UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Frequency analysis of RLCK circuits using iterative GMRES method and application of Schur complement

Diploma Thesis

Georgia Despoina Tzanetou

Supervisor: Nestor Evmorfopoulos

September 2025

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Frequency analysis of RLCK circuits using iterative GMRES method and application of Schur complement

Diploma Thesis

Georgia Despoina Tzanetou

Supervisor: Nestor Evmorfopoulos

September 2025

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάλυση συχνότητας κυκλωμάτων RLCK με επαναληπτική μέθοδο GMRES και εφαρμογή συμπληρώματος Schur

Διπλωματική Εργασία

Γεωργία Δέσποινα Τζανέτου

Επιβλέπων/πουσα: Νέστωρ Ευμορφόπουλος

Σεπτέμβριος 2025

Approved by the Examination Committee:

Supervisor **Nestor Evmorfopoulos**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Fotios Plessas**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Georgios Stamoulis**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Acknowledgements

I would like to express my sincere gratitude to my Professor Mr.Nestor Evmorfopoulos for his guidance and the trust he placed in me, and to Byron Moutafis and Kostas Kafousas, Senior R&D Engineers, for their ideas, time, and support during my internship at ANSYS. Above all, I am grateful to my mother for her unwavering support throughout my studies, and to my family and friends for their constant encouragement. Any errors are my own.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Georgia Despoina Tzanetou

Diploma Thesis

Frequency analysis of RLCK circuits using iterative GMRES method and application of Schur complement

Georgia Despoina Tzanetou

Abstract

This thesis focuses on the study of iterative methods for solving linear systems resulting from the simulation of electrical circuits using the Modified Nodal Analysis (MNA) method. The goal is to examine the Generalized Minimal Residual (GMRES) method as an alternative to direct methods, considering both memory efficiency and time performance. The subject is part of the larger scientific field of numerical analysis and circuit simulation, where the performance of the solver affects the scalability and practical application of methods in large systems.

In the context of optimizing the problem, the Schur complement is applied based on the number of circuit ports to reduce the dimension of the system. In addition, an Incomplete LU factorization with threshold (ILUT) is employed as a preconditioner to accelerate convergence. The analysis was performed on industrial-scale data, providing a realistic evaluation of the methods.

The results demonstrate that GMRES outperforms direct methods in terms of memory usage, making it attractive for large-scale applications. However, direct methods were found to remain faster in execution, so the choice of method depends on the priority at hand: memory or speed*.

Finally, the prospect of extending the approach through block-GMRES is reasonably examined, which can improve efficiency in cases with multiple right-hand sides.

Keywords:

GMRES (Generalized Minimal Residual), Modified Nodal Analysis (MNA), Schur complement, Sparse linear systems, Krylov subspace methods, ILU/ILUT preconditioner, Circuit simulation.

*Results during internship at ANSYS, which are held confidential.

Διπλωματική Εργασία

Ανάλυση συχνότητας κυκλωμάτων RLCK με επαναληπτική μέθοδο GMRES και εφαρμογή συμπληρώματος Schur

Γεωργία Δέσποινα Τζανέτου

Περίληψη

Στην παρούσα εργασία μελετάται η επιλογή επαναληπτικής μεθόδου για την επίλυση γραμμικών συστημάτων που προκύπτουν από την προσομοίωση ηλεκτρικών κυκλωμάτων με την Μέθοδο Τροποποιημένων Κομβικών Αναλύσεων (Modified Nodal Analysis – MNA). Στόχος είναι η διερεύνηση της μεθόδου GMRES ως εναλλακτική λύση απέναντι στις άμεσες μεθόδους, λαμβάνοντας υπόψιν την απόδοση τόσο σε μνήμη όσο και χρόνο. Το αντικείμενο αποτελεί μέρος του ευρύτερου επιστημονικού πεδίου της αριθμητικής ανάλυσης και της προσομοίωσης κυκλωμάτων, όπου η απόδοση του επιλυτή επηρεάζει την επεκτασιμότητα και την χρήση των μεθόδων σε μεγάλα συστήματα.

Στο πλαίσιο της βελτιστοποίησης του προβλήματος, εφαρμόζεται Schur complement με βάση τον αριθμό των θυρών του κυκλώματος για την μείωση της διάστασης του συστήματος καθώς και χρήση προρρυθμιστή ILU με σκοπό την επιτάχυνση της σύγκλισης. Η ανάλυση πραγματοποιήθηκε σε βιομηχανικής κλίμακας δεδομένα, δίνοντας μια ρεαλιστική αξιολόγηση των μεθόδων.

Τα αποτελέσματα δείχνουν ότι η GMRES υπερτερεί στη χρήση μνήμης συγκριτικά με τις άμεσες μεθόδους, γεγονός που την καθιστά ελκυστική σε εφαρμογές μεγάλης κλίμακας. Ωστόσο, οι άμεσες μέθοδοι διαπιστώθηκε ότι παραμένουν ταχύτερες σε εκτέλεση με αποτέλεσμα η επιλογή μεθόδου να εξαρτάται από την εκάστοτε προτεραιότητα, μνήμη ή ταχύτητα[†].

Εύλογα, τελικά, εξετάζεται η προοπτική επέκτασης της προσέγγισης μέσω της block-GMRES, η οποία μπορεί να βελτιώσει την αποδοτικότητα σε περιπτώσεις με πολλαπλά δεξιά μέλη.

[†]Αποτελέσματα κατά τη διάρκεια της πρακτικής άσκησης στην ANSYS, τα οποία παραμένουν απόρρητα.

Λέξεις-κλειδιά:

GMRES (Γενικευμένη Μέθοδος Ελαχίστων Υπολοίπων), Τροποποιημένη Κομβική Ανάλυση (MNA), Συμπλήρωμα Schur, Αραιά γραμμικά συστήματα, Μέθοδοι υποχώρων Krylov, Προσταθεροποίηση ILU/ILUT, Προσομοίωση κυκλωμάτων.

Table of contents

Chapter 1

Introduction

The analysis and simulation of electrical and electronic circuits is a fundamental step in the design of modern systems, from integrated circuits to complex communication networks. As integrated circuits are designed with smaller feature sizes and faster operation, complexity increases exponentially, resulting in large-scale mathematical models that are often computationally demanding to solve. Consequently, the need for more efficient numerical methods is becoming increasingly urgent.

1.1 Study Objectives

Challenges to classical methods for solving linear systems appear when looking at large-dimensional, sparse and often ill-conditioned cases. This paper studies the iterative GMRES (Generalized Minimal Residual) method for solving the linear system resulting from the Modified Nodal Analysis (MNA) method. As an iterative method, GMRES produces a sequence $x^{(k)}$ in Krylov subspaces that converges to the solution by minimizing the residual $\|b - Ax^{(k)}\|$, while direct methods such as LU factorization solve the system $Ax = b$ by factorizing A and give the exact solution in a finite number of steps. To improve efficiency, the 2-block representation of MNA is applied, as well as reduction techniques using Schur Complement based on the number of ports. In addition, ILUT (Incomplete LU factorization with Threshold) is used as a preconditioner to accelerate the convergence of GMRES.

1.1.1 Approach

The present work proposes a practical method for solving MNA systems and tests it on real data. Specifically, a large, sparse MNA matrix from an industrial application was used, and three approaches were implemented: (i) direct solution with LU, (ii) GMRES with ILUT preconditioning, and (iii) GMRES with ILUT preconditioning combined with a reduction via the Schur complement. The methods were compared in AC analysis, both in a frequency sweep and at a targeted frequency, under common tolerances and solver options. In terms of results, we report execution time, residual behavior, and convergence.

1.2 Thesis Structure

The thesis is structured as follows: Chapter 2 covers the theoretical basis of MNA and numerical solution approaches. Chapter 3 outlines the approach used and the implementation choices. Chapter 4 examines the experimental findings and conducts a comparative review. Finally, Chapter 5 summarises the thesis's findings and proposes directions for further research.

Chapter 2

Theoretical Background: MNA and Linear System Solvers

2.1 Modified Nodal Analysis

2.1.1 Introduction

The Modified Nodal Analysis (MNA) is one of the most widely used techniques for formulating the system equations of electrical circuits. It was first introduced in 1975 by Ho, Ruehli and Brennan [1] as a generalization of the classical Nodal Analysis, overcoming its limitations in handling voltage sources and inductors. Since then, MNA has been implemented as the core of industrial circuit simulators like SPICE.

2.1.2 How the MNA matrix is constructed

The Modified Nodal Analysis (MNA) is based on solving a linear system of the form

$$A \cdot x = b, \tag{2.1}$$

where A is the global admittance matrix, x is the vector of unknowns (node voltages and selected branch currents), and b is the excitation vector [2]. The generation of A and b is performed using the so-called *stamping process*. Each element in the circuit is associated with a predefined "stamp" that represents its contribution to the system equations, and this local pattern is inserted into the appropriate positions of the global matrix.

The connectivity of a circuit can be represented by a *directed graph*, where the graph

nodes correspond to the circuit nodes and the edges correspond to the circuit elements [3]. This topological information is encoded in the *reduced adjacency (or incidence) matrix* A of dimensions $(n-1) \times m$, defined as

$$a_{ij} = \begin{cases} +1 & \text{if branch } j \text{ leaves node } i, \\ -1 & \text{if branch } j \text{ enters node } i, \\ 0 & \text{otherwise.} \end{cases}$$

Based on this representation, Kirchhoff's laws can be written in compact matrix form:

$$u(t) = A^T v(t), \quad \text{(KVL)} \tag{2.2}$$

$$Ai(t) = 0, \quad \text{(KCL)}, \tag{2.3}$$

where $u(t)$ is the vector of branch voltages, $v(t)$ the node potentials, and $i(t)$ the branch currents [2]. The construction process proceeds element by element, and the stamps for the most common components are as follows:

- **Resistors**: contribute conductance values

  $$G = \frac{1}{R}$$

  to the corresponding diagonal and off-diagonal entries, depending on the terminal nodes.

- **Capacitors**: contribute entries proportional to the Laplace variable s (or $j\omega$ in the frequency domain), forming the capacitance submatrix.

- **Inductors**: require the introduction of an additional branch current variable. The inductor branch is connected to its terminal nodes via an incidence vector, and a constitutive equation (involving sL) is added to the system.

- **Voltage sources**: are also modeled by introducing a branch current variable, ensuring that Kirchhoff's Voltage Law (KVL) is enforced.

Classification of elements. The m elements of a circuit can be separated into two groups [2]:

- Group G_1: elements whose equations can be expressed as

  $$i_k(t) = g_k u_k(t) + c_k \frac{du_k(t)}{dt} + s_k(t),$$

  where g_k and c_k are constants (possibly zero), and $s_k(t)$ is a known excitation (e.g., resistors, capacitors, current sources).

- Group G_2: elements such as inductors and voltage sources, which cannot be represented in this form and therefore require the introduction of additional branch current variables.

This classification is fundamental, since the global MNA matrix is assembled by combining the contributions of G_1 into the matrices G and C, while G_2 defines the structure of the incidence matrix A and the inductance block L.

This *stamp methodology* makes the assembly of the MNA matrix entirely algorithmic: as the netlist of the circuit is parsed, each element contributes its local stamp, and the global system matrix is gradually formed. This systematic approach explains why MNA is especially suited for automated circuit simulation.

2.1.3 General Form of the MNA System

Block structure. The MNA system can be assembled in block matrix form:

$$\begin{bmatrix} G + sC & A^T \\ A & -sL \end{bmatrix} \begin{bmatrix} v \\ i \end{bmatrix} = \begin{bmatrix} j \\ e \end{bmatrix}, \tag{2.4}$$

where $A = A_2$ the incidence matrix relating currents to node voltages, and j, e denote the transformed source vectors.

Frequency-Domain Representation. Solving the lower equations of (2.5) and substituting to the upper equations, we can obtain the second-order expression of the system matrix (with respect to the complex frequency s):

$$A(s) = G + sC + \frac{1}{s} A_L^T L^{-1} A_L, \tag{2.5}$$

where A_L is the incidence matrix of the inductive elements. This form makes explicit the separate contributions of resistive, capacitive, and inductive devices, and is particularly convenient in model order reduction and iterative solution methods.

AC specialization. In steady-state (AC) analysis we use phasors with the convention $e^{j\omega t}$, so we simply set $s = j\omega$. With this substitution, (2.5) becomes

$$A(j\omega) = G + j\omega\, C + \frac{1}{j\omega}\, A_L^T L^{-1} A_L, \tag{2.6}$$

meaning that for each frequency ω we solve *one* complex linear system

$$A(j\omega)\, x(\omega) = b(\omega),$$

[2, 4]. In general circuits $A(j\omega)$ is complex and not Hermitian (often complex-symmetric), so Cholesky/CG are not applicable. Instead, nonsymmetric Krylov methods such as GMRES are appropriate, typically with preconditioning; here we use ILUT and, in the two-block formulation, apply it to the $A_{ii}(j\omega)$ subproblem. Across a frequency sweep, nearby systems are similar, so preconditioners or initial guesses can be recycled to lower the total cost.

2.1.4 Properties of the MNA method.

The resulting circuit matrix exhibits high sparsity, meaning that the majority of its entries are equal to zero. Exploiting this property through sparse matrix techniques, which avoid storing and manipulating zero entries, can significantly improve the efficiency of circuit simulation [2].

When a circuit includes elements with widely varying parameters, the resulting matrices may have a large condition number, making them ill-conditioned and highly sensitive to round-off errors. As Golub and Van Loan emphasize [5], the condition number $\kappa(A) = \|A\|\|A^{-1}\|$ determines the sensitivity of a linear system to perturbations: a problem is considered well-conditioned if $\kappa(A) \approx O(1)$ and ill-conditioned if $\kappa(A) \approx O(1/u)$, where u is the machine precision. In addition, conditioning is an inherent property of the problem and cannot be corrected by pivoting. In the context of MNA, circuits with widely varying element values often lead to ill-conditioned matrices, which directly affect the convergence of iterative solvers and the stability of direct methods [2].

In industrial designs, the application of MNA often leads to systems with hundreds of thousands of unknowns. Direct solvers, although robust, are frequently limited by memory requirements due to fill-in that occurs during factorization, where zero entries become nonzero and significantly increase both storage and computational cost [6]. Iterative solvers, on the other hand, are more suitable for large sparse systems but require effective preconditioning to converge in a reasonable number of iterations [7].

Because circuits include dynamic elements such as capacitors and inductors, the equations obtained from MNA are not purely algebraic but rather differential-algebraic equations (DAEs). As Najm explains [2], incorporating capacitors and inductors in the MNA formula-

tion leads to a system of DAEs rather than ordinary differential equations. Some variables, including inductor currents and voltages from sources, appear purely algebraically, while capacitors and inductors contribute time derivatives. Therefore, Laplace-domain analysis or time-integration techniques are often required to solve MNA systems.

2.2 Direct Methods

Let us assume that we have a linear system of equations as follows:

$$Ax = b, \quad A \in \mathbb{R}^{n\times n},\ x, b \in \mathbb{R}^n, \tag{2.7}$$

where A is a square and invertible matrix of dimension $n \times n$, and x, b are $n \times 1$ vectors. This system can be solved in two ways [8]:

- Directly, where the algorithm to be applied will perform only a finite number of operations for each coefficient of the system, and the end of the algorithm will give the solution, if it can be determined by the method used for the system.
- Iteratively, where the algorithm to be applied will attempt, through a series of iterations (operations on all positions of the system), to correct and ultimately solve the system.

The most famous and widely used direct method is Gaussian elimination, which is also the basis for LU factorization. Also, for symmetric positive definite matrices, Cholesky factorization is used. Direct solvers are robust and yield numerically stable results when appropriate pivoting strategies are used. However, they are computationally expensive for large-scale systems, with complexity on the order of $O(n^3)$ for dense problems, and in sparse cases they suffer from the phenomenon of *fill-in*, where zeros in A are replaced by nonzero entries during factorization, leading to increased memory requirements [6, 9].

2.2.1 Gaussian Elimination

Consider a linear system

$$Ax = b, \quad A \in \mathbb{R}^{n\times n},\ x, b \in \mathbb{R}^n,$$

with A nonsingular. Gaussian elimination reduces the matrix A to an upper triangular form U through a sequence of elementary row operations. The k-th elimination step chooses a *pivot*

a_{kk} and eliminates the entries below it by subtracting suitable multiples of row k from rows $i > k$. If

$$m_{ik} = \frac{a_{ik}}{a_{kk}}, \qquad i = k+1, \ldots, n,$$

then row i is updated as

$$\text{Row}_i \leftarrow \text{Row}_i - m_{ik}\,\text{Row}_k,$$

so that the entry (i, k) becomes zero. After $n-1$ steps we obtain an upper triangular matrix U and a transformed right-hand side c, yielding $Ux = c$, which is solved in $O(n^2)$ flops by backward substitution [5, 9]. The numerical stability of Gaussian elimination depends strongly on the pivot strategy. Partial pivoting, where rows are interchanged to maximize the magnitude of the pivot element, is commonly used to avoid divisions by small numbers and limit the growth of rounding errors [5, 9, 10]. The cost of Gaussian elimination for dense matrices is approximately $\frac{2}{3}n^3$ floating-point operations, which becomes prohibitive for very large n.

Pivoting and stability. Numerical stability depends on pivot elements.Without pivoting, the method may fail if a pivot is zero or very small. **Partial pivoting** (GEPP) improves stability by swapping rows to place the largest column element while **Complete pivoting**, which swaps both rows and columns, offers better stability but at a higher cost. **Scaled partial pivoting** balances row magnitudes before selecting pivots [10], making Gaussian elimination backward stable for a broad class of matrices [5].

Fill-in and sparse systems. Gaussian elimination introduces new nonzeros in positions that were originally zero, a phenomenon called *fill-in*, increasing memory usage and computational cost [9]. To minimize fill-in, graph-based reorderings like minimum degree and nested dissection are used [6, 10]. Modern sparse direct solvers efficiently implement these reorderings, exploiting memory locality and parallelism [10]. In circuit simulation, sparsity patterns from MNA matrices require specialized sparse LU solvers like KLU, which combine reordering with problem-specific optimizations [2].

2.2.2 LU Factorization

Gaussian elimination can be formulated as an LU factorization:

$$A = LU,$$

where the matrix A is decomposed into the product of a lower and an upper triangular matrix: with

$$L = \begin{bmatrix} 1 & & & 0 \\ m_{21} & 1 & & \\ \vdots & \ddots & \ddots & \\ m_{n1} & \cdots & m_{n,n-1} & 1 \end{bmatrix}, \qquad U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & u_{nn} \end{bmatrix}.$$

Here, L contains the elimination multipliers and U the resulting upper triangular matrix. The multipliers m_{ik} generated during elimination form the subdiagonal entries of L. In practice, pivoting is necessary for stability, leading to the factorization

$$PA = LU,$$

where P is a permutation matrix representing row swaps [9, 5].

Solving $Ax = b$ then reduces to two triangular systems:

$$Ly = Pb, \quad Ux = y.$$

Each triangular solve requires $O(n^2)$ operations, which is much cheaper than the $O(n^3)$ factorization. The factorization itself, however, can be reused for multiple right-hand sides, a common case in DC sweeps and transient analyses in circuit simulation [2].

For symmetric positive definite matrices, the LU factorization simplifies to **Cholesky decomposition**,

$$A = LL^T,$$

which is more efficient (about half the operations of LU) and does not require pivoting [5].

In summary, Gaussian elimination with partial pivoting is the standard robust method for dense systems. Its LU factorization formulation provides a practical and reusable framework for solving multiple right-hand sides. For sparse systems, fill-in and memory overhead make direct solvers less attractive for extremely large-scale problems, despite advances in reordering and sparse LU. These limitations motivate the use of iterative methods in very large circuit simulations, while direct methods remain indispensable when robustness and stability are the primary requirements [6, 7, 10].

Algorithm 1 LU Factorization with Partial Pivoting

for $k = 1$ to n **do**
  $x \leftarrow |a_{kk}|, m \leftarrow k$
  for $i = k + 1$ to n **do**
    if $|a_{ik}| > x$ **then**
      $x \leftarrow |a_{ik}|, m \leftarrow i$
    end if
  end for
  Interchange rows k and m
  for $i = k + 1$ to n **do**
    $l_{ik} \leftarrow a_{ik}/a_{kk}$
  end for
  for $i = k + 1$ to n **do**
    for $j = k + 1$ to n **do**
      $a_{ij} \leftarrow a_{ij} - l_{ik}\, a_{kj}$
    end for
  end for
end for

2.3 Iterative Methods

Iterative methods constitute a broad class of algorithms for solving linear systems of the form

$$Ax = b, \qquad A \in \mathbb{R}^{n \times n},$$

by generating a sequence of approximate solutions

$$x^{(0)},\ x^{(1)},\ \ldots,\ x^{(k)},$$

that converges to the exact solution x under suitable conditions. Unlike direct methods, which apply a finite sequence of algebraic transformations to produce the exact solution (up to round-off errors), iterative methods rely on successive refinements, exploiting the structure of A without explicitly factorizing it. This property makes them attractive for large-scale sparse systems where direct solvers are often memory-limited due to fill-in [11, 7, 12].

2.3.1 Stationary Iterative Methods

The classical starting point for iterative solvers is the family of stationary methods, derived by splitting the coefficient matrix A into convenient parts. Consider the decomposition

$$A = D - L - U,$$

where D is the diagonal of A, and $-L$, $-U$ are its strict lower and upper triangular parts, respectively. The generic stationary iteration is given by

$$x^{(k+1)} = M^{-1}\,(Nx^{(k)} + b), \qquad A = M - N.$$

Different choices of M lead to well-known methods:

- **Jacobi method:** $M = D$, yielding

$$x^{(k+1)} = D^{-1}\,(L + U)x^{(k)} + D^{-1}b.$$

- **Gauss–Seidel method:** $M = D - L$, using updated components of x as soon as they are available.

- **Successive Over-Relaxation (SOR):** an extension of Gauss–Seidel introducing a relaxation parameter ω, with $\omega = 1$ reducing to Gauss–Seidel.

The convergence of stationary methods depends on the spectral radius $\rho(M^{-1}N)$, and for general nonsymmetric, indefinite systems, convergence may be prohibitively slow [12, 7]. Despite their simplicity, these methods play a central role as building blocks for more advanced techniques.

2.3.2 Projection and Krylov Subspace Methods

Modern iterative solvers are based on projection principles: at iteration k, the next approximation $x^{(k)}$ is chosen from an affine subspace $x^{(0)} + \mathcal{K}_k$, where $\mathcal{K}_k$ is a suitably chosen subspace, such that the residual $r^{(k)} = b - Ax^{(k)}$ is orthogonal to a constraint space. The most widely used choice of $\mathcal{K}_k$ is the *Krylov subspace*,

$$\mathcal{K}_k(A, r^{(0)}) = \operatorname{span}\{r^{(0)}, Ar^{(0)}, A^2r^{(0)}, \ldots, A^{k-1}r^{(0)}\},$$

leading to the large family of Krylov subspace methods [7, 13].

Specializations of the general framework give rise to:

- **Conjugate Gradient (CG):** optimal for symmetric positive definite matrices.
- **Generalized Minimal Residual (GMRES):** minimizes the residual norm over Krylov subspaces, applicable to general nonsymmetric matrices.
- **BiCG and BiCGSTAB:** two-sided Krylov methods for nonsymmetric systems.
- **MINRES, QMR, TFQMR:** variants tailored for symmetric indefinite or nonsymmetric cases.

The power of Krylov methods lies in their ability to deliver rapid convergence when the eigenvalues of A are clustered or well-conditioned. Their cost per iteration is dominated by one sparse matrix–vector product and some inner products, making them well-suited for very large sparse problems.

2.3.3 Convergence and Preconditioning

The convergence rate of iterative methods is strongly linked to the spectral properties of A. For stationary iterations, convergence requires $\rho(M^{-1}N) < 1$. For Krylov methods, the distribution of eigenvalues influences how fast the residual norm decreases. Poor conditioning leads to stagnation or extremely slow convergence [11, 7].

To address this, *preconditioning* is essential. A preconditioner $P \approx A^{-1}$ transforms the system into

$$P^{-1}Ax = P^{-1}b,$$

with improved spectral properties. Two main variants are used in practice:

Left and right preconditioning. In **left preconditioning**, the transformed system is

$$P^{-1}Ax = P^{-1}b,$$

so that the residual being minimized is $r^{(k)} = P^{-1}(b - Ax^{(k)})$. In **right preconditioning**, one solves

$$AP^{-1}y = b, \qquad x = P^{-1}y,$$

so that the Krylov subspace is built for the preconditioned operator AP^{-1}, while the residual remains measured in the original space $r^{(k)} = b - Ax^{(k)}$ [7]. The two approaches are mathematically equivalent in exact arithmetic, since both systems have the same solution, but they

may lead to different convergence histories in finite precision. In addition, left preconditioning modifies the residual norm used in convergence checks, while right preconditioning alters the iterates themselves through the transformation $x = P^{-1}y$ [7].

Types of preconditioners. The simplest choice is **diagonal or Jacobi-type**, where $P = \mathrm{diag}(A)$. Block-Jacobi and SSOR are natural extensions, cheap to apply but usually weak in accelerating convergence [11, 7]. A more powerful class is **incomplete factorizations**, such as ILU for general matrices and Incomplete Cholesky (IC) for symmetric positive definite ones, where approximate LU/Cholesky factors are computed with restricted fill-in. Variants include ILU(0), ILU(p), and ILUT (dual threshold) [7, 13, 14]. These methods provide a strong balance between cost and effectiveness and are widely used in combination with Krylov solvers.

Other families, including polynomial preconditioners, sparse approximate inverses, and multilevel methods (e.g., algebraic multigrid), are also available but are less common in circuit simulation [13].

Residual and Stopping Criteria

In practice, iterative methods are terminated when the residual is sufficiently reduced rather than when the exact solution is obtained. A common measure is the *relative residual norm*,

$$\frac{\|r^{(k)}\|}{\|b\|} = \frac{\|b - Ax^{(k)}\|}{\|b\|},$$

which provides a scale-independent indicator of convergence. This quantity reflects the accuracy of the current approximation relative to the right-hand side b and is widely used as a stopping criterion. Depending on the application, typical tolerances range from 10^{-6} to 10^{-12} in double precision arithmetic. It is important to note, however, that a small residual norm does not always imply a small error in the solution, since $\|x - x^{(k)}\|$ also depends on the conditioning of A. Nevertheless, monitoring the relative residual offers a practical and computationally inexpensive way of controlling convergence in large-scale iterative solvers [11, 7].

2.3.4 Advantages and Limitations

Iterative methods offer several advantages over direct solvers:

- Lower memory requirements, since only sparse matrix–vector products are required.

- Better scalability for very large systems, especially on parallel architectures.
- Flexibility through preconditioning to tailor convergence.

On the other hand, their drawbacks include:

- Convergence is not guaranteed for all problems.
- Rate of convergence may be unacceptably slow without effective preconditioning.
- They yield approximate solutions that must be monitored for accuracy.

Despite their limitations, iterative methods have proven essential in large-scale scientific computing and circuit simulation. Krylov subspace techniques, most notably GMRES, are a key aspect of the current sparse linear solver toolbox. In fact, direct and iterative approaches have complimentary strengths: direct methods are durable, stable, and predictable, offering exact answers within machine accuracy; nevertheless, they become unfeasible for very large sparse systems due to high memory and computing requirements. Iterative methods on the other hand, scale better to large scale problems and handle sparsity more economically, but they rely significantly on preconditioning and may fail to converge in difficult circumstances. Based on Axelsson [11] and Saad [7], direct methods are preferable for small to medium-sized problems or when robustness is critical, while iterative methods are generally the method of choice for large-scale sparse systems, such as those arising from Modified Nodal Analysis.

Chapter 3

GMRES Method

3.1 Introduction

Saad and Schultz [15] developed in 1986 the GMRES (Generalized Minimal Residual) technique, which can solve both linear and nonlinear problems. The main idea of GMRES is straightforward: at each iteration the method expands a Krylov subspace generated by the initial residual and then selects, from this subspace, the vector that minimizes the Euclidean norm of the residual. In other words, the algorithm builds an orthonormal basis

$$\mathcal{K}_k(A, r_0) = \text{span}\{r_0, Ar_0, \ldots, A^{k-1}r_0\}, \tag{3.1}$$

with $r_0 = b - Ax_0$, and then solves a small least–squares problem to update the approximation [16]. This is typically done with the Arnoldi process, which produces an upper Hessenberg matrix together with the orthogonal basis.

In theory, GMRES can find the exact solution in at most N steps for an $N \times N$ system. In practice, however, it often converges in far fewer steps, which makes it attractive. The drawback is that the method has to store all basis vectors, and this may become a bottleneck for large problems. Furthermore, the rate of convergence strongly depends on the spectral properties of A. If the coefficient matrix is poorly conditioned, GMRES without any modifications can stagnate or converge extremely slowly.

To address this, preconditioning is almost always applied. The central idea is to replace the original system by an equivalent one that is easier to solve iteratively. Formally, a preconditioner $M \approx A$ is introduced so that either

$$M^{-1}Ax = M^{-1}b \quad \text{(left preconditioning)} \tag{3.2}$$

or

$$AM^{-1}y = b, \qquad x = M^{-1}y \quad \text{(right preconditioning)} \tag{3.3}$$

is solved instead [16]. More broadly, preconditioning has been recognized as essential for scalability in parallel implementations as well [17]. In this thesis we focus on the preconditioned version of GMRES, with particular attention to right preconditioning based on incomplete LU (ILU) factorizations, which remain among the most widely used practical techniques.

3.2 Krylov Subspace Theory and the Arnoldi Iteration

3.2.1 Basic Arnoldi Method

Krylov subspace methods belong to a large family of iterative solvers that rely on building a sequence of subspaces spanned by the residual and its images under repeated multiplication with the coefficient matrix. More precisely, suppose we want to solve $Ax = b$ and start from an initial guess x_0. The corresponding residual is $r_0 = b - Ax_0$. From this vector we can generate the Krylov subspaces. After m steps, the m-th Krylov subspace is

$$\mathcal{K}_m(A, r_0) = \mathrm{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0\}. \tag{3.4}$$

The basic idea is to keep the search for the solution inside the affine space $x_0 + \mathcal{K}_m(A, r_0)$ and to pick, among all possible vectors in there, the one that satisfies an optimality condition. For GMRES this is simply the vector that minimizes the residual norm [16].

Why is this effective? Because in many practical problems the action of A can be well captured in rather low-dimensional subspaces. So, instead of building large powers of A explicitly, we just expand the Krylov basis step by step. To make this work in finite precision arithmetic, the basis needs to stay orthonormal, otherwise the small least–squares problems we solve along the way become unstable. The steps of basic Arnoldi Algorithm are:

Algorithm 2 Arnoldi Iteration (Saad, Alg. 6.2 [7])

choose initial vector $v_1 = r_0/\|r_0\|_2$
for $j = 1, 2, \ldots, m$ **do**
    compute $w = Av_j$
    for $i = 1, \ldots, j$ **do**
        $h_{i,j} = v_i^\top w$
        $w = w - h_{i,j} v_i$
    end for
    $h_{j+1,j} = \|w\|_2$
    if $h_{j+1,j} = 0$ **then**
        stop
    else
        $v_{j+1} = w/h_{j+1,j}$
    end if
end for

We begin with the normalized residual $v_1 = r_0/\|r_0\|_2$ and then, at each step, we apply A to the latest vector, orthogonalize the result against all the previous ones, and normalize. After m steps we obtain an orthonormal family $\{v_1, \ldots, v_m\}$ and the following relation:

$$AV_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^\top, \tag{3.5}$$

that is, A acting on the current basis V_m can be split into a part in the span of V_m plus a rank-one correction in the direction of v_{m+1} [7]. If we enlarge the basis to $V_{m+1} = [V_m \; v_{m+1}]$ and extend H_m accordingly, this becomes the compact identity

$$AV_m = V_{m+1} \bar{H}_m. \tag{3.6}$$

and by projection we also have

$$V_m^\top A V_m = H_m. \tag{3.7}$$

This compact notation is what GMRES uses internally: it keeps the orthonormal basis vectors in V_m and the small Hessenberg matrix H_m, and with these two pieces of information the method can update the approximate solution efficiently.

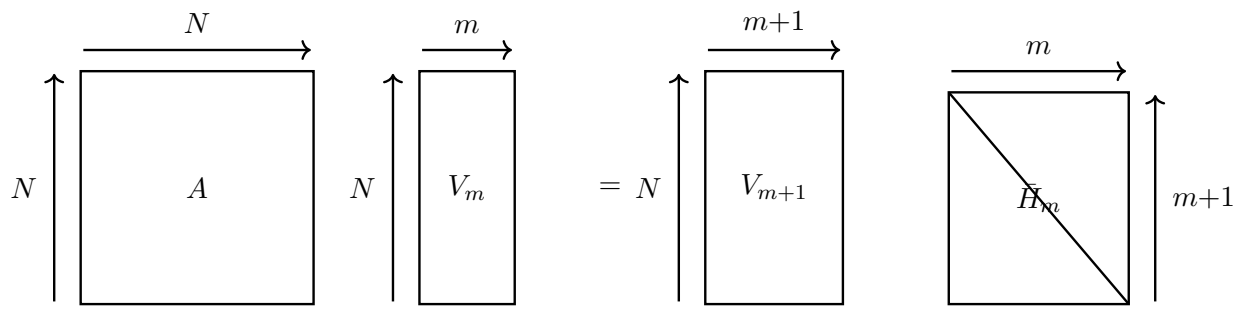


Figure 3.1: Compact Arnoldi relation: $AV_m = V_{m+1}\,\bar{H}_m$.

It is worth noting that the Arnoldi iteration not only provides the ingredients for GMRES but also produces useful spectral information. The eigenvalues of the Hessenberg matrix $\bar{H}_m$ serve as approximations to those of A, which sometimes can guide the choice of preconditioners or help explain convergence behavior.

$$H_m = \begin{pmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,m-1} & h_{1,m} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,m-1} & h_{2,m} \\ 0 & h_{3,2} & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & h_{m,m-1} & h_{m,m} \end{pmatrix}$$

Figure 3.2: Upper Hessenberg matrix H_m (nonzeros on and just below the diagonal).

In summary, the Arnoldi process is the computational backbone of GMRES: it builds the Krylov basis, maintains orthogonality, and reduces the large linear system to a sequence of smaller, structured least–squares problems.

3.2.2 Modified Gram–Schmidt Implementation

Different orthogonalization strategies have been proposed in the context of the Arnoldi process. Saad [7] discusses the Classical Gram–Schmidt (CGS) and the Modified Gram–Schmidt (MGS) procedures, and also mentions the use of Householder transformations and reorthogonalization steps to improve numerical stability. In the monograph of Van der Vorst [13], further variants are described: the Arnoldi process with CGS and with MGS, as well as the Lanczos algorithm for Hermitian matrices, the bi-Lanczos algorithm for nonsymmetric systems, and the complex symmetric Lanczos method for special cases.

In this thesis, the Modified Gram–Schmidt Arnoldi procedure is adopted, as it offers a good balance between robustness and computational cost, ensuring sufficient orthogonality of the Krylov basis throughout the GMRES iterations.

Compared to the classical Arnoldi process, the only difference lies in the orthogonalization step: in the classical Gram–Schmidt (CGS) version, the projections are accumulated in a single pass, while in the MGS version, the new vector is orthogonalized sequentially against each previously computed basis vector. This sequential subtraction makes MGS significantly more stable in finite precision arithmetic, although mathematically both variants are equivalent [7], [13].The following pseudocode summarizes the Arnoldi iteration when implemented with the Modified Gram–Schmidt (MGS) procedure.

Algorithm 3 Arnoldi Iteration with Modified Gram–Schmidt (MGS)

Choose initial vector $v_1 = r_0/\|r_0\|_2$.
for $j = 1, 2, \ldots, m$ **do**
    Compute $w = Av_j$.
    for $i = 1, \ldots, j$ **do**
        $h_{i,j} = v_i^\top w$
        $w \leftarrow w - h_{i,j} v_i$ (orthogonalize against v_i)
    end for
    $h_{j+1,j} = \|w\|_2$
    if $h_{j+1,j} = 0$ **then**
        Stop (Krylov subspace exhausted).
    else
        $v_{j+1} = w/h_{j+1,j}$
    end if
end for

3.3 The GMRES Method Algorithm.

The GMRES method combines the Arnoldi process with a least–squares minimization in order to construct successive approximations to the solution of $Ax = b$. The idea can be summarized step by step:

- Start with an initial guess x_0 and compute the residual $r_0 = b - Ax_0$.

- Normalize r_0 to obtain the first Krylov basis vector $v_1 = r_0/\|r_0\|_2$.

- Apply the Arnoldi process to generate an orthonormal basis $V_m = [v_1, \ldots, v_m]$ and the Hessenberg matrix $\bar{H}_m$ satisfying $AV_m = V_{m+1}\bar{H}_m$.

- Formulate the small least–squares problem

$$y_m = \arg\min_{y \in \mathbb{R}^m} \|\beta e_1 - \bar{H}_m y\|_2, \qquad \beta = \|r_0\|_2,$$

  and solve it efficiently using Givens rotations to update the QR factorization.

- Update the approximation as

$$x_m = x_0 + V_m y_m.$$

- Compute the residual norm $\|r_m\|_2$ and check convergence.

This description shows that each GMRES iteration consists of two essential tasks: (i) one Arnoldi step to expand the Krylov subspace, and (ii) one least–squares update to minimize the residual. The process continues until the residual norm is sufficiently small or until a prescribed number of iterations m has been reached [15]; [7].

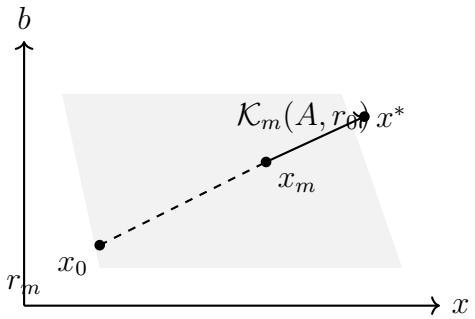


Figure 3.3: Geometric interpretation of GMRES: the approximation x_m is chosen in $x_0 + \mathcal{K}_m(A, r_0)$ such that the residual $r_m = b - Ax_m$ is minimized.

Next we summarize these steps in the form of a practical algorithm.

Algorithm 4 GMRES (modified Gram–Schmidt type) [13]

Require: $A \in \mathbb{R}^{N\times N}$, b, x_0, tolerance ε, max iterations m

$r_0 \leftarrow b - Ax_0$, $\beta \leftarrow \|r_0\|_2$, **if** $\beta = 0$ **then return** x_0

$v_1 \leftarrow r_0/\beta$, set $V_{m+1}[:,1] \leftarrow v_1$, $\bar{H}_m \leftarrow 0$, $g \leftarrow \beta e_1$

for $j = 1, 2, \ldots, m$ **do** ▷ Arnoldi process with MGS

  $w \leftarrow Av_j$

  for $i = 1, 2, \ldots, j$ **do**

    $h_{i,j} \leftarrow v_i^\top w$

    $w \leftarrow w - h_{i,j} v_i$

  end for

  $h_{j+1,j} \leftarrow \|w\|_2$;

  $v_{j+1} \leftarrow w/h_{j+1,j}$;

  for $i = 1, 2, \ldots, j-1$ **do** ▷ apply previous Givens

$$\begin{bmatrix} \bar{H}_m(i,j) \\ \bar{H}_m(i+1,j) \end{bmatrix} \leftarrow \begin{bmatrix} c_i & s_i \\ -s_i & c_i \end{bmatrix} \begin{bmatrix} \bar{H}_m(i,j) \\ \bar{H}_m(i+1,j) \end{bmatrix}$$

  end for

$$c_j \leftarrow \frac{|\bar{H}_m(j,j)|}{\sqrt{|\bar{H}_m(j,j)|^2 + |\bar{H}_m(j+1,j)|^2}}$$

$$s_j \leftarrow \frac{\bar{H}_m(j+1,j)}{\bar{H}_m(j,j)}\, c_j$$

$\bar{H}_m(j,j) \leftarrow c_j\, \bar{H}_m(j,j) + s_j\, \bar{H}_m(j+1,j)$

$\bar{H}_m(j+1,j) \leftarrow 0$

$$\begin{bmatrix} g_j \\ g_{j+1} \end{bmatrix} \leftarrow \begin{bmatrix} c_j & s_j \\ -s_j & c_j \end{bmatrix} \begin{bmatrix} g_j \\ 0 \end{bmatrix}$$

  if $|g_{j+1}|/\|b\|_2 \le \varepsilon$ **then** ▷ check convergence

    solve $\tilde{H}y = \tilde{g}$ with $\tilde{H} = \bar{H}_m(1{:}j, 1{:}j)$, $\tilde{g} = g(1{:}j)$ (back substitution)

    return $x_j = x_0 + V_j y$

  end if

end for

solve $\tilde{H}y = \tilde{g}$ with $\tilde{H} = \bar{H}_m(1{:}m, 1{:}m)$, $\tilde{g} = g(1{:}m)$

return $x_m = x_0 + V_m y$

3.4 Preconditioned GMRES.

The convergence of GMRES is influenced by the spectral properties of the coefficient matrix A, and for ill-conditioned or nonsymmetric systems, the plain method may slow down or stagnate. As we discussed in previous topic, preconditioning is almost always applies in order to improve the efficiency. In this thesis we use an Incomplete LU factorization (ILU) as a right preconditioner. The idea is to compute an approximate factorization

$$A \approx LU,$$

where L is lower triangular and U is upper triangular, but the factorization is incomplete: certain fill-ins are dropped according to a specified rule (ILU(0), ILUT, etc.). This produces a sparse and inexpensive approximation that is much cheaper to apply than the exact LU factorization.

With right preconditioning, we introduce the new variable y such that

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

where $M \approx A$ is the preconditioner (here $M = LU$ from ILU). The GMRES algorithm is then applied to the transformed system $AM^{-1}y = b$, so that Krylov vectors are generated by repeated multiplication with AM^{-1}. This has the advantage that the residuals are computed with respect to the original system $Ax = b$, which simplifies convergence monitoring [7]; [13].

In practice, constructing the ILU preconditioner involves two stages: a *symbolic factorization*, which determines the sparsity pattern of the incomplete factors L and U, and a subsequent *numeric factorization*, which fills in the actual values. The symbolic stage depends only on the structure of A and can therefore be reused if multiple systems with the same sparsity pattern are solved, while the numeric factorization must be recomputed whenever the matrix entries change. This distinction is particularly relevant for large-scale applications, as it can significantly reduce the overall setup cost of the preconditioner [7].

The structure of the algorithm is identical to the plain GMRES method described previously, except that every time a matrix–vector product Av_j is required, it is replaced by $AM^{-1}v_j$. In practice, this means solving two triangular systems with L and U to apply M^{-1} to a vector.

Algorithm 5 Preconditioned GMRES (right preconditioning, ILU)

Require: A, $M \approx A$ from ILU, b, x_0, tolerance ε, max iterations m

$r_0 \leftarrow b - Ax_0$, $\beta \leftarrow \|r_0\|_2$, if $\beta = 0$ return x_0
$v_1 \leftarrow r_0/\beta$, initialize V_{m+1}, $\bar{H}_m$, $g = \beta e_1$
for $j = 1, 2, \ldots, m$ **do**
    $z \leftarrow M^{-1} v_j$ (apply preconditioner)
    $w \leftarrow Az$
    for $i = 1, 2, \ldots, j$ **do**
        $h_{i,j} \leftarrow v_i^\top w$
        $w \leftarrow w - h_{i,j} v_i$
    end for
    $h_{j+1,j} \leftarrow \|w\|_2$; if $h_{j+1,j} = 0$ then break
    $v_{j+1} \leftarrow w/h_{j+1,j}$; store in V_{m+1}
    (apply Givens rotations to $\bar{H}_m$ and g as in plain GMRES)
    if residual $\|r_j\|_2$ small enough then stop
end for
solve $\tilde{H} y = \tilde{g}$ by back substitution
set $x_m = x_0 + M^{-1} V_m y$

The preconditioned GMRES algorithm described above forms the basis of the solver used in this work. In the following chapter, we turn from the theoretical background to the practical aspects of methodology and implementation, where the details of the ILU preconditioner, A matrix formulation and computational strategies are presented.

Chapter 4

Application of Schur Complement to GMRES

4.1 The Schur Complement

4.1.1 General Theory

The Schur complement is a classical concept in linear algebra and numerical analysis. Let $A \in \mathbb{R}^{n\times n}$ be written as a 2×2 block matrix

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \qquad A_{11} \in \mathbb{R}^{p\times p},\ A_{22} \in \mathbb{R}^{q\times q},\ n = p + q, \tag{4.1}$$

with $A_{12} \in \mathbb{R}^{p\times q}$ and $A_{21} \in \mathbb{R}^{q\times p}$.

We consider the block linear system

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \qquad x_1 \in \mathbb{R}^p,\ x_2 \in \mathbb{R}^q, \tag{4.2}$$

that is

$$A_{11}x_1 + A_{12}x_2 = b_1, \tag{4.3}$$

$$A_{21}x_1 + A_{22}x_2 = b_2. \tag{4.4}$$

If A_{11} is nonsingular, we can solve (4.3) for x_1:

$$x_1 = A_{11}^{-1}(b_1 - A_{12}x_2). \tag{4.5}$$

Substituting (4.5) into (4.4) yields the reduced system

$$Sx_2 = b_2 - A_{21}A_{11}^{-1}b_1, \tag{4.6}$$

where the matrix

$$S = A_{22} - A_{21}A_{11}^{-1}A_{12} \tag{4.7}$$

is called the *Schur complement of* A_{11} *in* A.

Conversely, if A_{22} is invertible, one can eliminate x_2 and obtain the alternative Schur complement

$$S = A_{11} - A_{12}A_{22}^{-1}A_{21}. \tag{4.8}$$

This reduction highlights the essential role of the Schur complement: it represents the exact effect of eliminating a subset of unknowns while keeping the system equivalent for the remaining ones. It appears in block Gaussian elimination, domain decomposition, constrained optimization, and saddle point problems [16],[18],[19],[20],[21].

4.1.2 Schur Complement on MNA formulation

Classical 2-block MNA. For a given frequency s, the standard MNA with node voltages v and inductor branch currents i_ℓ reads

$$\begin{bmatrix} G + s\,C & A_L^T \\ A_L & -\,s\,L \end{bmatrix} \begin{bmatrix} v \\ i_\ell \end{bmatrix} = \begin{bmatrix} e_i \\ 0 \end{bmatrix}, \tag{4.9}$$

where G and C are the conductance and capacitance matrices, L is the (block-diagonal) inductance matrix, A_L is the inductor incidence/coupling matrix, and e_i denotes a unit current injection at node/port i .

Condensation to the nodal "sum" form. From the lower block of (4.9) we have

$$A_L v - sL\,i_\ell = 0 \quad \Longrightarrow \quad i_\ell = \frac{1}{s}\,L^{-1}A_L\,v \qquad (s \neq 0,\ L \text{ nonsingular}).$$

Substituting in the upper block yields the condensed nodal system

$$\underbrace{\begin{bmatrix} A(s) & 0 \\ -\,A_L & sL \end{bmatrix}}_{\text{equivalent to (4.9)}} \begin{bmatrix} v \\ i_\ell \end{bmatrix} = \begin{bmatrix} e_i \\ 0 \end{bmatrix}, \qquad A(s) = G + sC + \tfrac{1}{s}A_L^{\mathsf{T}}L^{-1}A_L.$$

This is the matrix we will work with in the sequel.

Port/internal 2-block partition of the condensed matrix. Reorder the voltage unknowns so that port variables come first:

$$v = \begin{bmatrix} v_p \\ v_i \end{bmatrix}, \qquad A(s) = \begin{bmatrix} A_{pp}(s) & A_{pi}(s) \\ A_{ip}(s) & A_{ii}(s) \end{bmatrix}, \qquad b = \begin{bmatrix} b_p \\ b_i \end{bmatrix}.$$

Port Schur complement (reduced port system). Assuming $A_{ii}(s)$ is nonsingular (no floating internal sub-network at s), the port-reduced matrix is

$$\boxed{S_p(s) \;=\; A_{pp}(s) \;-\; A_{pi}(s)\, A_{ii}(s)^{-1}\, A_{ip}(s)}, \tag{4.10}$$

and the port-only system becomes

$$\boxed{S_p(s)\, v_p \;=\; b_p - A_{pi}(s)\, A_{ii}(s)^{-1} b_i}. \tag{4.11}$$

In our setting $b_p = e_i$ and $b_i = 0$, hence simply $S_p(s)\, v_p = e_i$.

Back-substitution for internal voltages. Once v_p is known,

$$A_{ii}(s)\, v_i \;=\; b_i - A_{ip}(s)\, v_p \quad\Longleftrightarrow\quad v_i \;=\; A_{ii}(s)^{-1}\big(b_i - A_{ip}(s)\, v_p\big), \tag{4.12}$$

which reduces to $A_{ii}(s)\, v_i = -A_{ip}(s)\, v_p$ for $b_i = 0$.

Computable form (no explicit inverses). In practice we never form inverses. Introduce the helper

$$T(s) \;:=\; A_{ii}(s)^{-1} A_{ip}(s) \quad \text{(i.e., solve } A_{ii}(s)\, T(s) = A_{ip}(s) \text{ column-wise)},$$

so that

$$S_p(s) \;=\; A_{pp}(s) - A_{pi}(s)\, T(s), \qquad v_i \;=\; -\,T(s)\, v_p \quad (b_i = 0). \tag{4.13}$$

This makes the implementation and complexity transparent: build T once (cost $\approx n_p$ solves with A_{ii}), solve the small port system, and recover v_i by a single triangular solve (or by applying T).

Equivalent formulations and intuition.

- **Admittance seen at the ports.** If $A(s)$ is symmetric (e.g., passive reciprocal networks) and $A_{ii}(s)$ is SPD, then the Schur complement $S_p(s)$ in (4.10) is also symmetric and represents the *driving-point admittance* perceived at the ports after eliminating the internal variables.

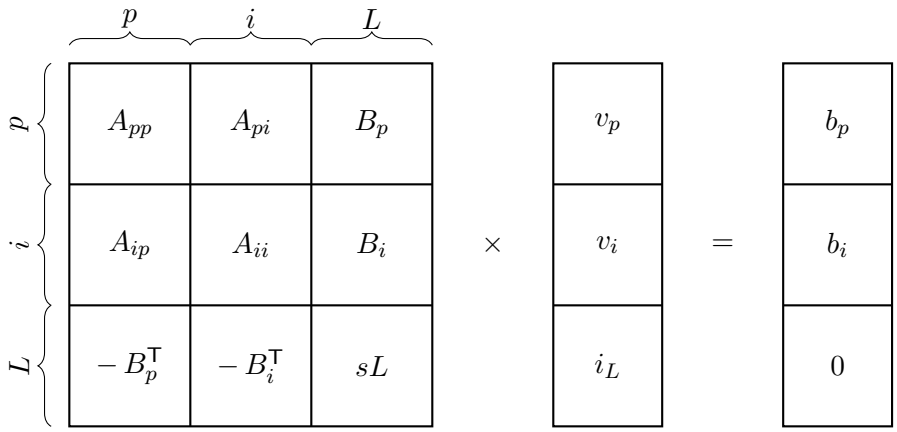


Figure 4.1: Classical MNA block structure with variables ordered as ports (p), internal nodes (i), and inductor currents (L).

- **No explicit inverse.** In practice we never form $A_{ii}(s)^{-1}$. Instead, we compute $T(s) := A_{ii}(s)^{-1}A_{ip}(s)$ by solving $A_{ii}(s)\,T(s) = A_{ip}(s)$ column-wise, and then set $S_p(s) = A_{pp}(s) - A_{pi}(s)\,T(s)$ as in (4.13).

- **Exact equivalence.** Under $A_{ii}(s)$ nonsingular, solving the reduced port system (4.11) and then back-substituting via (4.12) yields exactly the same (v_p, v_i) as solving the full condensed system.

Why this reduction is effective.

- The number of ports n_p is small, so the reduced system (4.11) is cheap to solve directly.

- The expensive work is confined to solves with $A_{ii}(s)$; we treat them iteratively with preconditioning (next subsection).

- For multiple port excitations, one can reuse $S_p(s)$ (or the intermediate $T(s)$) as well as any factors/preconditioners of $A_{ii}(s)$, amortizing setup costs.

4.2 Solution Strategy via Schur Complement and GMRES

We consider the solution of the linear system arising from the Modified Nodal Analysis (MNA) formulation in circuit simulation. The global MNA matrix at a given frequency is written as

$$A \;=\; G \;+\; j\omega\, C \;+\; \frac{1}{j\omega}\, A_L^{\mathsf{T}} S\, A_L, \tag{4.14}$$

where G and C are the conductance and capacitance matrices, while $A_L^{\mathsf{T}} S A_L$ represents the contribution of inductive and controlled sources.

Since direct solvers become prohibitive in terms of memory and runtime for large-scale systems, we adopt the Krylov method **GMRES**, which is well suited for nonsymmetric and indefinite matrices. Our implementation follows the port-based Schur reduction strategy presented as **Algorithm 3.1** in the cited dissertation [22], combined with an **ILU preconditioner** for the large internal block.

Procedure (Schur–Krylov with ILU):

1. **MNA matrix formation.** Assemble A as in (4.14); set the excitation at a selected port k via $b_p = e_k$, $b_{ii} = 0$.

2. **Block partitioning.** Reorder and split A into port vs. internal blocks

$$A = \begin{bmatrix} A_{pp} & A_{pi} \\ A_{ip} & A_{ii} \end{bmatrix}, \qquad x = \begin{bmatrix} x_p \\ x_{ii} \end{bmatrix}, \qquad b = \begin{bmatrix} b_p \\ b_{ii} \end{bmatrix}.$$

   The number of ports n_p is small, while A_{ii} is large and sparse.

3. **Port Schur complement.** Form

$$S_p \;=\; A_{pp} - A_{pi}\, A_{ii}^{-1} A_{ip},$$

   without explicitly inverting A_{ii}: compute $T = A_{ii}^{-1} A_{ip}$ by solving $A_{ii} T = A_{ip}$ (column- or block-wise). Reuse T across multiple excitations.

4. **Reduced port solve.** Solve the small system $S_p x_p = e_k$ (directly), which yields the port unknowns with cost $\mathcal{O}(n_p^3)$; n_p is typically very small.

5. **Internal right-hand side.** Assemble the right-hand side for the internal block

$$\mathrm{rhs} \;=\; -A_{ip}\, x_p \;=\; -A_{ip}\, S_p^{-1} e_k,$$

   coming from $A_{ip} x_p + A_{ii} x_{ii} = 0$.

6. **Preconditioning (ILU).** Compute an incomplete LU factorization of A_{ii} to obtain $M \approx A_{ii}$, and use M^{-1} as preconditioner (for this implementation right preconditioning is applied). This step targets clustering of the spectrum and reduction of GMRES iterations; parameters (drop tolerance / fill level) are tuned to balance fill and accuracy.

7. **Iterative solve (GMRES).** Solve $A_{ii}x_{ii} = \mathrm{rhs}$ with GMRES preconditioned by M^{-1} (typical settings: $\texttt{tol} = 10^{-6}$, $\texttt{maxit} = 400$;) and stop when the *relative residual* satisfies
$$\frac{\|r_k\|_2}{\|r_0\|_2} \le \texttt{tol}, \qquad r_k := b - A_{ii}x_{ii}^{(k)}.$$

8. **Assembly of the full solution.** Recover the total state vector as $x = \begin{bmatrix} x_p; & x_{ii} \end{bmatrix}$. If multiple port excitations are required, reuse the ILU factors of A_{ii} and the previously computed T to amortize costs.

This Schur–Krylov workflow exploits the small dimension of the port subsystem and treats the large internal block iteratively under ILU preconditioning, yielding substantial memory savings over direct solvers while preserving robustness for large-scale circuit simulations [22].

Algorithm 6 Schur Complement with ILU Preconditioning and GMRES

Require: Matrices G, C, A_L, S, frequency f, number of ports n_p, excitation port k

Ensure: Solution $x = \begin{bmatrix} x_p \\ x_{ii} \end{bmatrix}$

$A \leftarrow G + f\,C + \frac{1}{f} A_L^{\mathsf{T}} S A_L$ ▷ MNA system matrix

Partition A as $A = \begin{bmatrix} A_{pp} & A_{pi} \\ A_{ip} & A_{ii} \end{bmatrix}$, with n_p port variables

$e_k \leftarrow$ unit vector in $\mathbb{R}^{n_p}$ with 1 at position k

$S_p \leftarrow A_{pp} - A_{pi}\,A_{ii}^{-1} A_{ip}$ ▷ Schur complement

$x_p \leftarrow S_p^{-1} e_k$ ▷ port subsystem solution

$\mathrm{rhs} \leftarrow -A_{ip}\,x_p$ ▷ internal block right-hand side

$(L, U) \leftarrow \mathrm{ILU}(A_{ii})$ ▷ ILU preconditioner (MATLAB/MKL)

$x_{ii} \leftarrow \mathrm{GMRES}(A_{ii}, \mathrm{rhs}; \text{precond } (L, U), \texttt{tol}, \texttt{maxit})$

$x \leftarrow [x_p; x_{ii}]$

return x

Chapter 5

Implementation and Results

5.1 Chapter Summary

This chapter presents the practical implementation of the solver developed in the context of this work in the AC (phasor) setting, where $s = j\omega$ and each frequency point leads to a complex sparse linear system.. The core methodology is the application of the **Schur–GMRES with ILU preconditioning**, which is compared against two alternative approaches: (i) plain GMRES on the full matrix without Schur reduction, and (ii) direct LU factorization. All methods have been tested on **industrial matrices** and evaluated both on targeted frequencies and frequency sweeps, ensuring that the evaluation reflects realistic large-scale problems.

5.2 Objectives and Specifications

- **Input:** Matrix Market files `g.mtx`, `c.mtx`, `al.mtx`, `s.mtx` (conductance G, capacitance C, inductive incidence A_L, the inverse inductance matrix $S = L^{-1}$). For each frequency ω we assemble

$$A(j\omega) = G + j\omega\, C + \frac{1}{j\omega}\, A_L^T L^{-1} A_L,$$

  and construct the corresponding right-hand side $b(j\omega)$. Parsing is done with **CXSparse** (CSC), then converted/mapped to **Eigen** sparse types.

- **Output:** Solution vector $x(j\omega)$, GMRES convergence histories, execution time, and

accuracy metrics (relative residuals; deviation from the direct LU reference). For plotting, we also export *magnitude/phase* of selected quantities (Bode style).

- **Constraints:** AC (frequency-domain), complex-valued systems.
- **Goal:** Demonstrate in practice that the Schur–GMRES approach is beneficial compared to a direct solution, while preserving accuracy.

5.3 Architecture and Data Flow

5.3.1 Sparse Data Structures & Formats

- **Parsing/Storage:** Matrix Market files (`g.mtx`, `c.mtx`, `al.mtx`, `s.mtx`) are read with **CXSparse** (helper `io_mtx.cpp`) into CSC format and then converted into `Eigen::SparseMatrix<std::complex<double>>` (column-major).
- **Linear algebra:** Implemented using **Eigen** (Iterative + Direct modules). Direct solves are performed with the Eigen PARDISO interface `PardisoLU<SpMat>`. Preconditioning uses `Eigen::IncompleteLUT<std::complex<double>>`.
- **GMRES:** Custom right-preconditioned GMRES (no restart), recording the true residual at each iteration.
- **Outputs:** CSV files, relative errors.

5.3.2 Per-frequency pipeline

For each frequency f (with $\omega = 2\pi f$ and $j = \sqrt{-1}$), the system matrix is assembled as

$$A(j\omega) \;=\; G \;+\; j\omega\, C \;-\; \frac{j}{\omega}\,\Gamma, \quad \text{with } \Gamma = A_L^T\, S\, A_L,$$

where A_L is the incidence matrix of the inductive elements and $S = L^{-1}$ the inductance block provided in the dataset.The frequency sweep is logarithmic (`logspace`) e.g. over $[10^3, 10^9]$ Hz (configurable). The workflow is:

1. **Load & convert:** Read `g.mtx`, `c.mtx`, `al.mtx`, `s.mtx` via CXSparse–CSR, convert to `SpMat` (Eigen). Compute $\Gamma = A_L^T S A_L$ and form $A(j\omega)$.
2. **RHS:** Define $b = e_1$ (unit excitation at the first port/node).

3. **Partitioning:** With a given number of ports `nports`, reorder unknowns as

$$A = \begin{bmatrix} A_{pp} & A_{pi} \\ A_{ip} & A_{ii} \end{bmatrix}, \quad x = \begin{bmatrix} x_p \\ x_i \end{bmatrix}.$$

4. **Baseline (direct):** Solve the full $A(j\omega)$ using *PARDISO LU* as accuracy/time reference.

5. **Schur path:**

   (α′) Factorize A_{ii} with *PARDISO LU*.

   (β′) Compute $T = A_{ii}^{-1} A_{ip}$ via successive solves.

   (γ′) Form $S_p = A_{pp} - A_{pi} T$ and solve the small dense port system for x_p.

   (δ′) Form the internal RHS $A_{ii} x_i = -A_{ip} x_p$ and solve with **GMRES+IncompleteLUT** (right-preconditioning).

   (ε′) Assemble $x_{\text{Schur}} = [\, x_p;\ x_i \,]$ and check the residual on the full A.

6. **Plain GMRES path:** Solve the full $A(j\omega)$ with **GMRES+IncompleteLUT**, using the same tolerances/iteration limits, without Schur reduction.

7. **Metrics & logging:** For each frequency:

   - timings via `std::chrono`,
   - relative residual $\|Ax - b\|/\|b\|$ (LU, Schur+GMRES, GMRES-A),
   - *magnitude/phase* of the selected solution entry $x[k]$.

5.3.3 Fixed-frequency procedure

At a fixed frequency (e.g., 10^4 Hz), we perform a detailed comparison:

- Form S_p with **SparseLU** on A_{ii} for transparency,
- run **GMRES** on A_{ii} (Schur path) and on full A (plain path),
- log full convergence histories (true residual per iteration) in .csv file,
- directly compare LU vs GMRES on A_{ii} in terms of runtime and residual.

5.4 Pseudocode & Code Examples

5.4.1 Matrix Assembly (AC / phasor)

The following code reflects the implementation with CXSparse (parsing) and Eigen (assembly). Here `g.mtx`, `c.mtx`, `al.mtx`, `s.mtx` correspond to G, C, A_L, and $S = L^{-1}$ respectively.

```
// Convert CXSparse CSR -> Eigen SpMat<complex<double>> (column-major)
SpMat g = csr_to_eigen_complex(g_cs);
SpMat c = csr_to_eigen_complex(c_cs);
SpMat al = csr_to_eigen_complex(al_cs);
SpMat S  = csr_to_eigen_complex(s_cs);

vector<double> freqs = logspace(1e3, 1e9, 5);     // logarithmic sweep
for (double f : freqs) {
    double omega = 2.0 * M_PI * f;
    Complexd j(0.0, 1.0);

    // Γ = A_L^T * S * A_L
    SpMat alT = al.adjoint();
    SpMat Gamma = alT * S * al;
    Gamma.makeCompressed();

    // A(jω) = G + jω C - (j/ω) Γ
    SpMat A = g + j * omega * c - (j * (1.0 / omega)) * Gamma;
    A.makeCompressed();
}
```

5.4.2 Port/Interior Partition & Schur Complement + Inner GMRES

We reorder based on ports (4 here) (p first) and internals (i), form the 2×2 block matrix

$$A = \begin{bmatrix} A_{pp} & A_{pi} \\ A_{ip} & A_{ii} \end{bmatrix}.$$

```
// 1) Block extraction: [pp, pi; ip, ii]
SpMat  A_ip = A.block(nports, 0, n-nports, nports);     // ni x p
SpMat  A_ii = A.block(nports, nports, n-nports, n-nports);  // ni x ni
MatrixXcd A_pp = MatrixXcd(A.block(0, 0, nports, nports));     //p x p
MatrixXcd A_pi = MatrixXcd(A.block(0, nports, nports, n-nports));//p x ni

// RHS: b = e_1
VectorXcd b_full = VectorXcd::Zero(n);
b_full(0) = Complexd(1.0,0.0);

// 2) Factorize A_ii once with PARDISO (reused)
A_ii.makeCompressed();
PardisoLU<SpMat> fac_ii;
fac_ii.analyzePattern(A_ii);
fac_ii.factorize(A_ii);

// 3) Compute T = A_ii^{-1} A_ip column-by-column
const int p = nports;
```

```
const int ni = n - p;
MatrixXcd T(ni, p);
for (int c = 0; c < p; ++c) {

    VectorXcd rhs_col = VectorXcd( A_ip.col(c) );   // rhs = A_ip(:, c)
    T.col(c) = fac_ii.solve(rhs_col);   // T(:,c) = A_ii^{-1} * rhs_col
}

// 4) Dense Schur complement S_p = A_pp - A_pi * T, and port solve x_p
MatrixXcd Sschur = A_pp - A_pi * T;               // (p x p)
VectorXcd b_p = VectorXcd::Zero(p);  b_p(0) = Complexd(1,0);
VectorXcd b_i = VectorXcd::Zero(ni);              // here b_i = 0
VectorXcd z_bi = fac_ii.solve(b_i);               // A_ii^{-1} b_i
VectorXcd b_S = b_p - A_pi * z_bi;                // RHS for port system
VectorXcd x_p = Sschur.colPivHouseholderQr().solve(b_S);

// 5) Internal solve: A_ii x_i = -A_ip x_p, with GMRES
VectorXcd rhs_i = -A_ip * x_p;

GMRESResult inner = gmres_right_preconditioned(A_ii, rhs_i, tol,
                                               maxit, droptol, fill);
VectorXcd x_i = inner.x;

// Reconstruct full solution [x_p; x_i] and check residual on A
VectorXcd x_schur(n); x_schur << x_p, x_i;
double relres_schur = (A * x_schur - b_full).norm() / b_full.norm();
```

5.4.3 GMRES on the Full Matrix (no Schur)

For comparison we also solve the full system with right-preconditioned GMRES (IncompleteLUT) using the same tolerances.

```
set options: tol, maxit, droptol, fill;
GMRESResult full = gmres_right_preconditioned(A, b_full, tol, maxit,
                                              droptol, fill);
VectorXcd x_gmresA = full.x;
double relres_gmresA = (A * x_gmresA - b_full).norm() / b_full.norm();
```

5.4.4 Direct Method LU (PARDISO LU on full A)

We use Eigen's PARDISO interface as a direct baseline and accuracy/time reference.

```
PardisoLU<SpMat> pd;
pd.analyzePattern(A);
pd.factorize(A);
VectorXcd x_lu = pd.solve(b_full);
double relres_lu = (A * x_lu - b_full).norm() / b_full.norm();
```

5.4.5 GMRES (right-preconditioned, complex)

We solve $Ax = b$ with a *right* preconditioner $M \approx A$ (ILUT), i.e., solve $(AM^{-1})\, y = b$ and recover $x = M^{-1}y$. The implementation uses ILUT (drop tolerance, fill factor), complex

Arnoldi, and complex Givens rotations.

```
// Right-preconditioned GMRES with ILUT (complex)
// Returns solution, flags, iterations, residuals, time.
GMRESResult gmres_right_preconditioned(SpMat& A, VectorXcd& b,
                                       double tol, int maxit,
                                       double droptol, double fillfactor)
{
    // Initialisation

    const double bnorm = b.norm();
    if (bnorm == 0.0) { out.converged=true; ....}

    // --- Preconditioner M ≈ A (ILUT) ---
    Eigen::IncompleteLUT<Complexd> M;
    M.setDroptol(droptol);
    M.setFillfactor(fillfactor);
    M.compute(A);

    // --- Arnoldi/GMRES state ---
    VectorXcd r0 = b;                // initial residual (x0=0)
    const double beta = r0.norm();
    const int m = std::min(maxit, n);

    std::vector<VectorXcd> V; V.reserve(m+1);
    std::vector<VectorXcd> Z; Z.reserve(m);   // Z_j = M^{-1} v_j
    V.emplace_back(r0 / beta);

    MatrixXcd H = MatrixXcd::Zero(m+1, m);    // (m+1) x m Hessenberg
    std::vector<Complexd> cs(m, Complexd(0,0)), sn(m, Complexd(0,0));
    VectorXcd g = VectorXcd::Zero(m+1); g(0) = beta;

    out.resvec.push_back(beta / bnorm);       // true residual at iter 0
    int j_converged = -1;

    for (int j = 0; j < m; ++j) {
        // z_j = M^{-1} v_j ; w = A z_j
        VectorXcd z = apply_prec(V[j]);       Z.emplace_back(z);
        VectorXcd w = A * z;

        // Modified Gram-Schmidt
        for (int i = 0; i <= j; ++i) {
            H(i,j) = V[i].dot(w);
            w -= H(i,j) * V[i];
        }
        H(j+1,j) = w.norm();
        const Complexd hij = H(j+1, j);
        VectorXcd vnext = VectorXcd::Zero(n);
        if (st::abs(hij) > 0.0) {
            vnext = w / hij;
        }
        V.emplace_back(std::move(vnext));

        // Apply old Givens to column j
        for (int i = 0; i < j; ++i) {
            const Complexd h_ij = H(i,j), h_ipj = H(i+1,j);
            H(i,  j) = std::conj(cs[i]) * h_ij + std::conj(sn[i]) * h_ipj;
            H(i+1,j) = -sn[i] * h_ij + cs[i] * h_ipj;
        }

        // New Givens to zero H(j+1,j)
        const Complexd a = H(j,j), b_ = H(j+1,j);
```

```
        const double denom = std::sqrt(std::norm(a) + std::norm(b_));
        cs[j] = (denom==0.0) ? Complexd(1,0) : a / denom;
        sn[j] = (denom==0.0) ? Complexd(0,0) : b_ / denom;
        H(j,  j) = std::conj(cs[j]) * a + std::conj(sn[j]) * b_;
        H(j+1,j) = Complexd(0,0);

        // Update g and reported residual
        const Complexd gj = g(j), gjp1 = g(j+1);
        g(j)   = std::conj(cs[j]) * gj + std::conj(sn[j]) * gjp1;
        g(j+1) = -sn[j] * gj + cs[j] * gjp1;
        out.relres_reported = std::abs(g(j+1)) / bnorm;
        out.iters = j + 1;

        // Compute current x_k and true residual (for logging)
        MatrixXcd Rj = H.topLeftCorner(j+1, j+1);
        VectorXcd gh = g.head(j+1);
        VectorXcd y  = VectorXcd::Zero(j+1);      // backsolve Rj y = gh
        for (int i = j; i >= 0; --i) {
            Complexd s = gh(i);
            for (int k = i+1; k <= j; ++k) s -= Rj(i,k) * y(k);
            y(i) = s / Rj(i,i);
        }
        VectorXcd xk = VectorXcd::Zero(n);            // x_k = Σ y_k Z_k
        for (int k = 0; k <= j; ++k) xk += y(k) * Z[k];
        const double rr_true_k = (b - A * xk).norm() / bnorm;
        out.resvec.push_back(rr_true_k);

        if (out.relres_reported < tol) { j_converged = j; break; }
    }

    // Solution assembly (x = Σ y_k Z_k)
    const int jlast = (j_converged >= 0) ? j_converged : (out.iters - 1);
    if (jlast >= 0) {
        MatrixXcd R = H.topLeftCorner(jlast+1, jlast+1);
        VectorXcd gh = g.head(jlast+1);
        VectorXcd y  = VectorXcd::Zero(jlast+1);
        for (int i = jlast; i >= 0; --i) {
            Complexd s = gh(i);
            for (int k = i+1; k <= jlast; ++k)
                s -= R(i,k) * y(k);
            y(i) = s / R(i,i);
        }
        VectorXcd dx = VectorXcd::Zero(n);
        for (int k = 0; k <= jlast; ++k)
            dx += y(k) * Z[k];
        out.x = dx;
    }
    out.converged = (out.relres_reported < tol);
    out.relres_true = (b - A * out.x).norm() / bnorm;
    return out;
}
```

5.5 Results: runtime, convergence, and accuracy

The problem matrix has **nnz** $= 1{,}350{,}918$ and **cond**$(A) = 3.3803 \times 10^{18}$ (severely ill-conditioned). Solver settings: `maxit` $= 400$, `tol` $= 10^{-6}$, ILU with `droptol` $= 10^{-5}$ and `fill` $= 10$. The residual is always computed on the full system $\|b - Ax\|/\|b\|$. For the Schur

Table 5.1: Solver settings and problem characteristics.

Maximum iterations	$\texttt{maxit} = 400$
Convergence tolerance	$\texttt{tol} = 10^{-6}$ (global residual)
ILU parameters	$\texttt{droptol} = 10^{-5}$, $\texttt{fill} = 10$
Nonzeros in A	1,350,918
Condition number	$\mathrm{cond}(A) = 3.3803 \times 10^{18}$

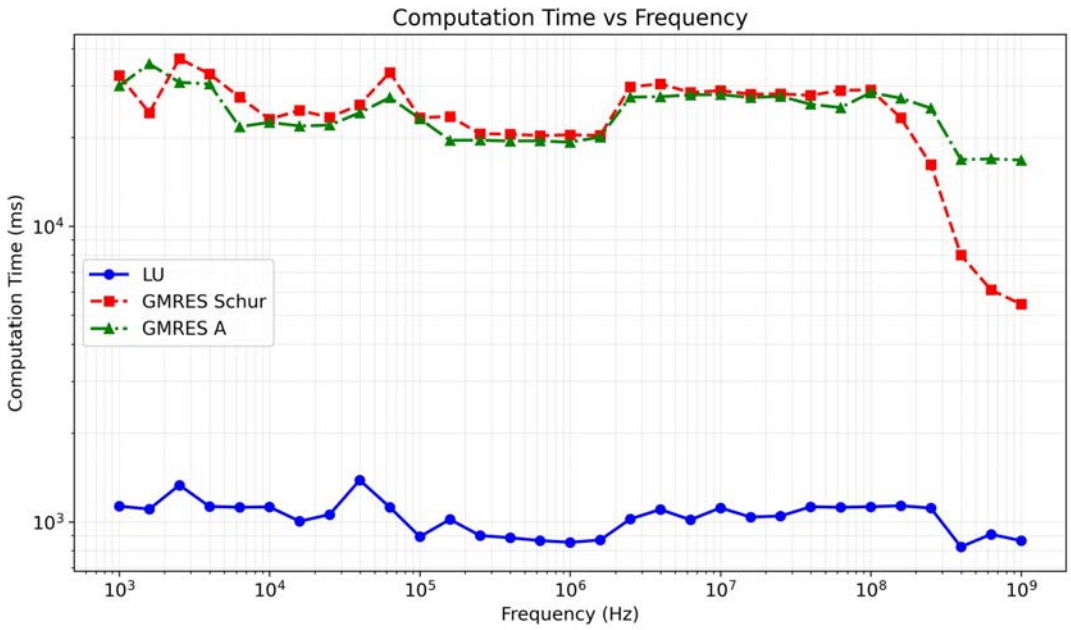
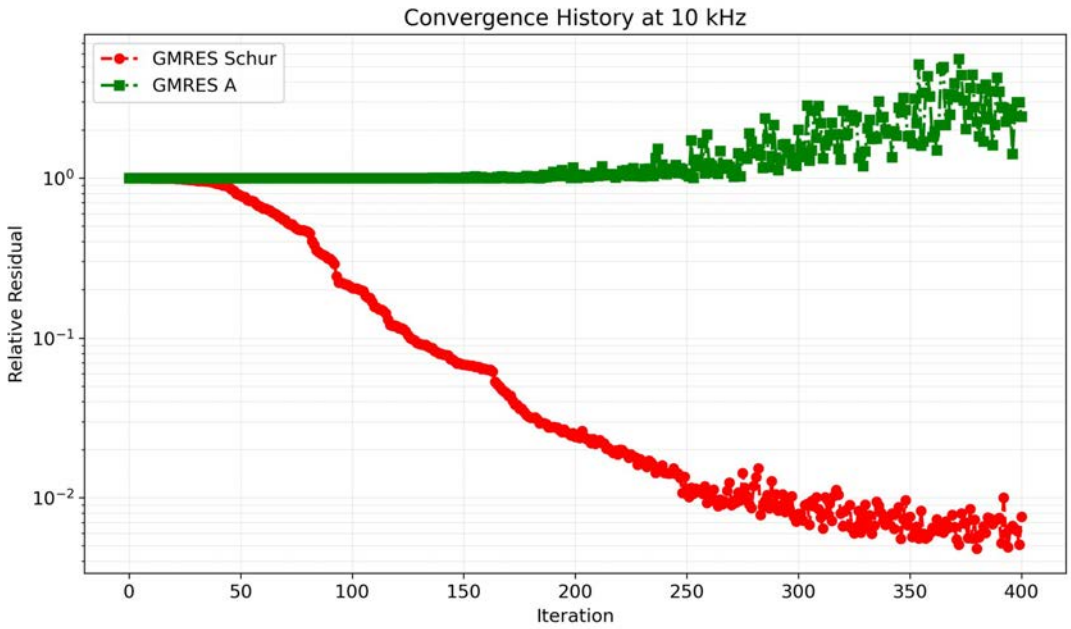
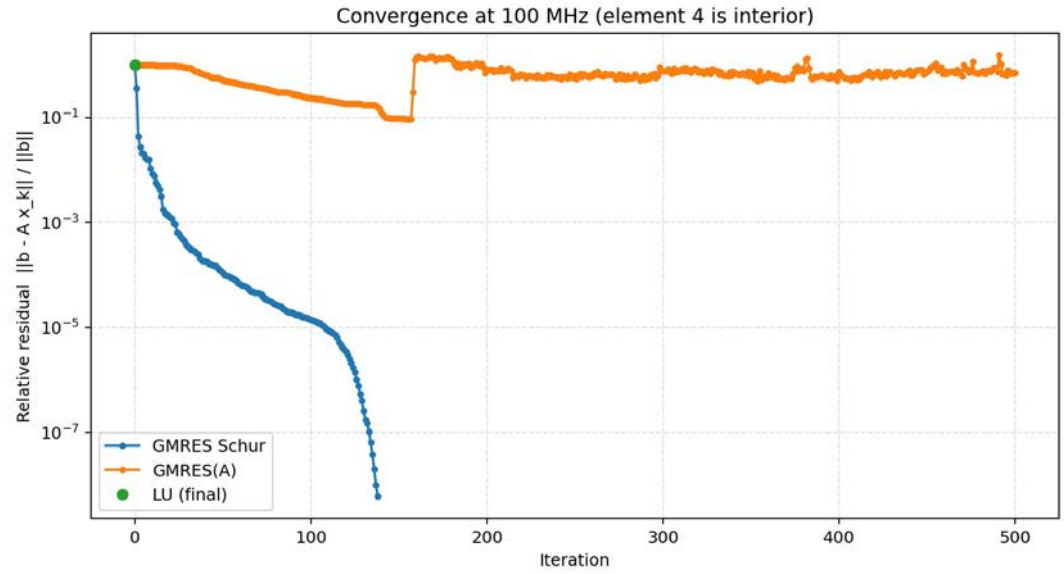


Figure 5.1: **Computation time vs frequency.** Solve time (1 RHS) versus frequency. LU is consistently faster in the *solve* ($\sim$ 0.8–1.3 s), while the iterative methods are in the tens of seconds. At high frequencies ($> 3 \times 10^8$ Hz) GMRES(Schur) improves. Here we solve-only, the setup cost is not included.

approach we reconstruct the interior unknowns by back–substitution, $x_{ii} = A_{ii}^{-1}(b_i - A_{ip}x_p)$, and form $x_{\text{iter\_schur}} = [\, x_p \,;\, x_{ii} \,]$.



(α′) Convergence at 10 kHz: relative residual $\|b - Ax_k\|/\|b\|$. GMRES(A) stays around 10^0, whereas GMRES(Schur) decreases smoothly below 10^{-2}.



(β′) Convergence at 100 MHz: GMRES(A) shows non stagnation around 10^{-1}, while GMRES(Schur) reaches $< 10^{-8}$ in $\sim 1.3 \times 10^2$ iterations.

Figure 5.2: **Convergence histories at low and high frequency.**

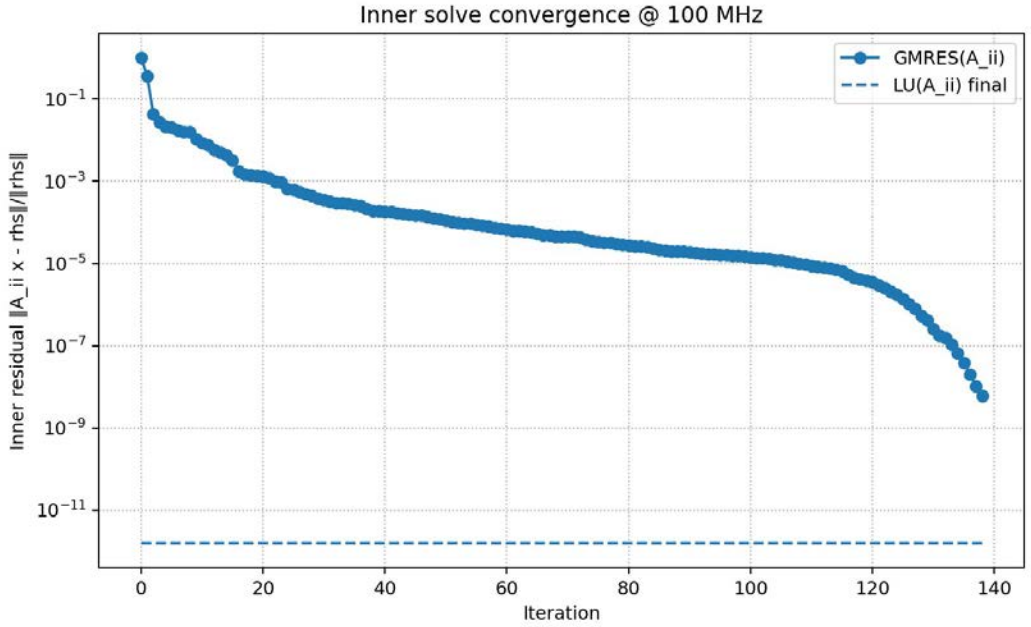


Figure 5.3: **Inner solve on A_{ii} at 100 MHz.** Inner residual $\|b_i - A_{ii}z\|/\|b_i\|$ drops from $\sim 10^{-1}$ to $\sim 10^{-8}$, sufficient for inexact preconditioning of the outer GMRES.

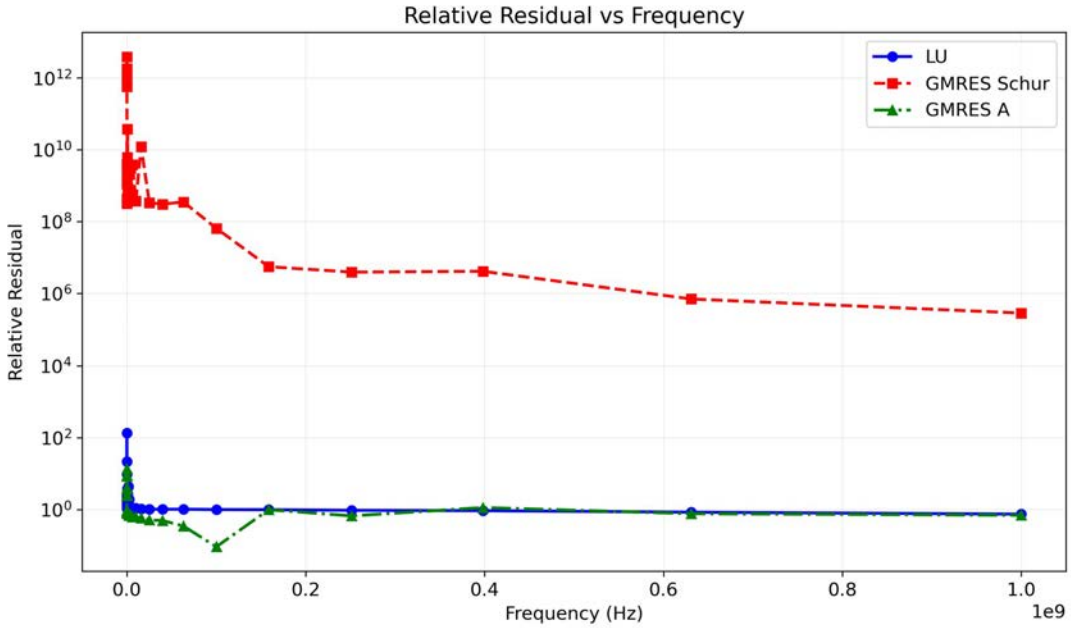


Figure 5.4: **Residual vs frequency.** Global residual $\|b - Ax\|/\|b\|$ across the sweep (log scale). In the final version we always measure the residual after full reconstruction $x = [x_p;\, x_{ii}]$.

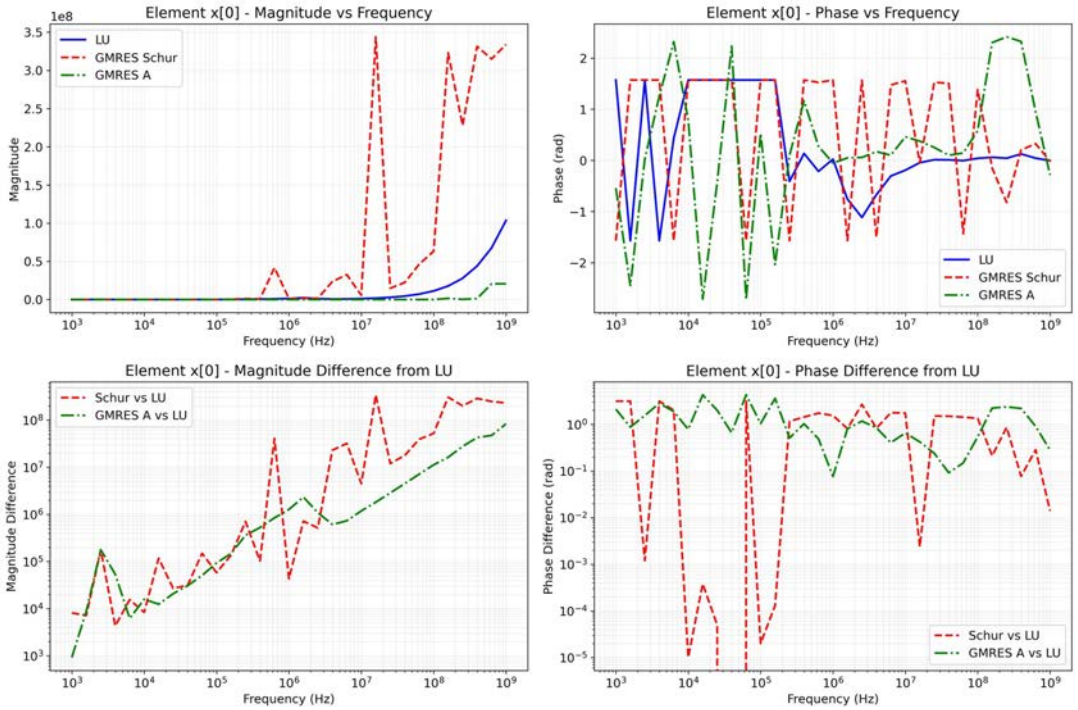


Figure 5.5: **Magnitude/phase sweep for $x[0]$.** If $x[0]$ is interior, deviations without back–substitution are expected; spikes tend to coincide with frequencies where outer GMRES does not converge. After full back-substitution and global residual check, the Schur solution aligns with LU within tolerance.

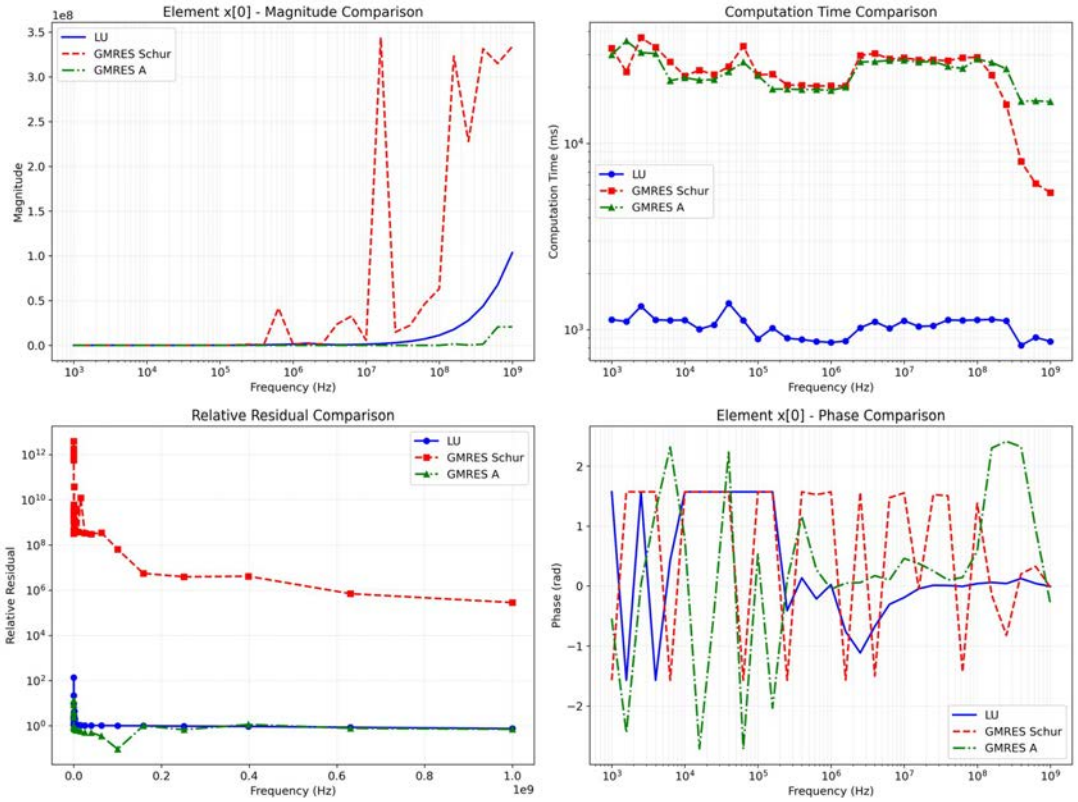


Figure 5.6: **Summary for $x[0]$.** Time, residual, magnitude, phase across methods.

5.5.1 Conclusions

The problem is strongly ill-conditioned (cond$(A) \approx 3.38\times10^{18}$), which explains the stagnation of GMRES(A) around 10^{0}–10^{-1} at low frequencies and its non-monotone behaviour at higher ones. As shown in Figure 5.2α′, at 10 kHz GMRES(A) barely makes progress, whereas GMRES on the Schur complement decreases the residual steadily. At 100 MHz, Figure 5.2β′ shows GMRES(Schur) reaching below 10^{-8} in roughly 1.3×10^{2} iterations, while GMRES(A) stays near 10^{-1}. We do not use restart; the small wiggles are consistent with numerical effects (loss of orthogonality / inexact operator). The inner solve on A_{ii} in Figure 5.3 drops from about 10^{-1} to 10^{-8}, which is adequate for inexact preconditioning.

Regarding cost and residual across the sweep, Figure 5.1 confirms that LU is consistently faster in solve-only time for one RHS (about 0.8–1.3 s), while the iterative methods are in the tens of seconds; at higher frequencies ($>3\times10^{8}$ Hz) GMRES(Schur) improves as the iteration count goes down. Importantly, Figure 5.4 also shows that even LU does not always maintain a very small *global* residual—this is not surprising for such an ill-conditioned system and without aggressive scaling; roundoff can be visible. Throughout, the error metric is the global $\|b - Ax\|/\|b\|$ after full reconstruction $x = [x_p; x_{ii}]$, and where GMRES(Schur) converges and this global residual is measured, the curve stays low and smooth; spikes line up with non-convergent points.

For accuracy, at the *ports* the Schur approach reproduces LU within tolerance (magnitude/phase) whenever the global residual meets the target; deviations of GMRES(A) coincide with stagnation regions. For *interior* unknowns such as $x[0]$ and $x[4]$, comparison must be done *after* back-substitution $x_{ii} = A_{ii}^{-1}(b_i - A_{ip}x_p)$ and permute-back; otherwise differences are expected. As seen in Figures 5.5 and 5.6, spikes without full reconstruction reflect either non-convergence or measurement on the reduced system. When GMRES(Schur) converges and back-substitution is applied, the interior values agree with LU within tolerance.

Chapter 6

Conclusion and future work

6.1 Summary and Conclusions

This thesis examines iterative solvers, namely GMRES, for the complex linear systems produced by Modified Nodal Analysis (AC, $s = j\omega$). The workflow splits variables into ports and internals and forms a Schur complement to eliminate the interior unknowns, so the Krylov solver works on a much smaller system. We also assess incomplete LU (ILUT) as a preconditioner to speed up convergence.

The numerical experiments revealed several significant findings. First, using the Schur complement was especially effective when the number of ports remained small in comparison to the overall system size, since it lowered the computational load on the iterative solver. Second, unpreconditioned GMRES frequently demonstrated delayed convergence, but the use of ILU resulted in a significant boost in performance. Third, compared with a direct LU factorisation, the iterative technique required significantly less memory[1], although the runtime was not necessarily shorter for small and medium-sized systems. Nonetheless, as the system size expanded, the iterative technique became more competitive and scalable, demonstrating its promise for dealing with large-scale problems where direct solvers are impractical.

In conclusion, the study found that combining Schur complement reduction with preconditioned GMRES provides a feasible and adaptable framework for addressing the types of systems encountered in circuit analysis.

[1] Results during internship at ANSYS, which are held confidential.

6.2 Future Work

The approach proposed here provides many obvious paths for further research and extension:

- **Advanced preconditioning techniques.** Future research might look at more advanced preconditioners, such as block-based techniques or multilevel ILU versions, with the objective of increasing resilience and convergence for larger and more difficult problems.

- **Parallel and accelerated implementations.** Adapting the solver to use multicore processors or GPUs would be a significant step towards lowering runtime and allowing the solution of problems that are presently computationally prohibitive.

- **Alternative Krylov subspace methods.** While this study focused on GMRES, other Krylov subspace approaches such as BiCGStab, QMR, or MINRES may be beneficial depending on the structure of the system.

- **Multiple right-hand sides.** Block-GMRES or similar techniques that can handle several right-hand sides simultaneously would be extremely useful in situations requiring efficient resolution of repeated excitations or input conditions.

Bibliography

[1] C.-W. Ho, A. E. Ruehli, and P. A. Brennan. The modified nodal approach to network analysis. *IEEE Transactions on Circuits and Systems*, 22(6):504–509, 1975.

[2] F. N. Najm. *Circuit Simulation*. Wiley, Hoboken, New Jersey, 2010.

[3] R. W. Freund. Sprim: Structure-preserving reduced-order interconnect macromodeling. In *Proc. IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 80–87, 2004.

[4] J. Przewocki, L. Kulas, and M. Mrozowski. Digital system interconnects analysis using model order reduction methods. In *Proc. IEEE/ACM Design, Automation and Test in Europe Conference (DATE)*, pages 628–633, 2002.

[5] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, Maryland, 4th edition, 2013.

[6] Timothy A. Davis. *Direct Methods for Sparse Linear Systems*. SIAM, Philadelphia, PA, 2006.

[7] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2nd edition, 2003.

[8] Σαρρής Ι., Καρακασίδης Θεόδωρος. *ΑΡΙΘΜΗΤΙΚΕΣ ΜΕΘΟΔΟΙ & ΕΦΑΡΜΟΓΕΣ ΓΙΑ ΜΗΧΑΝΙΚΟΥΣ με παραδείγματα στο Matlab*. Εκδόσεις Τζιόλα, 4th edition, 2017.

[9] Cornelis Vuik. Lecture notes: Iterative methods for linear systems, 2016.

[10] Gérard Meurant. *Direct Methods for Solving Linear Systems*. Springer, Paris, 2024.

[11] Owe Axelsson. *Iterative Solution Methods*. Cambridge University Press, Cambridge, UK, 1st paperback ed. edition, 1996.

[12] David R. Kincaid and Linda J. Hayes. *Iterative Methods for Large Linear Systems*. Academic Press, San Diego, CA, 1990.

[13] Henk A. van der Vorst. *Krylov Subspace Methods: Iterative Methods for Large Linear Systems*. Cambridge University Press, Cambridge, UK, 2003.

[14] M. Benzi. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, 182(2):418–477, 2002.

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

[16] Richard Barrett, Michael Berry, Tony F. Chan, James Demmel, June M. Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia, 2 edition, 1994.

[17] George E. Karniadakis and Robert M. Kirby II. *Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and Their Implementation*. Cambridge University Press, Cambridge, UK, 2003.

[18] Howard C. Elman. Approximate schur complement preconditioners on serial and parallel computers. *SIAM Journal on Scientific and Statistical Computing*, 10(3):581–605, 1989.

[19] Luc Giraud, Azzam Haidar, and Yousef Saad. Sparse approximations of the schur complement for parallel algebraic hybrid linear solvers in 3d. *Numerical Mathematics: Theory, Methods and Applications*, 3(3):276–294, 2010.

[20] Emmanuel Agullo, Luc Giraud, Abdou Guermouche, Azzam Haidar, and Jean Roman. Parallel algebraic domain decomposition solver for the solution of augmented systems. Technical Report RR-7516, INRIA, 2011.

[21] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, UK, 2004.

[22] Byron E. Moutafis. *Parallel computational methods for solving large linear systems*. Phd thesis, Democritus University of Thrace, 2020.