POLITECNICO DI TORINO

MASTER'S Degree in Electronic Engineering

MASTER's Degree Thesis

Fast Transient Analysis of System-Level Power Delivery Networks

Supervisors

Prof. Stefano GRIVET TALOCIA

Ph.D. Antonio CARLUCCI

Candidate

Alessandro GIAMBATTISTA

December 2024

Summary

Maintaining power integrity (PI) is one of the critical challenges in the analysis and simulation of modern systems, especially those operating at high performance, like AI. Power integrity verification refers to the ability of a power delivery network to provide stable, noise-free voltage to sensitive circuits. Instabilities in PI can lead to circuit malfunctions, signal distortion, and reduced overall performance. This thesis addresses the challenge of simulating linear systems subject to time-varying loads, which complicates PI analysis and makes direct simulation methods impractical. The main objective was to develop a novel simulation technique that reuses a transient solver based on implicit Euler for linear systems, to handle nonlinear loads that introduce feedback. The elimination of this feedback is achieved through an iterative method. Initially, the Waveform Relaxation method was employed but due to convergence issues, the Picard iteration method was adopted as a more robust and effective solution.

By using the Picard iteration method, this work successfully overcame the challenges introduced by time-varying loads, accelerating convergence and maintain good accuracy. A key contribution of this research was the introduction of the control parameter, g_{shift}, provide by Picard iteration method. This parameter enabled a systematic shift of the system descriptor of the Power Delivery Network, optimizing convergence. Through a sweep analysis, an optimal value of g_{shift} was identified, reducing the number of iterations required to achieve a desired level of accuracy.

The study highlighted that poorly conditioned system matrices in large-scale configurations presented a significant challenge during iterations, often leading to numerical errors and slow convergence. However the introducing g_{shift} significantly improved performance.

For istance, in a 2-core system configuration, the use of the shift parameter allowed us to achieve considerable accuracy with respect to the reference in just two iterations. Additionaly, in a 60-core system configuration, the use of g_{shift} led to a two-order magnitude reduction in error by the fourth iteration compared to the case without shifting, with noticeable improvements already observed in the initial iterations. This underscores the importance of optimizing the shift parameter to enhance simulation efficiency in complex systems.

These results demonstrate how the Picard iterative method can notably enhance the transient analysis of multicore systems in the presence of time-varying loads. This approach improves both speed and accuracy, overcoming the issues associated with direct methods, where time-varying loads cause slowdowns in the analysis of the PDN

In conclusion, this thesis presents an optimized iterative method that accelerates convergence while ensuring accurate and reliable results for the simulation of complex linear systems under time-varying load.

Acknowledgements

Sono passati ormai sei anni da quando ho intrapreso il mio percorso al Politecnico di Torino, e ancora oggi mi sembra difficile credere che questo capitolo così importante della mia vita stia per concludersi. Sei anni di esperienze, di crescita, di sfide, ma anche di enormi soddisfazioni. In questo lungo periodo, ho vissuto una moltitudine di emozioni contrastanti: la gioia di raggiungere traguardi significativi, la tristezza nei momenti più difficili, la rabbia davanti agli ostacoli che sembravano insormontabili e la frustrazione durante le fasi di incertezza. Tuttavia, ogni difficoltà ha contribuito a farmi crescere, facendomi scoprire una forza che non pensavo di avere.

Guardando indietro, mi rendo conto di come ogni esperienza, ogni sfida e anche ogni fallimento mi abbia insegnato qualcosa di essenziale. In ogni momento, sia nei successi che nelle difficoltà, ho trovato occasioni di riflessione che mi hanno permesso di evolvermi, diventando la persona che sono oggi. Questi sei anni non sono stati solo un percorso accademico, ma un viaggio di crescita e scoperta che ha inciso profondamente sulla mia vita, sul mio modo di pensare e di affrontare le sfide. Sebbene ogni giorno sia stato segnato da alti e bassi, posso dire con certezza che ogni esperienza ha avuto il suo valore e che oggi sono una persona migliore grazie ad esse.

In primo luogo, desidero esprimere la mia più profonda gratitudine alla mia famiglia, che ha sempre creduto in me e mi ha supportato nei momenti più difficili. Senza il loro amore, il loro sostegno e l'incoraggiamento continuo, non sarei mai riuscito a percorrere questo cammino con la forza e la determinazione che mi hanno sempre trasmesso.

Un grazie di cuore va anche ai miei amici di vecchia data e ai colleghi universitari, che mi hanno accompagnato in questo viaggio. Non dimenticherò mai il giorno in cui ho messo piede per la prima volta al Politecnico di Torino, una delle università più prestigiose in Italia e nel mondo. È stato un periodo ricco di sfide, esami, traguardi, ma anche di incontri speciali con persone che, pur seguendo percorsi diversi, hanno arricchito la mia esperienza universitaria in modi unici e indimenticabili.

In secondo luogo, desidero esprimere la mia sincera gratitudine al mio relatore, Prof. Stefano Grivet Talocia, e al dottorando Antonio Carlucci, per la loro pazienza, il supporto prezioso e i suggerimenti fondamentali che hanno contribuito in modo determinante alla realizzazione di questo lavoro. La loro esperienza, disponibilità e competenza sono state essenziali per il buon esito di questa tesi.

Un ringraziamento speciale va alla mia fidanzata, che è stata la persona con cui ho condiviso le mie incertezze, le mie paure e i miei sogni. Grazie per essere sempre al mio fianco, per avermi dato forza nei momenti più difficili e per avermi sostenuto in ogni passo di questo lungo percorso.

Infine, desidero dedicare questa tesi a mio padre colui che ha sempre creduto in me fin dal primo giorno e mi ha incoraggiato nei momenti di incertezza investendo tempo ed energie per vedermi crescere e raggiungere i miei obiettivi. Purtroppo, mio padre non potrà essere presente al traguardo di questa tesi, ma il suo amore, la sua guida e il suo esempio mi hanno accompagnato in ogni passo del mio cammino. Questa tesi è per lui, che mi ha insegnato il valore del duro lavoro, della perseveranza e della dedizione.

Table of Contents

List of Tables

List of Figures

Acronyms

FIVR
Fully Integrated Voltage Regulators

KCL
Kirchhoff's Current Law

MNA
Modified Nodal Analysis

MOR
Model Order Reduction

NA
Nodal Analysis

ODE
Ordinary Differential Equations

PDN
Power Delivery Network

SPI
Signal and Power Integrity

WR
Waveform Relaxation

Chapter 1

Introduction

1.1 Motivation and Objectives

As microprocessor power levels continue to rise due to the evolution of multicore architectures, significant challenges arise regarding stable and reliable power delivery. The increasing complexity and number of cores in these processors necessitate robust Power Delivery Networks (PDNs) that effectively distribute energy while maintaining system performance and integrity. Fully Integrated Voltage Regulators (FIVRs) are now a key component in power delivery systems. These devices, often implemented as multi-phase buck converters, dynamically regulate the voltage supplied to individual cores by adjusting their duty cycle based on feedback mechanisms, sensing the instantaneous output voltage, and comparing it with a voltage reference. However, analyzing the transient behavior of such systems introduces several difficulties due to the interplay between system size and nonlinearity. The structure of a typical PDN is illustrated in Fig. 1.1, where N_C cores are supplied by FIVRs distributed across N_P phases per core. Each regulator works to stabilize the power delivered to its respective core, and it is noticeable that the input network is shared from all cores, which furthermore increases the necessity to maintain power integrity due to the coupled noise present in each core. This task becomes increasingly complex as the number of cores grows. On the one hand, the size of the network introduces a significant challenge: describing every component in detail becomes computationally expensive and impractical. On the other hand, the presence of nonlinear components exacerbates the problem. Simulators like SPICE must continuously update their calculations to capture dynamic behavior accurately, which involves refactoring the Modified Nodal Analysis (MNA) matrix at each time step—a computational operation that can significantly slow down the analysis process. The goal of this work is to compute the voltages in a PDN model derived from a specific extraction procedure. This procedure involves formulating

the equations of the model, reducing its complexity using model-order reduction techniques, and subsequently linearizing it to obtain the final descriptor system. It has been observed that the direct method is very slow due to the presence of a time-varying term, which represents the activity of the chip. For this reason, an iterative method is proposed, where the time-varying term is kept constant during each iteration, and its variability is recovered through the iterative process.

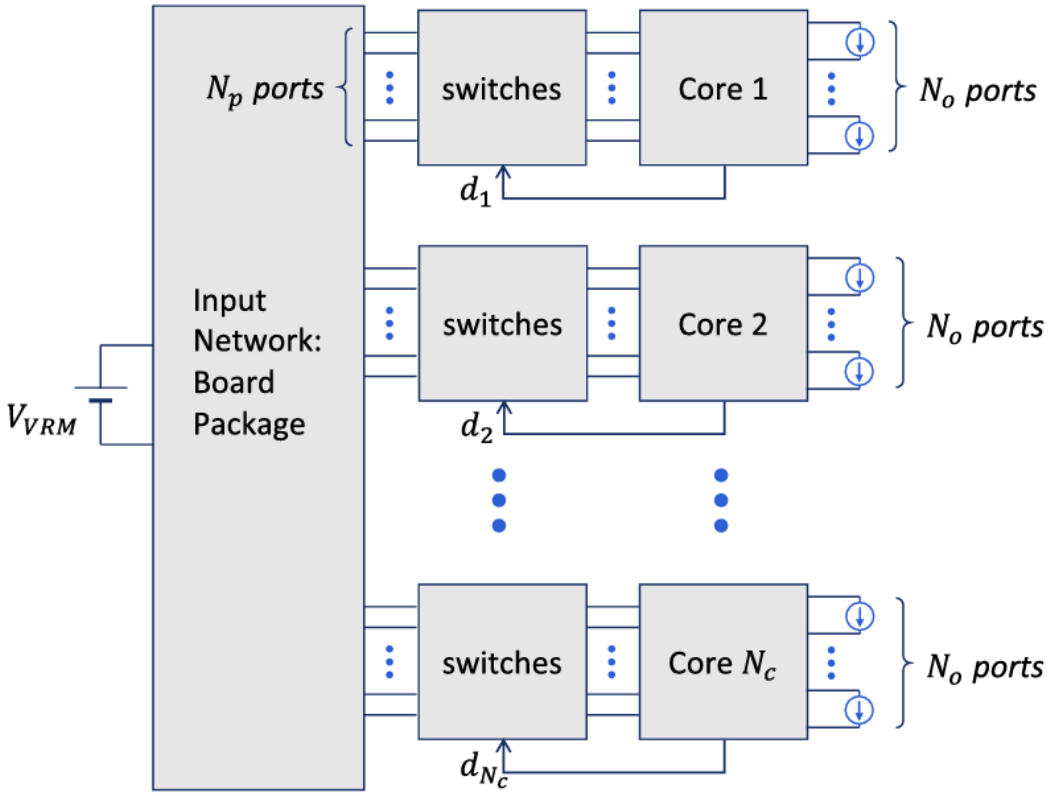


Figure 1.1: Multicore structure of a PDN [3]

The proposed approach is implemented in MATLAB and is designed to handle increasingly complex PDN configurations, scaling up to systems with as many as 60 cores. To achieve this, two iterative methods were adopted:

- **Waveform Relaxation (WR)**: This method was initially applied to simpler PDN models, but for some types of networks WR encountered issues with divergence, primarily due to unintended resonances.
- **Picard Method**: This is an alternative method introduced to overcome the limitations of WR. This method provides an iterative approach that takes

advantage of the error correction at each step to refine the solution until the desired convergence is reached. This method allowed us to introduce a control parameter to optimize convergence speed and accuracy.

As simulations progressed to more complex scenarios, managing numerical errors and addressing the increasing complexity of the PDN matrices, which were often poorly conditioned, became challenging. Preconditioning techniques were therefore applied to mitigate these issues and ensure sufficient accuracy.
This work not only demonstrates the potential of these methods for accelerating PDN verification in multicore systems but could also contribute to validating methods that could be used in future microprocessor designs.

1.2 Structure of the Document

This thesis is structured to help the reader from the theoretical foundations to the practical implementations and results of the method developed, offering insights into current challenges and future directions in PDN analysis.

- Chapter 2 gives an overview of the background on techniques used to improve and simplify the analysis of power integrity in power delivery networks, along with the challenges encountered in today's technological context.
- Chapter 3 explores the methods used in transient analysis, focusing on the Waveform Relaxation and Picard methods. It discusses their theoretical foundations and explains how these techniques are adapted to address specific challenges encountered in PDN simulations.
- Chapter 4 presents the results obtained from applying these methods, with a focus on accuracy and time analysis. It also discusses the implications of these findings and their potential impact on future research.
- In conclusion, Chapter 5 summarizes the key findings of the thesis and outlines potential directions for future work. It particularly focuses on enhancing the performance of the Picard method and exploring its applicability to even larger and more complex microprocessor systems.

Chapter 2

Background and Problem statement

2.1 Introduction

As microprocessors shift toward multicore architectures to serve the needs of AI and other emerging fields, power distribution network (PDN) analysis plays a more critical role in modern chip design. With the evolution of technology, power levels are increasing more, and as multicore platforms are most prevalent in the growing applications, such as High-Performance Computing (HPC) and Artificial Intelligence (AI) that use >100 cores. This trend gives rise to an increasing demand for robust voltage control techniques due to the potential failure of systems caused by changes in the supply voltages supplied to the power system. Hence, power integrity must be verified through transient analysis to ensure chips operate reliably.

2.2 Power Delivery Network

The **Power Delivery Networks (PDNs)** are critical architectures in integrated circuits, designed to efficiently and reliably supply power to various components, such as microprocessors or Integrated Circuits. As modern microprocessors become increasingly complex, requiring a steady power supply across numerous cores, the precise design of these systems has become essential. The effective operation of PDNs directly impacts the efficiency and stability of the entire device, ensuring that voltage levels remain within acceptable operational margins. This is where the concept of **Power Integrity** (PI) comes into play. In a real system every connection, material, and layout decision introduces unwanted effects—such as parasitic resistances, inductances, and capacitances— that can alter the balance

of power delivery. Understanding and addressing these issues is fundamental to designing a PDN that can meet the demands of modern computing.

2.2.1 Structure of the PDN

In a typical structure of a PDN depicted in Fig. 2.1 on one end, there is a power source—like a battery or a DC/DC converter—providing a theoretically stable voltage. On the other end lies a microprocessor or integrated circuit that draws current through interconnections on the PCB. However, this interconnection—the

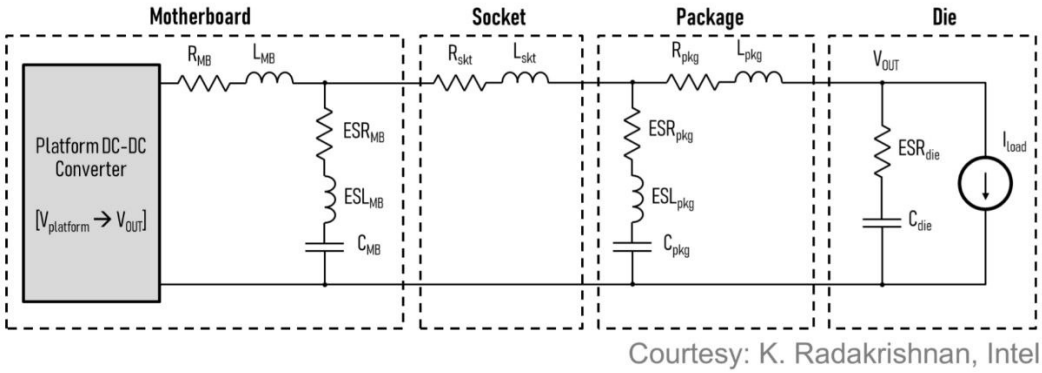


Figure 2.1: Typical PDN structure [7]

PDN—is far from perfect. Parasitic resistances in conductive paths lead to energy losses and voltage drops. Although these losses can have significant consequences, such as localized heating or reduced efficiency. Designers often mitigate these effects by using wide copper planes instead of wires, which reduces resistance and improves conductivity.
Similarly, parasitic inductances arise from loops formed by current flow. Larger loops result in higher inductance, which can lead to sudden voltage spikes during rapid changes in current demand. To address this issue, it is essential to keep current return paths close to outgoing paths; however, the size and layout of the system impose limitations.
An academic example showing the effect of different parasitic components on the power integrity is reported in figure 2.2.

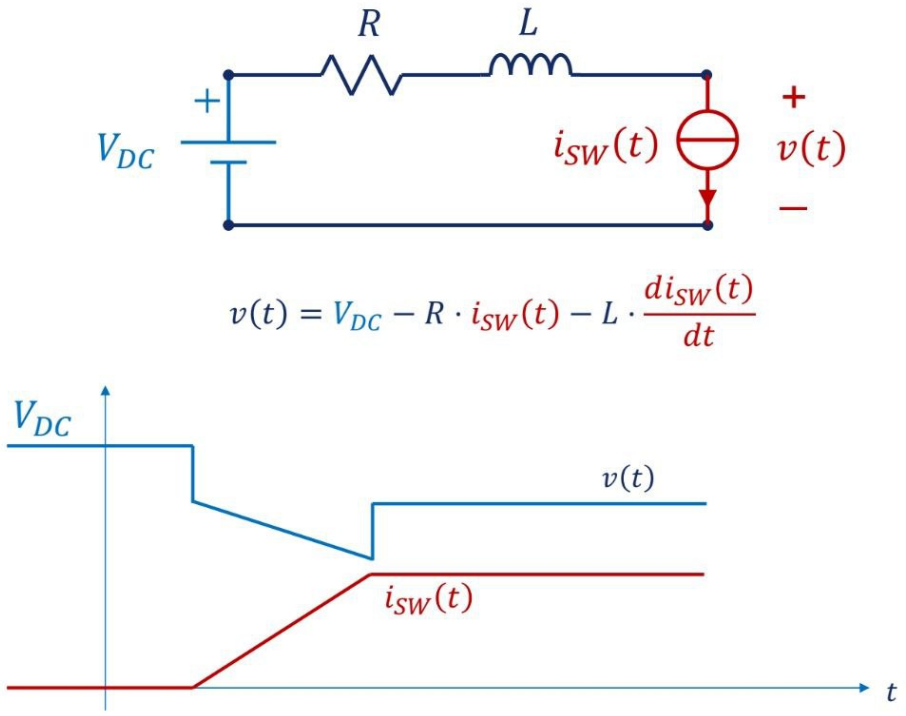


Figure 2.2: Effect of the inductance in the PDN

It is possible to notice by the equation how the derivative of the current can impact the level of the voltage compromising the correct behavior of the system. On the other hand, parasitic capacitances between power and ground planes can be beneficial as they help stabilize voltage levels. The closer and larger these planes are, the better their stabilizing effect becomes. Yet when all these parasitics interact, they create a PDN that deviates significantly from the expected idealized system. Instead of smooth, steady power delivery, fluctuating voltages, and ripple effects are encountered. This is where Power Integrity becomes crucial. PI focuses on managing these imperfections to ensure that power delivered to a circuit remains stable under demanding conditions. For instance, in a CMOS inverter, excessive fluctuations in supply voltage can slow down transistor switching speeds, leading to increased propagation delays and degraded signal transitions, see figure 2.3, where V_c is defined as $V_C = V_{DD} - V_{SS}$. The overall performance of the system can suffer significantly, especially in high-speed applications where timing is critical.

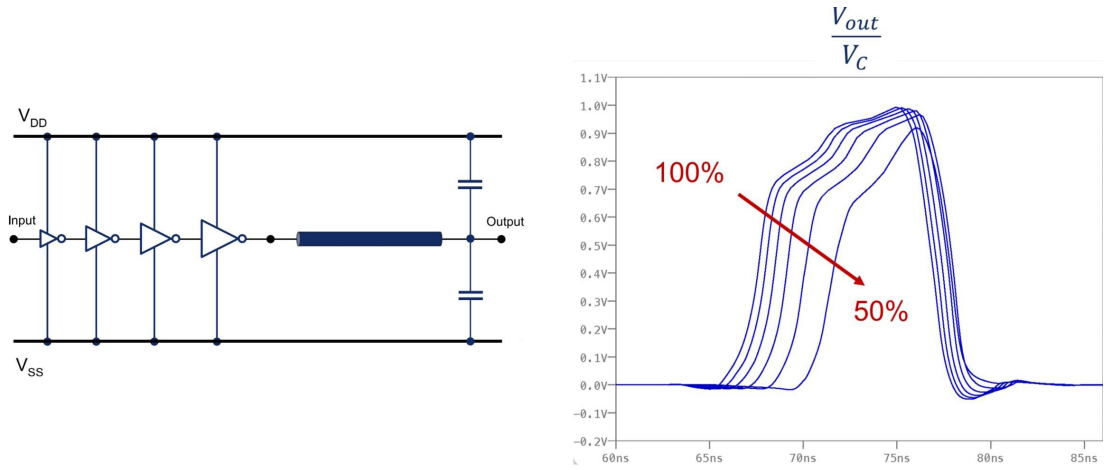


Figure 2.3: Degradation of the performance in cmos inverters [7]

2.2.2 PDN Impedance

During the experiments, WR and Picard methods were tested using the outlined PDNs before moving on to more complex microprocessor PDNs. One of the most crucial aspects of managing a PDN is its impedance, which significantly affects the stability and efficiency of power delivery to the processor cores. If impedance is not effectively controlled, power delivery can become inconsistent—resulting in voltage fluctuations or resonance effects that severely degrade system performance. Such issues can lead to power integrity challenges manifested as voltage dips that may even cause system failures. Therefore, understanding and managing PDN impedance is essential for ensuring reliable operation.
In terms of frequency response, voltage ripple is determined by multiplying the current drawn by the microprocessor and Z_{PDN}, leading to convolution between them over time. A significant challenge within PI is resonance that causes impedance to fluctuate rather than remain constant. To mitigate this problem, decoupling capacitors are often employed strategically within the design. By effectively exploiting their parasitic characteristics, designers can reduce issues related to impedance variability, see figure 2.4.

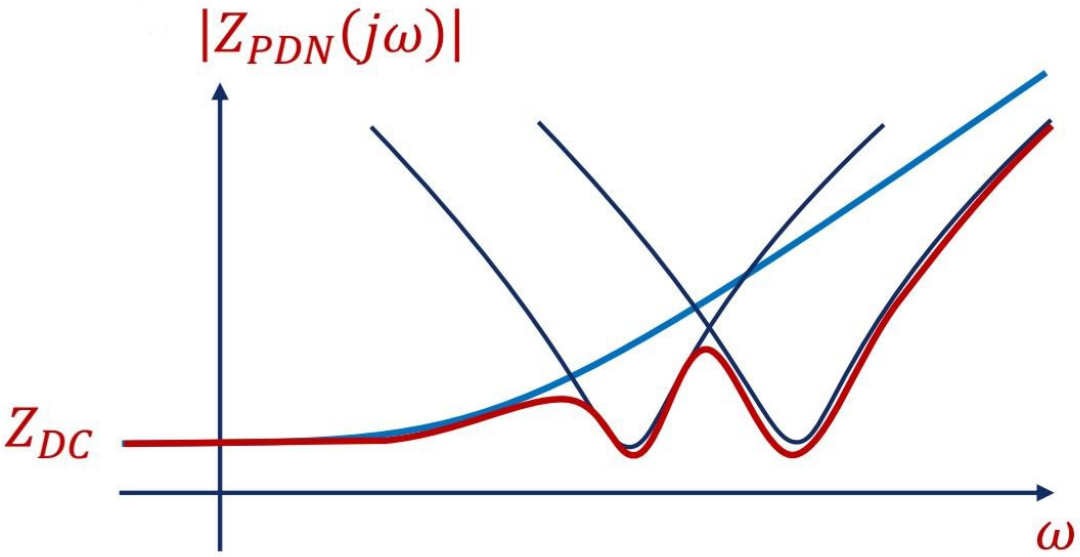


Figure 2.4: Behavior of decoupling capacitor

Looking at the graph in figure 2.4, it can be observed that initially, the impedance starts in a constant region where the PDN is purely resistive. As frequency increases, it begins to exhibit inductive behavior. By introducing two decoupling capacitors placed in parallel with the parasitic elements of the PDN, this trend has been dampened, resulting in a more linear response over a wider bandwidth.
To validate our simulation approach, graphs illustrating the admittance characteristics of various PDN configurations are presented. Each network demonstrates unique behaviors, with different degrees of resonance at both low and high frequencies. The graphs below highlight these behaviors, showcasing the inductive and capacitive elements that define the PDNs.
These admittances are defined as the admittance seen by our microprocessor, which, as shown in the graphs below, is modeled as a time-varying conductance. This conductance represents the chip's activity and is the reason why the direct method is not suitable. By analyzing these characteristics, it is possible to anticipate and address performance degradation in real-world applications.

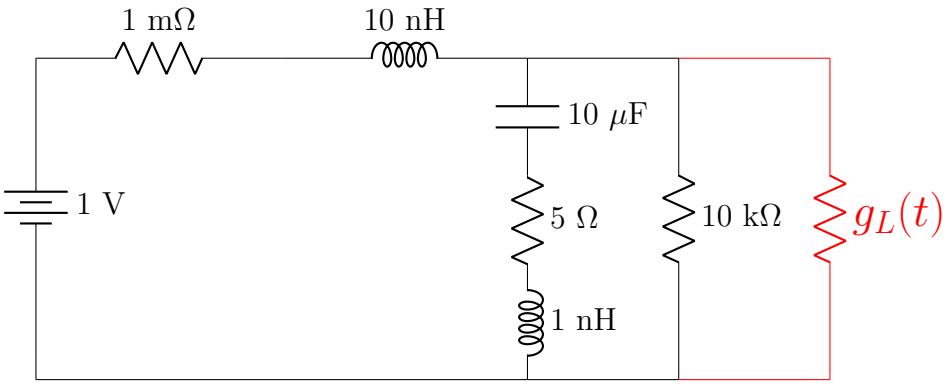


Figure 2.5: Simplified PDN circuit

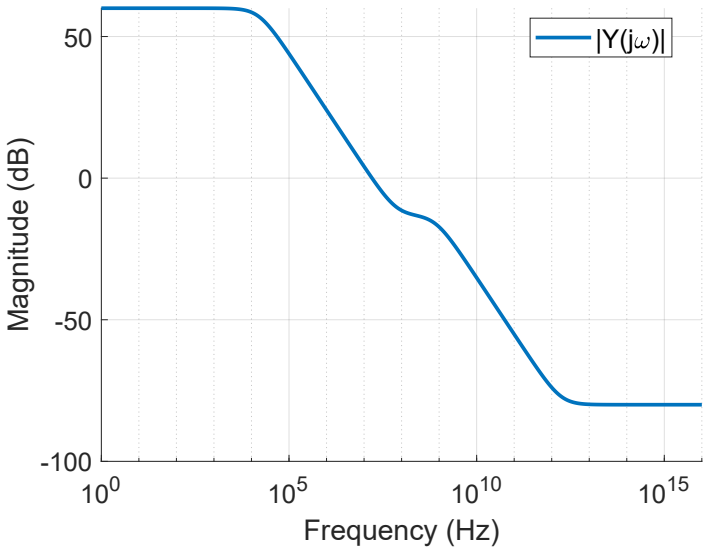


Figure 2.6: Admittance of the Simplified PDN

The next phase of our study involved moving from this simplified PDN to more detailed models. In these more detailed PDNs, more complex interactions among various components, including capacitors, inductors, and power sources are observed. As illustrated in the graphs below, the admittance characteristics of these networks become increasingly detailed, presenting additional challenges for maintaining power integrity in high-speed circuits.

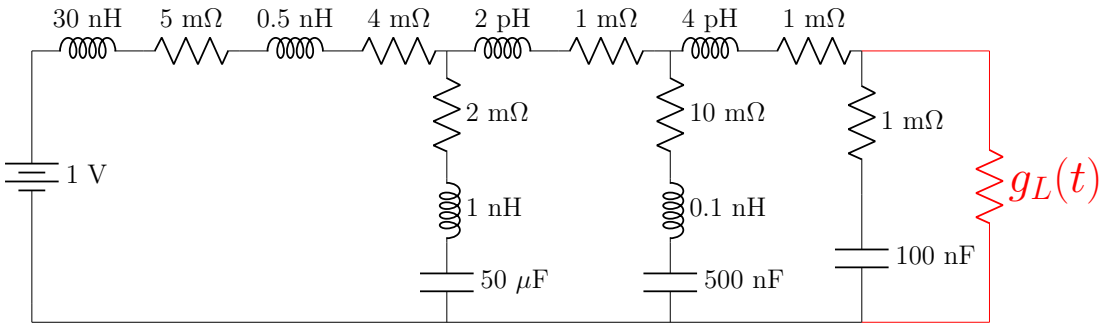


Figure 2.7: Full PDN circuit

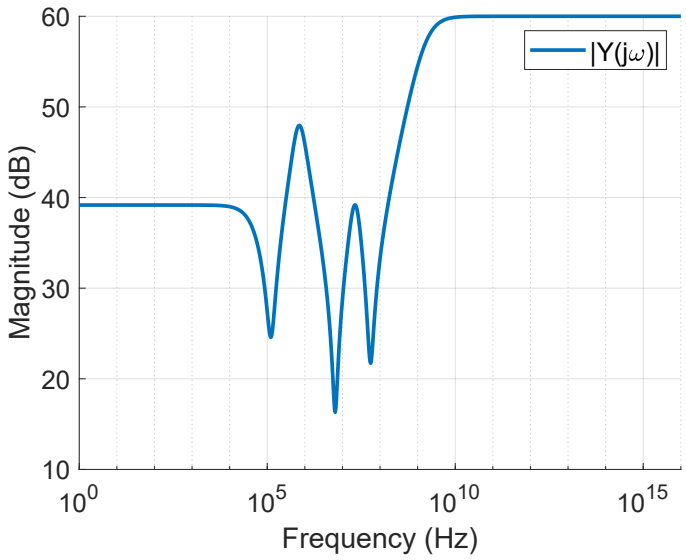


Figure 2.8: Admittance of the Full PDN

By adding a decoupling capacitors it is possible to enhance voltage stability by offering localized charge storage. This improvement contributes to the overall performance of the PDN by reducing fluctuations in power delivery. However, as demonstrated in the upcoming set of graphs, it is essential to carefully analyze the effect of these capacitors on impedance characteristics to ensure that they do not create new resonances at critical frequencies.

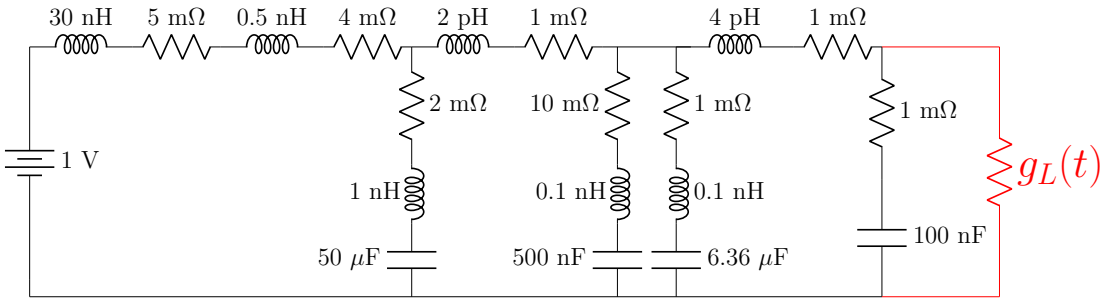


Figure 2.9: Full PDN cricuit with additional decoupling capacitors

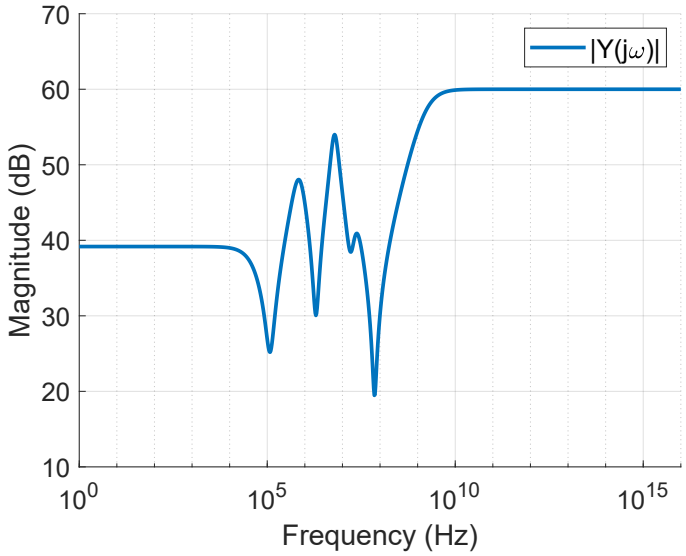


Figure 2.10: Admittance of the full PDN with additional decoupling capacitor

Conversely, a PDN that did not include decoupling capacitors was also analyzed. As anticipated, the absence of these capacitors resulted in less stable voltage levels, as reflected in the impedance and admittance graphs. This scenario highlights the critical role of decoupling capacitors in reducing noise and facilitating smooth power delivery to the processor cores.

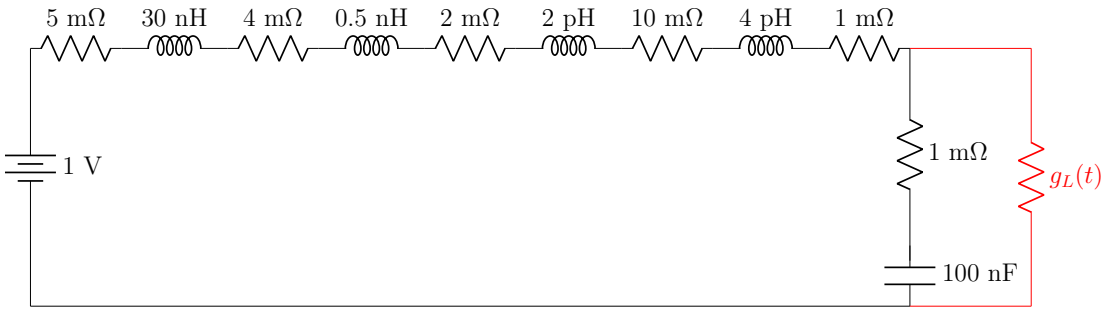


Figure 2.11: PDN Circuit without decoupling capacitors

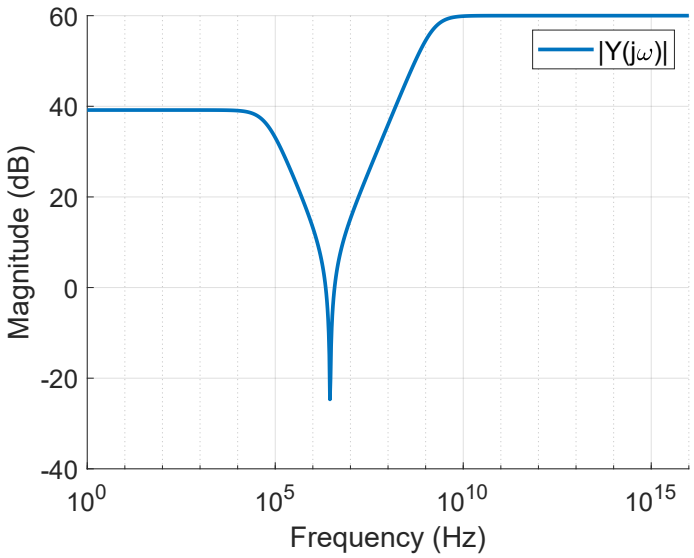


Figure 2.12: Admittance of the PDN without decoupling capacitors

2.2.3 PDN derivation

The Power Distribution Networks discussed above serve as the starting point for simulating the proposed method and verifying its accuracy. However, it is important to understand how they are derived from a structure, as depicted in Fig. 2.13.

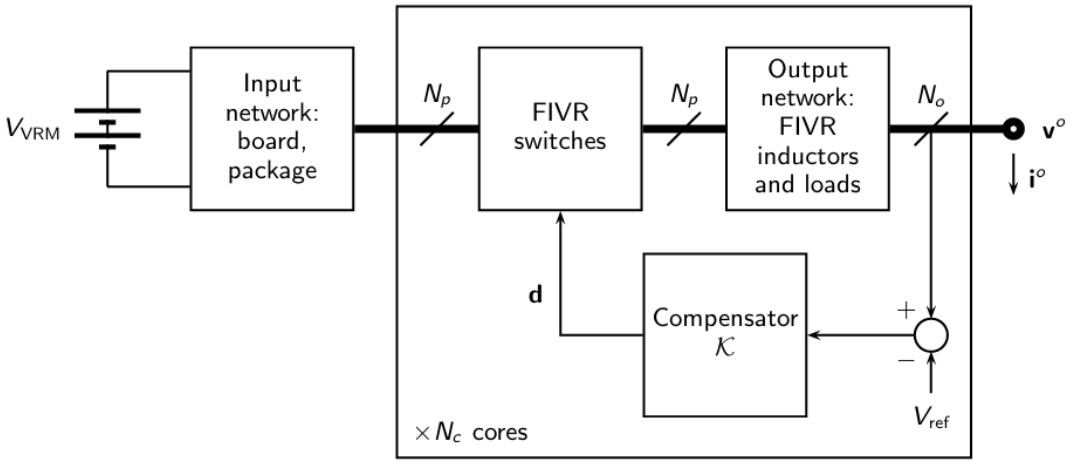


Figure 2.13: Multicore structure detailed [1]

The procedure to extract the descriptor system of the Power Delivery Networks is already documented in the article [1], below are reported the steps proposed by the authors.
Initially, the FIVR switches are represented by an ideal transformer that connects the input network to the output network for each core. The overall PDN system can be represented as follows:

$$\dot{\boldsymbol{x}} = \mathbf{A}\boldsymbol{x} + \mathbf{B}_w\boldsymbol{w} + \mathbf{B}_u\boldsymbol{u} \tag{1a}$$
$$\boldsymbol{z} = \mathbf{C}_z\boldsymbol{x} + \mathbf{D}_{zw}\boldsymbol{w} + \mathbf{D}_{zu}\boldsymbol{u} \tag{1b}$$
$$\boldsymbol{y} = \mathbf{C}_y\boldsymbol{x} + \mathbf{D}_{yw}\boldsymbol{w} + \mathbf{D}_{yu}\boldsymbol{u} \tag{1c}$$
$$\boldsymbol{w} = \Delta(\boldsymbol{d})\boldsymbol{z} \tag{1d}$$
$$\boldsymbol{i}^o(t) = \alpha(I_{\text{leak}})(\boldsymbol{v}^o)^3 + \boldsymbol{g}_L(t)\boldsymbol{v}^o \tag{1e}$$
$$\boldsymbol{x}_k = \mathbf{A}_k\boldsymbol{x}_k + \mathbf{B}_k(\boldsymbol{N}\boldsymbol{y} - \boldsymbol{V}_{\text{ref}}), \quad \boldsymbol{d} = \mathbf{C}_k x_k \tag{1f}$$

Where $\boldsymbol{v}_1$ and $\boldsymbol{i}_1$ are the vectors collecting all the voltages and currents at the $N_C N_P$ ports of the input network, and $\boldsymbol{v}_2$ and $\boldsymbol{i}_2$ correspond to the output networks. Next, introducing:

$$\boldsymbol{w} \triangleq \begin{pmatrix} \boldsymbol{i}_1 \\ \boldsymbol{v}_2 \end{pmatrix}, \quad \boldsymbol{z} \triangleq \begin{pmatrix} \boldsymbol{v}_1 \\ \boldsymbol{i}_2 \end{pmatrix}, \quad \Delta(\boldsymbol{d}) \triangleq \begin{pmatrix} 0 & -\Delta_1(\boldsymbol{d}) \\ \Delta_1(\boldsymbol{d}) & 0 \end{pmatrix}$$

It is possible to represent the FIVR switches with the equation $\boldsymbol{w} = \Delta(\boldsymbol{d})\boldsymbol{z}$ (1d). Additionally, the behavior of the subsystem shown in Figure 2.13, which includes both the input and output networks, can be described as a single coupled system. This system maps the inputs $V_{\text{VRM}}, \boldsymbol{i}^o, \boldsymbol{i}_1, \boldsymbol{v}_2$ to the output $\boldsymbol{v}_1, \boldsymbol{i}_2, \boldsymbol{v}^o$. By grouping V_{VRM} and $\boldsymbol{i}^o$ into the vector $\boldsymbol{u}$, and defining $\boldsymbol{y} \triangleq \boldsymbol{v}^o$, the state-space representation

for this subsystem can be formulated as in equations (1a), (1b), and (1c). Finally, the N_C compensators can be grouped into another linear subsystem, where the input is the vector of error signals for each core. The state-space equations for the compensator subsystem are given in (1e).
The challenge with this system is that the parameter x is high-dimensional. For this reason, an order reduction approach has been implemented based on projecting the first three equations, resulting in the following reduced system:

$$\dot{\boldsymbol{x}}_r = \hat{\mathbf{A}}\boldsymbol{x}_r + \hat{\mathbf{B}}_w\boldsymbol{w} + \hat{\mathbf{B}}_u\boldsymbol{u} \tag{2a}$$

$$\boldsymbol{z} = \hat{\mathbf{C}}_z\boldsymbol{x}_r + \mathbf{D}_{zw}\boldsymbol{w} + \mathbf{D}_{zu}\boldsymbol{u} \tag{2b}$$

$$\boldsymbol{y} = \hat{\mathbf{C}}_y\boldsymbol{x}_r + \mathbf{D}_{yw}\boldsymbol{w} + \mathbf{D}_{yu}\boldsymbol{u} \tag{2c}$$

$$\boldsymbol{w} = \boldsymbol{\Delta}(\boldsymbol{d})\boldsymbol{z} \tag{2d}$$

$$\dot{\boldsymbol{x}}_k = \mathbf{A}_k\boldsymbol{x}_k + \mathbf{B}_k(\boldsymbol{N}\boldsymbol{y} - \boldsymbol{V}_{\text{ref}}), \quad \boldsymbol{d} = \mathbf{C}_k\boldsymbol{x}_k \tag{2e}$$

Where $\hat{\mathbf{A}} = \mathbf{W}^T\mathbf{A}\mathbf{V}$, $\hat{\mathbf{B}}_w = \mathbf{W}^T\mathbf{B}_w$, $\hat{\mathbf{B}}_u = \mathbf{W}^T\mathbf{B}_u$, $\hat{\mathbf{C}}_z = \mathbf{C}_z\mathbf{V}$, and $\hat{\mathbf{C}}_y = \mathbf{C}_y\mathbf{V}$ are the reduced state-space matrices obtained via Petrov-Galerkin projection. Since the system is still nonlinear, a local linearization around an operating point has been performed, by separating each variable into its bias and small-signal components. This way, the resulting set of equations can be written in the compact linearized descriptor form:

$$\begin{aligned} \varepsilon\dot{\xi} &= \mathbf{A}\xi + \mathbf{B}\tilde{\boldsymbol{u}} \\ \tilde{\boldsymbol{y}} &= \mathbf{C}\xi + \mathbf{D}\tilde{\mathbf{u}} \end{aligned} \qquad \text{where } \xi = \begin{pmatrix} \tilde{\boldsymbol{x}} \\ \tilde{\boldsymbol{x}_k} \\ \tilde{\boldsymbol{z}} \end{pmatrix} \tag{2.1}$$

Thus, the PDNs are represented in matrix form as described above and analyzed using MATLAB simulations. The model order reduction and subsequent linearization were not implemented in the course of this work but reduced and linearized state-space descriptions of the PDN were provided as input for subsequent development. However, for the pre-validation of the code, the PDNs were represented as passive elements and to represent the dynamics of the system a set of equations is needed, which implies the necessity of solving differential equations. Therefore, describing these networks in matrix form requires understanding the Modified Nodal Analysis (MNA) method, which is essential for accurately formulating the circuit equations necessary for effective simulation.

2.3 Nodal Analysis: basic concepts

Nodal Analysis (NA) is a fundamental technique used to determine the voltages and currents in a resistive circuit with ideal current sources. The process involves breaking the circuit into n nodes and using Kirchhoff's Current Law (KCL) to

create a system of equations. These equations can then be solved to find the unknown voltages and currents within the circuit. To perform NA, it is necessary follow these steps:

1. **Identify Nodes:** Begin by identifying all nodes in the circuit.
2. **Select a Reference Node:** Choose one node as a reference point.
3. **Formulate Nodal Equations:** Write $n-1$ nodal equations based on Kirchhoff's Current Law, expressing currents in terms of conductances and node voltages.
4. **Construct the Matrix System:** The nodal equations can be expressed in matrix form as:
$$\mathbf{G_n} \boldsymbol{e} = \boldsymbol{a}$$
where:
   - $\mathbf{G_n}$ is the conductance matrix,
   - $\boldsymbol{e}$ is a vector of $n-1$ elements representing node voltages relative to the reference node,
   - $\boldsymbol{a}$ is a vector of $n-1$ elements that includes all the known current sources .
5. **Solve for Currents and Voltages:** This matrix system can then be solved to find all the unknown currents and voltages within the circuit.

2.3.1 Practical Example

To better understand how to apply Nodal Analysis (NA), let us consider a simple circuit with four nodes. This circuit is illustrated in Figure 2.14, where one node is designated as the reference node (marked by the red line), while nodes a, b, and c are the points where Kirchhoff's Current Law (KCL) will be applied. To begin, it is possible to recognize that each node represents a point where incoming and outgoing currents must balance according to Kirchhoff's Current Law (KCL). For each of these nodes, it is possible to express a relationship between the voltages at nodes e_a, e_b, and e_c relative to the reference node. For instance, at node a, it can be observed how the current is influenced by a combination of conductance and the potential difference relative to the other nodes, particularly node b and the reference node. By formulating this relationship, our first equation is derived.

$$1 \cdot e_a + \frac{1}{2}(e_a - e_b) = -3 \tag{2.2}$$

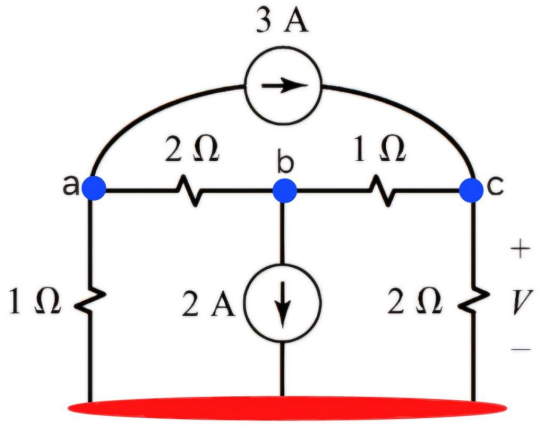


Figure 2.14: Circuit for Nodal Analysis

At node b, a similar relationship takes into account the conductance between nodes b and a, as well as the current flowing from node c. Now, it is possible to derive the second equation.

$$-\frac{1}{2}(e_a - e_b) - 1 \cdot (e_c - e_b) = -2 \tag{2.3}$$

At node c, the current relationship involves the conductance between c and b, as well as between c and the reference node. This leads us to formulate the third and last equation as follow:

$$-1 \cdot (e_c - e_b) - \frac{1}{2} \cdot e_c = 3 \tag{2.4}$$

At this stage, these three equations form the necessary system to solve for the voltages at nodes a, b, and c relative to the reference node. These relationships can be organized into a format that highlights the conductances between the nodes, resulting in a conductance matrix and two vectors: one representing the unknown node voltages and the other representing the known currents applied to the system.

$$\begin{bmatrix} \frac{1}{2}+1 & -\frac{1}{2} & 0 \\ -2 & \frac{1}{2}+1 & -1 \\ 0 & -1 & \frac{1}{2}+1 \end{bmatrix} \begin{bmatrix} e_a \\ e_b \\ e_c \end{bmatrix} = \begin{bmatrix} -3 \\ -2 \\ +3 \end{bmatrix}$$

This matrix formulation describes the currents balance at the nodes and allows us to solve the system of equations. Once the solutions are obtained, it is possible to determine the voltage values at nodes a, b, and c, which will also enable us to calculate the currents flowing through the circuit. This example illustrates how NA is used as a clear and effective methodology for understanding circuit behavior. It provides the groundwork for more complex analyses, such as Modified Node Analysis (MNA), which is beneficial for circuits with more sophisticated components and transient behavior.

2.4 Modified Nodal Analysis

Modified Nodal Analysis (MNA) is a technique used to formulate the equations of a circuit so that they can be solved as a system of differential equations. This method can be used to all types of circuits, including linear, nonlinear, dynamic, and static configurations. Moreover, it is particularly well-suited for CAD and circuit simulations tools like LTspice. Unlike standard nodal analysis, MNA stands out for its ability to handle a wider range of more complex circuits. It enables various type of simulations including constant sources, sinusoidal sources, transient analyses, and even symbolic analyses through the use of Laplace transforms.
The resulting equation from the MNA formulation takes the form:

$$\mathbf{G}\boldsymbol{x}(t) + \mathbf{C}\frac{d\boldsymbol{x}(t)}{dt} = \mathbf{B}\boldsymbol{w}(t), \tag{2.5}$$

where $\boldsymbol{x}(t)$ contains all the nodal voltages and any additional current variables, such as the currents flowing through inductors, ideal voltage sources, and controlled voltage sources. , $\boldsymbol{w}(t)$ collects all the independent sources in the circuit, $\mathbf{G}$ represents the conductance matrix, and $\mathbf{C}$ accounts for the capacitance and inductance parameters present in the circuit.
An interesting aspect of MNA is its ability to incorporate nonlinear elements found in the circuit, which are not addressed in standard nodal analysis. That allows us to analyze more complex circuits effectively. So, let us see a practical example of the MNA applied to our simpler PDN of figure 2.5.

Practical Example

Considering our simplest PDN depicted in figure 2.5 by getting the nodal equations including the inductors and capacitances, it is possible to get this matrix system:

$$\left[\begin{array}{ccccccc|c|c}
1000 & -1000 & & & & & & 1 & \\
-1000 & 1000 & & & & & & & 1 \\
 & & \frac{1}{10000} + g_l(t) & & & & & & -1 \\
 & & & & \frac{1}{5} & -\frac{1}{5} & & & \\
 & & & & -\frac{1}{5} & \frac{1}{5} & & & 1 \\
\hline
1 & & & & & & & & \\
 & 1 & & -1 & & & & & \\
 & & & & & 1 & & &
\end{array}\right]
\left[\begin{array}{c} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ i_{\text{in}} \\ i_1 \\ i_2 \end{array}\right]$$

Figure 2.15: System description of simplified PDN (Part 1)

$$+\left[\begin{array}{ccccc} & & & & \\ & 1\cdot 10^{-6} & -1\cdot 10^{-6} & & \\ & -1\cdot 10^{-6} & 1\cdot 10^{-6} & & \\ \hline & & & -1\cdot 10^{-9} & \\ & & & & -1\cdot 10^{-9} \end{array}\right] \frac{d}{dt}\begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ i_{\text{in}} \\ i_1 \\ i_2 \end{bmatrix} = \begin{bmatrix} \\ \\ \\ \\ V_{in} \\ \\ \\ \end{bmatrix}$$

Figure 2.16: System description of simplified PDN (Part 2)

This matrix represents the system of our simplified PDN, where the variables e_1, e_2, e_3, e_4, e_5 are the voltages at various nodes, and i_1, i_2, i_{in} indicate the currents in their respective branches. The **C** matrix includes all the contributions from both capacitive and inductive elements. Solving this system, which can be achieved through numerical methods, provides a clear and detailed insight into the dynamics of the PDN. This allows us to simulate the system's behavior under various operating conditions, both in a steady state and during transient events. Among the methods used in this thesis, the Euler Method was the method chosen for numerical calculations.

2.5 Euler Method

Once the circuit equations have been established using Modified Nodal Analysis (MNA) or, in the case of multicore systems, already provided in the form of a descriptor system, solving them numerically becomes necessary, particularly for dynamic systems where circuit behavior varies over time. One widely used method for tackling these differential equations is the Euler Method. It is one of the simplest numerical techniques available for solving ordinary differential equations (ODE). While it may be basic, it offers a straightforward approach to approximating solutions when analytical methods are impractical. The Euler Method lies in its strategy to estimate the solution of a differential equation by incrementally stepping forward from an initial condition in small time intervals. The Picard and WR methods in our code are based on the Euler Method.

2.5.1 Application on MNA

In the context of the MNA for PDNs the system of differential equations is of the form:

$$\mathbf{G}\boldsymbol{x}(t) + \mathbf{C}\frac{d\boldsymbol{x}(t)}{dt} = \mathbf{B}\boldsymbol{w}(t), \tag{2.6}$$

To solve numerically this system the backward Euler method is used, which is an implicit time-stepping method suitable for stiff systems. Considering the example shown in the section 2.2.2 Fig. 2.5, a descriptor system of the PDN in matrix form as illustrated in the section 2.4 Fig. 2.15 and 2.16 is obtained. From these figures, it can be seen that the time-varying parameter $\boldsymbol{g}_L(t)$ appears within the matrix $\mathbf{G}$. This represents a significant challenge, as to compute the solution of the system, it is necessary to invert this matrix at each time step. This operation becomes particularly slow, especially when dealing with multicore systems, where the number of elements and the order of the matrices is much larger than in the example considered.
To solve the system, the time t is discretize as $t_k = k\Delta t$ and so the derivative of $\dot{x}(t)$ is approximated as:

$$\dot{x}(t_k) \approx \frac{x(t_k) - x(t_{k-1})}{\Delta t}.$$

Substituting this approximation into the differential equation:

$$\mathbf{G}(t_k)x(t_k) + \mathbf{C}\frac{x(t_k) - x(t_{k-1})}{\Delta t} = \mathbf{B}w(t_k).$$

Next, multiplying by Δt to eliminate the fraction:

$$\Delta t\mathbf{G}(t_k)x(t_k) + \mathbf{C}(x(t_k) - x(t_{k-1})) = \Delta t\mathbf{B}w(t_k).$$

Rearranging the terms to isolate $x(t_k)$:

$$\Delta t\mathbf{G}(t_k)x(t_k) + \mathbf{C}x(t_k) - \mathbf{C}x(t_{k-1}) = \Delta t\mathbf{B}w(t_k).$$

Now, group the terms involving $x(t)$ on the left-hand side:

$$(\mathbf{G}(t_k)\Delta t + \mathbf{C})x(t_k) = \mathbf{C}x(t_{k-1}) + \Delta t\mathbf{B}w(t_k).$$

To solve for $x(t_k)$, the matrix $\mathbf{G}(t_k)\Delta t + \mathbf{C}$ has to be inverted:

$$x(t_k) = \left[\mathbf{G}(t_k)\Delta t + \mathbf{C}\right]^{-1}\left(\mathbf{C}x(t_{k-1}) + \Delta t\mathbf{B}w(t_k)\right).$$

Finally, the matrix $\mathbf{G}(t_k)$ can be decomposed as:

$$\mathbf{G}(t_k) = \mathbf{G_0} + \mathbf{G_L}(t_k),$$

where $\mathbf{G_0}$ is the constant part and $\mathbf{G_L}(t_k)$ accounts for the time-varying parameter $g_L(t_k)$.
Thus, the solution of the system, using the direct method, is given by the following algebraic equation at each time step:

$$x(t_k) = \left[(\mathbf{G_0} + \mathbf{G_L}(t_k)\Delta t) + \mathbf{C}\right]^{-1}\left(\mathbf{C}x(t_{k-1}) + \Delta t\mathbf{B}w(t_k)\right).$$

As noted, the first part of the equation involves inverting the matrix at each time step, which makes the process computationally expensive, especially when simulating the entire system dynamics.
To overcome this difficulty and make the calculation independent of the time-varying term, iterative techniques are employed. Specifically, the techniques of *Waveform Relaxation* and *Picard Iteration* have been studied, which allow solving the system more efficiently, by keeping $\mathbf{G_L(t)}$ constant and recovering its variability through the iterations.

2.5.2 Considerations

The Euler Method accuracy is dependent on the step size h: the smaller the h, the more accurate the approximation. Additionally, while the Euler Method is relatively easy to implement, it has its limitations. Being a first-order method, its precision is constrained and can result in significant errors, particularly when simulating " stiff " systems or circuits with rapidly changing behaviors. The Euler Method is commonly used in the simulation of dynamic systems, such as electrical circuits. In the case of application on Power Delivery Networks, it is useful because enabling in examining transient responses through the applicable differential equations based on the circuit dynamics. which are described by differential equations that evolve over time. The Euler Method allows us to solve these equations and analyze the system's behavior under various operating conditions.

2.6 Waveform Relaxation

The Waveform Relaxation (WR) method is an iterative technique used for solving systems of differential equations, particularly in circuit simulations. This method, instead of solving the entire system simultaneously, allows it to be divided into smaller, more manageable subsystems, thereby reducing computational cost and simulation time. Each subsystem can be solved independently and, in many cases, in parallel. This approach is described in the article [5], which, based on the Parallel Waveform Relaxation method, divides the multicore system into subsystems.
Each subsystem is solved over a defined time interval using initial estimates, then the solutions of the subsystems are combined, and the iterative process continues until the overall system converges to a solution. This approach optimizes computational efficiency, leveraging parallel computation to significantly reduce simulation time, making it ideal for analyzing large and complex systems. In our specific case, this method is introduced to decouple the solution of the descriptor system from the time-varying conductance, which represents the chip activity linking the instantaneous voltage and current on the load.
A schematic representation of the method is shown in Figure 2.17:

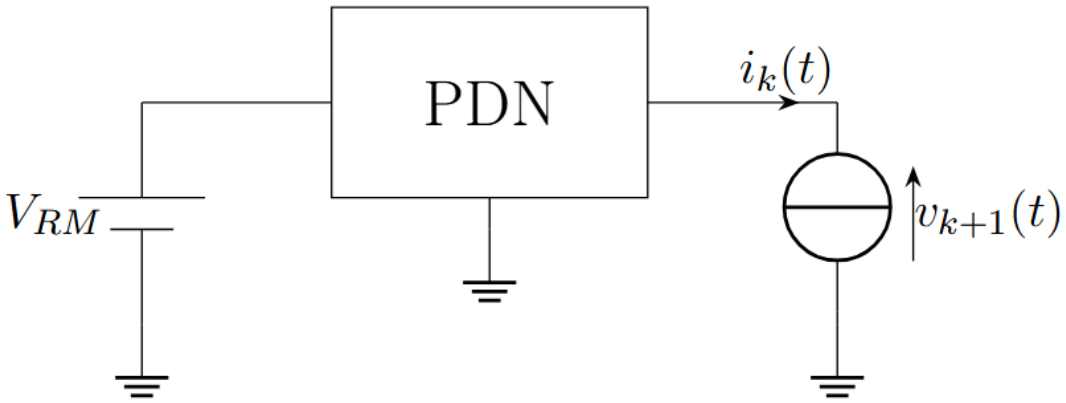


Figure 2.17: WR block diagram

In this block diagram, instead of directly considering the time-variant conductance it was applied a current generator whose value is proportional to the conductance, and the solution is computed iteratively with fixed time steps. During each iteration, the value of $\boldsymbol{g}_L(t)$ is considered constant, and the output voltage across the load is calculated accordingly. This process is repeated iteratively to account for the variability of the conductance over time. By doing so, $\boldsymbol{g}_L(t)$ no longer needs to be inverted at every step, but instead is treated as a constant parameter within the current generator.
As a result, the current takes the following form:

$$i_k(t) = \alpha \cdot V_{k-1}^3(t) + \boldsymbol{g}_L(t) \cdot V_{k-1}(t) \tag{2.7}$$

Where the parameter α depends on the leakage current, which is proportional to the cube of the voltage. This makes α an instantaneous term, present at every time step, alongside the term $\boldsymbol{g}_L(t)$, which describes the chip's activity. The value of $\boldsymbol{g}_L(t)$ varies: it is small when the chip is idle, meaning the voltage has little impact on the current, but increases significantly when the chip is active and requires more current, thereby increasing the current contribution from $\boldsymbol{g}_L(t)$.
In the representation shown in Figure 2.17, the current is treated as an independent input, which is known, while the voltage is an unknown output. This formulation allows us to eliminate the contribution of nonlinear terms within the system descriptor matrices, simplifying the overall system representation.

2.7 Picard iteration

Picard Iteration is an iterative method that gets the solution of a system by progressively refining the solution through iterative error correction, differing from

the traditional perturbation methods often employed in WR techniques. Like WR it is intended to eliminate the dependences on the time-variant element from the descriptor system. This iterative process accelerates convergence toward the system's solution, which will be demonstrated in the Results section.

2.7.1 Explanation by example

To understand how Picard Iteration functions, consider a system of ordinary differential equations (ODE) where the solution is expressed as a sum of successive iterations. Each iteration improves upon the previous approximation of the solution. Let us consider an ODE system as follow:

$$\frac{dy}{dt} = f(y,t), \quad y(t_0) = y_0. \tag{2.8}$$

Using the Picard iteration, the solution can be expressed as follows:

$$\begin{aligned} y_0(t) &= y_0, \\ y_1(t) &= y_0 + \int_{t_0}^{t} f(y_0(s), s)\, ds, \\ y_n(t) &= y_0 + \int_{t_0}^{t} f(y_{n-1}(s), s)\, ds. \end{aligned} \tag{2.9}$$

For instance, let us consider the differential equation

$$\frac{dy}{dt} = 2ty, \quad y(0) = 1. \tag{2.10}$$

Starting with the initial approximation $y_0(t)$:

$$y_0(t) = 1. \tag{2.11}$$

Calculating $y_1(t)$:

$$y_1(t) = 1 + \int_0^t 2s \cdot y_0(s)\, ds = 1 + \int_0^t 2s \cdot 1\, ds = 1 + t^2. \tag{2.12}$$

Calculating $y_2(t)$:

$$y_2(t) = 1 + \int_0^t 2s \cdot [1 + s^2]\, ds = 1 + t^2 + \frac{t^4}{2!}. \tag{2.13}$$

Proceeding with $y_3(t)$:

$$y_3(t) = 1 + \int_0^t 2s \cdot \left(1 + s^2 + \frac{s^4}{2!}\right) ds = 1 + t^2 + \frac{t^4}{2!} + \frac{t^6}{3!}. \tag{2.14}$$

The general form for $y_n(t)$ becomes:

$$y_n(t) = 1 + t^2 + \frac{t^4}{2!} + \cdots + \frac{t^{2n}}{n!}. \tag{2.15}$$

With each iteration, our solution gains precision. As the number of iterations approaches infinity, the solution converges to:

$$y(t) = \sum_{n=0}^{\infty} \frac{(t^2)^n}{n!} = e^{t^2}. \tag{2.16}$$

2.8 Numerical instability

The topic of numerical instability is a critical consideration in computational simulations, particularly when using real computers, which were employed for implementing the WR and Picard methods. Numerical instability arises when small errors in calculations lead to significant deviations from expected results, which becomes especially relevant in the context of solving ODE.
In our analysis, achieving accurate and reliable solutions is essential, but several sources can contribute to numerical instability. Round-off errors are one of the most common problems because a real computer has a defined level of numerical precision, and values beyond this precision are rounded, potentially introducing errors. Poorly conditioned problems also play a significant role because when a matrix is considered poorly conditioned that means that contains values that vary widely in magnitude, which can lead to instability in calculations.
Additionally, algorithmic issues must be addressed; employing robust algorithms that can effectively manage stability challenges is crucial to prevent inevitable instability.
To mitigate these challenges, several strategies can be implemented. Firstly, using robust methods for solving differential equations is vital. For instance, the implicit Euler method serves as a foundational algorithm for calculating solutions in our system. Secondly, tuning techniques can enhance stability by improving matrix conditioning. Lastly, careful computation practices are essential because when summing multiple values that may be affected by errors, it is advisable to sum smaller numbers first before adding larger ones to minimize potential errors.
As mentioned a useful tool to enhance stability within Matlab, the platform in use, is the `equilibrate()` function. This function modifies the matrix so that its values are scaled more closely together, thereby improving its conditioning and enhancing calculation stability. However, caution is necessary when dealing with very poorly conditioned matrices since rescaling them to turn back to the original system may inadvertently amplify existing errors, which could render the balancing process ineffective.

2.9 Linearization

An important aspect to take into consideration in the analysis of the PDN is the linearization process. As already discussed to get the descriptor system of the PDNs of the multicore system the first process consists of linearizing the system, particularly because these systems often have numerous nonlinear components. This is especially relevant when dealing with voltage regulators, which are commonly implemented as DC/DC converters, such as buck converters 2.18.

The buck converter includes nonlinear elements like switches, Mosfet, and diodes,

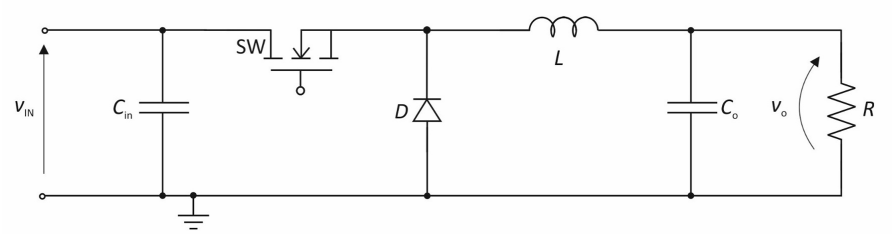


Figure 2.18: Structure of a Buck Converter

which are crucial for its functionality. In Continuous Conduction Mode (CCM), these switches alternate between opening and closing to prevent short circuits at the source. However, the nonlinear behavior of these components can complicate the analysis of the overall system. To effectively manage this complexity, it is often common to linearize the PDN. This technique involves creating an averaged model of the converter where the switching elements are represented as ideal transformers. By establishing a Linear Time-Invariant (LTI) system, further linearization methods can be applied alongside model reduction techniques that preserve the essential input-output behavior of the system. This approach simplifies the analysis of the PDN's behavior significantly. By transforming the system into a linearized model, engineers can leverage classical control theory and other analytical techniques to predict and optimize the PDN's performance more effectively. Linearization allows for a clearer understanding of how the system will respond under various conditions, facilitating better design and implementation of robust voltage control strategies. Nowadays, this process is vital for ensuring that modern electronic systems operate reliably and efficiently in an increasingly complex technological landscape.
Below is reported an example of how an average model of a buck converted is achieved.

Demonstration

The buck converter is primarily used in Continuous Conduction Mode (CCM), so the average model will be addressed under these conditions. In CCM, the diode

and the Mosfet operate alternately; when one is closed, the other is open, and vice versa.
To start, three nodes in the circuit are identified: the active node (a), where the active element, the Mosfet, is connected; the passive node (p), which is connected solely to the passive element, the diode; and finally, the common node (c), where both the Mosfet and diode are connected. Next, the voltages and currents at these nodes are defined, as shown in the figure 2.19. From this setup, it is clear that

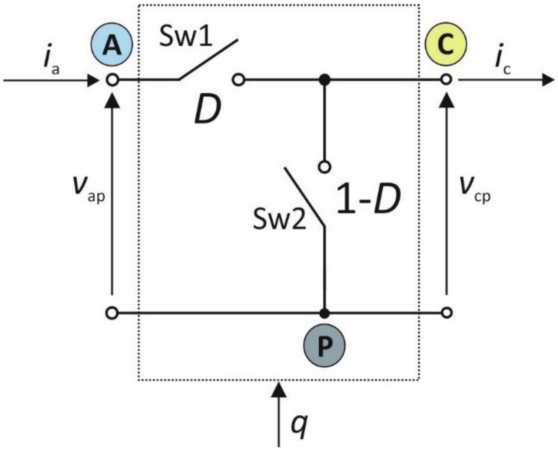


Figure 2.19: Buck converter for averaging model

the current $\overline{i}_a$ will be equal to $\overline{i}_c$ during the turn-on-time, while it will be zero when the MOSFET is off. Similarly, the voltage $\overline{v}_{cp}$ will be equal to $\overline{v}_{ap}$ when the MOSFET is on and will drop to zero when it is off. Thus, the behavior of these two parameters can be described as follows:

$$\begin{aligned} \overline{i}_a(t) &= \overline{i}_c(t) \cdot \overline{q}(t) \\ \overline{v}_{cp}(t) &= \overline{v}_{ap} \cdot \overline{q}(t) \end{aligned} \tag{2.17}$$

where $q(t)$ is the digital signal that controls the Mosfet. Taking the average value of these equations gives us:

$$\begin{aligned} \overline{i}_a &= \overline{i}_c \cdot D \\ \overline{v}_{cp} &= \overline{v}_{ap} \cdot D \end{aligned} \tag{2.18}$$

where D represents the duty cycle.
From these two final equations, it becomes possible to reconstruct an ideal transformer, as average values are considered. This leads us to create our averaged model of the buck converter as in figure 2.20, which can eventually be decomposed into DC and AC components for analyzing its control-to-output transfer function. This analysis is crucial for constructing an effective control network. Currently, an

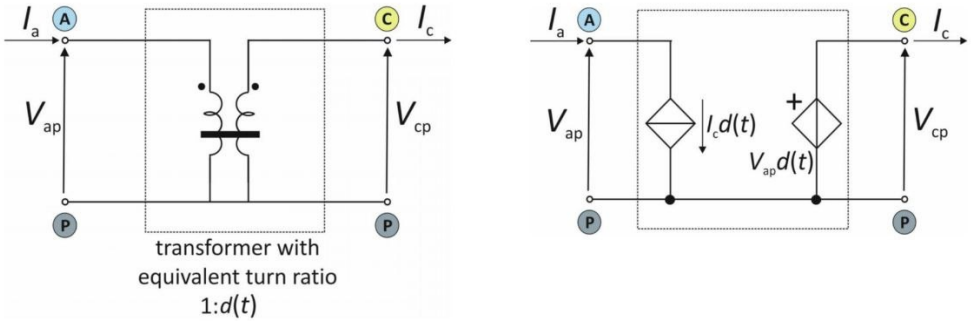


Figure 2.20: Representation of the averaging model of Buck converter in CCM [8]

average model was derived but to linearize it is necessary to identify the operating point and split each variable as follows:

$$\begin{aligned} i_a(t) &= \bar{i}_a + \tilde{i}_a \\ i_c(t) &= \bar{i}_c + \tilde{i}_c \\ d(t) &= \bar{d} + \tilde{d} \\ v_{AP}(t) &= \bar{v_{AP}} + \tilde{v_{AP}} \\ v_{CP}(t) &= \bar{v_{CP}} + \tilde{v_{CP}} \end{aligned} \tag{2.19}$$

Next, substituting in the equation 2.18:

$$\begin{aligned} \bar{i}_a + \tilde{i}_a &= (\bar{d} + \tilde{d})(\bar{i}_c + \tilde{i}_c) \\ \bar{v_{CP}} + \tilde{v_{CP}} &= (\bar{d} + \tilde{d})(\bar{v_{AP}} + \tilde{v_{AP}}) \end{aligned} \tag{2.20}$$

Finally, from these final equations is possible to distinguish the DC and AC terms and get the final circuit model as depicted in Fig. 2.21 :

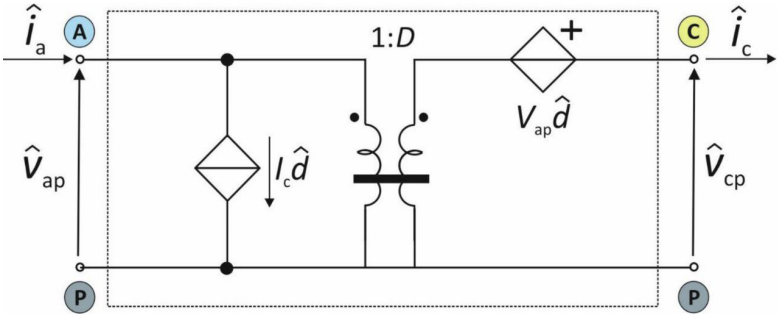


Figure 2.21: Linearize model of BUCK converter [8]

2.10 Review of current methods

With the continuous increase in power levels in modern microprocessors, Fully Integrated Voltage Regulators (FIVRs) are increasingly referenced regarding the architecture of the power delivery network (PDN) [6]. «These regulators provide fine-grain voltage regulation without overly complicating the distribution network», being positioned both on the die and on the package of the microprocessor.

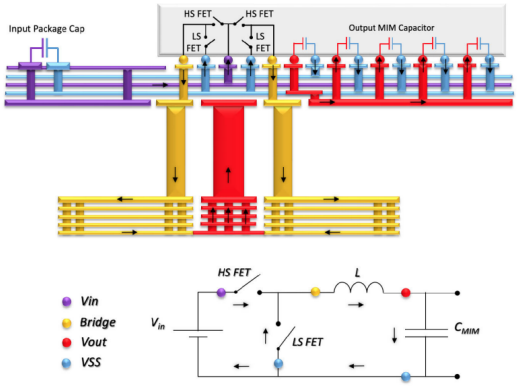


Figure 2.22: Example of Buck FIVR implementation [6]

However, since the input network is shared and each core has its IVR, there is noise coupling between the various cores. This issue highlights the need for a simulation framework capable of executing time-domain analyses to ensure power integrity. Several Model Order Reduction (MOR) approaches have been proposed to address this challenge (Articles: [3] [1] [2] [4]). The methodology proposed in the article [3] adopts a hierarchical process of MOR to reduce the complexity of output impedance dynamics, demonstrating excellent accuracy compared to reference simulations performed with HSPICE, with execution time acceleration ranging from 8× to 50×. Previous studies have shown that approaches based on rational macro modeling and moment matching through structured Krylov subspace projections can improve simulation efficiency, albeit without guarantees on accuracy limits (Article [1]). Moreover, waveform relaxation techniques, such as parallel waveform relaxation [5], have been developed to enhance efficiency and reduce execution times in transient power integrity verification. This approach has demonstrated a significant speedup compared to SPICE solutions, exceeding three orders of magnitude when applied to real PDN models of commercial multicore systems. So, the ongoing research in this field underscores the need for sophisticated methods to address the complexities of power integrity and signal integrity in the design of advanced microprocessors, laying the groundwork for future innovations in this critical area of electronic design.

2.11 Challenges

Despite the progress made over time, the issue of transient analysis in signal integration remains an open challenge. This is because new technologies will continue to emerge, each representing a unique starting point. As chip scaling progresses, the interaction between components becomes increasingly critical, and noise can affect the power levels in the network. Although advancements have been made, various difficulties still need to be addressed. The complexity of modern multicore systems generates significant coupling between the cores, creating power delivery noise that can compromise performance. Traditional simulation methods, such as SPICE-based approaches, struggle to scale according to the complexity required for multicore systems, leading to prolonged computation times and potential inaccuracies. Another challenge is the need for accurate modeling of the electromagnetic interactions between components, which can introduce additional non-linearities into the system. Ensuring that these interactions are accurately captured in transient simulations is crucial for reliable power integrity analysis.

2.12 Conclusion

In conclusion, while the challenges associated with the transient simulation of power distribution networks are significant, recent advancements in model reduction methods and advanced simulation techniques offer new opportunities to tackle these complexities. These developments not only enhance design efficiency but also increase the capability to meet the ever-growing performance demands of modern HPC and AI applications. It is essential that the evolution of these methods continues to ensure that future microprocessors can meet the increasing demands for performance and reliability, as demonstrated by recent studies.

Chapter 3

Methods

In this section, the methods and algorithms utilized for simulating PDN are described. Initially, our approach was centered around an algorithm based on WR to handle the problem of the time-variant load, which is particularly effective for problems that involve iterative calculations where the system's state evolves over time, such as in PDN simulations. However, as the simulations progressed, some efficiency challenges were encountered, which prompted us to transition to a different algorithm based on Picard iteration, which ultimately proved to be more effective for the PDNs studied.

3.1 Waveform Relaxation on-chip load

The first algorithm implemented was based on WR, a method commonly employed to solve large-scale nonlinear systems of equations. In the context of PDN, this technique involves iterating between calculating currents and updating voltages across the system until convergence is achieved. To clarify this process, a block diagram of the PDN already mentioned in the previous chapter can be referenced, illustrating how the WR method is applied in our case, see figure 3.1.
The approach begins with deriving the MNA for the PDN represented as passive elements. This step is essential as it establishes the groundwork for the system of equations that will govern our simulation and additionally establishes a basis for other subsequent steps in the algorithm which are initialization, current calculation, and voltage update via Matlab.
To establish the initial conditions, it was assumed that the chip was inactive, and therefore, the nominal voltage was considered. Starting with initial voltage values, the system's state is iteratively updated at each time step. The main idea behind WR is to adjust the voltage waveforms during each iteration, progressively refining our solution by adjusting the current sources. As iterations continue, the error

between consecutive iterations should decrease, ensuring convergence toward the desired reference value.
A representative flowchart of the logic of the code is shown in Figure 3.2. Unlike traditional methods like LTspice, which solve the circuit at each time step with specified accuracy, the WR uses a series of approximations. This distinction means that when comparing results from WR with those obtained from LTspice, careful consideration must be given to selecting an appropriate simulation step size, h, to ensure a good comparison between both methods.
Here's the logic of the code: At the beginning the output voltage was initialized at the nominal value then was derived the current as a function of the voltage as reported in the equation 3.1, then at each time interval t, it was calculated the output voltage over n samples, from this new output voltage it was evaluated the new current sources characteristic following again the equation reported below 3.1.

$$i_k = \alpha \cdot V_{k-1}^3(t) + \boldsymbol{g}_L(t) \cdot V_{k-1}(t) \tag{3.1}$$

The iterative process continues until the difference between consecutive voltages decreases below a predetermined threshold, indicating that convergence has been achieved. This iterative refinement is why WR is sometimes referred to as a perturbation method; small adjustments are made to the system at each step to bring the solution closer to its target.

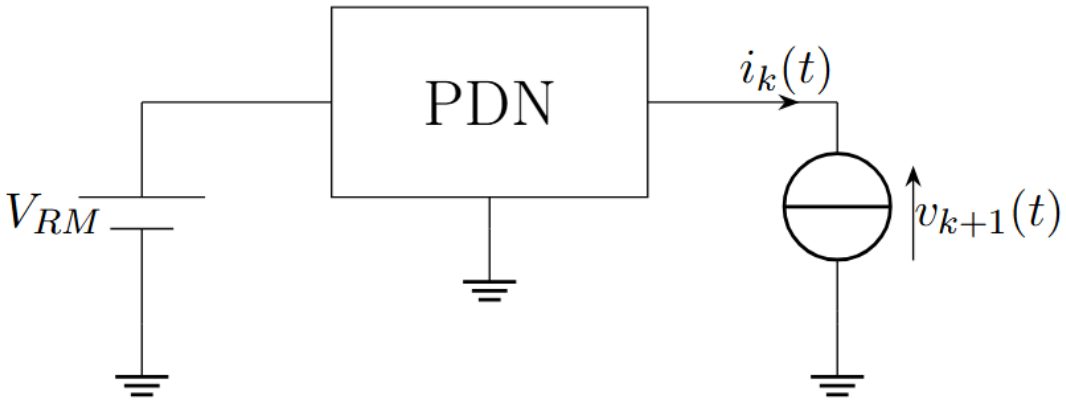


Figure 3.1: WR block diagram

Here is reported the flowchart of the WR used in the matlab code 3.2:

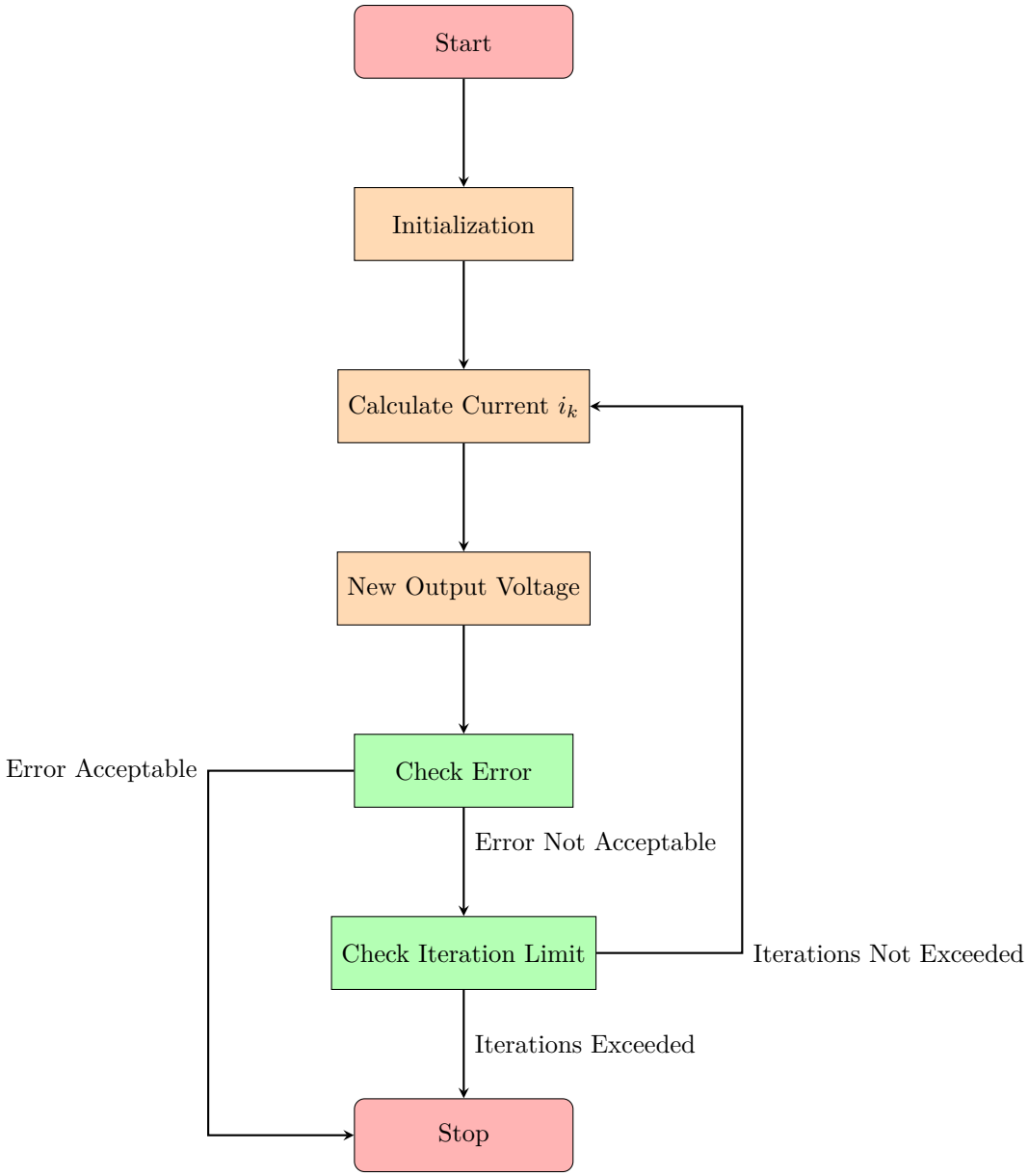


Figure 3.2: Flowchart of the perturbation approach WR

3.2 Picard method

The Picard method is an iterative technique that starts with an initial approximation and progressively refines this estimate through repeated iterations. Essentially, it integrates the given function step by step, gradually approaching the exact solution. This translates to performing additive operations iteratively, using the implicit Euler method to compute the additive terms until the solution meets the desired accuracy. Additionally, a translation term is introduced, which, if chosen correctly, can lead the system to converge even more rapidly. For this reason, the method was studied both with shifting and without shifting during the iterative process.

3.3 Mathematical Formulation

3.3.1 Notation

Symbol	Meaning
$\mathbf{E}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$	Descriptor system matrices - original system
$\boldsymbol{x}$	State variables - original system
$\boldsymbol{u}$	Input signals - original system
$\boldsymbol{y}$	Output signals - original system
$\boldsymbol{i}$	Input current - original system
$\boldsymbol{v}$	Output voltage - original system
$\underline{\mathbf{B}}, \underline{\mathbf{C}}, \underline{\mathbf{D}}$	Descriptor system matrices - original system without direct coupling
$\underline{\boldsymbol{x}}$	State variables - original system without direct coupling
$\underline{\boldsymbol{y}}$	Output signals - original system without direct coupling
$\underline{\boldsymbol{i}}$	Input signals - original system without direct coupling
N_p	Number of differential outputs per core
N_c	Number of cores
N_t	Number of total ports
$\mathbf{I}$	Identity matrix
$\hat{\mathbf{E}}, \hat{\mathbf{A}}, \hat{\mathbf{B}}, \hat{\mathbf{C}}$	Descriptor system matrices - original system without direct coupling, not explicitly using $\mathbf{D}$
Δt	Time increment
g_{shift}	Reference conductance - iteration with shifting
$\tilde{\boldsymbol{g}}$	Output conductance - original system shifted
$\tilde{\boldsymbol{x}}$	State variables - original system shifted
$\tilde{\boldsymbol{u}}$	Input signal - original system shifted
h	Discretization coefficient
k	Iteration number
$\mathbf{G}$	Diagonal representation of the output conductance

Table 3.1: Descriptor System Matrices and Variables

3.3.2 Descriptor systems: elimination of direct coupling

As discussed in Chapter 2, the descriptor system of a multicore system was derived through considerations of the system depicted in Fig. 3.3, and the resulting system equation is given in the equation 3.2.

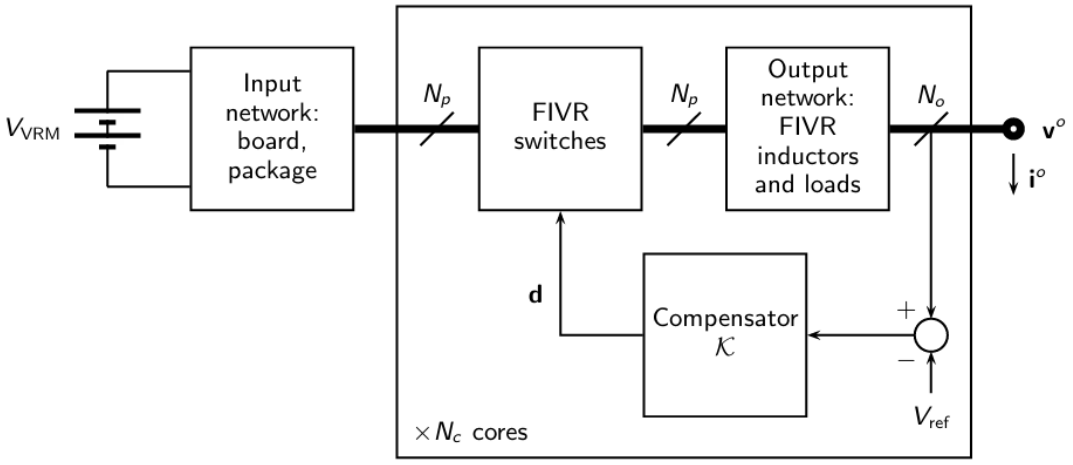


Figure 3.3: Schematic representation of multicore system

$$\begin{aligned} \mathbf{E}\dot{\boldsymbol{x}}(t) &= \mathbf{A}\boldsymbol{x}(t) + \mathbf{B}\boldsymbol{u}(t) \\ \boldsymbol{y}(t) &= \mathbf{C}\boldsymbol{x}(t) + \mathbf{D}\boldsymbol{u}(t) \end{aligned} \tag{3.2}$$

where

$$\boldsymbol{u}(t) = \begin{pmatrix} V_{in} \\ \boldsymbol{i}(t) \end{pmatrix}, \qquad \boldsymbol{y}(t) = \begin{pmatrix} I_{in} \\ \boldsymbol{v}(t) \end{pmatrix} \tag{3.3}$$

After that, the system will be split into multiple parts to separate the components related to the inputs and outputs which will help us to rewrite the system in a form that is independent of the input but dependent only on the output since the goal is to calculate only the output voltage. Therefore, the matrices **B**, **C**, and **D** will be rewritten as follows

$$\mathbf{B} = (\boldsymbol{B_1},\ \underline{\mathbf{B}}), \quad \mathbf{C} = \begin{pmatrix} \boldsymbol{C_1} \\ \underline{\mathbf{B}} \end{pmatrix}, \quad \mathbf{D} = \begin{pmatrix} \mathbf{D_{11}} & \mathbf{D_{12}} \\ \mathbf{D_{21}} & \underline{\mathbf{D}} \end{pmatrix} \tag{3.4}$$

where $\underline{\mathbf{B}}$, $\underline{\mathbf{C}}$ and $\underline{\mathbf{D}}$ have dimensions equal to : $\left(N_x, 2N_pN_c\right)$, $\left(2N_pN_c, N_x\right)$ and $\left(2N_pN_c, 2N_pN_{\mathbf{C}}\right)$ respectively. Now, having separated the input from the outputs in the matrices, the system can be rewritten as follows:

$$\begin{aligned} \mathbf{E}\dot{\boldsymbol{x}}(t) &= \mathbf{A}\boldsymbol{x}(t) + \boldsymbol{B_1} \cdot V_{in} + \underline{\mathbf{B}} \cdot \boldsymbol{i}(t) \\ \boldsymbol{y}(t) &= \underline{\mathbf{C}}\boldsymbol{x}(t) + \mathbf{D_{21}} \cdot V_{in} + \underline{\mathbf{D}} \cdot \boldsymbol{i}(t) \end{aligned} \tag{3.5}$$

Assuming the system is in steady state for negative times, the initial conditions are defined as follows $\boldsymbol{i}(t) = 0$, $\boldsymbol{y}(t) = \boldsymbol{y_{DC}} \quad \forall t < 0$

$$\begin{cases} 0 = \mathbf{A} \cdot x_{DC} + \boldsymbol{B_1} \cdot V_{in} \\ \boldsymbol{y_{DC}} = \underline{\mathbf{C}} \cdot x_{DC} + \mathbf{D_{21}} \cdot V_{in} \end{cases} \rightarrow \begin{cases} x_{DC} = -\mathbf{A}^{-1}(\boldsymbol{B_1} \cdot V_{in}) \\ \boldsymbol{y_{DC}} = (-\underline{\mathbf{C}} \cdot \mathbf{A}^{-1} \cdot \underline{\mathbf{B}} + \mathbf{D_{21}}) \cdot V_{in} \end{cases} \tag{3.6}$$

Now, let us assume: $\underline{\boldsymbol{x}}(t) = \boldsymbol{x}(t) - x_{DC} \quad \underline{\boldsymbol{y}}(t) = \boldsymbol{y}(t) - \boldsymbol{y_{DC}} \quad \underline{\boldsymbol{i}}(t) = \boldsymbol{i}(t) - I_{DC} = \boldsymbol{i}(t)$
This leads to the following results

$$\begin{aligned} \mathbf{E}\dot{\underline{\boldsymbol{x}}}(t) = \mathbf{E}\dot{\boldsymbol{x}}(t) = \mathbf{A} \cdot \underline{\boldsymbol{x}}(t) + \mathbf{A} \cdot x_{DC} + \boldsymbol{B_1} \cdot V_{in} + \underline{\mathbf{B}} \cdot \boldsymbol{i}(t) \\ \underline{\boldsymbol{y}}(t) = \boldsymbol{y}(t) - \boldsymbol{y_{DC}} = \underline{\mathbf{C}} \cdot \underline{\boldsymbol{x}}(t) + \underline{\mathbf{C}} \cdot x_{DC} + \mathbf{D_{21}} \cdot V_{in} + \underline{\mathbf{D}} \cdot \boldsymbol{i}(t) - \boldsymbol{y_{DC}} \end{aligned} \tag{3.7}$$

Substituting equation 3.6 will yield the final result of this system:

$$\begin{cases} \mathbf{E}\dot{\underline{\boldsymbol{x}}}(t) = \mathbf{A} \cdot \underline{\boldsymbol{x}}(t) + \underline{\mathbf{B}} \cdot \boldsymbol{i}(t) \\ \underline{\boldsymbol{y}}(t) = \underline{\mathbf{C}} \cdot \underline{\boldsymbol{x}}(t) + \underline{\mathbf{D}} \cdot \boldsymbol{i}(t) \end{cases} \tag{3.8}$$

note: $\underline{i}$ and $\underline{\boldsymbol{y}}$ are the "small-signal" current and voltage after removing the DC values, respectively and $\underline{\boldsymbol{x}}(t) = \boldsymbol{x}(t) - x_{DC}$; $\underline{\boldsymbol{x}}(0) = 0$
Now, since $\mathbf{u}$(t) include the output current and V_{IN}, and the goal is to have an output equation dependent solely on the output voltage, the process of eliminating the matrix D from the system will be carried out. To achieve this, the system will be reconstructed as follows:

$$\begin{aligned} \underbrace{\begin{pmatrix} \mathbf{E} & 0 \\ 0 & 0 \end{pmatrix}}_{\hat{\mathbf{E}}} \frac{d}{dt} \begin{pmatrix} \underline{\boldsymbol{x}}(t) \\ \underline{\mathbf{D}} \cdot \boldsymbol{i}(t) \end{pmatrix} &= \underbrace{\begin{pmatrix} \mathbf{A} & 0 \\ 0 & -I \end{pmatrix}}_{\hat{\mathbf{A}}} \begin{pmatrix} \underline{\boldsymbol{x}}(t) \\ \underline{\mathbf{D}} \cdot \boldsymbol{i}(t) \end{pmatrix} + \underbrace{\begin{pmatrix} \underline{\mathbf{B}} \\ \underline{\mathbf{D}} \end{pmatrix}}_{\hat{\mathbf{B}}} \boldsymbol{i}(t) \\ \boldsymbol{y}(t) &= \underbrace{\begin{pmatrix} \underline{\mathbf{C}} & I \end{pmatrix}}_{\hat{\mathbf{C}}} \underbrace{\begin{pmatrix} \underline{\boldsymbol{x}}(t) \\ \underline{\mathbf{D}} \cdot \boldsymbol{i}(t) \end{pmatrix}}_{\hat{\mathbf{w}}} + \boldsymbol{y_{DC}} \end{aligned} \tag{3.9}$$

Thus, rewriting the system yields the following form:

$$\begin{aligned} \hat{\mathbf{E}}\dot{\hat{\mathbf{w}}}(t) = \hat{\mathbf{A}}\hat{\mathbf{w}}(t) + \hat{\mathbf{B}}\boldsymbol{i}(t) \\ \boldsymbol{y}(t) = \hat{\mathbf{C}}\hat{\mathbf{w}}(t) + \boldsymbol{y_{DC}} \end{aligned} \tag{3.10}$$

where $\hat{\mathbf{E}}, \hat{\mathbf{A}}, \hat{\mathbf{B}}, \hat{\mathbf{C}}$ have dimension equal to: $\left(N_x + N_t,\ N_x + N_t\right)$ $\left(N_x + N_t,\ N_x + N_t\right)$ $\left(N_x + N_t,\ N_t\right)$ $\left(N_t,\ N_x + N_t\right)$ respectively, where N_t is equal to $N_p \cdot 2 \cdot N_c$
From this point, the Picard algorithm is described in progressively more complex cases, starting from the scalar case and advancing towards the multiport and multicore cases.

3.3.3 The scalar case

in this section vogliamo andare ad analizzare il metodo di picard in un caso scalare andando ad analizzare il caso senza lo shift e con lo shift partiamo per tanto dal caso piu semplici in cui abbiamo a che fare con elementi scalari.

Direct solution

Using the formulation described in equation 2.5 in Chapter 2, the MNA system obtained in the scalar case is as follows:

$$\begin{aligned} \mathbf{C}\dot{\boldsymbol{x}}(t) + \mathbf{G}(t)\boldsymbol{x}(t) &= \mathbf{B}\boldsymbol{u}(t) \\ \mathbf{G}(t) &= \mathbf{G_0} + \mathbf{G_1}\boldsymbol{g}(t) \\ \boldsymbol{y}(t) &= L^T\boldsymbol{x}(t) \end{aligned} \tag{3.11}$$

Then defining: $\mathbf{E} = \mathbf{C}$, $\mathbf{A} = -\mathbf{G_0}$, $\mathbf{A_1} = -\mathbf{G_1}$, the final matrix equation will be:

$$\mathbf{E}\dot{\boldsymbol{x}}(t) = \mathbf{A}\boldsymbol{x}(t) + \mathbf{A_1}\boldsymbol{g}(t)\boldsymbol{x}(t) + \mathbf{B}\boldsymbol{u}(t) \tag{3.12}$$

To derive the conditions for $t = 0$:

$$\begin{aligned} \boldsymbol{x}(0) &= -(\mathbf{A} + \mathbf{A_1}\boldsymbol{g}(0))^{-1}\mathbf{B}\boldsymbol{u}(0) \\ \boldsymbol{y}(0) &= L^T\boldsymbol{x}(0) \end{aligned} \tag{3.13}$$

Whereas to derive the condition for $t > 0$ (implicit Euler scheme with time step Δt, $x_h \approx x(h\Delta t)$, $u_h = u(h\Delta t)$, $y_h \approx y(h\Delta t)$, $g_h = g(h\Delta t)$):

$$\begin{aligned} x_h &= \left[\mathbf{E} - \Delta t\left(\mathbf{A} + \mathbf{A_1}g_h\right)\right]^{-1}\left(\mathbf{E}x_{h-1} + \Delta t\mathbf{B}u_h\right) \\ y_h &= L^T x_h \end{aligned} \tag{3.14}$$

Iterations without shifting

Using the MNA again, the following system can be derived:

$$\begin{aligned} \mathbf{C}\dot{\boldsymbol{x}}(t) + \mathbf{G}(t)\boldsymbol{x}(t) &= \mathbf{B}\boldsymbol{u}(t) \\ \mathbf{G}(t) &= \mathbf{G_0} + \mathbf{G_1}\boldsymbol{g}(t) \end{aligned} \tag{3.15}$$

Then defining: $\mathbf{E} = \mathbf{C}$, $\mathbf{A} = -\mathbf{G_0}$, $\mathbf{A_1} = -\mathbf{G_1}$, the final matrix equation will be:

$$\mathbf{E}\dot{\boldsymbol{x}}(t) = \mathbf{A}\boldsymbol{x}(t) + \mathbf{A_1}\boldsymbol{g}(t)\boldsymbol{x}(t) + \mathbf{B}\boldsymbol{u}(t) \tag{3.16}$$

To solve the system iteratively without shifting, the process begins with $\hat{x}^{(0)}(t) = 0$. For the first iteration ($k = 1$), the system equation becomes:

$$\mathbf{E}\dot{\boldsymbol{x}}^{(1)}(t) = \mathbf{A}\boldsymbol{x}^{(1)}(t) + \mathbf{B}\boldsymbol{u}(t) \tag{3.17}$$

At $t = 0$:

$$\boldsymbol{x}^{(1)}(0) = -\mathbf{A}^{-1}\mathbf{B}\boldsymbol{u}(t) \tag{3.18}$$

For $t > 0$, the solution can be iteratively updated using:

$$\boldsymbol{x}_h^{(1)}(t) = (\mathbf{E} - \Delta t\mathbf{A})^{-1}(\mathbf{E}\boldsymbol{x}_{h-1}^{(1)} + \Delta t\mathbf{B}\boldsymbol{u}(t)) \tag{3.19}$$

For the next iterations, so $k > 1$:

$$\mathbf{E}\dot{\boldsymbol{x}}^{(k)} = \mathbf{A}\boldsymbol{x}^{(k)} + \mathbf{A_1}\boldsymbol{g}(t)\boldsymbol{x}(t)^{(k-1)} \tag{3.20}$$

At $t = 0$:

$$\boldsymbol{x}^{(k)}(0) = -\mathbf{A}^{-1}\mathbf{A_1}\boldsymbol{g}(0)\boldsymbol{x}^{(k-1)}(0) \tag{3.21}$$

For $t > 0$, the solution can be iteratively updated using:

$$\boldsymbol{x}_h^{(k)} = (\mathbf{E} - \Delta t\mathbf{A})^{-1}(\mathbf{E}x_{h-1}^{(k)} + \mathbf{A_1}\boldsymbol{g}(t)x_h^{(k-1)}) \tag{3.22}$$

Finally, the final solution can be expressed as a series:

$$\begin{aligned} \boldsymbol{x}_{\text{tot}}(t) &= \sum_{k=0}^{\text{max iteration}} \boldsymbol{x}^{(k)}(t) \\ \boldsymbol{y}(t) &= L^T\boldsymbol{x}_{\text{tot}}(t) \end{aligned} \tag{3.23}$$

Iterations with shifting

The initial representation of the MNA system is always as follows

$$\begin{aligned} \mathbf{C}\dot{\boldsymbol{x}}(t) + \mathbf{G}(t)\boldsymbol{x}(t) &= \mathbf{B}\boldsymbol{u}(t) \\ \mathbf{G}(t) &= \mathbf{G_0} + \mathbf{G_1}\boldsymbol{g}(t) \end{aligned} \tag{3.24}$$

Then defining: $\mathbf{E} = \mathbf{C}$, $\mathbf{A} = -\mathbf{G_0}$, $\mathbf{A_1} = -\mathbf{G_1}$.

$$\mathbf{E}\dot{\boldsymbol{x}}(t) = \mathbf{A}\boldsymbol{x}(t) + \mathbf{A_1}\boldsymbol{g}(t)\boldsymbol{x}(t) + \mathbf{B}\boldsymbol{u}(t) \rightarrow \text{Bilinear state-space} \tag{3.25}$$

A reference conductance g_{shift} is now choosen, and it is generally $g_{shift} \neq \boldsymbol{g}(0)$. Then considering this $\boldsymbol{g}(t) = \tilde{\boldsymbol{g}}(t) + g_{shift}$. with $\boldsymbol{x}(t) = \tilde{\boldsymbol{x}}(t) + x_0$, $\boldsymbol{u}(t) = \tilde{\boldsymbol{u}}(t) + u_0$, the following equation is obtained:

$$\begin{aligned} \mathbf{E}\dot{\tilde{\boldsymbol{x}}}(t) =& \mathbf{A}\tilde{\boldsymbol{x}}(t) + \mathbf{A}x_0 + \mathbf{A_1}\tilde{\boldsymbol{x}}(t)\tilde{\boldsymbol{g}}(t) + \mathbf{A_1}\tilde{\boldsymbol{x}}(t)g_{shift} + \\ &+ \mathbf{A_1}x_0\tilde{\boldsymbol{g}}(t) + \mathbf{A_1}x_0g_{shift} + \mathbf{B}\tilde{\boldsymbol{u}}(t) + \mathbf{B}u_0 \end{aligned} \tag{3.26}$$

Then, the equation above can be split as follow

$$\text{split}: \quad \begin{cases} 0 = \mathbf{A}x_0 + \mathbf{A_1}x_0 g_{shift} + \mathbf{B}u_0 \\ \mathbf{E}\dot{\tilde{\boldsymbol{x}}}(t) = (\mathbf{A_0} + \mathbf{A_1}g_{shift})\tilde{\boldsymbol{x}}(t) + \mathbf{A_1}x_0\tilde{\boldsymbol{g}}(t) + \mathbf{B}\tilde{\boldsymbol{u}}(t) + \mathbf{A_1}\tilde{\boldsymbol{x}}(t)\tilde{\boldsymbol{g}}(t) \end{cases} \tag{3.27}$$

Then defining that new matrix and vectors:

$$\tilde{\mathbf{w}}(t) = \begin{pmatrix} \tilde{\boldsymbol{u}}(t) \\ \tilde{\boldsymbol{g}}(t) \end{pmatrix} \quad \bar{\mathbf{A}} = (\mathbf{A} + \mathbf{A_1}g_0) \quad \bar{\mathbf{B}} = (\mathbf{B} \quad \mathbf{A_1}x_0) \tag{3.28}$$

$$\mathbf{E}\dot{\tilde{\boldsymbol{x}}}(t) = \bar{\mathbf{A}}\tilde{\boldsymbol{x}}(t) + \bar{\mathbf{B}}\tilde{\mathbf{w}}(t) + \underbrace{\mathbf{A_1}\tilde{\boldsymbol{x}}(t)\tilde{\boldsymbol{g}}(t)}_{\text{bilinear}} \tag{3.29}$$

note: if $\tilde{\boldsymbol{u}}(t) = 0$, $\bar{\mathbf{B}} = B$, $\tilde{\mathbf{w}}(t) = \tilde{\boldsymbol{g}}(t)$ and this is our case:

$$\mathbf{E}\dot{\tilde{\boldsymbol{x}}}(t) = \bar{\mathbf{A}}\tilde{\boldsymbol{x}}(t) + \bar{\mathbf{B}}\tilde{\boldsymbol{g}}(t) + \mathbf{A_1}\tilde{\boldsymbol{x}}(t)\tilde{\boldsymbol{g}}(t) \tag{3.30}$$

For the iteration, the process begin with $\tilde{\boldsymbol{x}}^{(0)}\ (t) = 0$. For $k = 1$,

$$\mathbf{E}\dot{\tilde{\boldsymbol{x}}}^{(1)}(t) = \bar{\mathbf{A}}\boldsymbol{x}^{(1)}(t) + \mathbf{B}\tilde{\boldsymbol{g}}(t) \tag{3.31}$$

From the equation above the following result is obtained:

$$\begin{aligned} \tilde{\boldsymbol{x}}^{(1)}(0) &= -\bar{\mathbf{A}}^{-1}\mathbf{B}\tilde{\boldsymbol{g}}(0) \\ \tilde{\boldsymbol{x}}_h^{(1)} &= (\mathbf{E} - \Delta t\bar{\mathbf{A}})^{-1}[\mathbf{E}\tilde{\boldsymbol{x}}_{h-1}^{(1)} + \Delta t B\tilde{\boldsymbol{g}}_h] \end{aligned} \tag{3.32}$$

whereas for $k > 1$

$$\mathbf{E}\dot{\tilde{\boldsymbol{x}}}^{(k)}(t) = \bar{\mathbf{A}}\tilde{\boldsymbol{x}}^{(k)}(t) + \mathbf{A_1}\tilde{\boldsymbol{x}}^{(k-1)}(t)\tilde{\boldsymbol{g}}(t) \tag{3.33}$$

again the following result is obtained:

$$\begin{aligned} \tilde{\boldsymbol{x}}^{(k)}(0) &= -\bar{\mathbf{A}}^{-1}\mathbf{A_1}\tilde{\boldsymbol{x}}^{(k-1)}(0)\tilde{\boldsymbol{g}}(0) \\ \tilde{\boldsymbol{x}}_h^{(k)} &= (\mathbf{E} - \Delta t\bar{\mathbf{A}})^{-1}[\mathbf{E}\tilde{\boldsymbol{x}}_{h-1}^{(k)} - \Delta t\mathbf{A_1}\tilde{\boldsymbol{x}}_h^{(k-1)}\tilde{\boldsymbol{g}}_h] \end{aligned} \tag{3.34}$$

3.3.4 The multiport case

Formulation

Consider the following system

$$\begin{aligned} \hat{\mathbf{E}}\hat{\mathbf{w}}(t) &= \hat{\mathbf{A}}\hat{\mathbf{w}}(t) + \hat{\mathbf{B}}\boldsymbol{i}(t) \\ \boldsymbol{y}(t) &= \hat{\mathbf{C}}\hat{\mathbf{w}}(t) + \boldsymbol{y_{DC}} \end{aligned} \tag{3.35}$$

where this matrices have already been defined in the descriptor system. The input currents are found as follows

$$\boldsymbol{i}(t) = -\mathbf{Q}\mathbf{G}(t)\mathbf{Q}^T\boldsymbol{y}(t) = -\mathbf{Q}\mathbf{G}(t)\mathbf{Q}^T(\hat{\mathbf{C}}\hat{\mathbf{w}} + \boldsymbol{y_{DC}}) \tag{3.36}$$

Using this equation in the original system,

$$\begin{aligned} \hat{\mathbf{E}}\dot{\hat{\mathbf{w}}}(t) &= \hat{\mathbf{A}}\hat{\mathbf{w}}(t) - \hat{\mathbf{B}}\mathbf{Q}\mathbf{G}(t)\mathbf{Q}^T\hat{\mathbf{C}}\hat{\mathbf{w}}(t) - \hat{\mathbf{B}}\mathbf{Q}\mathbf{G}(t)\mathbf{Q}^T\boldsymbol{y_{DC}} \\ \boldsymbol{y}(t) &= \hat{\mathbf{C}}\hat{\mathbf{w}}(t) + \boldsymbol{y_{DC}} \end{aligned} \tag{3.37}$$

Iteration without shifting

To solve the system iteratively without shifting, the initial condition is set as $\hat{\mathbf{w}}^{(0)}(t) = 0$. For the first iteration ($k = 1$), the system equation becomes:

$$\hat{\mathbf{E}}\dot{\hat{\mathbf{w}}}^{(1)}(t) = \hat{\mathbf{A}}\hat{\mathbf{w}}^{(1)}(t) - \hat{\mathbf{B}}\mathbf{Q}\mathbf{G}(t)\mathbf{Q}^T\boldsymbol{y_{DC}} \tag{3.38}$$

At $t = 0$:

$$\hat{\mathbf{w}}^{(1)}(0) = \hat{\mathbf{A}}^{-1}\hat{\mathbf{B}}\mathbf{Q}\mathbf{G}(0)\mathbf{Q}^T\boldsymbol{y_{DC}} \tag{3.39}$$

For $t > 0$, the solution can be iteratively updated using:

$$\hat{\mathbf{w}}_h^{(1)} = (\hat{\mathbf{E}} - \Delta t\hat{\mathbf{A}})^{-1}\left(\hat{\mathbf{E}}\hat{\mathbf{w}}_{h-1}^{(1)} - \Delta t\hat{\mathbf{B}}\mathbf{Q}\mathbf{G}(t)\mathbf{Q}^T\boldsymbol{y_{DC}}\right) \tag{3.40}$$

For the next iterations, so $k > 1$:

$$\hat{\mathbf{E}}\dot{\hat{\mathbf{w}}}^{(k)}(t) = \hat{\mathbf{A}}\hat{\mathbf{w}}^{(k)} - \hat{\mathbf{B}}\mathbf{Q}\mathbf{G}(t)\mathbf{Q}^T\hat{\mathbf{C}}\hat{\mathbf{w}}^{(k-1)}(t) \tag{3.41}$$

At $t = 0$:

$$\hat{\mathbf{w}}^{(k)}(0) = \hat{\mathbf{A}}^{-1}\hat{\mathbf{B}}\mathbf{Q}\mathbf{G}(0)\mathbf{Q}^T\hat{\mathbf{C}}\hat{\mathbf{w}}^{(k-1)}(0) \tag{3.42}$$

At $t > 0$, again in a iterative way:

$$\hat{\mathbf{w}}_h^{(k)} = (\hat{\mathbf{E}} - \Delta t\hat{\mathbf{A}})^{-1}\left(\hat{\mathbf{E}}\hat{\mathbf{w}}_{h-1}^{(k)} - \Delta t\hat{\mathbf{B}}\mathbf{Q}\mathbf{G}(t)\mathbf{Q}^T\hat{\mathbf{C}}\hat{\mathbf{w}}_h^{(k-1)}\right) \tag{3.43}$$

The solution $\hat{\mathbf{w}}$ can then be expressed as a series:

$$\hat{\mathbf{w}}_{\text{tot}}(t) = \sum_{k=0}^{\text{max iteration}} \hat{\mathbf{w}}^{(k)}(t) \tag{3.44}$$

So, the output y is given by:

$$\boldsymbol{y}(t) = \left(\hat{\mathbf{C}} \sum_{k=0}^{\text{max iteration}} \hat{\mathbf{w}}^{(k)}(t)\right) + \boldsymbol{y_{DC}} \tag{3.45}$$

Iteration with shifting

Starting from the equation:

$$\begin{aligned}\hat{\mathbf{E}}\dot{\hat{\mathbf{w}}}(t) &= \hat{\mathbf{A}}\hat{\mathbf{w}}(t) - \hat{\mathbf{B}}\mathbf{Q}\mathbf{G}(t)\mathbf{Q}^T\hat{\mathbf{C}}\hat{\mathbf{w}}(t) - \hat{\mathbf{B}}\mathbf{Q}\mathbf{G}(t)\mathbf{Q}^T\boldsymbol{y_{DC}} \\ \boldsymbol{y}(t) &= \hat{\mathbf{C}}\hat{\mathbf{w}}(t) + \boldsymbol{y_{DC}}\end{aligned} \tag{3.46}$$

Assume again:

$$\mathbf{G}(t) = \tilde{\boldsymbol{g}}(t) + \mathbf{G_{shift}}, \quad \hat{\mathbf{w}}(t) = \tilde{\hat{\mathbf{w}}}(t) + \hat{w}_{shift} \tag{3.47}$$

where for simplicity, let us recall $\tilde{\hat{\mathbf{w}}}(t) = \tilde{\mathbf{w}}(t)$ and $\hat{w}_{shift} = w_{shift}$. Then:

$$\begin{aligned}\hat{\mathbf{E}}\dot{\tilde{\mathbf{w}}}(t) = &\hat{\mathbf{A}}\tilde{\mathbf{w}}(t) + \hat{\mathbf{A}}w_{shift} - \hat{\mathbf{B}}\mathbf{Q}\mathbf{G_{shift}}\mathbf{Q}^T\hat{\mathbf{C}}\tilde{\mathbf{w}}(t) - \hat{\mathbf{B}}\mathbf{Q}\mathbf{G_{shift}}\mathbf{Q}^T\hat{\mathbf{C}}w_{shift} \\ &- \hat{\mathbf{B}}\mathbf{Q}\tilde{\boldsymbol{g}}(t)\mathbf{Q}^T\hat{\mathbf{C}}\tilde{\mathbf{w}}(t) - \hat{\mathbf{B}}\mathbf{Q}\tilde{\boldsymbol{g}}(t)\mathbf{Q}^T\hat{\mathbf{C}}w_{shift} - \hat{\mathbf{B}}\mathbf{Q}\tilde{\boldsymbol{g}}(t)\mathbf{Q}^T\boldsymbol{y_{DC}} \\ &- \hat{\mathbf{B}}\mathbf{Q}\mathbf{G_{shift}}\mathbf{Q}^T\boldsymbol{y_{DC}}\end{aligned} \tag{3.48}$$

Splitting

$$\begin{cases} 0 = \hat{\mathbf{A}}w_{shift} - \hat{\mathbf{B}}\mathbf{Q}\mathbf{G_{shift}}\mathbf{Q}^T\hat{\mathbf{C}}w_{shift} - \hat{\mathbf{B}}\mathbf{Q}\mathbf{G_{shift}}\mathbf{Q}^T\boldsymbol{y_{DC}} \\ \hat{\mathbf{E}}\dot{\tilde{\mathbf{w}}} = \hat{\mathbf{A}}\tilde{\mathbf{w}} - \hat{\mathbf{B}}\mathbf{Q}\mathbf{G_{shift}}\mathbf{Q}^T\hat{\mathbf{C}}\tilde{\mathbf{w}} - \hat{\mathbf{B}}\mathbf{Q}\tilde{\boldsymbol{g}}(t)\mathbf{Q}^T\hat{\mathbf{C}}\tilde{\mathbf{w}} - \hat{\mathbf{B}}\mathbf{Q}\tilde{\boldsymbol{g}}(t)\mathbf{Q}^T\hat{\mathbf{C}}w_{shift} + \\ \qquad - \hat{\mathbf{B}}\mathbf{Q}\tilde{\boldsymbol{g}}(t)\mathbf{Q}^T\boldsymbol{y_{DC}} \end{cases} \tag{3.49}$$

Then, defining

$$\mathbf{A_c} = \hat{\mathbf{A}} - \hat{\mathbf{B}}\mathbf{Q}\mathbf{G_{shift}}\mathbf{Q}^T\hat{\mathbf{C}}, \quad \mathbf{B_c} = -\begin{bmatrix}\hat{\mathbf{B}} & \hat{\mathbf{B}}\end{bmatrix}, \quad \mathbf{U}(t) = \begin{pmatrix}\mathbf{Q}\tilde{\boldsymbol{g}}(t)\mathbf{Q}^T\hat{\mathbf{C}}w_{shift} \\ \mathbf{Q}\tilde{\boldsymbol{g}}(t)\mathbf{Q}^T\hat{\mathbf{C}}\boldsymbol{y_{DC}}\end{pmatrix} \tag{3.50}$$

As a result the following equation was obtained

$$\hat{\mathbf{E}}\dot{\tilde{\mathbf{w}}}(t) = \mathbf{A_c}\tilde{\mathbf{w}}(t) + \mathbf{B_c}\mathbf{U} - \underbrace{\hat{\mathbf{B}}\mathbf{Q}\tilde{\boldsymbol{g}}(t)\mathbf{Q}^T\hat{\mathbf{C}}\tilde{\mathbf{w}}(t)}_{\text{bilinear}} \tag{3.51}$$

For $k = 1$:

$$\hat{\mathbf{E}}\dot{\tilde{\mathbf{w}}}^{(1)}(t) = \mathbf{A_c}\tilde{\mathbf{w}}^{(1)}(t) + \mathbf{B_c}\mathbf{U}(t) \tag{3.52}$$

At $t = 0$:

$$\tilde{\mathbf{w}}^{(1)}(0) = -{\mathbf{A_c}}^{-1}\mathbf{B_c}\mathbf{U}(0) \tag{3.53}$$

For $t > 0$:

$$\tilde{\mathbf{w}}_h^{(1)} = \left(\hat{\mathbf{E}} - \Delta t\mathbf{A_C}\right)^{-1}\left(\hat{\mathbf{E}}\tilde{\mathbf{w}}_{h-1}^{(1)} + \Delta t\mathbf{B_c}\mathbf{U_h}\right) \tag{3.54}$$

For $k > 1$:

$$\hat{\mathbf{E}}\dot{\tilde{\mathbf{w}}}^{(k)}(t) = \mathbf{A_c}\tilde{\mathbf{w}}^{(k)}(t) - \hat{\mathbf{B}}\mathbf{Q}\tilde{\boldsymbol{g}}(t)\mathbf{Q}^T\hat{\mathbf{C}}\tilde{\mathbf{w}}^{(k-1)}(t) \tag{3.55}$$

At $t = 0$:

$$\tilde{\mathbf{w}}^{(k)}(0) = {\mathbf{A_c}}^{-1} \left(\hat{\mathbf{B}}\mathbf{Q}\tilde{\boldsymbol{g}}(t)\mathbf{Q}^T\hat{\mathbf{C}}\tilde{\mathbf{w}}^{(k-1)}(0) \right) \tag{3.56}$$

For $t > 0$:

$$\tilde{\mathbf{w}}_h^{(k)} = \left(\hat{\mathbf{E}} - \Delta t A_{\mathbf{C}} \right)^{-1} \left(\hat{\mathbf{E}}\tilde{\mathbf{w}}_{h-1}^{(k)} - \Delta t \hat{\mathbf{B}}\mathbf{Q}\tilde{\boldsymbol{g}}(t)\mathbf{Q}^T\hat{\mathbf{C}}\tilde{\mathbf{w}}_h^{(k-1)} \right) \tag{3.57}$$

The solution $\hat{\mathbf{w}}$ can then be expressed as a series:

$$\begin{aligned} \hat{\mathbf{W}}_{\text{tot}} &= \sum_{k=0}^{\text{max iteration}} \hat{\mathbf{w}}^{(k)}(t) \\ y &= \hat{\mathbf{C}} \cdot \hat{\mathbf{w}}_{\text{tot}} + \hat{\mathbf{C}} \cdot \mathbf{w}_{\text{shift}} + \boldsymbol{y_{DC}} \end{aligned} \tag{3.58}$$

Chapter 4

Results

4.1 Waveform Relaxation

Several simulations were conducted on various PDN models, to assess the reliability of the WR method. Our approach involved two modes of analysis: a software-based analysis using MATLAB, which allowed us to implement and test the method numerically, and a circuit analysis with LTspice, enabling us to compare the results obtained with theoretical expectations and verify the accuracy of our method. This initial phase aimed to evaluate the WR method's effectiveness in addressing power distribution challenges by exploring various configurations and scenarios within the network.
However, during our simulations, convergence issues were encountered with certain specific PDN configurations. This observation was crucial as it marked a turning point that led us to consider a methodological shift, which will be elaborated upon in the following section.
The results are illustrated through several figures that compare different PDN cases and the accuracy of the method used. From the PDNs illustrated in Chapter 2 Sec. 2.2.2, it was evaluated via LTspice the exact solution by directly using the time-variant conductance $g_L(\mathrm{t})$.

Simplified PDN

The objective is to see that the output voltage evaluated via WR on Matlab tends toward the exact solution calculated in this case with LTspice, Fig. 4.1. Then, the error at each time step was evaluated and the trend was toward 0 as the iterations increased, as expected, see figure 4.2. For all the other PDNs analyzed, the graphs should also show the convergence of the method, matching the WR results with those from LTspice.

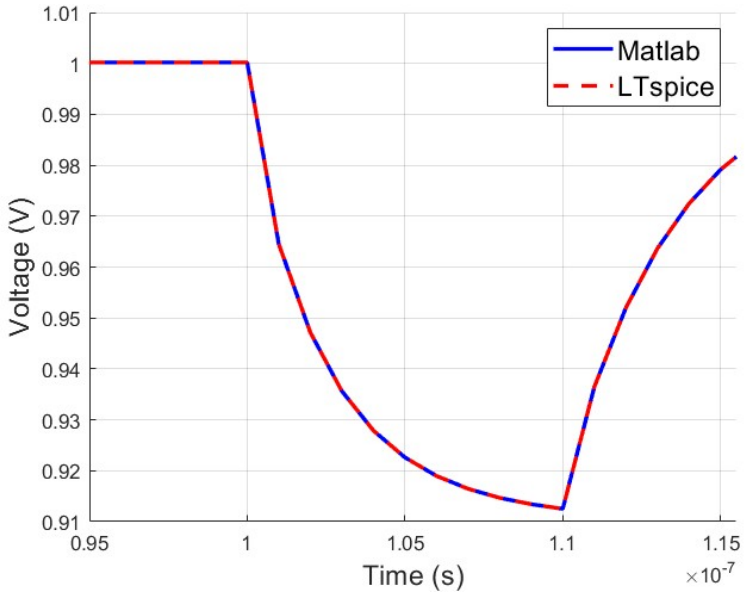


Figure 4.1: Comparison between exact case and WR by Matlab-simplified PDN

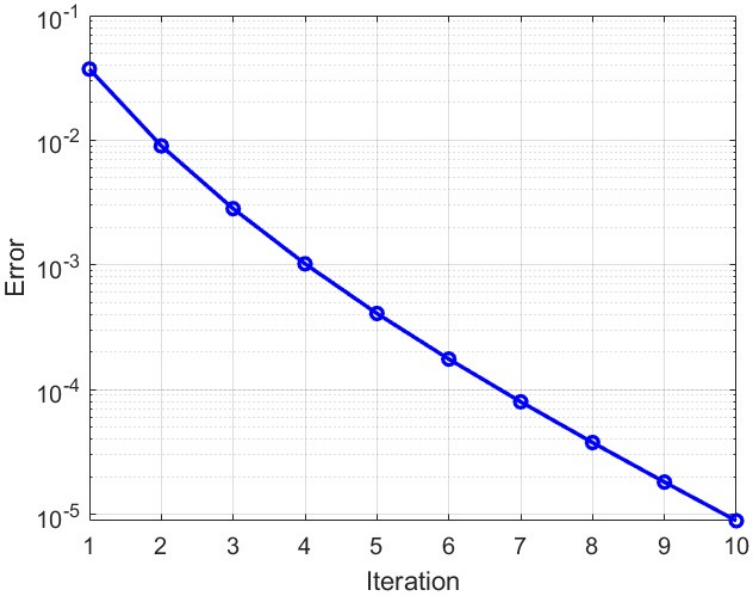


Figure 4.2: Error vs Reference of the simplified PDN using WR

PDN with decoupling capacitors

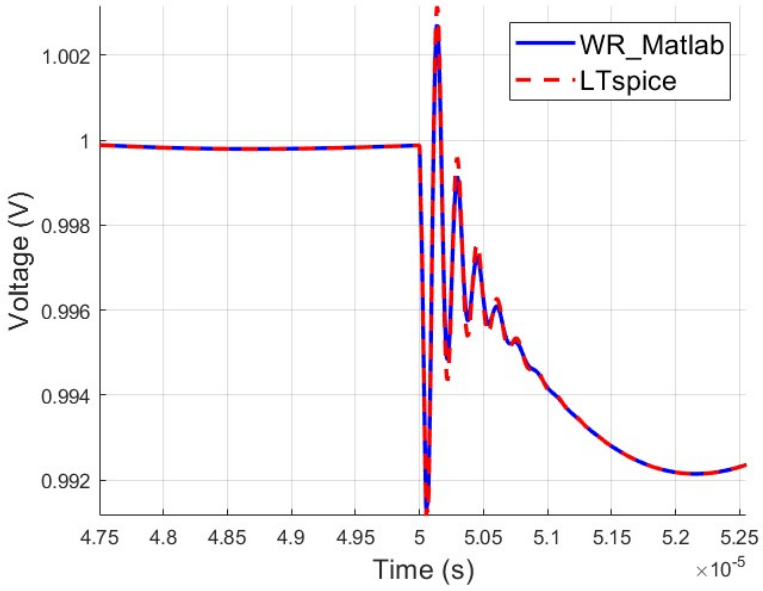


Figure 4.3: Comparison between exact case and WR by Matlab-full PDN

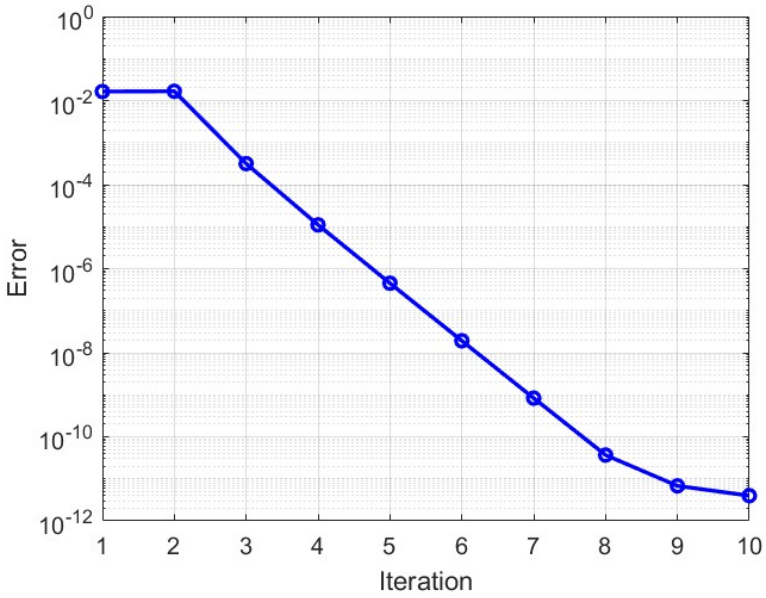


Figure 4.4: Error vs Reference of the full PDN using WR

PDN with extra decoupling capacitors

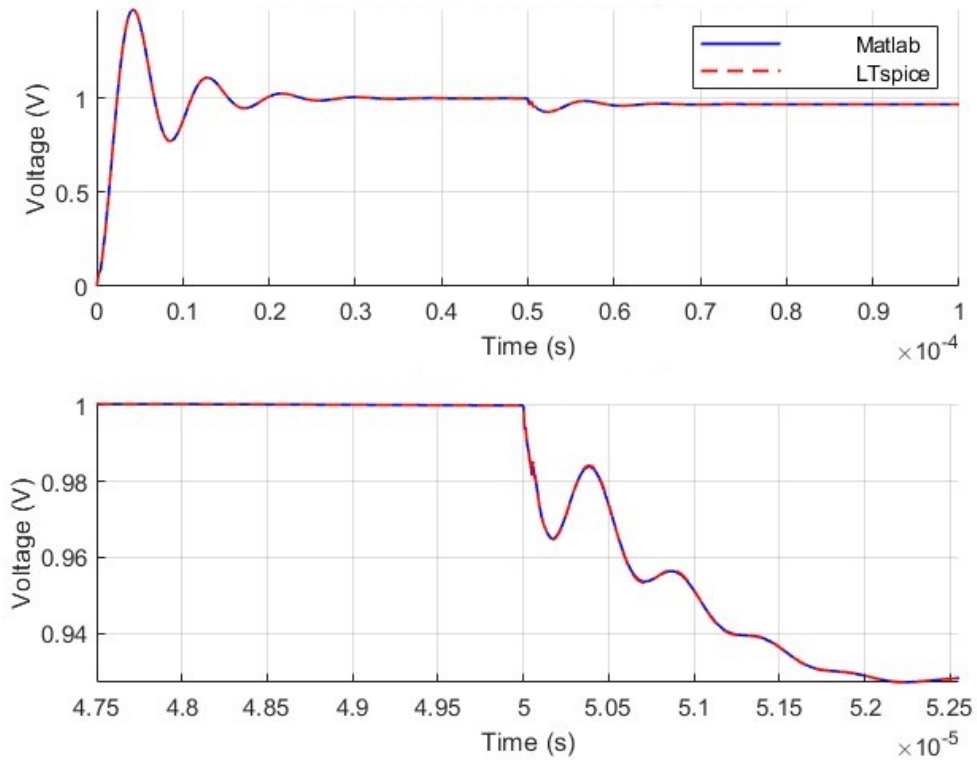


Figure 4.5: Comparison between exact case and WR by Matlab-full PDN with extra decap

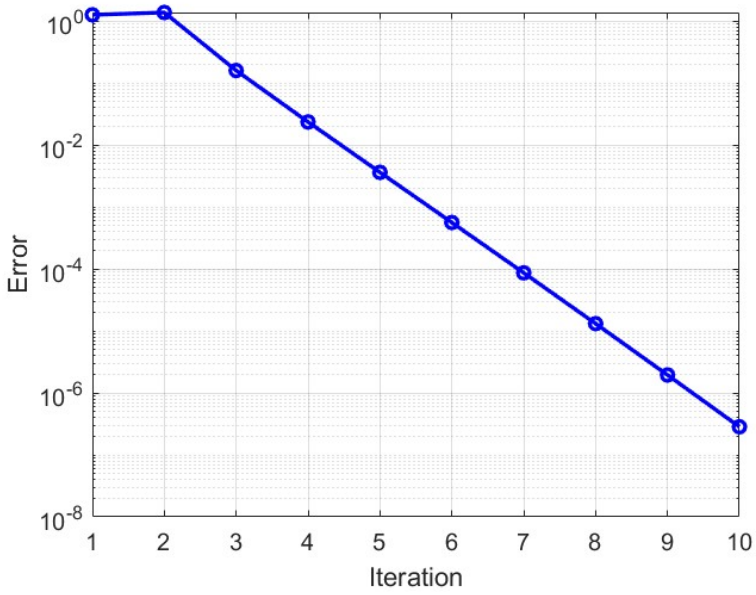


Figure 4.6: Error vs Reference of the full PDN with additional decoupling capacitor using WR

PDN without decoupling capacitors

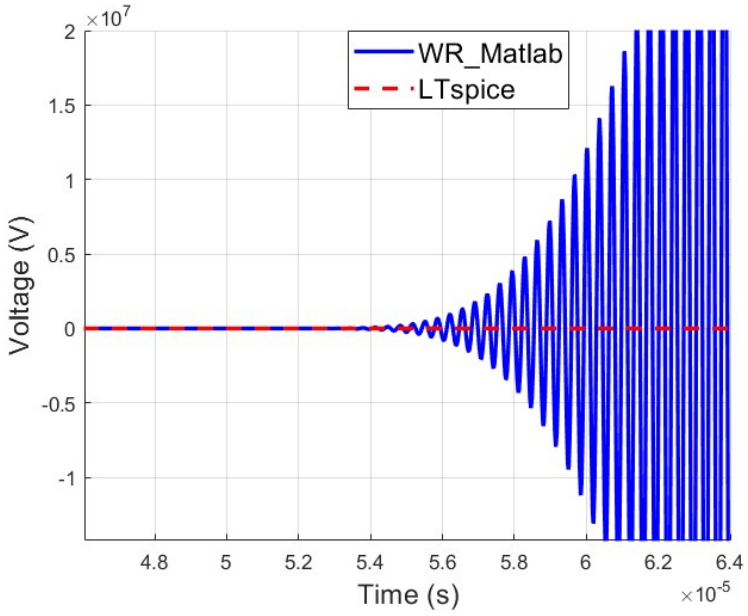


Figure 4.7: Comparison between exact case and WR by Matlab-full PDN with no decap

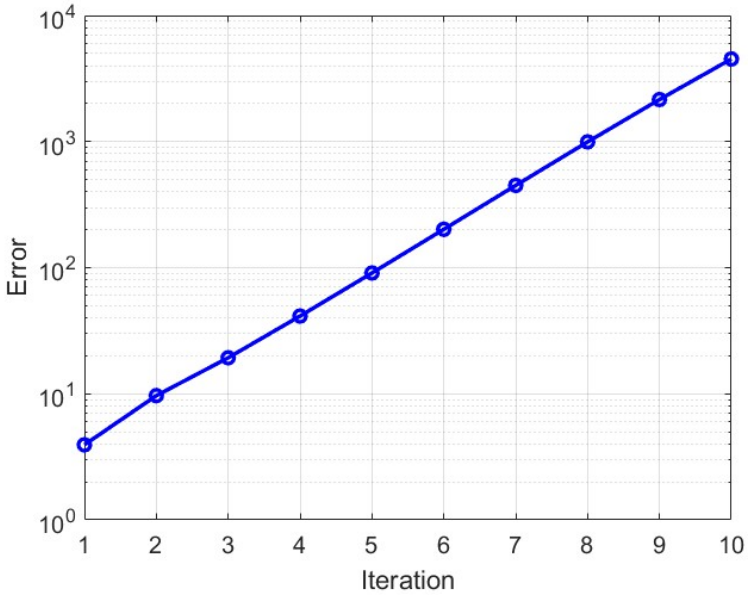


Figure 4.8: Error vs Reference of the PDN without decoupling capacitor using WR

The method applied demonstrated a significant reduction in error as the iterations progressed; however, this was not the case for all configurations. Specifically, the PDN with decoupling capacitors, as depicted in section 2.2.2 Fig. 2.7 and the PDN with additional decoupling capacitance Fig. 2.9 exhibited relatively rapid convergence, achieving more than sufficient accuracy within just a few iterations. In contrast, this positive trend was less pronounced in the simplified PDN scenario (Sec. 2.2.2 Fig. 2.5). On the other hand, a particularly critical situation arose in PDN that does not use decoupling capacitance, an essential component for maintaining voltage stability and compensating for impedance across various operational frequencies. In these cases, the expected gradual reduction of error did not happen, highlighting the need for further modifications to our approach.

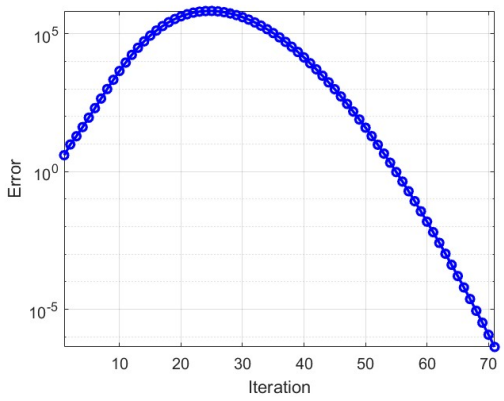


Figure 4.9: Error vs Reference with different load WR

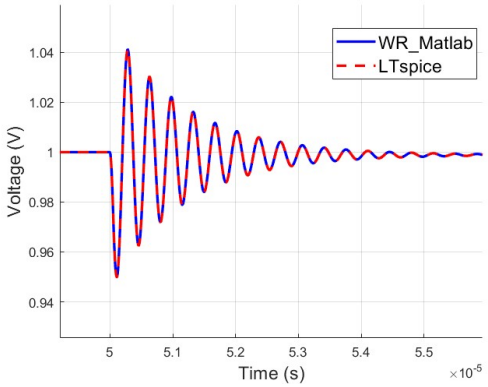


Figure 4.10: Last iteration WR vs exact case

However, varying load conditions were found to potentially mitigate this divergence, allowing for a return to a convergent behavior, as illustrated in the figure 4.9 and 4.10. However, this alternative solution did not fully align with the primary objective of our research, which aimed to achieve rapid and accurate verification. To address these challenges effectively, the Picard method is employed, this latter generates new outputs by cumulatively summing errors, unlike the WR method.

4.2 Picard iteration test

A series of preliminary tests were conducted to evaluate the effectiveness of the Picard method, mirroring the approach taken with the Waveform Relaxation (WR) method discussed earlier. The primary goal was to ensure that the Picard method would not encounter the same convergence issues observed with WR, ultimately providing a more robust solution. As previously mentioned, the process began with the matrix formulation using MNA, and then the Picard method was applied alongside the implicit Euler approach to solve the ODE equation.

4.2.1 Preliminary test

As with the WR method, a preliminary analysis of the Picard method was conducted. In this phase, the same PDN networks are used. Results are reported below:

Simplified PDN

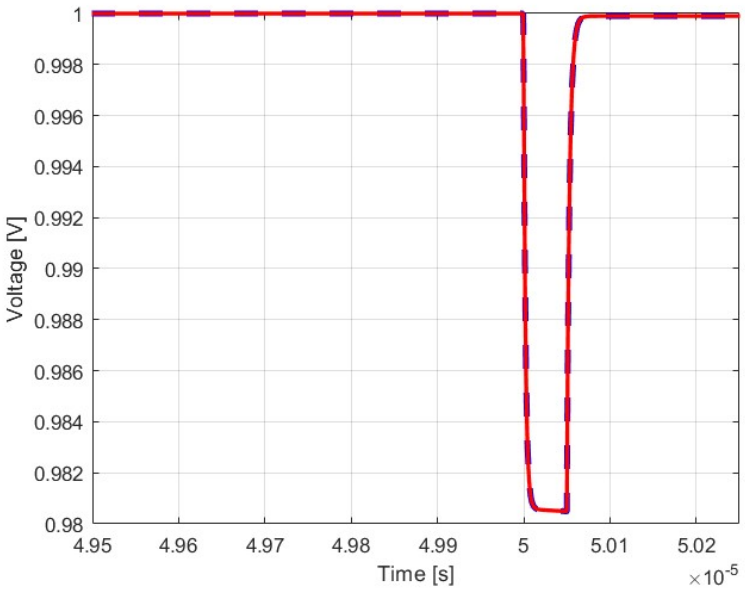


Figure 4.11: Comparison between exact case and Picard by Matlab-full PDN with no decap

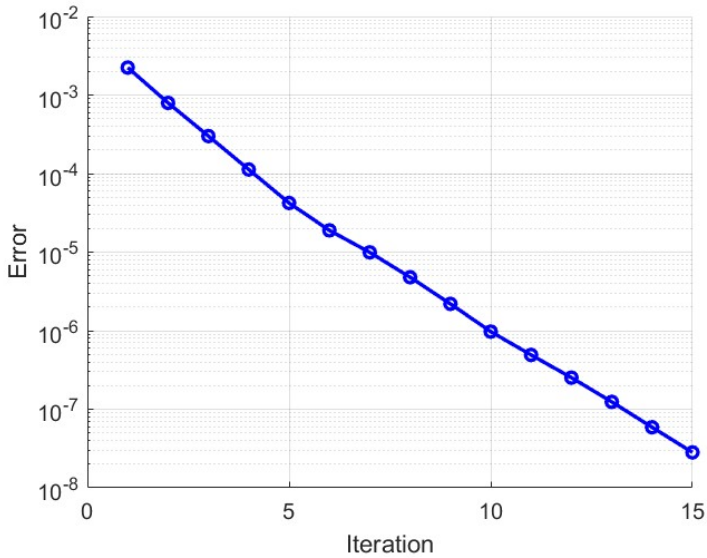


Figure 4.12: Error vs Reference simplified model Picard

PDN with decoupling capacitors

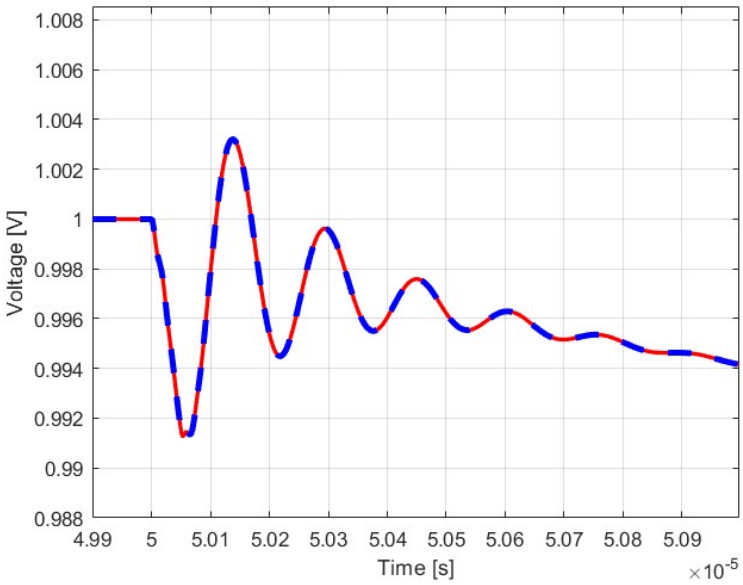


Figure 4.13: Comparison between exact case and Picard by Matlab- PDN with decaps

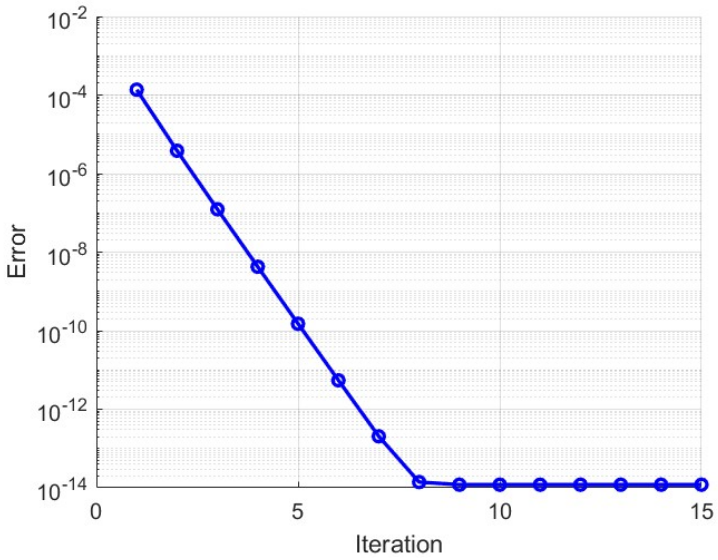


Figure 4.14: Error vs Reference full PDN Picard

PDN with extra decoupling capacitors

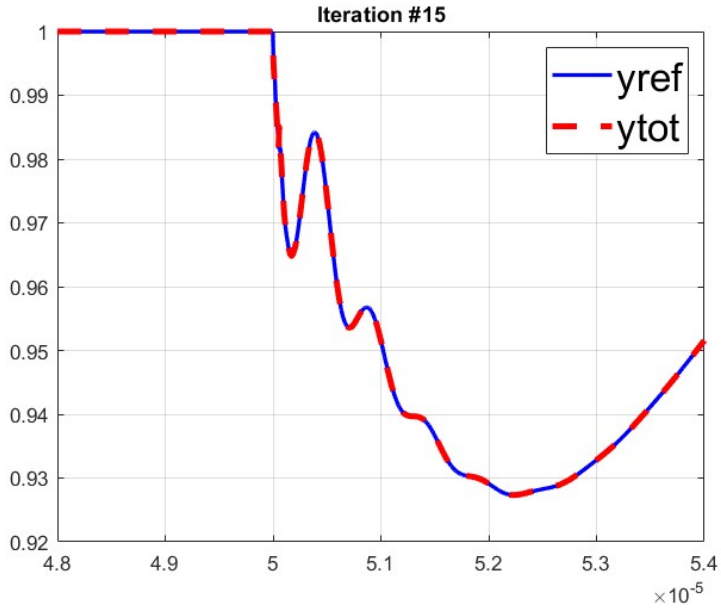


Figure 4.15: Comparison between exact case and Picard by Matlab- PDN with extra decaps

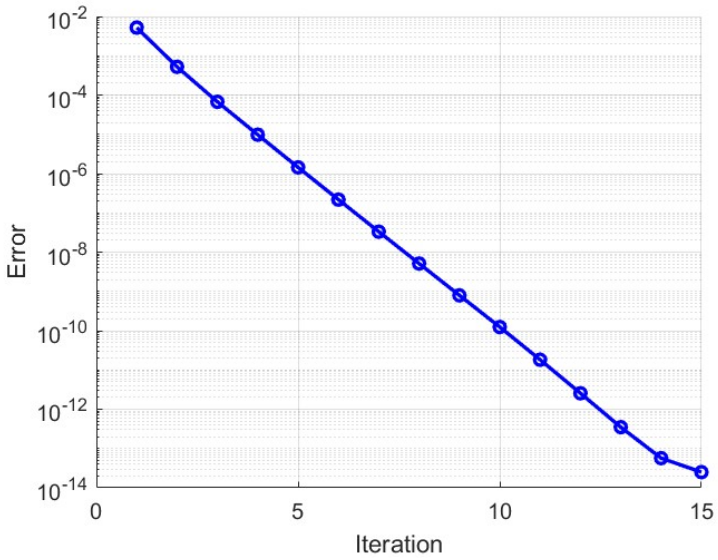


Figure 4.16: Error vs Reference full PDN extra decap Picard

PDN with no decoupling capacitors

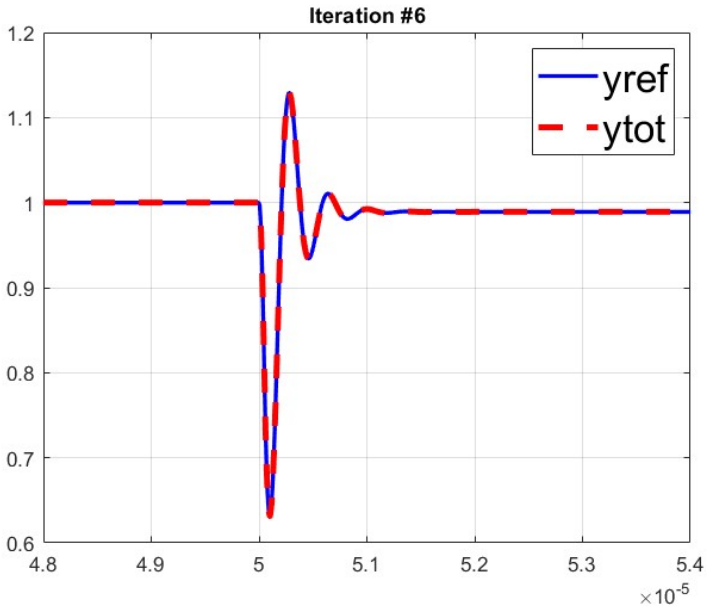


Figure 4.17: Comparison between exact case and Picard by Matlab- PDN with no decaps

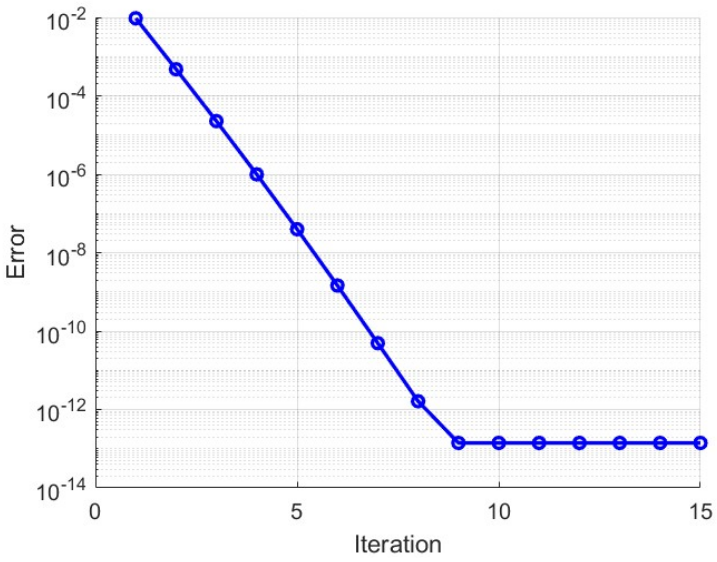


Figure 4.18: Error vs Reference PDN with no decaps Picard

From these graphs, it is evident that the method is performing well. This conclusion is supported by the close match between the output voltage obtained using the direct method and the results generated by the Picard method. This result is further verified by examining the error graphs, which demonstrate a minimal deviation between the two approaches as the iteration increases. In addition, the error tends to decrease more rapidly with the Picard method than with WR. Notably, in the case of the simplified PDN, while the WR method initially exhibited a hyperbolic trend in error reduction, the Picard method demonstrated a little more linear progression but was quite similar. Another observation is that in cases where the PDN lacked decoupling capacitors—where WR struggled to converge—the Picard method successfully resolved this issue. Notably, a considerable speed of convergence was achieved even under these challenging conditions. Overall, it is clear that the Picard method performs exceptionally well for simple PDNs, but its versatility extends beyond this. As previously mentioned, the Picard method incorporates a control parameter that can further accelerate convergence by appropriately modifying the system. These results demonstrate that the divergence problem encountered with WR is no longer present when using the Picard method. The error converges swiftly toward machine precision, establishing an excellent foundation for more advanced simulations that will be discussed in later sections.

Once it was confirmed that the Picard method converges without the issues seen in the Waveform Relaxation (WR) approach, the next step involved investigating how variations in the control parameter g_{shift} influence the convergence rate. This analysis was conducted across all types of PDN configurations examined during our preliminary tests, and the results are illustrated in the figures below.

- Comparing error against reference while varying $gshift$

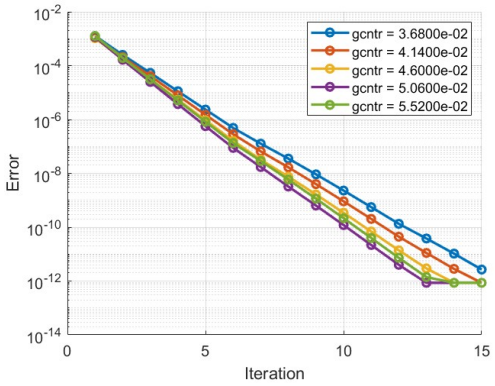


Figure 4.19: Error vs Reference simplified model varying g_{shift} Picard

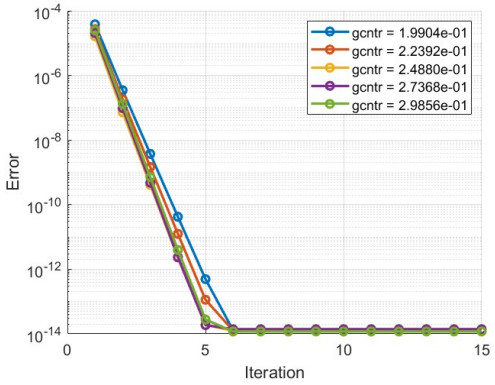


Figure 4.20: Error vs Reference full PDN varying g_{shift} Picard

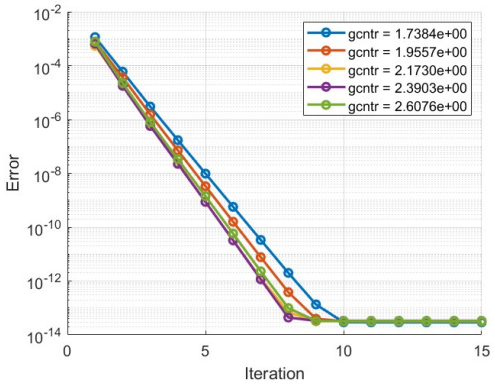


Figure 4.21: Error vs Reference full PDN extra decap varying g_{shift} Picard

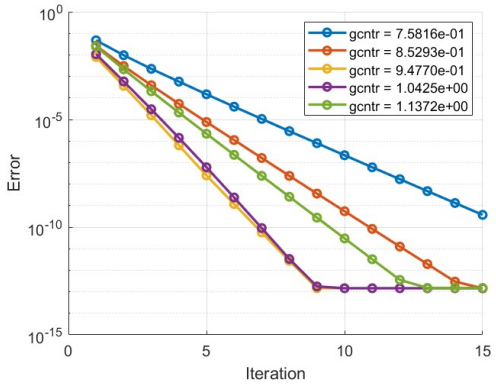


Figure 4.22: Error vs Reference PDN with no decaps varying g_{shift} Picard

Examining the graphs reveals how the convergence behavior is affected by adjustments to the control parameter g_{shift}. Specifically, in configurations without decoupling capacitors, variations in this parameter lead to significant fluctuations in the convergence rate. In contrast, networks equipped with decoupling capacitors demonstrate more stable convergence that improves gradually. This observation suggests that systems lacking decoupling capacitors are more sensitive to the selection of g_{shift}. This led us to consider whether there exists an optimal value for g_{shift} that could significantly enhance both accuracy and speed in our simulations. To investigate this further, it was conducted an additional analysis where g_{shift} was varied and recorded the number of iterations required to achieve our target accuracy.

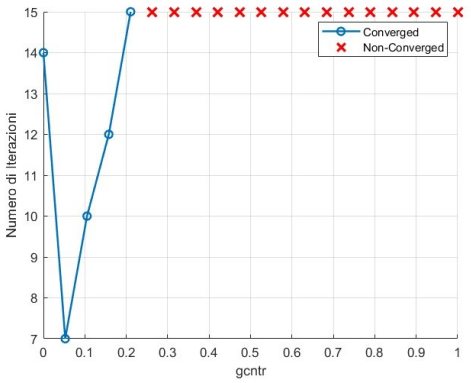


Figure 4.23: Simplified model step vs $gshift$ Picard

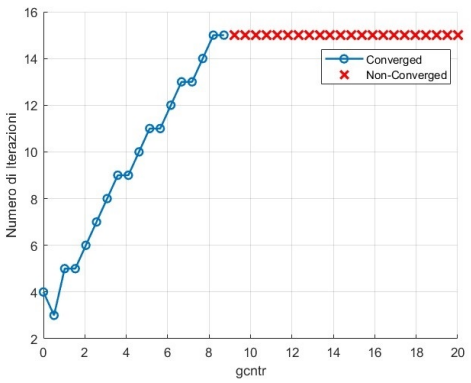


Figure 4.24: Full PDN step vs $gshift$ Picard

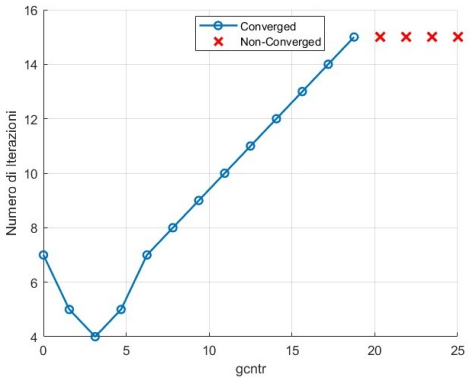


Figure 4.25: Full PDN extra decap step vs $gshift$ Picard

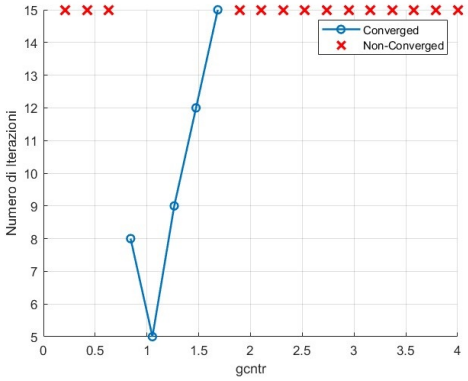


Figure 4.26: PDN with no decaps step vs $gshift$ Picard

These graphs highlight the importance of carefully selecting the value of g_{shift} to achieve rapid convergence with the method. Conversely, an inappropriate choice of this parameter can hinder reaching the desired accuracy. It is particularly interesting to note that each graph reveals a well-defined minimum point, confirming the existence of an optimal g_{shift} value that significantly accelerates convergence. Each network configuration appears to have its unique optimal point. In particular, it was observed that as the control parameter moves away from the case without any shift ($g_{\text{shift}} = 0$), the number of iterations required to achieve the desired accuracy increases substantially, especially for PDN configurations lacking decoupling capacitance. On the other hand, for networks equipped with decoupling capacitors, the choice of g_{shift} becomes less critical, resulting in more stable convergence that is less sensitive to variations in this parameter. These findings are quite promising, suggesting that by developing more advanced techniques to determine the optimal g_{shift} value in future research, it may be possible to minimize the number of iterations needed to achieve the desired accuracy, thereby optimizing the simulation process. Once this optimal value is identified, the impedance behavior of the PDN and the optimal g_{shift} value can be represented in a single graph to explore the relationship between these two parameters. Additionally, further graphs were generated to examine the behavior of the poles as g_{shift} varied. Below are the results of this analysis.

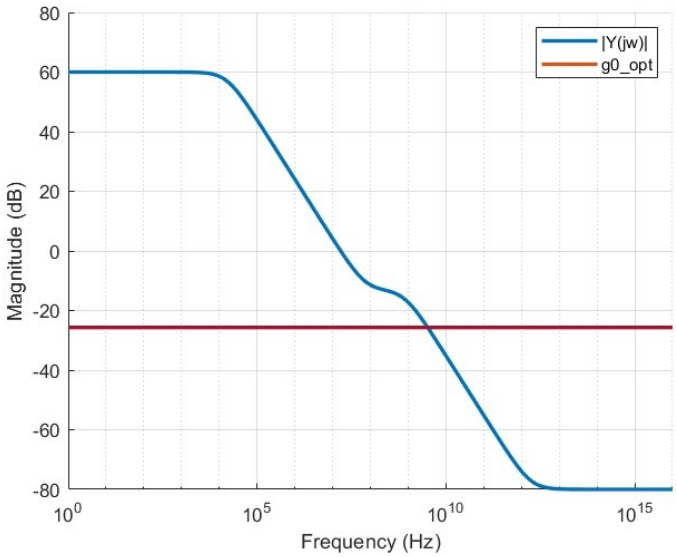


Figure 4.27: Simplified model Admittance vs $gshift$ Picard

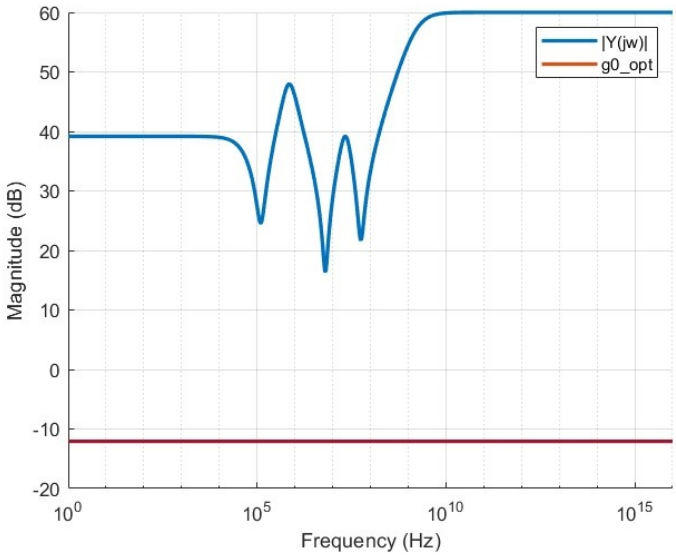


Figure 4.28: Full PDN Admittance vs $gshift$ Picard

The graphs illustrate how varying the values of g_{shift} causes the system's poles to shift, as expected. Additionally, a series of g_{shift} values are presented alongside the optimal case. This allows us to see how the system has been modified to achieve accuracy more quickly. Below is reported a simplified table that summarizes the

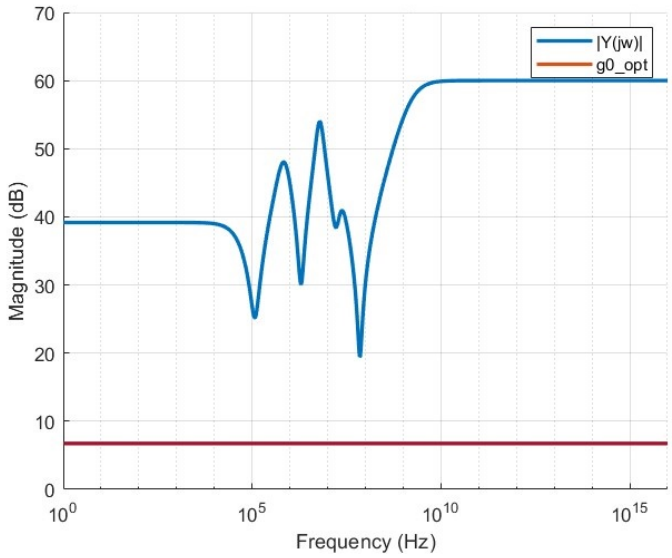


Figure 4.29: Full PDN extra decap Admittance vs gcntrl Picard

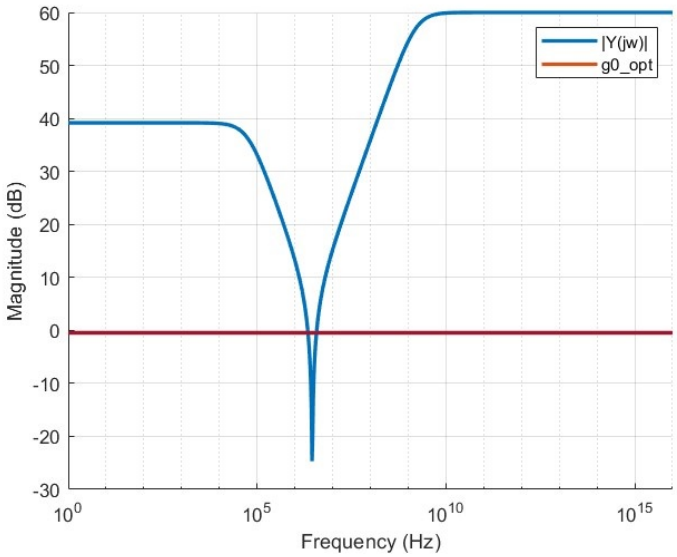


Figure 4.30: PDN with no decaps Admittance vs gcntrl Picard

accuracies achieved by the two methods.

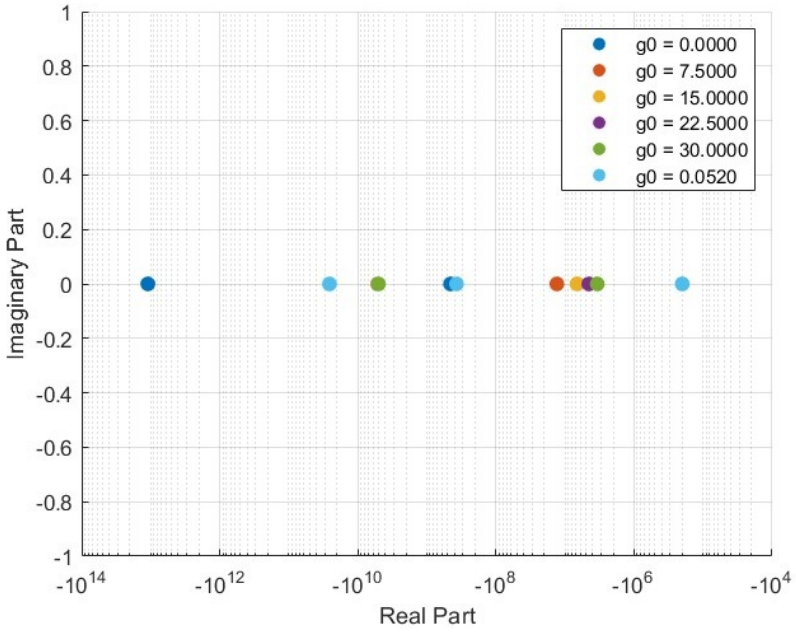


Figure 4.31: simplified model poles vs gcntrl Picard

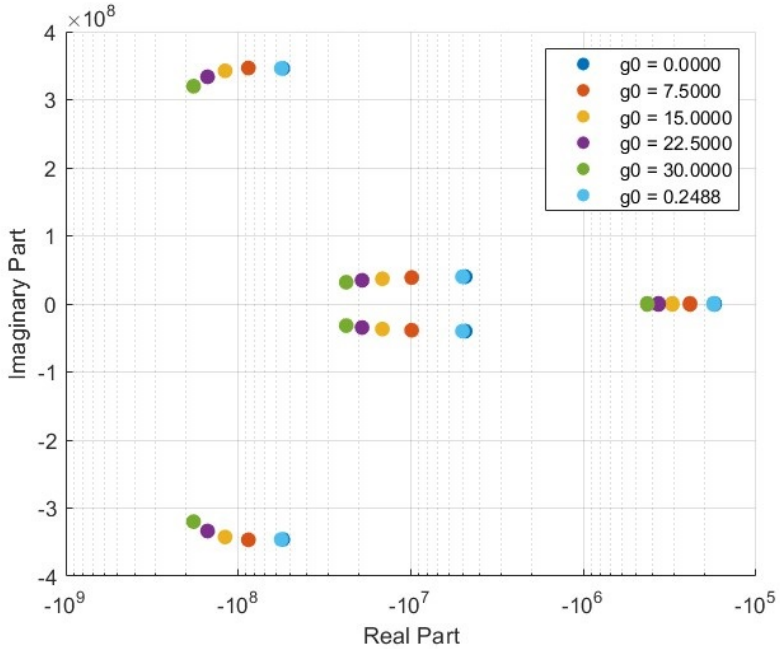


Figure 4.32: Full PDN poles vs gcntrl Picard

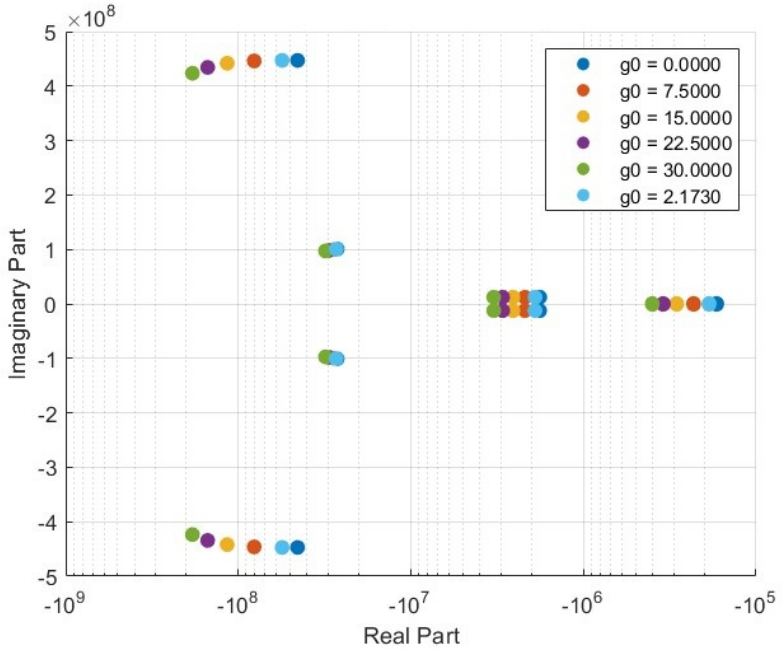


Figure 4.33: Full PDN extra decap poles vs gcntrl Picard

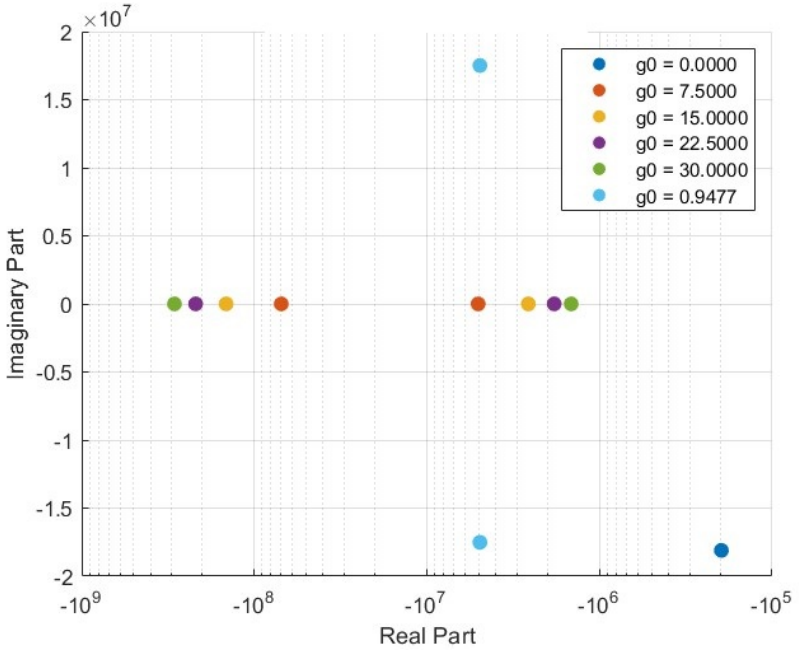


Figure 4.34: PDN with no decaps poles vs gcntrl Picard

PDN Analyzed	WR Method	Picard Method
Simplified Model	$5.7 \cdot 10^{-7}$	$9.9 \cdot 10^{-12}$
Full PDN	$3 \cdot 10^{-9}$	$1.1 \cdot 10^{-14}$
Full PDN with Extra Decap	$5.8 \cdot 10^{-7}$	$2 \cdot 10^{-14}$
PDN without Decap	*not converge*	$1.5 \cdot 10^{-13}$

Table 4.1: Comparison of accuracies between WR and Picard methods for the analyzed PDNs

Now that the method has been shown to converge and an optimal value for g_{shift} has been identified for faster convergence, attention can shift to a more realistic scenario.

4.2.2 Multiport case 2 core

After analyzing the effectiveness of the Picard method in relatively simple contexts, attention can now shift to examining more complex cases. To begin, a two-core network configuration was considered. Each core is equipped with 57 differential ports, resulting in a multi-port system with a total of 114 outputs per core. In other words, the transition is from a single-port system to a multi-port one, ultimately managing a total of 228 outputs. This transition helps to underscore the importance of ensuring accuracy in the results while striving to achieve maximum precision in the shortest time possible. Additionally, it highlights the challenges that arise in designing these cores, particularly in ensuring that the power delivery network functions correctly and efficiently supplies power. To address these challenges, the detailed system formulation described in Chapter 3 will be referenced. The first step is to verify, as done previously, that the Picard method converges. For this reason, an analysis was conducted without applying any shift. The results obtained are presented below.ù

First of all, the exact representation of the PDN behavior is needed. Then, it

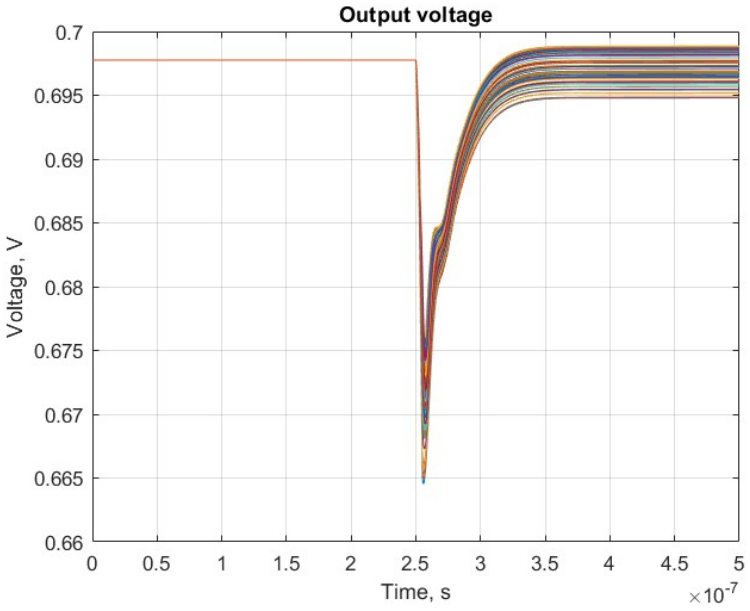


Figure 4.35: Output voltage reference - exact case

was applied the Picard Method without considering the control parameter.

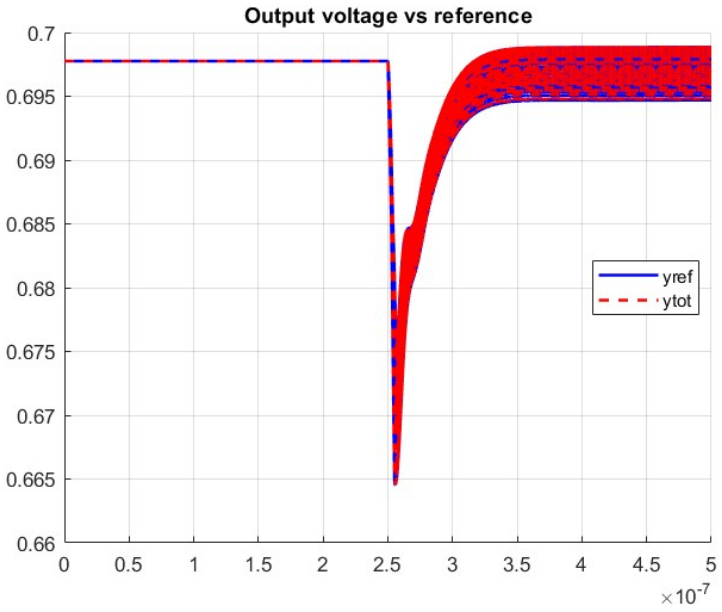
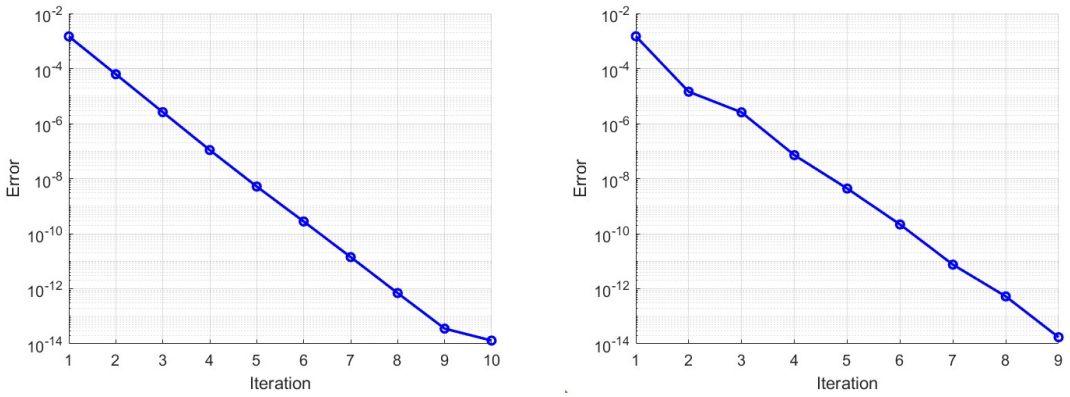


Figure 4.36: Comparison output voltage - 2 core case

As expected, the method converged 4.36, and that was confirmed by the error graph depicted in figure 4.37.

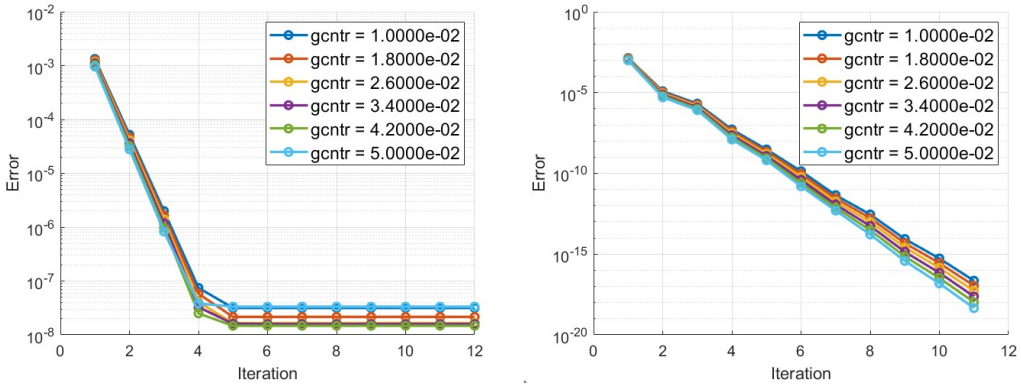


a) Error vs Reference #2 core test without shift

b) Error vs Previous iteration #2 core test without shift

Figure 4.37: Case without shift

The figures presented illustrate the results of the Picard method applied to the multi-port two-core case without any shifts. The first observation highlights the error behavior, which aligns perfectly with our expectations. Specifically, there is a significant reduction in error as the number of iterations increases. It is not only important to confirm that the system converges but also to evaluate how the error changes relative to the previous iteration. Notably, after just a few iterations —specifically 3 — an accuracy on the order of 1×10^{-6} was achieved, which is more than sufficient to confirm the method's convergence. Once it was established that the method converges, attention turned to examining how variations in the control parameter g_{shift} affect its performance. This step is crucial as it aims to determine whether an optimal value of g_{shift} exists that could expedite the process and further enhance the method's efficiency. Given the preliminary cases indicating that well-designed PDNs lead to faster convergence times, the effect of the parameter g_{shift} was explored by selecting smaller values. To achieve this, an analysis was conducted by sweeping the g_{shift} parameter around zero to observe how it influences the system's convergence. As illustrated in the graph, the system's convergence speed varies both positively and negatively, showing an improvement in accuracy during the initial steps. However, after a certain point, the slope of the curve reaches a "plateau," where the reduction in error slows down. Ideally, this plateau should not exist, as the system is expected to approach machine precision theoretically. This behavior suggests that numerical errors may be preventing the method from achieving MATLAB's machine precision. To investigate this issue



a) Error vs Reference #2 core test with shift

b) Error vs Previous iteration #2 core test with shift

Figure 4.38: Case with shift

further, a more detailed analysis was conducted to identify the source of the error. It was found that poor conditioning of the system could be one of the contributing factors to this behavior.

Matrix Conditioning Analysis

As mentioned in Chapter 2, when using software like MATLAB, various errors can arise due to several factors. One of the most common and easily identifiable issues relates to matrix conditioning. During the simulations, particular attention was given to the conditioning of the matrices involved in the system, especially focusing on matrix A_c. This matrix is crucial for calculating the initial conditions. Additionally, another critical aspect is the resulting matrix M_{result}, which must be inverted during the solution process (as outlined in Chapter 3, Section 3.3.4).

$$M_{\text{result}} = \hat{E} - dt \cdot A_c.$$

The analysis revealed that the time step plays a crucial role in the conditioning of the system and, consequently, in the numerical performance of the model. A larger time step tends to amplify the conditioning, making the system more numerically unstable. Additionally, the conditioning of the matrices is also significant. To mitigate the negative effects associated with these aspects, MATLAB's `equilibrate` function for matrix balancing was used, which enhances numerical stability and facilitates the solution of the system by rescaling the matrix to reach a better conditioning. To illustrate the effects of the adjustments made, two different cases are presented, with a high shift value chosen to clearly differentiate this behavior.

For the first case, the scenario with $g_{\text{shift}} = 0$ was considered and two analyses were conducted. The first analysis involved the conditioning of the matrices A_c, and then the resulting matrix $M_{\text{risultante}}$ for two different values of dt: one less accurate and the other more accurate, since only the latter depends on dt. Then, the `equilibrate` function was applied to all three cases to observe how the conditioning changed.

case 1

$$\begin{array}{rl} \text{cond}(A_c) & = 1.402 \cdot 10^{25}, \\ \text{cond}(M_{\text{result}})\Big|_{dt=5\cdot10^{-10}} & = 7.18 \cdot 10^{23}, \\ \text{cond}(M_{\text{result}})\Big|_{dt=5\cdot10^{-11}} & = 9.189 \cdot 10^{21} \end{array} \xrightarrow{\texttt{equilibrate()}} \begin{array}{l} = 5.313 \cdot 10^{12}, \\ = 1.363 \cdot 10^{11}, \\ = 1.67 \cdot 10^{10} \end{array}$$

case 2

$$\begin{array}{rl} \text{cond}(A_c) & = 1.04 \cdot 10^{26}, \\ \text{cond}(M_{\text{result}})\Big|_{dt=5\cdot10^{-10}} & = 1.953 \cdot 10^{23}, \\ \text{cond}(M_{\text{result}})\Big|_{dt=5\cdot10^{-11}} & = 1.14 \cdot 10^{22} \end{array} \xrightarrow{\texttt{equilibrate()}} \begin{array}{l} = 2.745 \cdot 10^{13}, \\ = 1.364 \cdot 10^{11}, \\ = 1.67 \cdot 10^{10} \end{array} \quad (1)$$

The results obtained from the analyses conducted on the two scenarios, one with $g_{\text{shift}} = 0$ and the other with $g_{\text{shift}} = 1$, illustrate the influence of the time step dt on the numerical behavior of the matrices A_c and M_{result}, focusing on the numerical conditioning of these systems.

Case 1: $g_{\text{shift}} = 0$

In the first case, the matrix A_c exhibits an extremely high condition number of $\text{cond}(A_c) = 1.402 \times 10^{25}$. This value indicates that the system represented by the matrix A_c is highly poorly conditioned, suggesting that small perturbations in data or numerical errors can lead to significant variations in the solution. When examining the resulting matrix M_{result} with a time step of $dt = 5 \times 10^{-10}$, the condition number is $\text{cond}(M_{\text{result}}) = 7.18 \times 10^{23}$, which is lower than that of A_c, but still quite high. By reducing the time step to $dt = 5 \times 10^{-11}$, there is an improvement in the condition number of M_{result}, which decreases to $\text{cond}(M_{\text{result}}) = 9.189 \times 10^{21}$. This demonstrates that using smaller time steps leads to greater numerical stability, allowing for more precise and reliable evaluations of solutions while mitigating rounding errors and other numerical inaccuracies, even though the values remain elevated. The implementation of MATLAB's equilibrate function significantly enhances the condition number. The condition number for matrix A_c improves from an order of magnitude around 10^{25} to approximately 10^{12}, while for the resulting matrix, it reduces by about eleven orders of magnitude. This results in a system with less poorly conditioned matrices. Although these condition numbers

are still relatively high, this approach substantially reduces potential errors, albeit only up to a certain extent.

Case 2: $g_{\text{shift}} = 1$

In the second case, the matrix A_c has a condition number of $\text{cond}(A_c) = 1.04 \times 10^{26}$. This value is extremely high, indicating a severe ill-conditioning of the matrix. The resulting matrix M_{result}, calculated with a time step of $dt = 5 \times 10^{-10}$, has a condition number of $\text{cond}(M_{\text{result}}) = 1.953 \times 10^{23}$. When further reducing the time step to $dt = 5 \times 10^{-11}$, the condition number for the resulting matrix decreases to $\text{cond}(M_{\text{result}}) = 1.14 \times 10^{22}$. This further improvement highlights the importance of using smaller time steps, which contribute to greater numerical stability and reduced error. In this case as well, applying the equilibrate function to matrix A_c results in a significantly lower condition number of $\text{cond}(A_c) = 2.745 \times 10^{13}$. Although this value remains high, it is considerably lower than what was obtained without using this function, suggesting that it has positively impacted the numerical conditioning of the matrix. For the resulting matrix M_{result} calculated with a time step of $dt = 5 \times 10^{-10}$, its condition number further decreases to $\text{cond}(M_{\text{result}}) = 1.364 \times 10^{11}$, indicating a clear improvement compared to when no balancing was applied. This value suggests that using equilibrate has had a beneficial effect on enhancing the system's numerical stability and reducing risks associated with ill-conditioning. Finally, by further reducing the time step to $dt = 5 \times 10^{-11}$, the condition number for M_{result} drops again to $\text{cond}(M_{\text{result}}) = 1.67 \times 10^{10}$. This additional improvement confirms that employing smaller time steps alongside matrix balancing significantly enhances numerical stability and solution accuracy. Overall, the results obtained for both cases, with $g_{\text{shift}} = 0$ and $g_{\text{shift}} = 1$, demonstrate that the choice of time step plays a crucial role in determining the numerical conditioning of these systems. In Case 1, with $g_{\text{shift}} = 0$, although matrix A_c is highly poorly conditioned, using smaller time steps greatly improves numerical stability. In Case 2, with $g_{\text{shift}} = 1$, while the initial conditioning of matrix A_c remains high, notable improvements in the condition number for M_{result} are evident even when employing smaller time steps. In both scenarios, a clear trend emerges: smaller time steps lead to enhanced numerical stability, reducing condition numbers for resulting matrices and improving solution accuracy. While neither case achieves ideal conditioning levels for their resulting matrices, adopting smaller time steps significantly improves stability and reliability in numerical solutions while mitigating instability effects and enhancing result quality. Below are summarized results for both discussed cases:

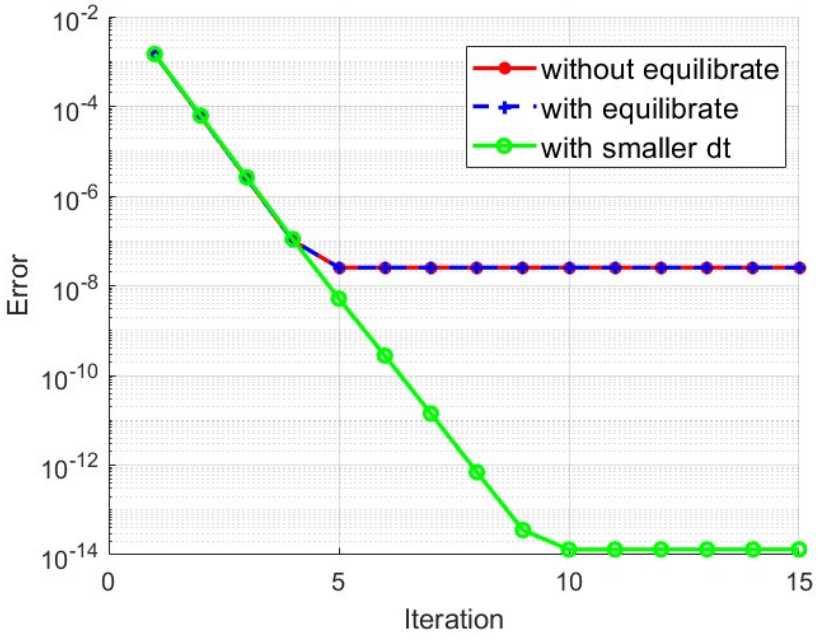


Figure 4.39: Error graphs for the case $g_{\text{shift}} = 0$

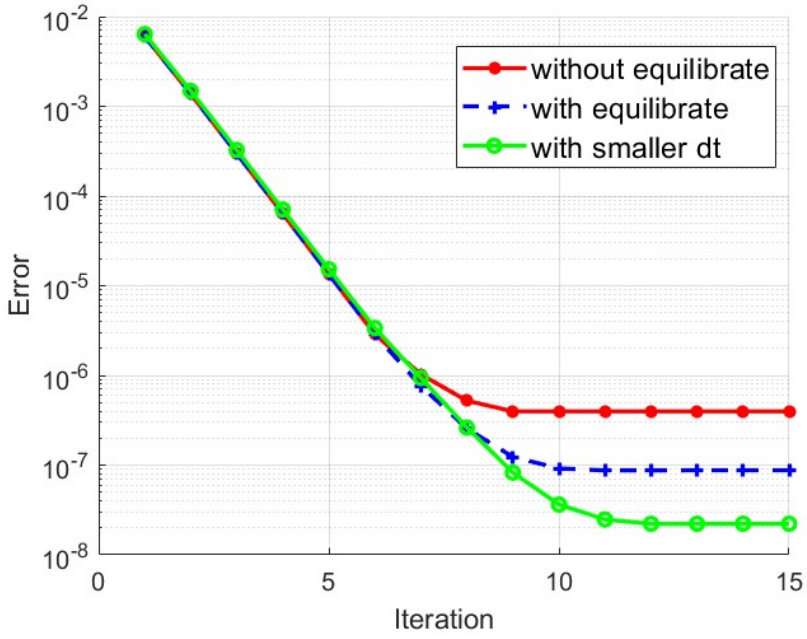


Figure 4.40: Error graphs for the case $g_{\text{shift}} = 1$

It can be observed that, unlike the case $g_{\text{shift}} = 1$, a good precision of the error with respect to the reference has been achieved. This result is primarily attributable to the fact that in this situation, the initial conditions did not experience significant numerical errors. This allowed for greater stability in the calculations and significantly reduced the impact of matrix conditioning errors, thanks to the use of the `equilibrate()` function and the appropriate choice of the time step dt. In the case of $g_{\text{shift}} = 1$, the situation changed due to the increased sensitivity of the system to the initial conditions, which will negatively impact accuracy. It is important to note that the matrix $\hat{E}$ exhibits a structural singularity. The introduction of a value of $g_{\text{shift}} > 0$ has rendered the matrix A_c poorly conditioned. Since in this case, the matrix A_c cannot be bypassed, the numerical error also reflects in the initial conditions, negatively impacting the results. The table reported in 4.2 illustrates all the data analized about the condition of the matrix under the use of the equilibration dt and gshift.

	No Equilibration		**Equilibration**		**Eq. + Small** dt	
	$g_{\textbf{shift}} = 0$	$g_{\textbf{shift}} = 1$	$g_{\textbf{shift}} = 0$	$g_{\textbf{shift}} = 1$	$g_{\textbf{shift}} = 0$	$g_{\textbf{shift}} = 1$
A_c	$1.4 \cdot 10^{25}$	$1.04 \cdot 10^{26}$	$5.31 \cdot 10^{12}$	$2.745 \cdot 10^{11}$	$5.31 \cdot 10^{12}$	$2.745 \cdot 10^{11}$
$M_{\textbf{result}}$	$7.18 \cdot 10^{23}$	$1.953 \cdot 10^{23}$	$1.363 \cdot 10^{11}$	$1.364 \cdot 10^{11}$	$1.67 \cdot 10^{10}$	$1.67 \cdot 10^{10}$
Accuracy	$2.54 \cdot 10^{-8}$	$3.973 \cdot 10^{-7}$	$3.52 \cdot 10^{-8}$	$8.76 \cdot 10^{-8}$	$6.67 \cdot 10^{-15}$	$2.1 \cdot 10^{-8}$

Table 4.2: Conditioning values of A_c and M_{result} for different approaches and g_{shift} values, along with the precision achieved in each configuration.

To minimize numerical errors, it was adopted a strategy of summing the smaller values before the larger ones when calculating the total error. This approach aimed to limit the impact of rounding errors and enhance overall accuracy in our computations.

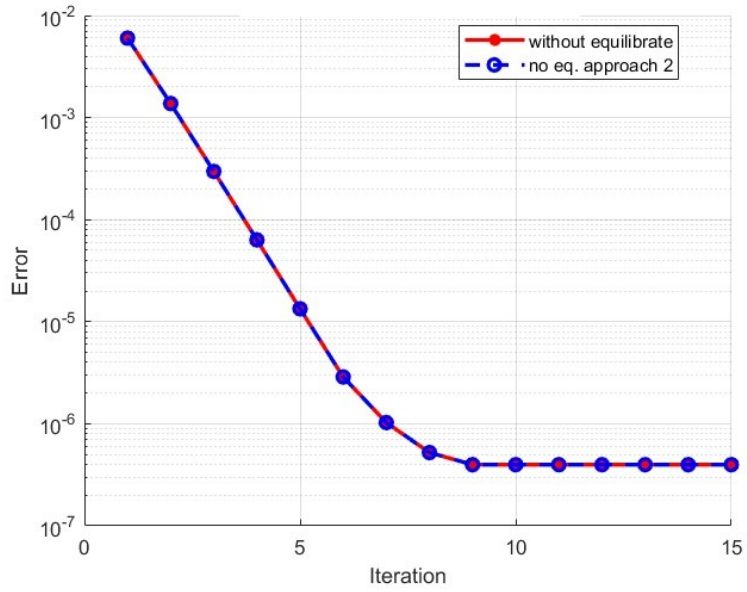
Sum of errors Inverse

By summing the ordered error vectors at each iteration, the vector y_{tot} is obtained. Unlike the previously used approach, where from previous sums were accumulated, this method constructs y_{tot} at each iteration starting from zero and summing the values of y_{niter} calculated in ascending order.
Below is reported a piece of code used in our simulation:

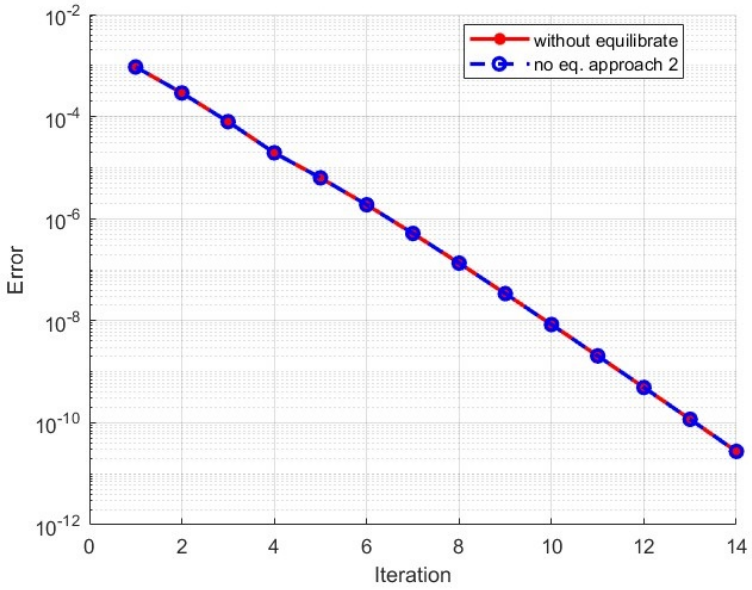
```
if niter > 1
    for idx = niter:-1:1
        % Sum the error from the smaller to the larger.
        ytot_acc = ytot_acc + y{idx};
    end
else
    % if first iteration
    ytot_acc = y{1};
end
```

In this setup, ytot_acc accumulates all the errors that occur between one iteration and the previous one, which will be used to calculate the final solution of the system.

The results illustrated in Figure 4.41 present the error graphs relative to the reference value (a) and the previous iteration (b). This test was conducted under conditions similar to those of the worst-case scenario shown in Figure 4.40, utilizing a large time step dt and without applying the `equilibrate()` function. Consequently, the observed trend in error remained unchanged from the initial findings, indicating that this test did not yield significant alterations in the results for the case of $g_{\text{shift}} = 1$. In the scenario with $g_{\text{shift}} = 0$, a worst-case analysis was also conducted under conditions identical to those previously discussed. The results, displayed in Figure 4.42, illustrate the error in relation to both the reference value and the previous iteration. Despite this thorough examination, the operations performed did not lead to any significant variations in the outcomes. This observation suggests that this particular aspect is not a critical factor in addressing the main problem and can therefore be excluded from further analysis. Overall, these insights emphasize that while various approaches were tested to enhance numerical stability and reduce errors, they did not yield significant differences in performance for

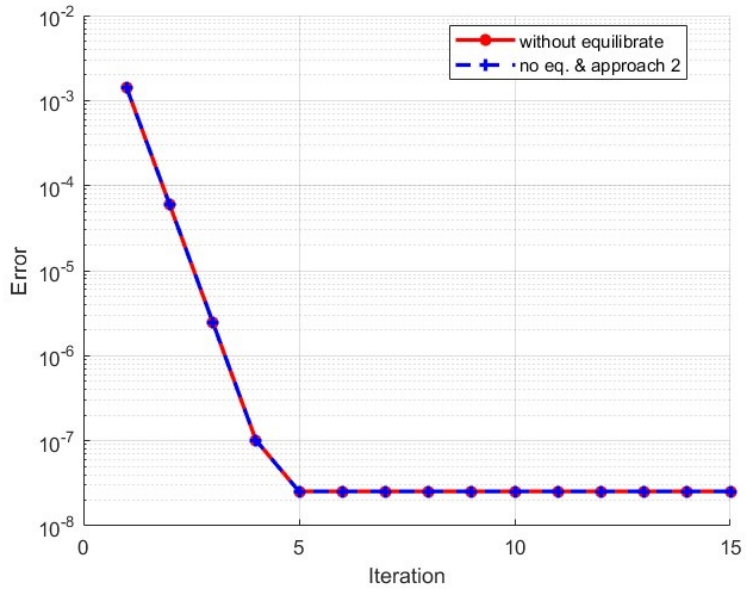


a) Error with respect to the reference: with shift

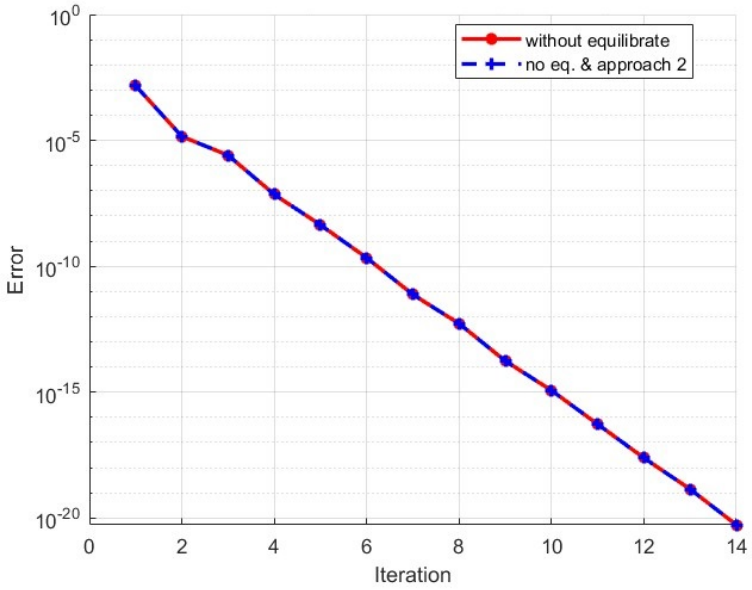


b) Error with respect to the previous iteration: with shift

Figure 4.41: Error graphs for the case $g_{\text{shift}} = 1$using different approach



a) Error with respect to the reference: no shift



b) Error with respect to the previous iteration: no shift

Figure 4.42: Error graphs for the case $g_{\text{shift}} = 0$ using different approach

either case. Consequently, it can be concluded that this specific method of error summation may not be essential for achieving the desired results.

Considerations of d_t

So far, it has been observed that using a small time step dt, which enhances accuracy, has led the system to converge towards lower values, effectively reducing the plateau observed in the case of $g_{\text{shift}} = 1$. However, this finding is somewhat counterintuitive because smaller dt values are generally expected to worsen numerical errors, especially when dealing with poorly conditioned matrices. While smaller time steps improve the analysis by making it closer to a continuous model, they can also increase numerical errors. To investigate this paradox further, a sweep of different dt values was conducted to determine if larger values could yield better convergence and vice versa. Below are the cases with $g_{\text{shift}} = 0$ and $g_{\text{shift}} = 1$.
It can be observed that, by using an increasingly smaller time step, the system does not necessarily tend to reach the reference with greater accuracy. As highlighted by the graphs, some larger values of dt may result in a smaller error compared to smaller values. The same analysis was conducted using the `equilibrate()` function.
It can be observed that, even in this case, some smaller values of dt cause the error to saturate at higher levels compared to larger values of dt. In the case of $g_{shift} = 1$, this result is even more evident.

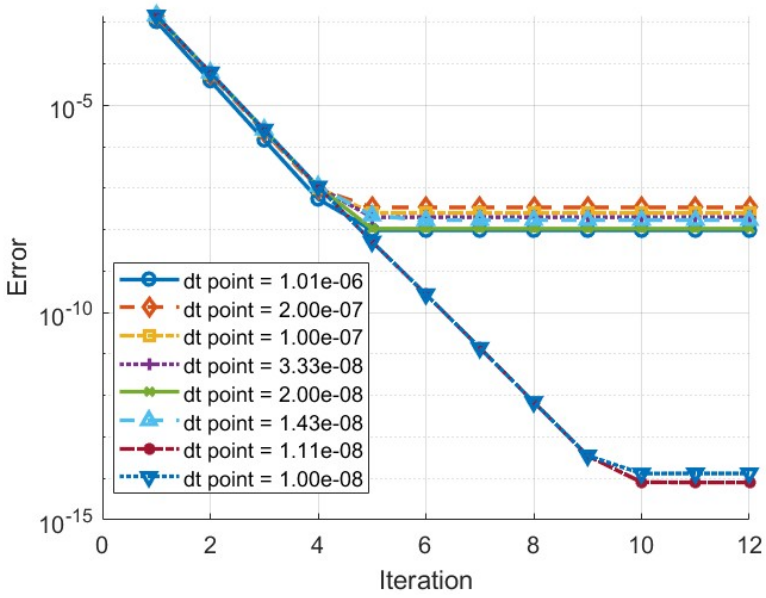


Figure 4.43: Error as a function of d_t for the case $g_{\text{shift}} = 0$ without the use of `equilibrate()`.

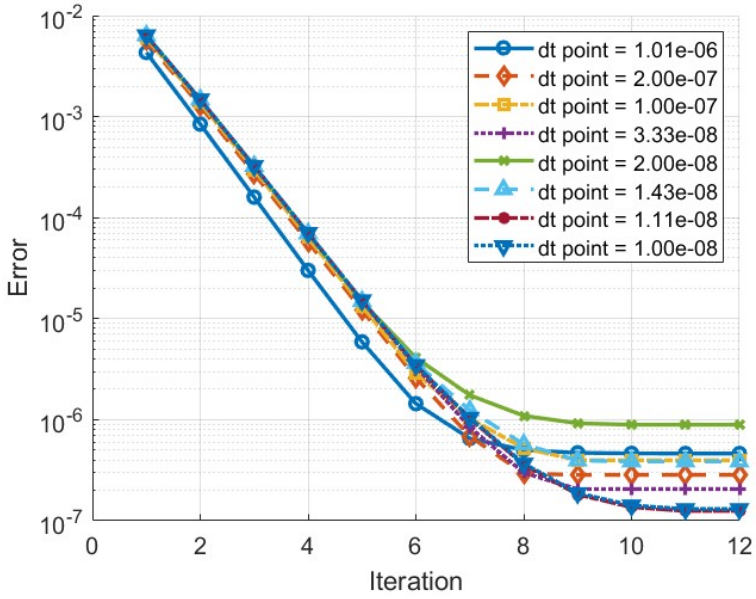


Figure 4.44: Error as a function of d_t for the case $g_{\text{shift}} = 1$ without the use of `equilibrate()`.

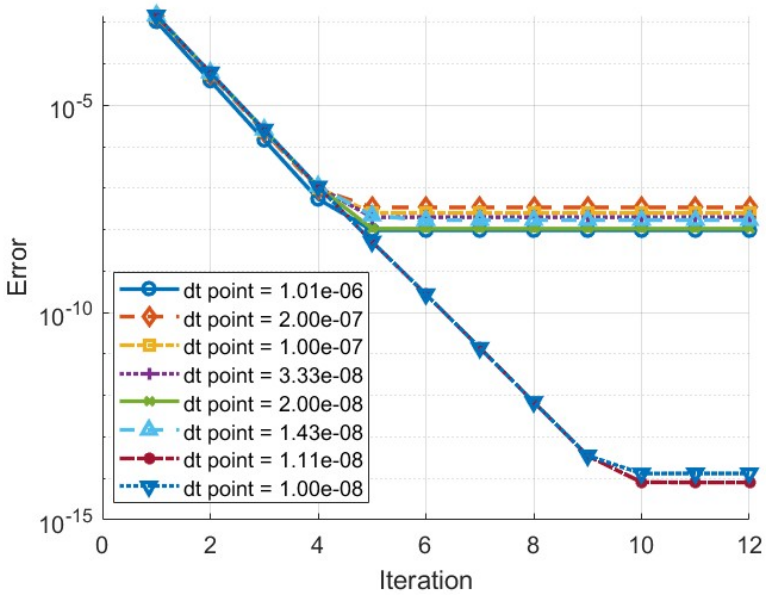


Figure 4.45: Error as a function of d_t for the case $g_{\text{shift}} = 0$ with the use of `equilibrate()`.

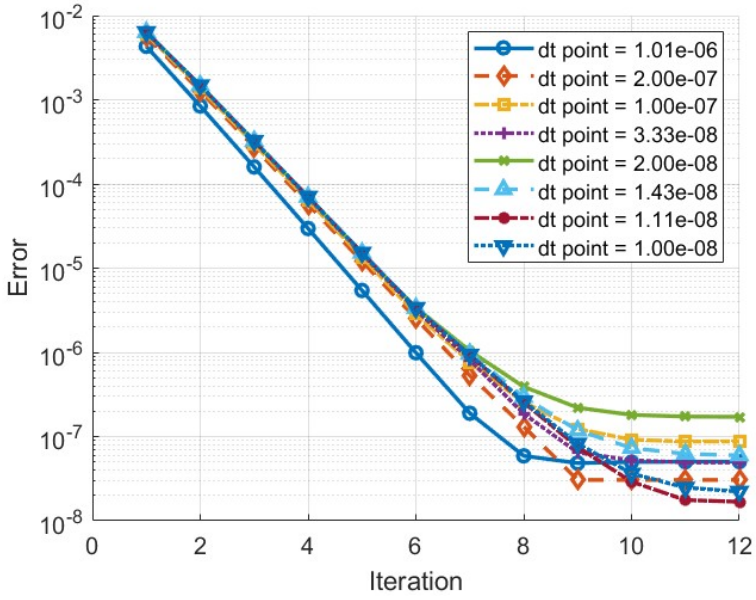


Figure 4.46: Error as a function of d_t for the case $g_{\text{shift}} = 1$ with the use of `equilibrate()`.

Optimal gshift

So far, the analysis has focused on the performance of the method in relation to machine precision. However, it is essential to clarify that the primary goal is not necessarily to achieve machine precision. Instead, the aim is to optimize accuracy during the initial iterations of the process. The specific objective is to identify the value of g_{shift} that allows for reaching a target precision, such as 1×10^{-5}, in the fewest iterations possible.
To achieve this, a sweep across various values of g_{shift} was conducted, similar to the preliminary phases. The intent was to identify which value would most efficiently reduce error while achieving the desired precision. The results of this investigation are illustrated in Figure 4.47, revealing seven potential values of g_{shift} that meet the target precision criteria.

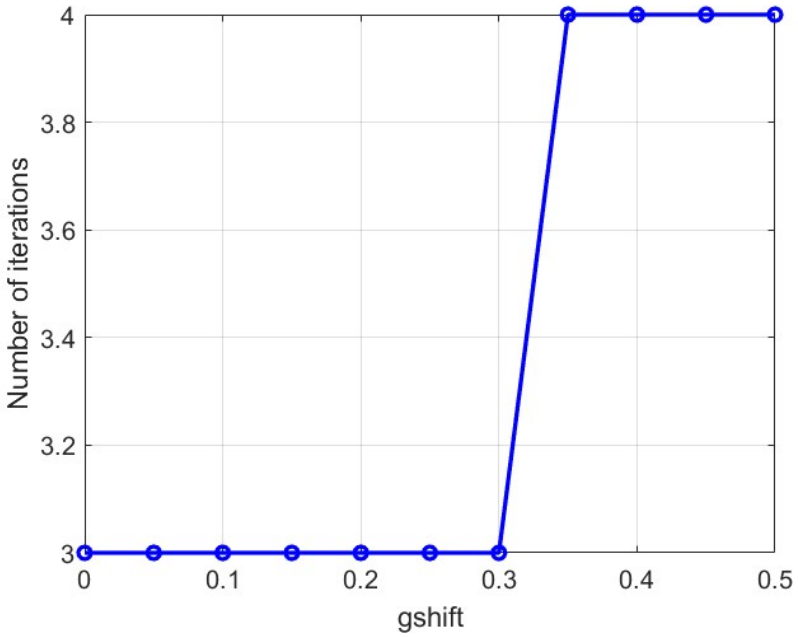


Figure 4.47: Step optimal gshift 2-core cases

However, simply reaching this precision is not sufficient; it is also crucial to consider the accuracy associated with each value.
To address this aspect, the error linked to each g_{shift} value was evaluated, with results presented in Figure 4.48. This analysis indicates that while multiple values of g_{shift} can achieve the target precision, one particular value stands out for its ability to minimize error and provide more reliable performance.

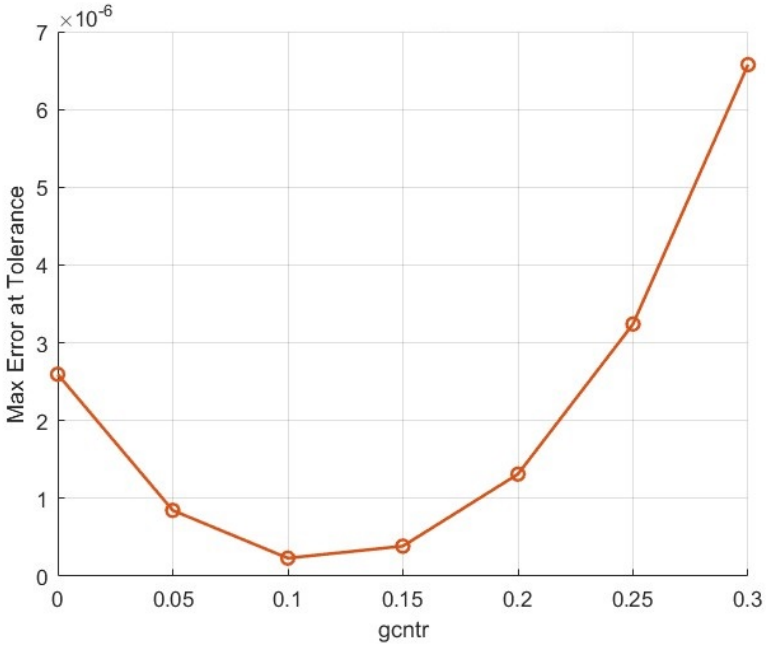


Figure 4.48: Error optimal gshift 2-core cases

Figure 4.49 presents a comparison between system performance when using the optimal g_{shift} value and when no shift is applied. This comparison highlights the significant impact of the optimal parameter. Specifically, utilizing $g_{\text{shift}} = 0.1$ enhances accuracy, particularly in the early iterations. By the fourth iteration, the error with the optimal shift drops to 1×10^{-8}, compared to approximately 1×10^{-7} without any shift. This improvement underscores the crucial role of the optimal g_{shift} in accelerating convergences. Now that it has been demonstrated that the method is effective and that an optimal value exists even for the two-core case, attention has shifted to a more comprehensive analysis involving a 60-core microprocessor.

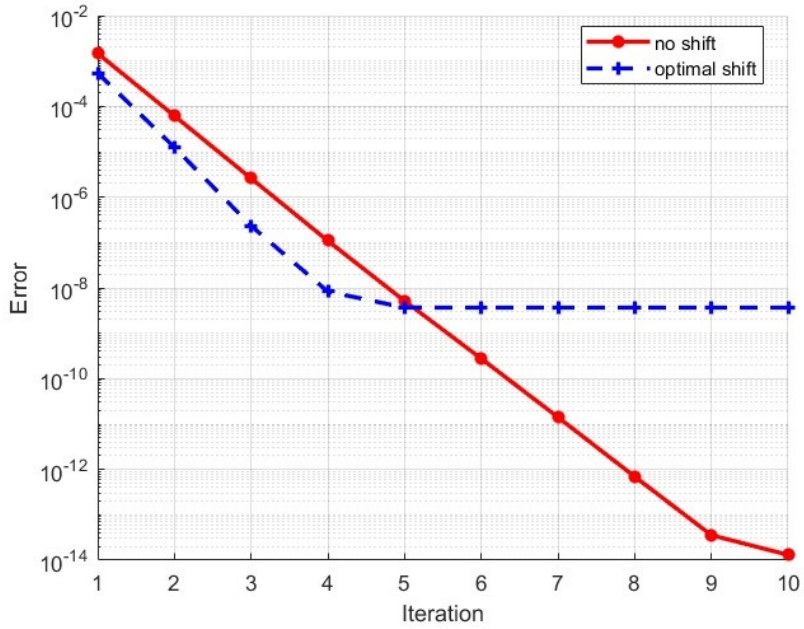


Figure 4.49: Comparison between optimal shift and no shift 2-core case

4.2.3 Multiport case 60 core

After successfully analyzing the performance of the method with two cores, where each core was equipped with 57 differential ports—resulting in a total of 228 outputs—the focus has now shifted to a multi-port system. This new configuration features 57 single-ended ports extended across 60 cores, bringing the total to an impressive 3,420 ports. Given this substantial increase in complexity, the demand for optimal reliability in the method becomes even more critical as the analysis now involves a significantly larger system. The first step in this expanded analysis was to verify that the method converged effectively without applying any shifts.

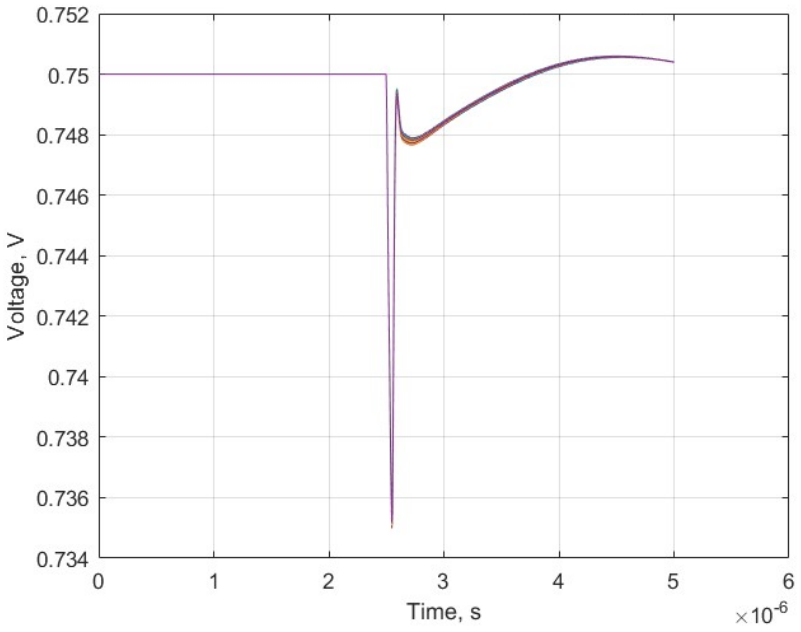


Figure 4.50: Output voltage reference - exact case: 60 core

For the 60-core case, the output voltage reference is reported in figure 4.50.

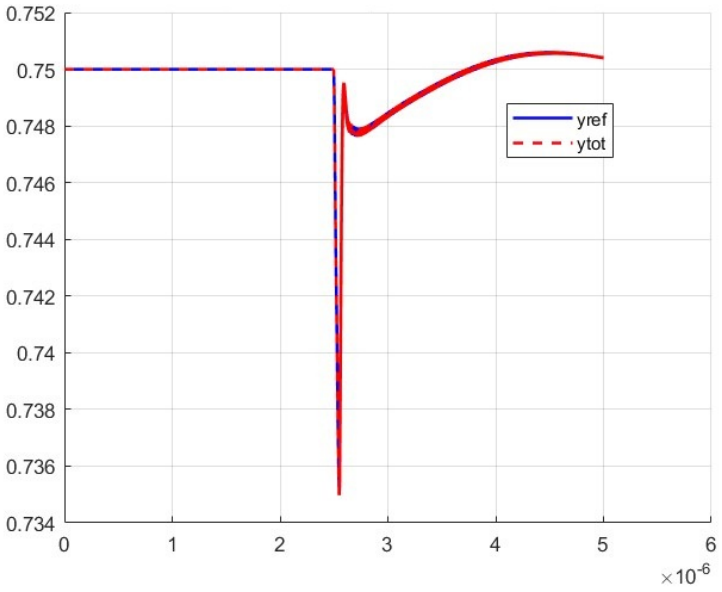


Figure 4.51: Comparison output voltage reference - 60 core

As shown in Fig. 4.51, the convergences of the method are reached, indeed the error graphs, see Fig. 4.52, reveal a perfectly linear trend, indicating that the error decreases as the number of iterations increases. This outcome aligns with expectations and suggests that the method is functioning positively. Following this successful verification, attention turned to examining the case where a shift was applied. It is also important to note that a thorough assessment of matrix conditioning was conducted. The results indicated that the conditioning of these matrices is significantly better compared to what was observed in the two-core scenario. Specifically, the condition numbers were recorded as follows:

$$\text{cond. } A_c = 1.2 \cdot 10^{15}$$
$$\text{cond. } M_{res} = 4.45 \cdot 10^{14}$$

With these improved conditioning values, it was anticipated that results would demonstrate continuous convergence similar to the previous case without shifts, as illustrated in Figure 4.52. Additionally, it was expected that the error would stabilize at a level significantly lower than what was observed in the two-core case, especially given that matrix conditioning improved from $1.04 \cdot 10^{26}$ to $1.2 \cdot 10^{15}$, representing a difference of 11 orders of magnitude. Despite these improvements, it is crucial to recognize that the system remains poorly conditioned, although not as severely as before. Consequently, the `equilibrate()` function was still utilized to enhance numerical stability.

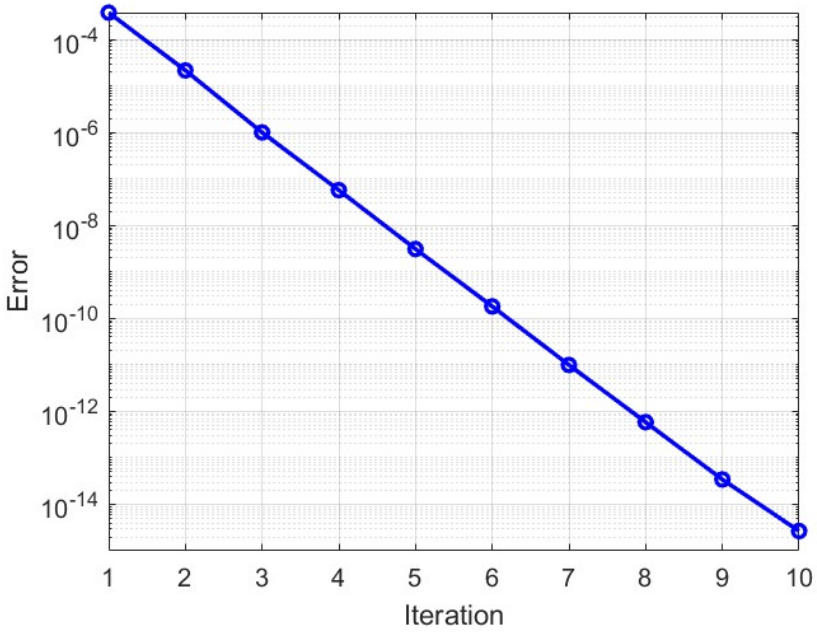


Figure 4.52: Errors Multiport case 60 core no shift

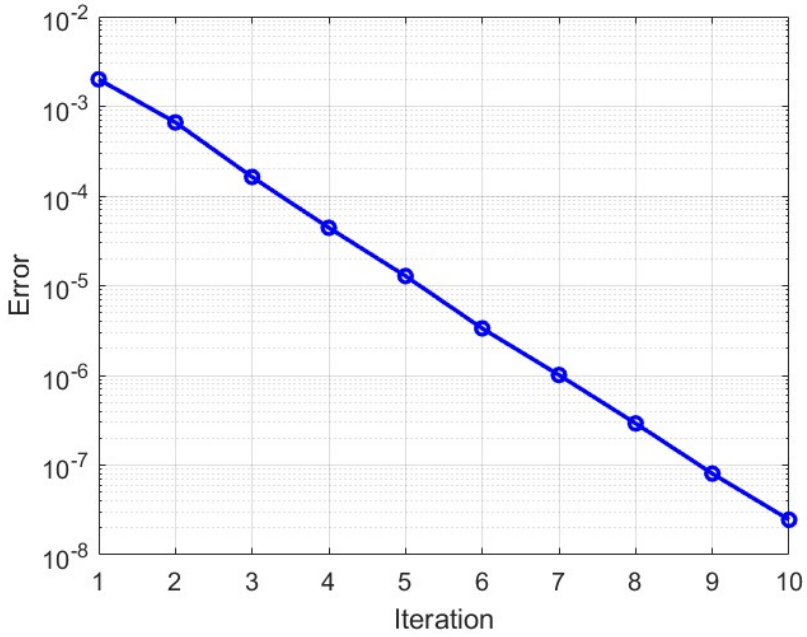


Figure 4.53: Errors for a Multiport 60 core with shift

This result confirms what I was anticipating; the trend of the graph, reflects what was seen in the case without a shift. As mentioned, due to the issues encountered in calculating the initial conditions, a less pronounced slope was evident. A comparison

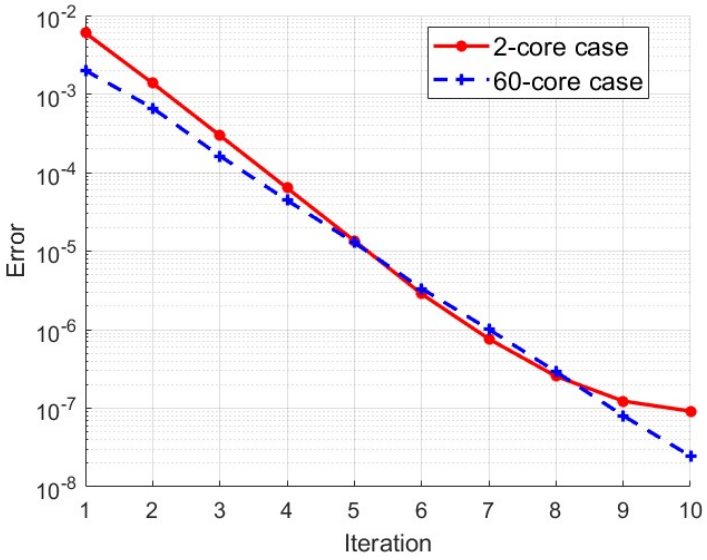


Figure 4.54: 60-core vs 2-core case with shift

between the two simulations further illustrated improved behavior resulting from enhanced system conditioning. The graphs clearly show how these advancements contribute to better overall performance.
As a counterproof, a sweep of the g_{shift} values was performed to calculate the conditioning of the matrix A_c to determine if there are any values for which the system is well-conditioned. The result is shown in Figure 4.55, from 0 to 11, it shows an almost constant trend, after which it increases linearly as g_{shift} increases.

4.2.4 Optimal gshift

As explained in the case of the 2-core system, the primary interest is about the value of g_{shift} that allows us to achieve a specified tolerance in the fewest number of iterations. Therefore, in this case, as well, a sweep of g_{shift} was conducted to determine the optimal value. As shown in the graph, a value was found for which an accuracy of less than 10^{-5} is achieved in just two iterations. The choice of the maximum number of iterations to reach this result was guided by the graph 4.52, as it presents a precision slightly above 1×10^{-5} at the second iteration, with the aim of lowering it further. Obviously, the third iteration would not have been necessary; a reduction of the target to 1×10^{-7} would have been required, but

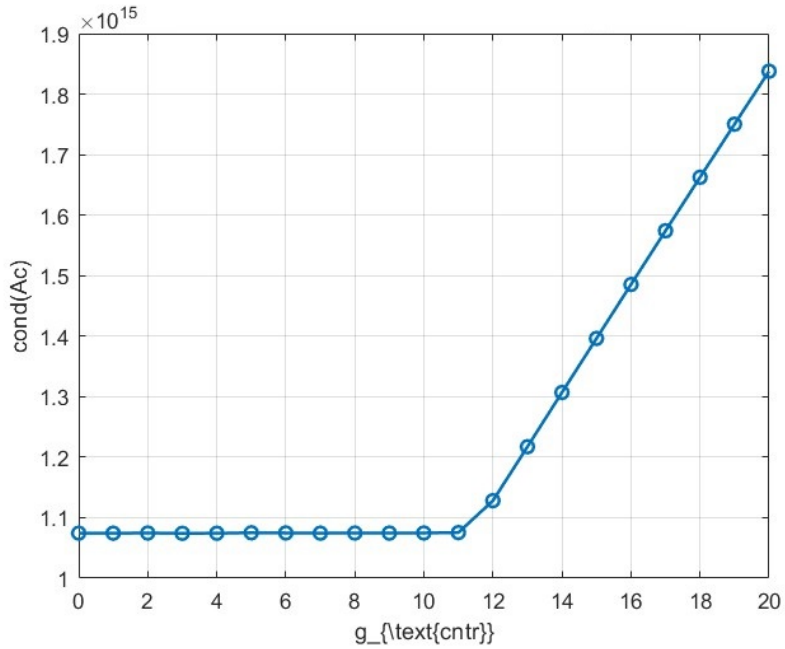


Figure 4.55: Conditioning of A_c vs g_{shift}

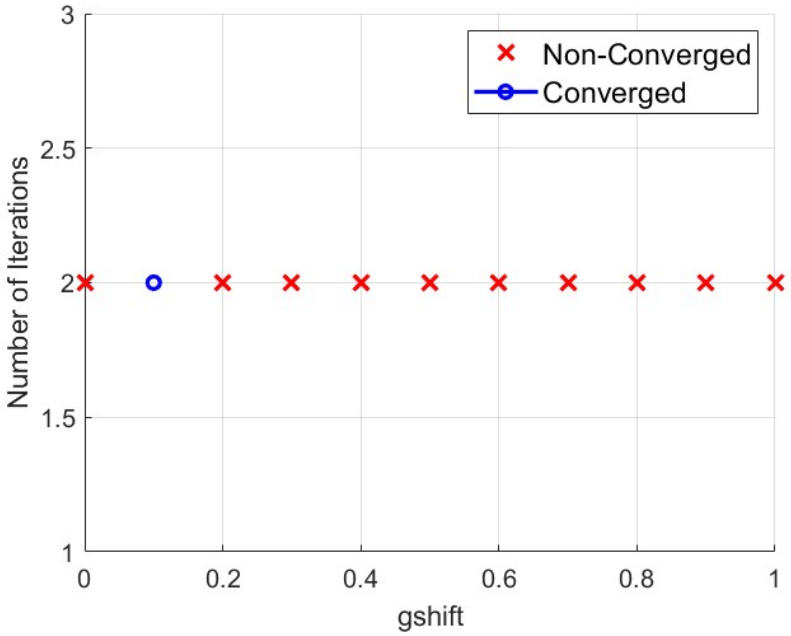


Figure 4.56: Optimal gshift 60-core cases

since 1×10^{-5} was more than sufficient, the decision was made to stop at that value. It is noticeable that there exists a value of g_{shift} that allows us to achieve this result. That value is 0.1. This experiment, confirms not only the validity of the code but also reinforces that optimizing shift parameters can significantly enhance performance across different configurations.
The graph below illustrates the comparison between the case without a shift and the one with the optimal shift. As can be observed, as the number of iterations increases, the error in the case with the shift decreases significantly compared to the case without any shift. This is an excellent outcome, as it shows that after just four iterations, it reached an accuracy that is two orders of magnitude better, meaning the error relative to the reference is now two times smaller. By the eighth iteration, this error reduces even further, reaching a decrease of three orders of magnitude. These results clearly confirm the critical role that g_{shift} plays in enhancing our analysis. The substantial improvement in accuracy demonstrates how effectively this parameter can optimize performance, allowing for quicker convergence and more reliable results. The findings underscore the importance of carefully selecting shift values to maximize efficiency and accuracy in the systems analysis.

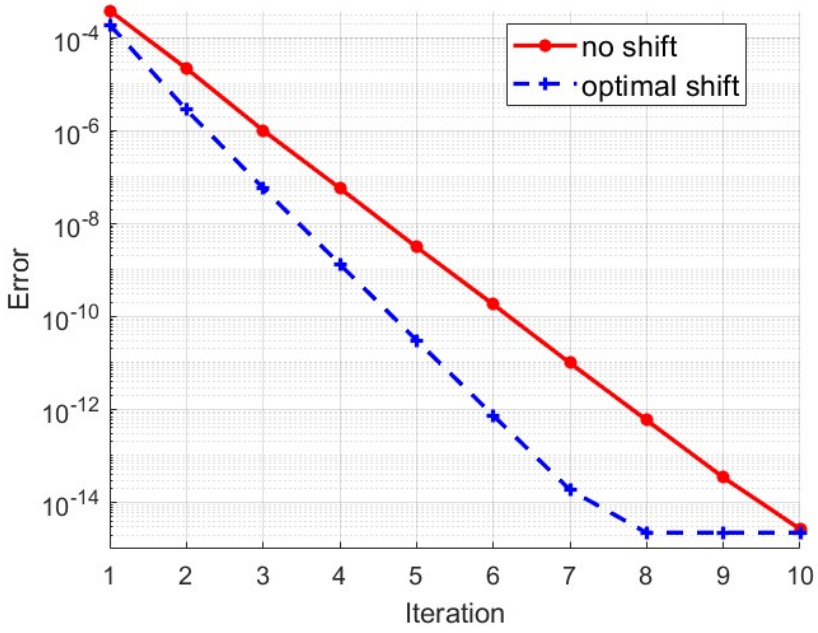


Figure 4.57: Comparison between optimal gshift and no shift 60-core cases

Chapter 5

Conclusion

In this thesis, the problem of simulating linear systems in the presence of time-varying loads, which makes direct simulation methods impractical, is addressed. Therefore, the main goal of this work was to develop a new simulation method that reuses a transient solver based on implicit Euler, originally designed for linear systems, in the presence of nonlinear loads that induce feedback. Initially, the Waveform Relaxation method was used, but due to convergence issues, the Picard method was chosen instead. By utilizing the Picard iteration method, it was possible to avoid the challenges associated with direct solutions due to the time-varying load. This allowed us to significantly accelerate the convergence process. Furthermore, was successfully identified the optimal value of the parameter g_{shift}, enables us to achieve a specific level of precision with the minimum number of iterations possible.

The analysis highlighted several fundamental challenges associated with conditioning in large systems, where the system matrix exhibits high condition numbers. This often leads to significant errors during iterations, making it difficult to reach the required precision. However, the results obtained by introducing g_{shift} demonstrated a significant improvement. In the 60-core configuration, a reduction in error was observed starting from the early iterations, achieving two orders of magnitude gain in accuracy by the fourth iteration. This result not only showcases the effectiveness of optimizing the shift parameter but also confirms that improved performance is attainable even in complex systems.

Comparative tests between the method with and without g_{shift} underscored the importance of this parameter. By applying $g_{\text{shift}} = 0.1$, the error continued to decrease more rapidly and steadily, reaching a reduction of three orders of magnitude by the eighth iteration compared to the case without any shift. Additionally, improvements in matrix conditioning observed during the tuning process contributed to a more stable solution, consistently lowering error levels.

5.1 Future work

These findings have significant practical implications. An optimized iterative method for complex systems, such as those analyzed in this thesis, can be applied across various engineering fields, including advanced circuit simulation and large-scale network management. The ability to achieve precision and stability with fewer iterations not only saves computational time and resources but also enhances overall calculation efficiency. Looking ahead, there are two promising directions for further development of the method presented here. First, dynamic refinement of g_{shift} could allow for real-time adjustments to this parameter, further improving convergence speed. Second, exploring applications of the method to problems with different characteristics or integrating it with more advanced preconditioning techniques could make it even more robust and versatile.
The proposed method not only accelerates convergence but also ensures precise and reliable results.

Bibliography

[1] A. Carlucci, S. Grivet-Talocia, S. Mongrain, S. Kulasekaran, and K. Randhakrishnan, *A structured Krylov subspace projection framework for fast power integrity verification*, 2023 IEEE 27th Workshop on Signal and Power Integrity (SPI), 2023, pp. 1-4.

[2] A. Carlucci, S. Grivet-Talocia, S. Mongrain, S. Kulasekaran, and K. Randhakrishnan, *Balancing-based model reduction for fast power integrity verification*, 2023 IEEE 32nd Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS), 2023, pp. 1-3.

[3] A. Carlucci, S. Grivet-Talocia, T. Bradde, S. Mongrain, S. Kulasekaran, and K. Randhakrishnan, *A compressed multivariate macromodeling framework for fast transient verification of system-level power delivery network*, IEEE Transactions on Component, Packaging and Manufacturing Technology, vol. 13, no. 10, 2023, pp. 1-4.

[4] A. Carlucci, S. Grivet-Talocia, T. Bradde, S. Kulasekaran, and K. Randhakrishnan, *Structured model order reduction of system-level power delivery networks*, IEEE ACCESS, Early access [ONLINE], 2024, pp. 1-4.

[5] A. Carlucci, A. Moglia, S. Grivet-Talocia, S. Kulasekaran, and K. Randhakrishnan, *Fast transient simulation of system-level power delivery networks via parallel waveform relaxation*, IEEE Transactions on Component, Packaging and Manufacturing Technology, 2024, pp. 1-13.

[6] E. A. Burton, G. Schhrom, F. Paillet, J. Douglas, W. J. Lambert, K. Randhakrishnan, and M. J. Hill, *FIVR– Fully integrated voltage regulators on 4th generation Intel Core^{TM} SoCs*, 2014 IEEE Applied Power Electronics Conference and Exposition - APEC 2014, 2014, pp. 432-439.

[7] Prof. Stefano Grivet-Talocia, *Advanced design for signal integrity and compliace*, Materiale del corso, Corso di Laurea in Ingegneria Elettronica, Politecnico di Torino, 2024.

[8] Prof. Francesco Musolino, *Power Electronics*, Materiale del corso, Corso di Laurea in Ingegneria Elettronica, Politecnico di Torino, 2023.