2013.

[42] N. Xie, E. Bompard, R. Napoli and F. Torelli, "Widely convergent method for finding solutions of simultaneous nonlinear equations," *Elect. Power Syst. Res.*, vol. 83, no. 1, pp. 9-18, Feb. 2012.

[43] F. Torelli and A. Vaccaro, "A second order dynamic power flow model," *Elect. Power Syst. Res.*, vol. 126, pp. 12-20, Sept. 2015.

[44] F. Milano, "Analogy and Convergence of Levenberg's and Lyapunov-Based Methods for Power Flow Analysis," *IEEE Trans. Power Syst.*, vol. 31, no. 2, pp. 1663-1664, March 2016.

[45] R. Pourbagher and S. Y. Derakhshandeh, "Application of high-order Levenberg–Marquardt method for solving the power flow problem in the ill-conditioned systems," *IET Gener. Transmiss. Distrib.*, vol. 10, no. 12, pp. 3017-3022, Sept. 2016.

[46] R. Pourbagher and S. Y. Derakhshandeh, "A powerful method for solving the power flow problem in the ill-conditioned systems," *Int. J. Elect. Power Energy Syst.*, vol. 94, pp. 88-96, Jan. 2018.

[47] S. Y. Derakhshandeh, R. Pourbagher and A. Kargar, "A novel fuzzy logic Levenberg-Marquardt method to solve the ill-conditioned power flow problem," *Int. J. Elect. Power Energy Syst.*, vol. 99, pp. 299-308, July 2018.

[48] K. Tang, S. Dong, J. Shen, C. Zhu and Y. Song, "A Robust and Efficient Two-Stage Algorithm for Power Flow Calculation of Large-Scale Systems," *IEEE Trans. Power Syst.*, vol. 34, no. 6, pp. 5012-5022, Nov. 2019.

[49] S. Y. Derakhshandeh and R. Pourbagher, "Application of high-order Newton-like methods to solve power flow equations," *IET Gener. Transmiss. Distrib.*, vol. 10, no. 8, pp. 1853-1859, May 2016.

[50] C. Chun and B. Neta, "Developing high order methods for the solution of systems of nonlinear equations," *Appl. Math Comput.*, vol. 342, pp. 178-190, Feb. 2019.

[51] Y. Zhang, Y. Zhang, B. Wu and J. Zhou, "Power injection model of STATCOM with control and operating limit for power flow and voltage stability analysis," *Elect. Power Syst. Res.*, vol. 76, no. 12, pp. 1003-1010, Aug. 2006.

[52] P. Zhu, G. Taylor andM. Irving, "Performance analysis of a novel Q-limit guided continuation power flow method," *IET Gener. Transmiss. Distrib.*, vol. 3, no. 2, pp. 1042-1051, Dec. 2009.

[53] J.C. Butcher, *Numerical Methods for Ordinary Differential Equations*, 2nd ed. Chichester, UK: Wiley, 2008.

[54] G. Dahlquist and A. Björck. *Numerical Methods in Scientific Computing*, vol 1. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics (SIAM), 2008.

[55] R. Bulirsch and J. Stoer, "Numerical treatment of ordinary differential equations by extrapolation methods," *Numer. Math.*, vol. 8, no. 1, pp. 1-13, 1966.

[56] S. Weerakoon and T.G.I. Fernando, "A variant of Newton's method with accelerated third-order convergence," *Appl. Math. Lett.*, vol. 13, no. 8, pp. 87-93, Nov. 2000.

[57] P. Phohomsiri and F.E. Udwadia, "Acceleration of Runge-Kutta integration schemes," *Discrete Dyn. Nature Soc.*, vol. 2, pp. 307-314, Sept. 2004.

[58] L.F. Richardson and J.A. Gaunt, "The deferred approach to the limit," *Philos. Trans. Roy. Soc. A*, vol. 226, no. 636-646, pp. 299-349, Jan. 1927.

[59] W. B. Gragg, "On extrapolation algorithms for ordinary initial value problems," *J. Soc. Ind. Appl. Math.: Ser. B, Numer. Anal.*, vol. 2, no. 3, pp. 384-403, 1965.

[60] W.H. Press, S.A. Teukolsky, W.T. Vetterling and B.P. Flannery. *Numerical Recipes, 3rd ed*. Cambridge, U.K.: Cambridge Univ. Press, 2007.

[61] G. Bader and P. Deuflhard, "A semi-implicit mid-point rule for stiff systems of ordinary differential equations," *Numer. Math.*, vol. 41, no. 3, pp. 373-398, 1983.

[62] P. Deuflhard, "Recent Progress in Extrapolation Methods for Ordinary Differential Equations," *SIAM Rev.*, vol. 27, no. 4, pp. 505-535, 1985.

[63] K.R. Geldhof, T.J. Vyncke, F.m.l.l.D., Belie, L. Vandevelde, J.A.A. Melkebeek and R.K. Boel, "Embedded Runge-Kutta methods for the integration of a current control loop in an SRM dynamic finite element model," *IET Sci. Meas. Tech.*, vol. 1, no. 1, pp. 17-20, Jan. 2007.

[64] J.R. Dormand, P.J. Prince, "A family of embedded Runge-Kutta formulae", *J. Comput. Appl. Math.*, vol. 6, no. 1, pp. 19-26, Mar. 1980.

[65] I.K. Argyros and S. George, "Iterative regularization methods for nonlinear ill-posed operator equations with m-accretive mappings in banach spaces," *Acta Math. Scientia*, vol. 35, no. 6, pp. 1318-1324, Nov. 2015.

[66] I.K. Argyros and S. Hilout. *Computational Methods in Nonlinear Analysis*, New Jersey: World Scientific Public Company, 2013.

[67] M.L.N. Gonçalves, "Inexact Gauss-Newton like methods for injective-overdetermined systems of equations under a majorant condition," *Numer. Algorithms*, vol. 72, pp. 377-392, 2016.

[68] G. Huang, S. Noschese and L. Reichel, "Regularization matrices determined by matrix nearness problems," *Linear Algebr. Appl.*, vol. 502, pp. 41-57, Aug. 2016.

[69] L. Dykes, G. Huang, S. Noschese and L. Reichel, "Regularization matrices for discrete ill-posed problems in several space dimensions," *Numer. Linear Algebr.*, vol. 25, no. 4, Aug. 2018, Art no. e2163.

[70] F. Milano, "Analogy and Convergence of Levenberg's and Lyapunov-Based Methods for Power Flow Analysis," *IEEE Trans. Power Syst.*, vol. 31, no. 2, pp. 1663-1664, Mar. 2016.

[71] R. Seydel, *Practical Bifurcation and Stability Analysis: From Equilibrium to Chaos*. New-York: Springer-Verlag, 1994.

[72] J. Lee and H.-D. Chiang, "Convergent regions of the Newton homotopy method for nonlinear systems: theory and computational applications," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 48, no. 1, pp. 51-66, Jan. 2001.

[73] J. Grainger and W. Stevenson, *Power System Analysis*. New York: McGraw- Hill; 2004.

[74] P. Lynch, "Richardson Extrapolation: The Power of the 2-gon," *Math. Today*, vol. 39, no. 5, pp. 159-160, Jan. 2003.

[75] J. Scheffel, C. Håkansson, "Solution of systems of nonlinear equations - a semi-implicit approach," Appl. Numer. Math., vol. 59, pp. 2430-2443, Oct. 2009.

[76] I.K. Argyros and S. George, "Regularization Methods for Ill-Posed Problems with Monotone Nonlinear Part," *Punjab Univ. J. Math.*, vol. 46, no. 1, pp. 25-38, 2014.

[77] D.K.R. Babajee, M.Z. Dauhoo, M.T. Darvishi, A. Karami and A. Barati, "Analysis of two Chebyshev-like third order methods free from second derivatives for solving systems of nonlinear equations," *J. Comput. Appl. Math.*, vol. 233, no. 8, pp. 2002-2012, Feb. 2010.

[78] C.S. Liu, "A Dynamical Tikhonov Regularization for Solving Ill-posed Linear Algebraic Systems," *Acta Applicandae Mathematicae*, vol. 123, pp. 285-307, 2003.

[79] J.A. Ezquerro and M.A. Hernández, "An optimization of Chebyshev's method," *J. Complex.*, vol. 25, no. 4, pp. 343-361, Aug. 2009.

[80] M.S. Petkovic, B. Neta, L.D. Petkovic and J. Dzunic, *Multipoint Methods for Solving Nonlinear Equations*. Cambridge, MA, USA: Academic Press; 2013.

[81] W.H. Press, S.A. Teukolsky, T.H. Vetterling and B.P Flannery, *Numerical Recipes: The art of scientific computing*, 3rd ed. Cambridge, UK: Cambridge University Press; 2007.

[82] M.T. Darvishi and A. Barati, "A fourth-order method from quadrature formulae to solve systems of nonlinear equations," *Appl. Math. Comput.*, vol. 188, no. 1, pp. 257-261, May 2007.

[83] M.A. Noor and M. Waseem, "Some iterative methods for solving a system of nonlinear equations," *Comput. Math. Appl.*, vol. 57, no. 1, pp. 101-106, Jan. 2009.

[84] M.A. Krasnoselskii, "Two remarks on the method of successive approximations," *Uspekhi Mat. Nauk*, vol. 10, no. 1(63), pp. 123-127, 1955.

[85] W.R. Mann, "Mean value methods in iterations," *Proc. Amer. Math. Soc.*, vol. 4, pp. 506-510, June 1953.

[86] S. Ishikawa, "Fixed points by a new iteration method," *Proc. Amer. Math. Soc.*, vol. 44, no. 1, pp. 147-150, May 1974.

[87] R. P. Argawal, D. O'Regan and D. R. Sahu, "Iterative Construction of Fixed Points of Nearly Asymptotically Nonexpansive Mappings," *J. Nonlinear Convex Analysis*, vol. 8, no. 1, pp. 61-79, 2007.

[88] D. R. Sahu, "Applications of the S-iteration process to constrained minimization problems and split feasibility problems," *Fixed Point Th.*, vol. 12, no. 1, pp. 187-204, 2011.

[89] D. R. Sahu, K. K. Singh and V. P. Singh, "Some Newton-like methods with sharper error estimates for solving operator equations in Banach spaces," *Fixed Points Th. Appl.*, no. 78, 2012.

[90] R.F. King, "Tangent methods for nonlinear equations," *Numerische Mathematik*, vol. 18, pp. 298-304, Aug. 1971.

[91] L. Chen and Y. Ma, "A new modified King-Werner method for solving nonlinear equations," *Comput. Math. Appl.*, vol. 62, no. 10, pp. 3700-3705, Nov. 2011.

[92] T.J. McDougall and S.J. Wotherspoon, "A simple modification of Newton's method to achieve convergence of order $1+\sqrt{2}$," *Appl. Math. Lett.*, vol. 29, pp. 20-25, Mar. 2014.

[93] M. Frontini and E. Sormani, "Some variant of Newton's method with third-order convergence," Appl. Math. Comput., vol. 140, no. 2-3, pp. 419-426, Aug. 2003.

[94] A.Y. Özban, "Some New Variants of Newton's Method," Appl. Math. Lett., vol. 17, no. 6, pp. 677-682, June 2004.

[95] M. Frontini and E. Sormani, "Third-order methods from quadrature formulae for solving systems of nonlinear equations," Appl. Math. Comput., vol. 149, no. 3, pp. 771-782, Feb. 2004.

[96] A. Cordero and J.R. Torregrosa, "Variants of Newton's method for functions of several variables," Appl. Math. Comput., vol. 183, no. 1, pp. 199-208, Dec. 2006.

[97] J.F. Traub, *Iterative Methods for the Solution of Equations*. New York: Chelsea; 1982.

[98] T. Lotfi, P. Bakhtiari, A. Cordero, K. Mahdiani and J.R. Torregrosa, "Some new efficient multipoint iterative methods for solving nonlinear systems of equations," *Int. J. Comput. Math*., vol. 92, no. 9, pp. 1921-1934, Aug. 2015.

[99] A. Cordero, J.L. Hueso, E. Martínez and J.R. Torregrosa, "A modified Newton-Jarratt's composition," *Numer. Algor*., vol. 5, no. 1, pp. 87-99, Sept. 2010.

[100] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "Matpower: Steady-State Operations, Planning and Analysis Tools for Power Systems Research and Education," *IEEE Trans. Power Syst*., vol. 26, no. 1, pp. 12-19, Feb. 2011.

[101] A.B. Birchfield, T. Xu, K.M. Gegner, K.S. Shetye and T.J. Overbye, "Grid Structural Characteristics as Validation Criteria for Synthetic Networks," *IEEE Trans. Power Syst*., vol. 32, no. 4, pp. 3258-3265, July 2017.

[102] [online]. Available: http://www.pserc.cornell.edu/matpower/ [Accessed 5 Feb. 2020]

[103] C. Josz, S. Fliscounakis, J. Maeght and P. Panciatici, "AC Power Flow Data in Matpower and QCQP Format: iTesla, RTE Snapshots, and PEGASE," 2016, arXiv:1603.01533. [Online]. Available: https://arxiv.org/abs/1603.01533 [v3 Accessed 5 Feb. 2020]

[104] S. Fliscounakis, P. Panciatici, F. Capitanescu and L. Wehenkel, "Contingency Ranking With Respect to Overloads in Very Large Power Systems Taking Into Account Uncertainty, Preventive, and Corrective Actions," *IEEE Trans. Power Syst.*, vol. 28, no. 4, pp. 4909-4917, Nov. 2013.

[105] V. Karakaya, K. Dogan, Y. Atalan and N.E.H. Bouzara, "The local and semilocal convergence analysis of new Newton-like iteration methods," *Turkish J. Math.*, vol. 42, pp. 735-751, 2018.

[106] Z. Shen, H.-D. Chiang, Y. Tang and N. Zhou "An online line switching methodology with look-ahead capability to alleviate power system overloads based on a three-stage strategy," *Int. J. Elect. Power Energy Syst.*, vol. 115, Feb. 2020, 105500

[107] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics; 2003.