Simulation of dense mutual inductance systems with approximate matrix inversion

Ifigeneia Apostolopoulou
ifiaposto@gmail.com

October, 2013

University of Thessaly
Department of Electrical and Computer Engineering

Advisor: Nestoras Evmorfopoulos
Co-Advisor: Georgios Stamoulis

Contents

List of Tables

List of Figures

List of Algorithms

Περίληψη

Όσο η γεωμτρία των κυκλωμάτων γίνεται πιο πυκνή και η συχνότητα λειτουργίας αυξάνεται, τα on-chip φαινόμενα επαγωγής γίνονται ολοένα και πιο σημαντικά στην ανάλυση χρονισμού και θορύβου των ολοκληρωμένων κυκλωμάτων VLSI. Η μέθοδος PEEC (The Partial Element Equivalent Circuit) έχει ευρέως χρησιμοποιηθεί για να μοντελοποιήσει τα φαινόμενα επαγωγής. Ωστόσο, ο πίνακας επαγωγών L που προκύπτει είναι μεγάλος και πυκνός, καθιστώντας την προσομοίωση του κυκλώματος ασύμφορη λόγω των μεγάλων απαιτήσεων σε μνήμη και χρόνο. Έχει παρατηρηθεί ότι ο αντίστροφος πίνακας επαγωγών $K = L^{-1}$ *μπορεί να θεωρηθεί προσεγγιστικά αραιός. Συνεπώς, το να θεωρήσουμε τον πίνακα K αραιό μπορεί να κάνει το πρόβλημα της προσομοίωσης της αμοιβαίας επαγωγής των κυκλωμάτων διαχειρίσιμο.*

Στην παρούσα διπλωματική εργασία μελετάμε διάφορες μεθόδους αραιής και προσεγγιστικής αντιστροφής του πίνακα L προκειμένου να επιτύχουμε γρήγορη και ακριβή προσομοίωση των φαινομένων αμοιβαίας επαγαγωγής. Επιπλέον προτείνουμε έναν εναλλακτικό αλγόριθμο ο οποίος προσεγγιστικά υπολογίζει κάποιες προδιαγεγραμμένες θέσεις του αντιστρόφου ενός πίνακα. Ακόμη, τροποποιούμε την ανάλυση κόμβων ώστε να γίνεται χρήση του αραιού προσεγγιστικού πίνακα $\widetilde{K}$ *αντί του L στη μεταβατική ανάλυση κυκλωμάτων. Τέλος, παρουσιάζουμε πειραματικά αποτελέσματα απο συγκεκριμμένες γεωμετρίες κυκλωμάτων προκείμενου να δείξουμε την επιτάχυνση και την ακρίβεια που επιτυγχάνεται όταν κάθε μία μέθοδος προσεγγιστικής και αραίης αντιστροφής εφαρμόζεται στον εκάστοτε πίνακα επαγωγών.*

Abstract

As technology shrinks further and clock speed increases, on-chip inductance effects become more significant in timing and noise analysis of VLSI circuits. The Partial Element Equivalent Circuit (PEEC) method has been widely used to model on-chip inductance effects. However, the resulting inductance matrix L *is large and dense which makes the simulation impractical due to the enormous demands on computational time and memory. It has been recognized that the reluctance or susceptance matrix* $K = L^{-1}$*, the inverse of the inductance matrix, is approximately sparse. Consequently, sparsification of the reluctance matrix can be applied to make the problem of the simulation of the inductance effects tractable.*

In this thesis, we explore several techniques for approximate sparsification of the K *matrix at the simulation level so as to achieve fast and accurate inductance simulation. Furthermore, we propose an alternative algorithm which approximately captures selected entries of the inverse of a matrix. Moreover, the nodal analysis formulation is modified in order to make use of the sparse approximate matrix* $\widetilde{K}$ *instead of* L *in transient analysis. Finally, results from specific cases are given to demonstrate the efficiency and accuracy achieved by each approximate sparsification method.*

1

Introduction

1.1 Problem Description

With the aggressive scaling of VLSI technology, the accurate modeling of inductance effects has become a problem of crucial importance. The inductive coupling effect becomes more important because of higher frequency signal content, denser geometries, more metal layers and the reduction of resistance by copper and capacitance by low-k dielectric. Inductance effect is present not only in IC but also in on-chip interconnects such as power grids, clock nets and bus structures. It causes signal overshoot, undershoot, oscillations and aggravates crosstalk and power-grid noises.

The major problem of inductance modeling is the uncertainty of return paths. Since inductance is a function of a closed loop, the return path is difficult to predict in advance before simulation. The partial equivalent elements circuit (PEEC) method based on partial inductances was proposed to overcome the difficulty in finding complete current loops in a real chip environment. In this approach, each conductor is segmented and for high frequencies, further subdivided into filaments. Each filament is represented by a resistor and a self-inductor in series and a capacitor to ground. Mutual inductors and capacitors are inserted as necessary for accurate modeling. The partial self and mutual inductances are defined with the assumption of infinite return paths (see section 2.2). However, the PEEC model results in a huge number of circuit elements and the partial inductance matrix L is large and dense. Consequently, direct simulation of the full L matrix is usually impractical , owning to the enormous demands it places on computation time and memory.

To effectively reduce the mutual inductance terms and speedup the simulation, sparsification is crucial. Many approaches sparsify either the inductance matrix L or its inverse $K = L^{-1}$, which is called susceptance or reluctance matrix. The mutual susceptance terms drop off much faster with distance than the corresponding mutual inductance terms. As a consequence, the reluctance matrix can be considered approximately sparse. However, $K = L^{-1}$ needs to be computed first, which is prohibitively expensive. In order to avoid full inversion, several techniques have been developed to obtain an approximate value for the nonzero entries in K matrix and are presented in this thesis.

1.2 Literature Overview

In this section, we briefly review existing methods that tackle the problem of unrealistic demands on both simulation time and memory, which stems from the large and full inductance matrix L. The obvious approach is that of the direct truncation of L, i.e., setting to zero off-diagonal entries that are smaller in magnitude than a threshold parameter. However, this approach can lead to unstable simulation, since the truncated L may not be symmetric and positive definite (SPD, see section 4.2). An alternative approach, which is based on the sparsification of L and guarantees the positive definiteness of the truncated L matrix, was proposed in [33]. However, this approach can lead to a high degree of inaccuracy. In [21], it was shown that the off-diagonal entries in the susceptance or reluctance matrix $K = L^{-1}$ diminish much faster than those in L and the sparsification of K was proposed. The simplest technique is that of direct truncation, where L is inverted and the small off-diagonal terms are truncated. In [21], a highly effective sparsification method of the K matrix at the extraction level, the K-method, was proposed: a small inductance sub-matrix of wires strongly coupled to the wire of interest is extracted and then inverted, and the corresponding row (or column) of the wire in the inverse forms the significant entries in the approximate inverse matrix denoted by $\widetilde{K}$ (see section 3.1). In [26], the mathematical foundation of the K-method is offered. Moreover, in order for the K-method to be applicable, a sparsity pattern of K, i.e., a coupling window of each wire, should be a priori known. In [34], a systematic approach for determining the coupling window and an incremental computation method of $\widetilde{K}$ is proposed. Since commercial simulation tools do not support the use of the reluctance matrix K, a double inversion method is proposed in [31], [20]. Specifically, once the sparse approximation of K is obtained, it is inverted. The resulting inductance $\tilde{L}$ contains far fewer significant entries than the original matrix L, hence it is much easier to sparsify. To preserve positive definiteness of $\tilde{L}$, the magnitude of canceled off-diagonals should be added to the corresponding diagonal element. However, this technique requires a large matrix inversion which can be expensive. Furthermore, the significant entries of $\tilde{L}$ and L can be significantly different resulting in a loss of simulation accuracy.

While the sparsification of the reluctance matrix K offers better simulation accuracy compared to those achieved by the truncation of the inductance matrix, the stability of this approximation is not guaranteed either, i.e., the sparse approximate reluctance matrix $\widetilde{K}$ may not be SPD. The stability of the K-method was proved in [22] based on the diagonal dominance of the reluctance matrix K and the fact that it has positive diagonal entries (7.2.3). The diagonal dominance property is derived from the assumption that $K_{ij} < 0\,, i \neq j$. However, if the targeting circuit presents irregular geometry, positive off-diagonal entries may occur. Therefore, there is no guarantee that the K matrix is diagonally dominant and the sparsification of K may lead to unstable simulation. In [23], it was noted that the reluctance matrix K is diagonally dominant when all conductors are sufficiently discretized and a reluctance extractor and simulator was proposed. Furthermore, it is considered that the reluctance elements K_{ij} can be directly extracted by setting unit magnetic flux on conductor i and zero to others (see sections 2.2, 2.3 for more details). However, the extraction of K elements is not supported by current commercial inductance extraction tools. In [27], an approximate sparsification technique of L^{-1}, which preserves the SPD property of the approximate sparse reluctance matrix when the reluctance matrix K can be adequately approximated by a banded matrix, was proposed. Specifically, it is proved that the approximate sparse reluctance matrix $\widetilde{K} = \tilde{L}^{-1}$ is SPD, the band entries of $\tilde{L}$ matrix match the corresponding entries of L and $\tilde{L}$ can be entirely constructed using only these band entries. The work in [28] constitutes a generalization of [27] where the sparse approximate reluctance matrix $\widetilde{K}$ is considered a multi-banded matrix. As a consequence, this method is applicable to 3D- interconnect structures. However, the previously mentioned techniques may not yield acceptable simulation accuracy when the conductors appear in irregular geometry and the corresponding reluctance matrix presents arbitrary sparsity pattern.

1.3 Thesis Contribution

Several window-based approaches have been developed to approximate $\widetilde{K}$ at the extraction level [34], [22], [20]. These approaches focus on determining the coupling window to be considered or, equivalently, the entries of the reluctance matrix $\widetilde{K}$ that should be approximated. However, all these techniques, given the coupling window of each conductor, perform a small sub-matrix inversion to obtain the entries of the corresponding column of $\widetilde{K}$, as it had already been proposed in the K-method [21]. Consequently, the point of interest was to determine which entries should be approximated rather than how to get their approximate value. Furthermore, if there are c_i nonzero entries in the i^{th} column of the $n \times n$ reluctance $\widetilde{K}$ matrix, the overall computation of $\widetilde{K}$ costs $\sum_{i=1}^{n} c_i^3$. For small window sizes, i.e., the constant c_i is small and the $\widetilde{K}$ presents a high sparsity ratio $\varepsilon = \left(1 - \frac{\#nonzeros}{\#entries}\right)$, K-method turns out to be extremely fast and has complexity $O(n)$. However, when large-sized inductance matrices are considered, in order to adequately capture the inductance effect a larger window, i.e., a greater c_i, should be used hence K-method can become expensive.

In this thesis, we first demonstrate the need for larger inductance windows in order to provide accuracy comparable to what is obtained when the full and exact K matrix is used, when the targeting matrix is large. Furthermore, we explore alternative approximate inverse techniques which can become more efficient and accurate compared to the K-method when larger window sizes are used so as to preserve a high degree of accuracy. We should note that we exploit the sparsity of K at the simulation level, as it was proposed in [29]. To this purpose, we review and incorporate into the simulation tool preconditioning techniques based on sparse approximate inverses [1], [2], [7], [8], [16] to obtain the sparse approximate matrix $\widetilde{K}$ which is subsequently used into the transient analysis (see section 4.3). Furthermore, we note that several approaches have been developed to obtain an approximation of the diagonal of the inverse of a matrix [9], [10], [13], [14], [15]. We observe that the method described in [9] can be extended in order to approximate not only the diagonal but also arbitrary entries of the inverse of the matrix. Consequently, we develop an algorithm which constitutes a selected approximate matrix inversion, the PBAPINV method, which turns out to be highly accurate compared to the K-method and more efficient when the matrix $\widetilde{K}$ is less sparse or large enough so that a small window size is inadequate to provide high degree of accuracy. Moreover, we develop a reluctance simulator, the GKC-simulator, which utilizes the nodal analysis formulation so that the computations are done with conductance, reluctance and the capacitance matrices.

1.4 Thesis Outline and Overview

In Chapter 2, we provide the theoretical background of the inductive properties of electric circuits, the concept of the partial inductance is described and the definition, the physical meaning and the properties of the reluctance matrix K are presented. In Chapter 3, we review the sparse approximate inverse methods. Specifically, the mathematical background of the K-method is extensively described in Section 3.1 while in Section 3.4 the PBAPINV method is developed. In Chapter 4, the fundamentals of circuit simulation are briefly described (see [17] for more details) while in Section 4.4 the details of the GKC-simulator are presented. In Chapter 5, we demonstrate the experimental results when each of the approximate inverse methods of Chapter 2, are used into the simulation tool and a comparison in terms of accuracy and runtime is given. Furthermore, in Appendix 7.1 we offer selected code segments of the simulator and the implementation of PBAPINV in C. In Appendix 7.2, some supplementary mathematical proofs are given. Finally, Chapter 6 presents our conclusion and further research avenues.

2

Background

2.1 Inductive Properties of Electric Circuits

Inductance represents the capability of a circuit to store energy in the form of a magnetic field and is defined on current loops. In this section, a general N-loop system is considered with currents $\underline{I}_1, \underline{I}_2, \dots, \underline{I}_i, \dots, \underline{I}_j, \dots, \underline{I}_n$ which are uniformly distributed on the cross section of each loop. The magnetic field $\underline{B}$, induced by each current, is interrelated with the electric field $\underline{E}$ and current as determined by Maxwell's equations:

$$\nabla \underline{D} = \rho \tag{2.1.1}$$

$$\nabla \underline{B} = 0 \tag{2.1.2}$$

$$\nabla \times \underline{H} = \underline{J} + \frac{\partial \underline{D}}{\partial t} \tag{2.1.3}$$

$$\nabla \times \underline{E} = -\frac{\partial \underline{B}}{\partial t} \tag{2.1.4}$$

$$\underline{D} = \varepsilon \underline{E} \tag{2.1.5}$$

$$\underline{B} = \mu \underline{H} \tag{2.1.6}$$

$$\underline{J} = \sigma \underline{E} \tag{2.1.7}$$

The magnetic field $\underline{B}$ can also be expressed as $\underline{B} = \nabla \times \underline{A}$ where $\underline{A}$ is the magnetic vector potential, which is not unique because $\underline{A}' = \underline{A} + \nabla\varphi$ where φ a scalar potential. The second term in (2.1.3) corresponds to the displacement current, which is considered negligible compared to the actual current flowing in the conductor, hence $\frac{\partial D}{\partial t} \approx 0$. Moreover, using the Coulomb gauge $\nabla \underline{A} = 0$ and substituting into (2.1.3) we obtain the Poisson's equation for the magnetic field potential $\nabla^2 \underline{A} = -\mu \underline{J}$. Considering the boundary condition $\lim_{\underline{r} \to \infty} \underline{A}(\underline{r}) = \underline{0}$, this equation has unique solution:

$$\underline{A}(\underline{r}) = \frac{\mu}{4\pi} \int_{V} \frac{\underline{J}(\underline{r}')}{|\underline{r} - \underline{r}'|} d\underline{r}' \qquad (2.1.8)$$

A set of N^2 inductances is defined for a system of N loops as

$$L_{ij} = \frac{\Phi_{ij}}{I_j} \text{ for } \underline{I}_k=0 \text{ if } k \neq j \qquad (2.1.9)$$

, where Φ_{ij} represents the magnetic flux in loop i due to current $\underline{I}_j$:

$$\Phi_{ij} = \iint_{S_i} \underline{B}_j \cdot \underline{n}\, ds \qquad (2.1.10)$$

, where S_i is a smooth surface bounded by the loop i, $\underline{B}_j$ is the magnetic field created by the current in the loop j, and $\underline{n}$ is a unit vector normal to the surface element ds.Two isolated current loops i and j are shown in Figure 2.1. In the special case where two circuit loops are the same, the coefficient in (2.1.9) is referred to as a loop self- inductance; otherwise, it is referred to as a mutual inductance.

Substituting $\underline{B}_j = \nabla \times \underline{A}_j$and using Stoke's Theorem, the loop flux is expressed as:

$$\Phi_{ij} = \iint_{S_i} (\nabla \times \underline{A}_j) \cdot \underline{n}\, ds = \oint_{l_i} \underline{A}_j\, d\underline{l} \qquad (2.1.11)$$

, where $\underline{A}_j$ is vector potential created by the current $\underline{I}_j$.The magnetic vector potential of loop j using (2.1.8) is

$$\underline{A}_j(\underline{r}) = \frac{\mu}{4\pi} \int_{V_j} \frac{\underline{J}_j(\underline{r}')}{|\underline{r} - \underline{r}'|} d\underline{r}' = I_j \frac{\mu}{4\pi} \oint_{l_j} \frac{d\underline{l}'}{|\underline{r} - \underline{r}'|} \qquad (2.1.12)$$

, where $|\underline{r} - \underline{r}'|$ is the distance between the loop element $d\underline{l}'$ and the point of interest $\underline{r}$. Substituting (2.1.12) into (2.1.11) yields:

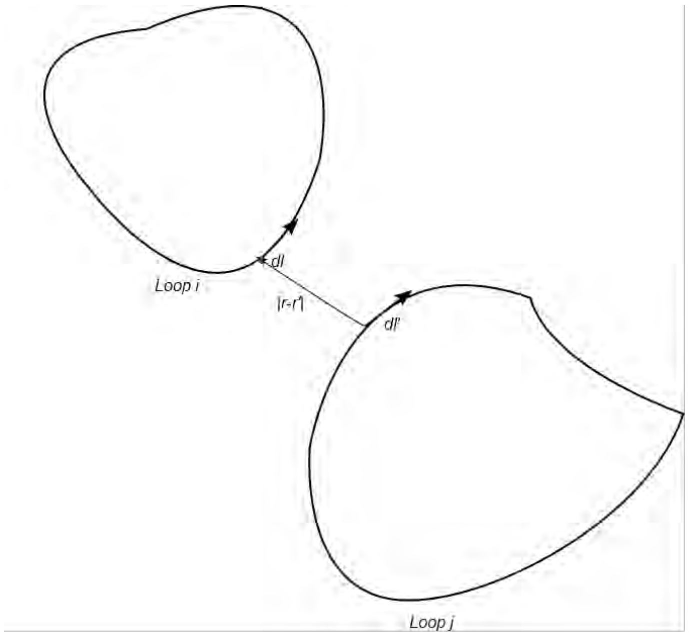

Figure 2.1: Two isolated complete Current Loops *i* and *j*

$$\Phi_{ij} = I_j \frac{\mu}{4\pi} \oint_{l_i} \oint_{l_j} \frac{d\underline{l} \cdot d\underline{l}'}{|\underline{r} - \underline{r}'|} \tag{2.1.13}$$

$$L_{ij} = \frac{\mu}{4\pi} \oint_{l_i} \oint_{l_j} \frac{d\underline{l} \cdot d\underline{l}'}{|\underline{r} - \underline{r}'|} \tag{2.1.14}$$

Note that the integration in (2.1.11), (2.1.12), (2.1.13) and (2.1.14) is performed in the direction of current flow. Furthermore, we note that the finite cross-sectional dimensions of the conductors are neglected in the transition between the general volume integral to a more constrained but simpler contour integral in (2.1.12). Thus, the loop conductor is confined to an infinitely thin filament. The thin filament approximation is acceptable only when the cross-sectional dimensions of the conductors are much smaller compared to the distance $|\underline{r} - \underline{r}'|$ between any loop points. As a consequence, this approach cannot be used to determine the self-inductance. To account for the finite cross-sectional dimensions of the conductors, both (2.1.11) and (2.1.12) are amended to include an explicit integration over the conductor cross-sectional area a,

$$\Phi_{ij} = \frac{1}{I_i} \oint_{l_i} \int_{\alpha_i} \underline{A}_j J_i d\underline{l}\, d\alpha \tag{2.1.15}$$

$$\underline{A}_j(\underline{r}) = \frac{\mu}{4\pi} \oint_{l_j} \int_{a_j} \frac{J_j d\underline{l}' da'}{|\underline{r} - \underline{r}'|} \tag{2.1.16}$$

, where a_i and a_j are the cross sections of the segments, $d\underline{l}$ and $d\underline{l}'$, da and da' are the differential elements of the respective cross sections, J_i and J_j are the current density distributions over the wire cross section a_i and a_j respectively such that $d\underline{J} = Jd\underline{l}da$ and $I_i = \int_{a_i} J_i \, da$. The only constrained imposed by the above formulations is that the current flow has the same direction across the areas a_i and a_j. These formulas can be further simplified assuming a uniform current distribution (i.e., $I = aJ$). Then, the magnetic flux is transformed into

$$\Phi_{ij} = \frac{\mu}{4\pi} \frac{I_j}{a_i a_j} \oint_{l_i} \oint_{l_j} \int_{a_i} \int_{a_j} \frac{d\underline{l} \cdot d\underline{l}'}{|\underline{r} - \underline{r}'|} \, dada' \tag{2.1.17}$$

Finally,

$$L_{ij} = \frac{\mu}{4\pi} \frac{1}{a_i a_j} \oint_{l_i} \oint_{l_j} \int_{a_i} \int_{a_j} \frac{d\underline{l} \cdot d\underline{l}'}{|\underline{r} - \underline{r}'|} dada' \tag{2.1.18}$$

From (2.1.18), for $i \neq j$ we obtain the mutual inductance while for $i = j$ we obtain the self inductance.

2.2 Partial Inductance

For integrated circuits associated with rather complicated on-chip structures where not deliberately designed inductors may appear (parasitic inductance), it is difficult to correctly estimate the current loop, therefore the concept of partial inductance is developed, which is defined on wire segments rather than current loops.

The loop inductance, as defined in (2.1.18), can be deconstructed into more basic elements if the two loops are broken into segments, as shown in Figure 2.2. The loop i is broken into N_i segments $S_1, S_2, \ldots, S_{N_i}$ and loop j into N_j segments $S_1', S_2', \ldots, S_{N_j}'$. The definition of the loop inductance can be rewritten as:

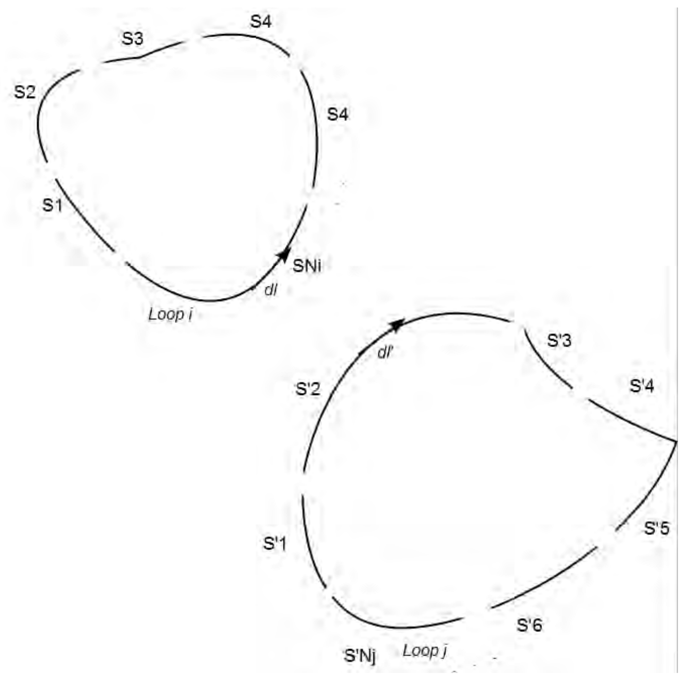

Figure 2.2: Two complete Current Loops broken into Segments

$$L_{ij} = \frac{\mu}{4\pi}\frac{1}{\alpha_i a_j}\sum_{i=1}^{N_i}\sum_{j=1}^{N_j}\int_{S_i}\int_{S_j'}\int_{a_i}\int_{a_j}\frac{d\underline{l}\cdot d\underline{l}'}{\left|\underline{r}-\underline{r}'\right|}dada' \tag{2.1.19}$$

The integration along segments S_i and S_j' in (2.1.19) is performed in the direction of current flow.

Partial inductance is defined as the argument of the double summation in (2.1.19) for the conductor segments:

$$\mathcal{L}_{ij} = \frac{\mu}{4\pi}\frac{1}{\alpha_i a_j}\int_{S_i}\int_{S_j'}\int_{a_i}\int_{a_j}\frac{\left|d\underline{l}\cdot d\underline{l}'\right|}{\left|\underline{r}-\underline{r}'\right|}dada' \tag{2.1.20}$$

The partial inductances are denoted by $\mathcal{L}_{ij}$ in order to distinguish them from the loop inductances L_{ij}. Then (2.1.19) is written as

$$L_{ij} = \frac{\mu}{4\pi}\frac{1}{\alpha_i a_j}\sum_{i=1}^{N_i}\sum_{j=1}^{N_j}S_{ij}\mathcal{L}_{ij} \tag{2.1.21}$$

, where $S_{ij} = \pm 1$ is the sign of the scalar product $d\underline{l}\cdot d\underline{l}'$, which depends on the direction of current flow in the conductor segments.

It is vital for the understanding of the concept of partial inductances their relation to the magnetic flux to be established. Specifically, partial inductance $\mathcal{L}_{ji}$ is associated with the magnetic flux created by the current of the segment S_i through the virtual loop which the segment S_j forms through infinity, as it is shown in Figure 2.3.

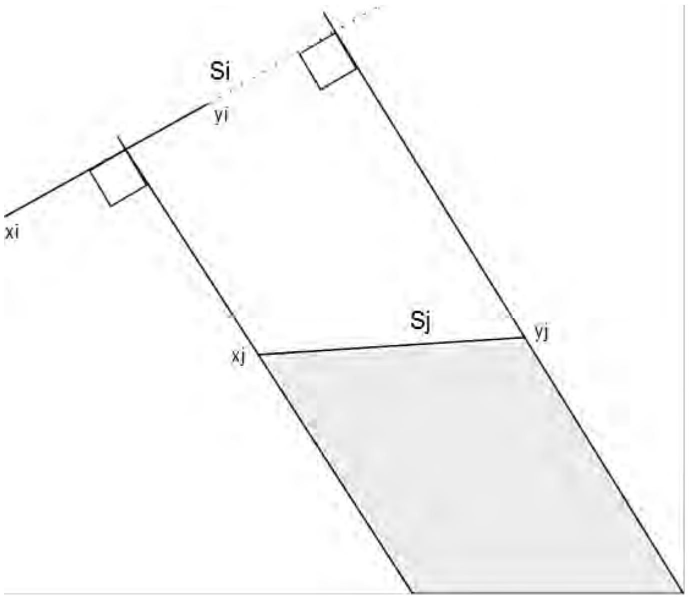

Figure 2.3: Loop Definition of Partial Inductance

In other words,

$$\mathcal{L}_{ji} = \frac{1}{I_i} \int_{a_{ij}} \underline{B_i} \cdot d\underline{s} \tag{2.1.22}$$

, where a_{ij} is the area bounded at the ends by the conductor segment S_j and infinity, and on the sides by two straight lines which go through the end points x_j and y_j of segment S_j and are normal to the line connecting the end points x_i and y_i of segment S_i. For simplicity, we assume infinitely thin segments. Hence, (2.1.20) becomes

$$\mathcal{L}_{ji} = \frac{\mu}{4\pi} \int_{S_i} \int_{S_j} \frac{|d\underline{l} \cdot d\underline{l}'|}{|\underline{r} - \underline{r}'|} \tag{2.1.23}$$

and

$$\underline{A}_i(\underline{r}) = I_i \frac{\mu}{4\pi} \int_{S_i} \frac{d\underline{l}'}{|\underline{r} - \underline{r}'|} \tag{2.1.24}$$

Using Stoke's Theorem,

$$\int_{a_{ij}} \underline{B_i} \cdot d\underline{s} = I_i \frac{\mu}{4\pi} \oint_{l_{ij}} \int_{S_i} \frac{d\underline{l} \cdot d\underline{l}'}{|\underline{r} - \underline{r}'|} \tag{2.1.25}$$

The path l_{ij} can be restricted to the portion from x_j to y_j, i.e., S_j, because magnetic vector potential $\underline{A_i}$ is zero at infinity and normal to $d\underline{l}$ on the two perpendicular paths, since it is in the direction of I_i.

By defining each segment as forming its own return loop with infinity, partial inductances can be used without a priori knowledge of the actual current loops. Moreover, the partial inductance model contains all the magnetic interactions which were contained in the loop inductance values, as it can be derived from (2.1.21).

2.3 Circuit Element K

2.3.1 Definition of the Reluctance matrix K

As we have already seen in the previous section, each element (i, j) in the partial inductance matrix is given by

$$\mathcal{L}_{ij} = \frac{\mu}{4\pi} \frac{1}{\alpha_i a_j} \int_{l_i} \int_{l_j} \int_{a_i} \int_{a_j} \frac{d\underline{l} \cdot d\underline{l}'}{r_{ij}} dada' \tag{2.3.1}$$

, where a_i, a_j are cross-sections of segments i and j, and r_{ij} is the distance between two points in segments i and j (we consider that the sign is incorporated into the definition of the partial inductance). For a $n \times n$ partial inductance matrix the following linear system equation can be derived:

$$\begin{bmatrix} \mathcal{L}_{11} & \mathcal{L}_{12} & \dots & \mathcal{L}_{1n} \\ \mathcal{L}_{21} & \mathcal{L}_{22} & \dots & \mathcal{L}_{2n} \\ \vdots & \vdots & & \vdots \\ \mathcal{L}_{n1} & \mathcal{L}_{n2} & \dots & \mathcal{L}_{nn} \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ \vdots \\ I_n \end{bmatrix} = \begin{bmatrix} \Phi_1 \\ \Phi_2 \\ \vdots \\ \Phi_n \end{bmatrix} \tag{2.3.2}$$

, where I_i is the current running along conductor segment i, and Φ_i is the total flux flowing through the virtual loop from segment i to infinity. The inverse of the system can be written as:

$$\begin{bmatrix} K_{11} & K_{12} & \dots & K_{1n} \\ K_{21} & K_{22} & \dots & K_{2n} \\ \vdots & \vdots & & \vdots \\ K_{n1} & K_{n2} & \dots & K_{nn} \end{bmatrix} \begin{bmatrix} \Phi_1 \\ \Phi_2 \\ \vdots \\ \Phi_n \end{bmatrix} = \begin{bmatrix} I_1 \\ I_2 \\ \vdots \\ I_n \end{bmatrix} \tag{2.3.3}$$

, where $K = \mathcal{L}^{-1}$, the inverse of the inductance matrix, which is called susceptance or reluctance matrix. From (2.3.3) the physical meaning of K_{ij} can be derived: K_{ij} is the induced current along conductor i when the total magnetic flux for the conductor j is equal to one and those for all other conductors are set to zero. For example, as shown in Figure 2.4, to obtain the fourth column of K, we apply unit flux to conductor 4 and set zero to all others. The induced currents other than the fourth are the off-diagonal terms in the fourth column of K. Moreover, K_{ij} is positive if the current direction along the conductor i and the flux of the conductor j follow the right-hand rule; it is zero otherwise.

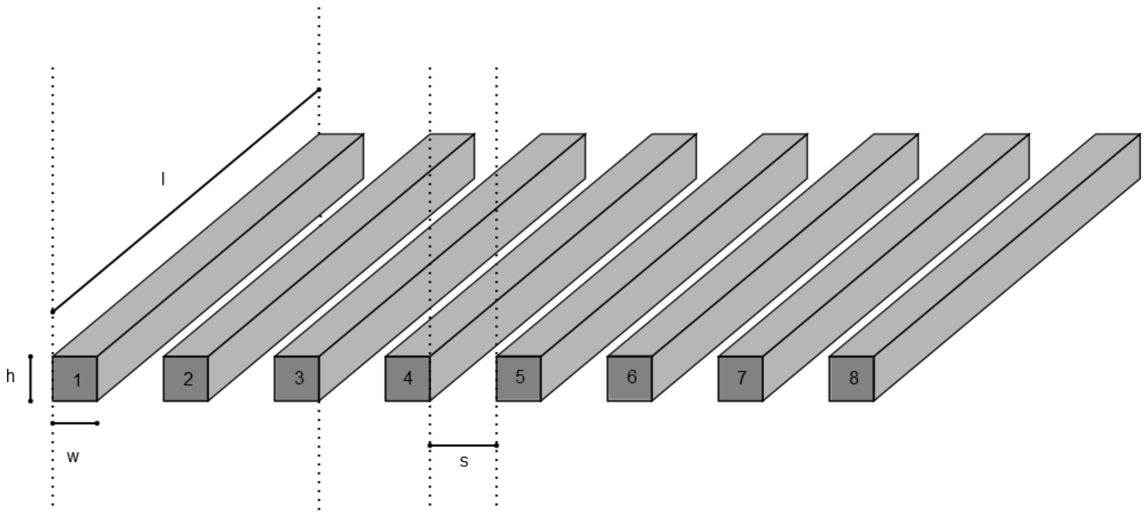

Figure 2.4: Layout Example with 8 Parallel Conductors

2.3.2 Properties of the Reluctance matrix K

First of all, the reluctance matrix K is SPD as the inverse of an SPD matrix. Furthermore, the off-diagonal elements of K exhibit a much faster decrease compared to the decrease of the off-diagonal values into the $\mathcal{L}$ matrix.

The following is a small example of an 8 conductor bus to demonstrate the idea of how off-diagonal elements in K decrease much faster than that of the partial inductance matrix. The parameters for this example are as follows: width and height for each conductor is 2um, length is 40um, spacing is 5um, sigma is 3.77e7. We calculate the partial inductance matrix using FastHenry[24]. The extracted inductance matrix is as follows:

$$\mathcal{L} = \begin{bmatrix} 11.4 & 4.26 & 2.54 & 1.79 & 1.38 & 1.11 & 0.94 & 0.81 \\ 4.26 & 11.4 & 4.26 & 2.54 & 1.79 & 1.38 & 1.11 & 0.94 \\ 2.54 & 4.26 & 11.4 & 4.26 & 2.54 & 1.79 & 1.38 & 1.11 \\ 1.79 & 2.54 & 4.26 & 11.4 & 4.26 & 2.54 & 1.79 & 1.38 \\ 1.38 & 1.79 & 2.54 & 4.26 & 11.4 & 4.26 & 2.54 & 1.79 \\ 1.11 & 1.38 & 1.79 & 2.54 & 4.26 & 11.4 & 4.26 & 2.54 \\ 0.94 & 1.11 & 1.38 & 1.79 & 2.54 & 4.26 & 11.4 & 4.26 \\ 0.81 & 0.94 & 1.11 & 1.38 & 1.79 & 2.54 & 4.26 & 11.4 \end{bmatrix} pH$$

and the corresponding reluctance matrix K is

$$K = \begin{bmatrix} 103.39 & -33.91 & -7.60 & -3.96 & -2.6 & -1.95 & -1.63 & -1.84 \\ -33.91 & 114.48 & -31.45 & -6.33 & -3.16 & -2.03 & -1.55 & -1.63 \\ -7.60 & -31.45 & 115.01 & -31.19 & -6.18 & -3.07 & -2.03 & -1.94 \\ -3.96 & -6.34 & -31.19 & 115.13 & -31.14 & -6.18 & -3.16 & -2.60 \\ -2.60 & -3.16 & -6.18 & -31.14 & 115.13 & -31.19 & -6.33 & -3.96 \\ -1.95 & -2.03 & -3.08 & -6.18 & -31.19 & 115.01 & -31.45 & -7.60 \\ -1.63 & -1.55 & -2.03 & -3.16 & -6.33 & -31.45 & 114.48 & -33.91 \\ -1.85 & -1.63 & -1.95 & -2.60 & -3.96 & -7.60 & -33.91 & 103.39 \end{bmatrix} \times 10^9\ H^{-1}$$

As we could see, the off-diagonal term $\mathcal{L}_{17}$ in $\mathcal{L}$ matrix is 0.94/11.4 or 8.2% of the partial self-inductance $\mathcal{L}_{11}$, while $|K_{17}|$ in K matrix is only 1.63/103.39 or 1.6% of self-reluctance term K_{11}. The reason of this fast decay of off-diagonal values stems from the physical meaning of the reluctance matrix.

Suppose that the magnetic flux of the j^{th} conductor is set to one while the magnetic flux along all the other conductors is set to zero. As already mentioned, the induced current along the i^{th} conductor yields the value K_{ij}. Therefore, the activated conductor j must carry positive current while, in order for the magnetic flux of all the other conductors to remain at zero, they must carry current at the opposite direction. Specifically, as it is shown in Figure 2.5, the neighbor conductors will carry an opposite current that cancels the magnetic field induced by the current along the j^{th} conductor. Furthermore, the magnetic field generated by each neighbor cancels part of the magnetic field induced on the aggressor line j. In other words, the current along the $(j+1)^{th}$ neighbor segment will also induce current in the $(j+2)^{th}$ conductor. As a consequence, the induced current by the $(j+1)^{th}$ conductor shields the induced current by the j^{th} conductor in $(j+2)^{th}$ segment to go further. As depicted in Figure 2.5, the $(j+2)^{th}$ conductor results in a shorter arrow, accounting for the overall effect (not a signal active line). That is the physical explanation of the locality property and shielding effect of K matrix. A more detailed explanation of this fundamental property of the reluctance matrix can be found in [30].

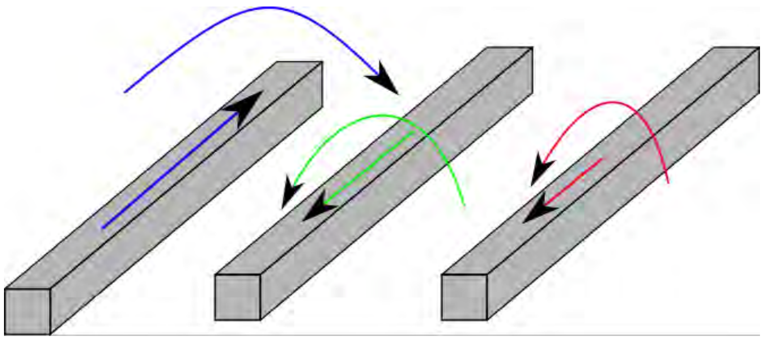

Figure 2.5: An Example to explain the Locality Property of $\boldsymbol{K}$ matrix

From the previous discussion, one may derive that $K_{ii} > 0$ and $K_{ij} < 0, i \neq j$. However, this is not the general case where each conductor may be divided into segments or filaments and the conductors may have irregular geometry or unequal dimensions. In such cases, in the previous example the coupling effect between conductors $(j+1)$ and (j+2) may be much stronger than the coupling effect between j and $(j+2)$. Consequently, the overall effect causes conductor $j+2$ to carry positive current and $K_{j+2,j} > 0$.

Based on the locality property of K matrix, the faraway mutual reluctances can be truncated without sacrificing accuracy, compared to truncation techniques applied to $\mathcal{L}$ matrix, yielding a sparse version of K matrix. The use of a sparse version of K matrix can greatly speedup the transient simulation of a circuit as we will see in the next sections.

Furthermore, positive definiteness of the resulting sparse K matrix should be guaranteed for stable circuit simulation. In [22], a proof for the diagonal dominance property of K matrix, based on the assumption that $K_{ij} < 0 \; \forall i, j, i \neq j$, is given. A reluctance matrix K which is diagonally dominant with positive diagonal elements is also SPD (7.2.3). In that case, the arbitrary truncation of any off-diagonal entry results in an SPD sparse matrix and stability is ensured. However, as we have already mentioned, positive off-diagonal values may occur. In [23], it has been proven that under sufficiently fine discretization of the conductors stability can be ensured. However, the sparsification of the K-matrix is constrained at the extraction level while the finer discretization may lead to a K matrix of greater size. In [25], SPD-remedy techniques under insufficient discretization by enforcing the positive definiteness, either directly or indirectly by enforcing the diagonal dominance, of the K matrix are proposed. The combination of an accurate and effective sparse approximate inversion method and an accurate SPD- remedy technique may lead to the adoption of the reluctance element K by the commercial circuit simulation tools.

3

Sparse Approximate Inversion Methods

3.1 Approximate Inversion via small matrix inversions

First, we provide two theorems as the basis of this approximate inversion method.

Theorem 3.1. *Suppose A is a $n \times n$ non-singular matrix, and B is the inverse of A. A_m is a minor of A with order m formed by rows $i_1, i_2, \dots, i_m$ and columns $j_1, j_2, \dots, j_m$. B_{n-m} is the matrix remained in matrix B after deleting columns $i_1, i_2, \dots, i_m$ and rows $j_1, j_2, \dots, j_m$. Then,*

$$|B_{n-m}| = \frac{(-1)^{\sum(i_k+j_k)}}{|A|} |A_m| \tag{3.1.1}$$

Proof:

We first consider the special case that $i_k = j_k$ $(1 \le k \le m)$. Thus, A_m is the top left corner of A and B_{n-m} is at the right bottom corner of B. In block format:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

, where A_{11} and B_{11} are $m \times m$ matrices; A_{22} and B_{22} are $(n-m) \times (n-m)$ matrices. Suppose A_{11} is invertible, hence [7.2.1]

$$|A| = |A_{11}||A_{22} - A_{21}A_{11}^{-1}A_{12}| \tag{3.1.2}$$

Moreover, according to (7.2.2)

$$|B_{22}| = \frac{1}{|A_{22} - A_{21}A_{11}^{-1}A_{12}|} \tag{3.1.3}$$

Therefore,

$$|B_{22}| = \frac{|A_{11}|}{|A|} \tag{3.1.4}$$

If $|A_{11}| = 0$, then $|B_{22}|$ should also be zero. Otherwise, if $|B_{22}| \neq 0$, since $A = B^{-1}$, following similar step, we can also argue that $|A_{11}| = \frac{|B_{22}|}{|B|} \neq 0$, a contradiction. So $|B_{22}| = \frac{|A_{11}|}{|A|}$ for all cases.

Now we extend the conclusion to general case. By making $(i1 - 1) + (i_2 - 2) + \cdots + (i_m - m) = (i_1 + i_2 + \cdots + i_m) - \frac{m(m+1)}{2}$ successive interchanges of adjacent rows using the permutation matrix P_1 and, correspondingly $(j_1 - 1) + (j_2 - 2) + \cdots + (j_m - m) = (j_1 + j2 + \cdots + j_m) - \frac{m(m+1)}{2}$ column interchanges using the permutation matrix P_2, the resulting matrix is $\widetilde{A} = P_1 A P_2^T$ and $\widetilde{B}^{-1} = P_2 A^{-1} P_1^T$. The relative positions of elements in A_m and B_{n-m} are preserved, so we have $\tilde{A}_{11} = A_m$ and $\widetilde{B}_{22} = B_{n-m}$. Furthermore, $|\tilde{A}| = (-1)^{i_1+i_2+\cdots+i_m+j_1+j_2+\cdots+j_m-m(m+1)}|A|$. Hence,

$$|B_{n-m}| = |\tilde{B}_{22}| = \frac{|\tilde{A}_{11}|}{|\tilde{A}|} = \frac{(-1)^{\Sigma(i_k+j_k)}}{|A|}\,|A_m|$$

Theorem 3.2 *Suppose A is a $n \times n$ nonsingular matrix, and B is the inverse of A. For the r^{th} row of A matrix, only the entries at columns $i_1, i_2, \ldots, i_m$ are non-zero, the rest are zeroes. Let B_m be the* sub-matrix *of B formed by rows $i_1, i_2, \ldots, i_m$ and columns $j_1, j_2, \ldots, j_m$. If there exists p , $1 \leq p \leq m$ such that $j_p = r$, then the p^{th} row of the inverse of B_m is equal to the r^{th} row of A (omitting the zero entries in the latter):*

$$B_m^{-1}(p,:) = A(r,:)$$

Similar conclusion holds for the columns of A.

Proof:

Using Cramer's law we have:

$$B_m^{-1}(p,q) = \frac{(-1)^{p+q}}{|B_m|}\,|B_{m(q,p)}| \;,1 \leq q \leq m \tag{3.1.5}$$

, where $B_{m(q,p)}$ is the matrix that is obtained by deleting row q and column p from the sub-matrix B_m.

Using theorem 3.1 we have:

$$|B_{m(q,p)}| = \frac{(-1)^{\Sigma_{k\neq q} i_k + \Sigma_{k\neq p} j_k}}{|A|}\,|A_{n-m+1}| \tag{3.1.6}$$

, where A_{n-m+1} refers to the sub-matrix of A formed by deleting columns $i_1, i_2, \ldots, i_{q-1}, \ldots, i_{q+1}, \ldots, i_m$ and rows $j_1, j_2, \ldots, j_{p-1}, \ldots, j_{p+1}, \ldots, jm$.

Using theorem 3.1 we also have:

$$|B_m| = \frac{(-1)^{\Sigma(i_k+j_k)}}{|A|}\,|A_{n-m}| \tag{3.1.7}$$

, where A_{n-m} refers to the sub-matrix of A formed by deleting columns $i_1, \ldots, i_m$ and rows $j_1, \ldots, j_m$.

The $\left(r-(p-1)\right)^{th}$ row of A_{n-m-1} comes from the r^{th} row of A and there is only one nonzero element in it. It is the $\left(r-(p-1), i_q-(q-1)\right)$ entry (or the (r, i_q) entry of A). Using Laplace Expansion on this row we get:

$$|A_{n-m+1}| = (-1)^{r-(p-1)+i_q-(q-1)} A(r, i_q)|A_{n-m}| \qquad (3.1.8)$$

Combining (3.1.5), (3.1.6), (3.1.7), (3.1.8) and the fact that $j_p = r$, we obtain $A_m^{-1}(p,q) = A(r, i_q)$.

Theorem 3.2 provides the basis for a matrix inversion algorithm of an $n \times n$ invertible matrix L when its inverse $K = L^{-1}$ exhibits a priori known sparsity pattern. Specifically, we consider $A = K$, $B = L$ and we apply the theorem 3.2 for every row r, $0 \le r \le n-1$. The application of the theorem to a row r of L is depicted in Figure 3.1. The predetermined positions of nonzero entries in row r are denoted by $i_1, i_2, \dots, i_m$. These nonzero entries can be computed exactly from the p^{th} row of the inverse of a sub-matrix of A, formed by the intersection of the rows $i_1, i_2, \dots, i_m$ and the arbitrarily selected columns $j_1, j_2, \dots, j_p, \dots, j_m$ where p, $1 \le p \le m$ such that $j_p = r$. Consequently, only a subset of the entries of the L matrix, rather than the entire L matrix, can be used to compute its whole exact inverse. When the diagonal of the inverse of the matrix is nonzero then $i_p = r$ for some p, $1 \le p \le m$ hence we can consider $j_k = i_k\ \forall k, 1 \le k \le m$.

If m (the number of nonzero entries in r^{th} row K) is a small constant, compared to the size of the matrix n, for *every* row r, $1 \le r \le n$ of K the above mentioned algorithm becomes very efficient and has complexity $O(n)$. However, it remains efficient only for high sparsity ratios of K.

If K is not exactly sparse, but it exhibits a decay property, i.e., many of the entries of K are small, it can be considered approximately sparse. The positions $i_1, i_2, .., i_m$ in row r are expected to contain the m largest entries of the r^{th} row of K while the other entries are considered approximately zero. Hence, the p^{th} row of the inverse of the sub-matrix which corresponds to the r^{th} row, does not contain the exact values of the inverse's r^{th} row but an approximation of it.

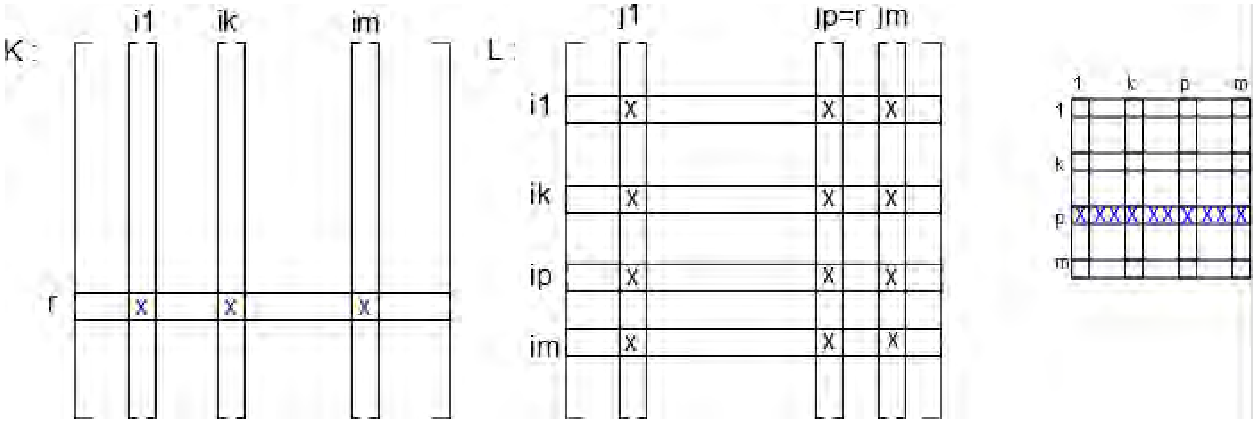

Figure 3.1: Inversion of a Matrix via small Matrix Inversions

3.2 Approximate Inversion via Frobenius Norm Minimization

This method constitutes a commonly used explicit preconditioning technique ([1], [2]) which tries to compute a preconditioner M such that $M \approx A^{-1} \Leftrightarrow AM \approx I$ (or $M \approx A^{-1} \Leftrightarrow MA \approx I$) in some sense and the number of nonzero entries in M are much less than those in A. Although a sparse inverse does not always exist, it often occurs that most entries in A^{-1} are relatively small, hence they can be cut off.

Specifically, the approximate inversion can be reduced to an optimization problem, as follows:

$$Find\ the\ matrix\ M\ which\ minimizes\ the\ \parallel AM - I \parallel\ where\ \parallel . \parallel\ is\ a\ matrix\ norm.$$

Of course, if no restriction is placed on M, the exact inverse will be found. Thus a sparsity pattern for M is prescribed.

If Frobenius norm is used as a matrix norm, fast convergence is ensured and inherent parallelism emerges. Remember that the Frobenius norm of a $n \times n$ matrix is defined as: $\parallel A \parallel_F = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} |a_{ij}|^2$. Consequently,

$$\parallel AM - I \parallel_F^2 = \sum_{k=0}^{n-1} \parallel (AM - I)e_k \parallel_2^2 \quad where\ e_k = (0, \dots, 0, 1, 0, \dots, 0)^T \tag{3.2.1}$$

The solution of (3.2.1) decouples into n independent least square problems:

$$\min_{m_k} \parallel Am_k - e_k \parallel_2, \ \ k = 0, 1, \dots, n-1 \tag{3.2.2}$$

Thus, we can solve (3.2.2) in parallel with each least square problem computing a column of M.

Let S_k be a vector indexing the nonzero entries of m_k. Then

$$\parallel Am_k - e_k \parallel_2 = \parallel A(:, S_k)m_k(S_k) - e_k \parallel_2 \tag{3.2.3}$$

Depending on the sparsity of A, some of the rows of $A(:, S_k)$ may be zero. Let T_k be a vector indexing the nonzero rows of $A(:, S_k)$. Then

$$\parallel Am_k - e_k \parallel_2 = \parallel A(T_k, S_k)m_k(S_k) - e_k \parallel_2 \tag{3.2.4}$$

Consequently, the least square problem in (3.2.2) can be reduced to the one in (3.2.4) with matrix $A(T_k, S_k)$ which has size $|T_k| \times |S_k|$.

The difficulty lies in determining a sparsity pattern G of the underlying approximate inverse which allows a good approximation of A^{-1}. Sparsity can be achieved either statically, on the basis of the positions, or dynamically, on the basis of values, of the fill-ins. In [2], SPAI algorithm is introduced, which offers a method to dynamically capture the sparsity structure of the approximate inverse. However, such adaptive methods tend to be expensive. Research has also been done on the selection of a sparsity pattern, based on the sparsity pattern S of A, in a preprocessing step, so that a sparse approximate inverse can be computed directly by minimizing (3.2.1) ([3], [4]).

If matrix A has too many entries, to be used as a basis of the sparsity pattern of M, sparsification should be applied to A as well e.g using a thresholding technique ([5], [6]). Below, we represent some of the proposed heuristics, where the sparsified version of A is denoted by $\tilde{A}$ and the sparsity structure of A by S.

1. $\tilde{a}_{ij} = a_{ij}$ if $|a_{ij}| > \tau \times \max\{\alpha_{ij}\}$, where $\tau \in (0,1)$ is a threshold parameter.
2. For a fixed positive integer k, with $k \ll n$, find the k largest entries in each column of A. Then S is the set of positions (i,j) of the resulting $k \times n$ entries.
3. For each column, find the row indices of the k largest entries. Then for each row index i found, the same search is performed on column i. The new row indices found, are added to the previous ones to determine the nonzero pattern of the column. This heuristic is called the neighbours of neighbours and can be easily extended by performing several iterations instead of one.
4. $\tilde{a}_{ij} = a_{ij}$ if $|a_{ij}| > \tau \times \max\{a_{ij}\}$ and $i \in V_j$, where $\tau \in (0,1)$ is a threshold parameter and V_j indexes the k largest elements in j^{th} column of A. Thus in each column we retain the k largest entries on the condition that they are above the threshold.

The 1 and 2 heuristics are the simplest but the first one does not strictly limit the number of fill-ins of $\tilde{A}$, hence it can lead to expensive approximate inversion. Heuristic 4 is a combination of 1 and 2. The threshold τ is useful from a computational point of view even when a column contains more than k elements larger than the threshold, because it reduces the number of elements to be sorted in order to find the k largest. A disadvantage common to all above mentioned heuristics is the need to specify good values of the parameters involved.

When S is obtained, we can simply consider $G = S$, where G is the sparsity structure of A^{-1} or we can apply S to approximate sparsity pattern techniques so as to acquire G.

3.3 Approximate Inversion via AINV algorithm

The AINV algorithm [7] constitutes an approximate inverse preconditioning technique, as well. The difference is that AINV results in a factorized form of the approximate inverse rather than a single matrix. Moreover, the AINV method does not require that the sparsity pattern be known in advance. Its basic idea is the construction of a preconditioner by means of *incomplete conjugate Gram-Schmidt (or A −orthogonalization) process*.

Definition 3.1. *Two nonzero vectors u, v are A-orthogonal or conjugate with respect to A, where A is a symmetric positive definite matrix, if $\langle u, v\rangle_A = 0$, where $\langle u, v\rangle_A = \langle Au, v\rangle = \langle u, A^T v\rangle = \langle u, Av\rangle = u^T Av$.*

Theorem 3.3. *If A is a $n \times n$ SPD matrix, $\{z_1, z_2, \dots, z_n\}$ a set of n conjugate directions and $Z = [z_1, z_2, \dots, z_n]$, i.e., the i^{th} column of Z is z_i, then*

$$Z^T AZ = D = \begin{bmatrix} p_1 & 0 & \dots & 0 \\ 0 & p_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & p_n \end{bmatrix} \textit{ where } p_i = z_i^T A z_i \qquad (3.3.1)$$

It follows that

$$A^{-1} = ZD^{-1}Z^T \qquad (3.3.2)$$

, hence a factorization of A^{-1} is obtained. As a consequence, the approximate inversion reduces to the construction of a set of n conjugate vectors which can be structured by means of conjugate Gram-Schmidt process applied to any set of linearly independent vectors $u_1, u_2, \dots, u_n$. The resulting matrix Z is unit upper triangular in order to satisfy the root-free Cholesky decomposition of A: $A = LDL^T$ and $Z = L^{-T}$.

Gramm-Schmidt Conjugation. *Given a set of n linearly independent vectors $u_0, u_1, \dots, u_{n-1}$ and a $n \times n$ SPD matrix A, a set of n A-orthogonal vectors $z_1, z_2, \dots, z_n$ is constructed by subtracting from u_i any components that are not A-orthogonal to $z_1, \dots, z_{i-1}$. Hence,*

$$z_i = u_i - \sum_{k=1}^{i-1} c_{ik} z_k \qquad for\ i = 1,2,\dots,n \tag{3.3.3}$$

To find the constants c_{ik} for $i = 1,2,\dots,n$ and $k = 0,1,\dots,i-1$ in (3.3.3), we have:

$$\begin{gathered} z_i^T A z_k = u_i^T A z_k - \sum_{j=1}^{i-1} c_{ik} z_k^T A z_j = u_i^T A z_k - c_{ik} z_k^T A z_k = 0 \\ \Rightarrow c_{ik} = \frac{u_i^T A z_k}{z_k^T A z_k} \end{gathered} \tag{3.3.4}$$

Denoting the i^{th} row of A by a_i^T and considering $u_i = e_i$ for computational convenience, the inverse factorization algorithm can be written as follows:

```
1.  Input: $A \in R^{n,n}$ an SPD matrix.
2.  Output: $Z = [z_1, z_2, \dots, z_n]$ and $D = diag(p_1, p_2, \dots, p_n)$ such that
    $A^{-1} = ZD^{-1}Z^T$
3.  $z_i^0 = e_i \quad i = 1(1)n$
4.  for $i = 1,2,\dots,n$
5.      for $j = i, i+1, \dots, n$
6.          $p_j^{i-1} = a_i^T z_j^{i-1}$
7.      end
8.      if $i = n$ go to 13.
9.      for $j = i+1, \dots, n$
10.         $z_j^i = z_j^{i-1} - \frac{p^{i-1}}{p_i^{i-1}} z_i^{i-1}$
11.     end
12. end
13. $z_i = z_i^{i-1}, p_i = p_i^{i-1},\ i = 1(1)n$
14. return $z_i, p_i\ i = 1(1)n$
```

Algorithm 1: the Inverse Factorization Algorithm

The sparsity in Z factor can be preserved following three different approaches:

1. Using a drop tolerance, in order to determine whether an entry of Z should be preserved or removed at step 10 of the algorithm. This fact means that the sparisity structure of Z dynamically changes at each step of the algorithm.
2. Dropping all newly added fill-in elements outside of a preset sparsity pattern.
3. Skipping some z-vector updates at step 10 of the algorithm when the coefficient p_j^{i-1}/p_i^{i-1} is in some sense “small”.

Strategy 1 is suggested since it offers better numerical results.

If the incomplete inverse fractorization process is successfully completed, one obtains a unit upper triangular matrix $\tilde{Z}$ and a diagonal matrix $\widetilde{D}$ with positive diagonal entries such that, according to (3.3.2), $\tilde{Z}\widetilde{D}\tilde{Z}^T \approx A^{-1}$ which is an SPD matrix as well. It is shown in [7] that an incomplete inverse factorization of A^{-1} exists, in exact arithmetic, for arbitrary values of the drop tolerance and for any choice of the sparsity pattern of $\tilde{Z}$ when A is an H-matrix.

Definition 3.2. *A matrix $A = [a_{ij}]$ is an H-matrix if $\hat{A}$ matrix, where*

$$\hat{a}_{ij} = \begin{cases} -|a_{ij}| & when\ i \neq j \\ \ \ |a_{ij}| & when\ i = j \end{cases}$$

, has eigenvalues with positive real parts.

An SPD diagonally dominant matrix is an H-matrix. However, for general (non-H) matrices a zero or negative pivot p_i may occur at step 6 of the algorithm, which leads to AINV’s breakdown.

Algorithmic modifications can be implemented in order to avoid breakdown for general SPD matrices so as to restore the SPD property of the approximate inverse. Specifically, when some pivot p_i is too small or negative, it is replaced by:

$$p_i = \max\{\sqrt{\varepsilon_M}, \mu\sigma\theta\}$$

, where

ε_M, machine precision

$\mu = 0.1$, a relaxation parameter

$\sigma = \max_{\{i \le k \le n-1\}}\{\, p_k^{i-1}\}$

$\theta = \| z_i^{i-1} \|_\infty \neq 0$

However, there is no guarantee that a good approximate inverse will be produced.

3.4 Selected Approximate Inversion via Probing Technique

In this section, an alternative method to capture selected entries of the inverse of a matrix is presented, which constitutes an extension of [9] where an estimator for the diagonal of the matrix inverse is described.

3.4.1 Basic Idea

Let $A \in C^{n,n}$ and $V_s \in R^{n,s}$. Then:

$$M = (AV_sV_s^T) \times D^{-1}(V_sV_s^T) \Rightarrow$$

$$\begin{bmatrix} m_{11} & \cdots & m_{1j} & \cdots & m_{1n} \\ \vdots & & \vdots & & \vdots \\ m_{i1} & \cdots & m_{ij} & \cdots & m_{in} \\ \vdots & & \vdots & & \vdots \\ m_{n1} & \cdots & m_{nj} & \cdots & m_{nn} \end{bmatrix} =$$
$$\begin{bmatrix} a_{11} & \cdots & a_{1j} & \cdots & a_{1n} \\ \vdots & & \vdots & & \vdots \\ a_{i1} & \cdots & a_{ij} & \cdots & a_{in} \\ \vdots & & \vdots & & \vdots \\ a_{n1} & \cdots & a_{nj} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1s} \\ u_{21} & u_{22} & \cdots & u_{2s} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ u_{n1} & u_{n2} & \cdots & u_{ns} \end{bmatrix} \times \begin{bmatrix} u_{11} & u_{21} & \cdots & \cdots & u_{n1} \\ u_{12} & u_{22} & \cdots & \cdots & u_{n2} \\ \vdots & \vdots & & & \vdots \\ u_{1s} & u_{2s} & \cdots & \cdots & u_{ns} \end{bmatrix} \times$$
$$D^{-1}(V_sV_s^T) \Rightarrow$$

$$\begin{bmatrix} m_{11} & \cdots & m_{1j} & \cdots & m_{1n} \\ \vdots & & \vdots & & \vdots \\ m_{i1} & \cdots & m_{ij} & \cdots & m_{in} \\ \vdots & & \vdots & & \vdots \\ m_{n1} & \cdots & m_{nj} & \cdots & m_{nn} \end{bmatrix} =$$
$$\begin{bmatrix} a_{11} & \cdots & a_{1j} & \cdots & a_{1n} \\ \vdots & & \vdots & & \vdots \\ a_{i1} & \cdots & a_{ij} & \cdots & a_{in} \\ \vdots & & \vdots & & \vdots \\ a_{n1} & \cdots & a_{nj} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} (r_1,r_1) & \cdots & (r_1,r_j) & \cdots & (r_1,r_n) \\ \vdots & & \vdots & & \vdots \\ (r_j,r_1) & \cdots & (r_j,r_j) & \cdots & (r_j,r_n) \\ \vdots & & \vdots & & \vdots \\ (r_n,r_1) & \cdots & (r_n,r_j) & \cdots & (r_n,r_n) \end{bmatrix} \times D^{-1}(VsVs^T) \tag{3.4.1}$$

, where $r_i = u_i^T$ is the i^{th} row of V_s , $D(V_sV_s^T) = \begin{bmatrix} (r_1,r_1) & & \\ & \ddots & \\ & & (r_n,r_n) \end{bmatrix}$ and $D^{-1}(V_sV_s^T)$ the inverse of $D(V_sV_s^T)$, assuming that $(r_i,r_i) \neq 0$ for $i = 1(1)n$.

Hence, from (3.4.1), we have:

$$m_{ij} = a_{ij} + \frac{\sum_{\substack{l=1 \\ l\neq j}}^{n} a_{il}\,(r_l,r_j)}{(r_j,r_j)} \quad ,i,j = 1(1)n \tag{3.4.2}$$

If we set $M = A$, the equation (3.4.2) holds, of course, only if $s = n$ and the matrix $V_s = [v_1, v_2, \dots, v_n]$ has n orthogonal columns $v_1, v_2, \dots, v_n$.

Given a set of matrix index pairs $L_A, L_A = \{(i,j)\} \subseteq \{1,2,\dots,n\} \times \{1,2,\dots,n\}$ where $A \in C^{n,n}$, we define matrix $L(A) = [\hat{a}_{ij}] \in C^{n,n}$ as follows:

$$\hat{a}_{ij} = \begin{cases} 0 & if \ \ (i,j) \notin L_A \\ a_{ij} & if \ \ (i,j) \in L_A \end{cases}, \qquad i,j = 1(1)n$$

Furthermore, we assume that the matrix A exhibits sparse structure which is denoted by $S_A = \{(i,j)\} \subseteq \{1,2,\dots,n\} \times \{1,2,\dots,n\}$. Then, considering $M = A$ equation (3.4.2) leads to the following proposition:

Proposition 3.1. *Let $A = [a_{ij}] \in C^{n,n}$ and $V_s \in R^{s,n}$ be of full rank with $s \leq n$. Then, the equation*

$$L(A) = L(AV_sV_s^T)D^{-1}(V_sV_s^T) \qquad (3.4.3)$$

holds if l^{th} row of V_s is orthogonal to the j^{th} row for every l such that $(i,l) \in S_A$ and every j such that $(i,j) \in L_A$ for i=1(1)n. In other words:

$$L(A) = L(AV_sV_s^T)D^{-1}(V_sV_s^T) \Leftrightarrow$$
$$\forall (i,j) \in L_A \wedge \forall (i,l) \in S_A : (r_l, r_j) = 0$$

When $L_A = \{\ (i,i), i = 1(1)n\}$, i.e., $L(A) = D(A)$, proposition 3.1 reduces to the following one:

Proposition 3.2. *Let $A = [a_{ij}] \in C^{n,n}$ and $V_s \in R^{s,n}$ be of full rank with $s \leq n$. Then, the equation*

$$D(A) = D(AV_sV_s^T)D^{-1}(V_sV_s^T) \qquad (3.4.4)$$

holds if i^{th} row of V_s is orthogonal to all those rows j of V_s for which $a_{ij} \neq 0$.

Proposition 3.1 suggests that if the sparsity structure of matrix A is a priori known, then the s columns of V_s can be selected in such a way such that each element $\hat{a}_{ij}$ of matrix $L(A)$ has no contributions from elements of matrix A except for the element a_{ij} or equivalently $\sum_{\substack{l=1 \\ l \neq j}}^{n} a_{il}\,(r_l, r_j)/(r_j, r_j) = 0$ in (3.4.2).

Moreover, the above mentioned proposition offers a way to compress matrix V_s. Note that the columns of V_s which satisfy proposition 3.1 are not necessarily orthogonal $\left((r_l, r_j) = 0\ \forall (l,j), l \neq j\right)$, which implies that $s = n$. Consequently we can construct a matrix V_s with $s \ll n$, where the value of s depends on L_A and S_A as we will see later.

Furthermore, proposition 3.1 suggests a way to estimate arbitrary entries of a matrix A, i.e. , $L(A)$, based on the action of the matrix on the columns $v_1, v_2, \dots, v_n$ of Vs. In fact, computing in some way the matrix $X_s = AV_s$ where V_s a matrix which satisfies proposition 3.1, leads to the computation of $L(A)$ according to (3.4.3).

Finally, it is clear that when matrix A is dense, preposition 3.1 is satisfied only when $s = n$ and matrix V_s is fully orthogonal. However, if the elements of A exhibit a certain decay property, then matrix A can be considered approximately sparse and equation (3.4.3) results in the following:

$$\widetilde{L(A)} = L(\tilde{A}) \approx L(AV_sV_s^T)D^{-1}(V_sV_s^T) \qquad (3.4.5)$$

, where $\tilde{A}$ is a sparsified version of A and $\widetilde{L(A)}$ the approximation of $L(A)$. Approximate sparsity pattern techniques can be used so as to acquire a prescribed sparsity structure $S_{\tilde{A}}$ of $\tilde{A}$. Then, $S_{\tilde{A}}$ and L_A can be applied directly to proposition 3.1 to approximate $L(A)$.

3.4.2 Construction of the Probing Vectors

Once the pattern of the sparsified matrix $\tilde{A}$ is prescribed, we proceed to the construction of a set of probing vectors which form the, henceforth called, probing matrix V_s. To this purpose, a modified adjacency graph associated with L_A and $S_{\tilde{A}}$ is constructed according to Algorithm 2:

```
1.  Input: $L_A, S_{\tilde{A}}, n$.
2.  Output: the modified adjacency graph $G(V, E)$.
3.  $V = \{1,2,\dots,n\}$
4.  $E = \{\}$
5.  for $i = 1,2,\dots,n$
6.      for each $(i,j) \in L_A$ do
7.          for each $(i,l) \in S_{\tilde{A}}$ do
8.              $E = E \cup (l,j)$
9.          end
10.     end
11. end
```

Algorithm 2: Construction of the Modified Adjacency Graph

As it can be seen, a set of edges is added to the graph G of n vertexes in order to preserve each element $(i,j) \in L_A$. If $L_A = \{\ (i,i),\ i = 1(1)n\}$ then the graph G reduces to the adjacency graph of $\tilde{A}$. Note that the elements $(i,j) \in L_A$ may add many common edges. Furthermore, the outer loop is inherently capable of parallel execution by concatenating the sub-graph that corresponds to each row i of $L(\tilde{A})$ at the end.

Subsequently, the graph is colored so that no adjacent vertices have the same color. Ideally, we want to color the graph with the smallest number of colors, but this is known to be a NP-hard problem. Therefore, we rely on heuristic techniques to find a coloring with an acceptably small number of colors. Algorithm 3 presents a well-known greedy algorithm for this task:

```
1. Input: the modified adjacency graph $G(V, E)$ corresponding to a $n \times n$
   matrix.
2. Output: colors of the vertices of the graph.
3. for $i = 1,2,\dots,n$
4.     set $color(i) = 0$
5. end
6. for $i = 1,2,..,n$
7.     set $color(i) = \min\{k > 0 | k \neq color(l)\ \forall (l,i) \in E\}$
8. end
```

Algorithm 3: Greedy Coloring Algorithm

After coloring the vertices of the modified adjacency graph that corresponds to L_A and $S_{\tilde{A}}$, the probing matrix V_s can be constructed as follows:

$$(V_s)_{jk} = \begin{cases} 1 & color(j) = k \\ 0 & otherwise \end{cases} \tag{3.4.6}$$

As a result, the number of colors of the graph is equal to s and according to (3.4.6) each row of Vs contains exactly one nonzero entry, yielding $D(V_sV_s^T) = D^{-1}(V_sV_s^T) = I$. Therefore, (3.4.5) becomes:

$$\widetilde{L(A)} = L(\tilde{A}) \approx L(AV_sV_s^T) \tag{3.4.7}$$

In addition, the constructed probing matrix V_s satisfies the conditions of proposition 3.1.

Proposition 3.3. *Suppose that the vertices of the modified adjacency graph $G(V,E)$ corresponding to L_A and $S_{\tilde{A}}$ are colored by Algorithm 3. Let V_s be constructed according to (3.4.6). Then*

$$(r_j, r_l) = 0 \ \forall j \ \forall l : \exists i \text{ such that } (i,j) \in L_A \ \wedge (i,l) \in S_{\tilde{A}} \tag{3.4.8}$$

Proof

If an index i such that $(i,j) \in L_A \ \wedge (i,l) \in S_{\tilde{A}}$ exists, then according to Algorithm 2 there is an edge (l,j) in graph G, i.e., $(l,j) \in E$. Consequently, according to Algorithm 3, $color(l) \neq color(j)$. Furthermore, the l^{th} and j^{th} row of V_s consist of zeros except for the $color(l)^{th}$ and $color(j)^{th}$ entry respectively. Hence, the two rows are orthogonal, and the proposition follows immediately.

Example. *Let $A \in C^{8,8}$ be a matrix whose entries L_A are to be approximated and $\tilde{A}$ a sparse version of A. The left part of* Figure 3.2 *depicts $S_{\tilde{A}}$ and L_A. The right part of* Figure 3.2 *depicts the adjacency matrix of the corresponding graph constructed by Algorithm 2 and the color of each vertex obtained according to Algorithm 3. The number of colors required is $s = 4$. Using equation (3.4.6), the probing matrix V_s is given by*

$$V_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

, while the matrices $X_4 = AV_4$ and $X_4V_4^T$ are given by

$$X_4 = \begin{bmatrix} a_{00} & a_{01} & \circledast & \circledast \\ a_{10} & a_{11} & \circledast & \circledast \\ a_{22} & \circledast & \circledast & \circledast \\ a_{33} & a_{34} & \circledast & \circledast \\ a_{43} & a_{44} & \circledast & \circledast \\ \circledast & \circledast & a_{55} & \circledast \\ \circledast & \circledast & a_{66} & a_{67} \\ \circledast & \circledast & a_{76} & a_{77} \end{bmatrix}, X_4V_4^T = \begin{bmatrix} a_{00} & a_{01} & \circledast & \circledast & \circledast & \circledast & \circledast & \circledast \\ a_{10} & a_{11} & \circledast & \circledast & \circledast & \circledast & \circledast & \circledast \\ \circledast & \circledast & a_{22} & \circledast & \circledast & \circledast & \circledast & \circledast \\ \circledast & \circledast & \circledast & a_{33} & a_{34} & \circledast & \circledast & \circledast \\ \circledast & \circledast & \circledast & a_{43} & a_{44} & \circledast & \circledast & \circledast \\ \circledast & \circledast & \circledast & \circledast & \circledast & a_{55} & \circledast & \circledast \\ \circledast & \circledast & \circledast & \circledast & \circledast & \circledast & a_{66} & a_{67} \\ \circledast & \circledast & \circledast & \circledast & \circledast & \circledast & a_{76} & a_{77} \end{bmatrix}$$

We observe that another coloring algorithm could swap the colors of the 7^{th} *and* 8^{th} *vertexes, assigning the same color to the vertexes* 6 *and 8. Consequently, the probing method presented here differs from the standard probing techniques known in the literature [12], [11], where two vertexes* k *and* l *can have the same color only if the* k^{th} *and* l^{th} *columns of* $\tilde{A}$ *have no nonzero entries in the same row positions.*

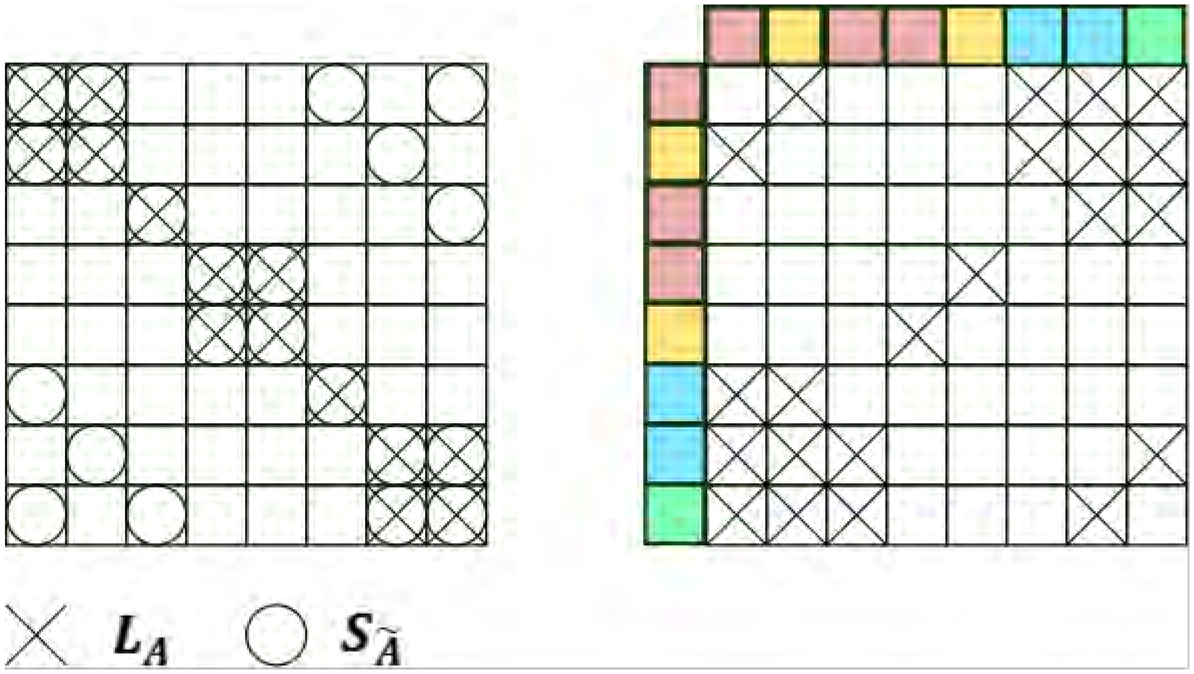

Figure 3.2: Construction and Coloring of the Modified Adjacency Graph

3.4.3 Selected Approximate Matrix Inversion

Let us assume that $L \in R^{n,n}$, $K = L^{-1}$ and we want to approximate some prescribed entries L_K of the inverse matrix K. We also consider that $K \approx \widetilde{K}$, where $\widetilde{K}$ is a sparse matrix with sparsity pattern denoted by $S_{\widetilde{K}}$. We apply (3.4.7) for $A = K$. The only thing left is to compute the matrix $X_s = KV_s$. By observing that $X_s = KV_s \Leftrightarrow K^{-1}X_s = V_s \Leftrightarrow LX_s = V_s$, it follows that:

If $X_s = [x_1, x_2, \dots, x_s], V_s = [v_1, v_2, \dots, v_s]$ *and* $X_s = KV_s$ *, the columns of* X_s *can be obtained by solving the linear systems* $Lx_i = v_i,\ i = 1,2, \dots, s$.

When $s \ll n$, the computation of X_s , and, as a consequence, of $\widetilde{L(K)}$, is much less expensive than solving the full sequence of n linear systems to obtain the whole and exact inverse K by a straightforward use of equations. The sequence of linear systems can be solved by a direct or iterative method depending on the dimension and the condition of the matrix.

Finally, from $\widetilde{L(K)} = L(\widetilde{K}) \approx L(X_s V_s^T)$, it can be easily seen that the element (i, j) of $\widetilde{L(K)}$ can be extracted from X_s as follows:

$$\widetilde{L(K)}(i, j) \approx X_s(i, color(j))$$

, where the probing matrix V_s is constructed according to (3.4.6).

The resulting algorithm of the method is presented below.

```
1.  **Input:** $L \in C^{n,n}, L_K, S_{\widetilde{K}}$ .
2.  **Output:** $\widetilde{L(K)} \in C^{n,n}$.
3.  Construct the modified adjacency graph $G(V,E)$ corresponding to $L_K$
    and $S_K$ from **Algorithm 2**.
4.  Color graph $G(V,E)$ according to **Algorithm 3**.
5.  Set $s = number\ of\ colors$.
6.  Construct the probing matrix $V_s = [v_1, v_2, \dots, v_s]$ from (3.4.6)
7.  **for** $i = 1,2,\dots,s$
8.        Solve $Lx_i = v_i$
9.  **end**
10. $X_s = [x_1, x_2, \dots, x_s]$
11. **for each** $(i,j) \in L_K$
12.       $\widetilde{L(K)}(i,j) = X_s(i, color(j))$
13. **end**
```

Algorithm 4: Selected Approximate Inversion via Probing Technique

As it can be seen, solving the sequence of s linear systems is the computationally most demanding step in the algorithm. However, the probing method can be parallelized by solving the sequence of the linear systems in parallel.

Finally, it should be mentioned that the accuracy of $\widetilde{L(K)}$ depends on the prescribed $S_{\widetilde{K}}$ and the degree of the decreasing trend of the elements in K.

4

GKC-Simulation

4.1 MNA formulation

Given a circuit, let $V = \{0,1,\dots,n-1\}$ be a set of n nodes or vertices (representing the circuit as a directed graph), where the reference node is denoted by the integer 0, and $E = \{e_1, e_2, \dots, e_m\}$ a set of m edges or branches of the circuit. The adjacency matrix, $A \in R^{n-1,m}$, which represents the connectivity of the circuit, can be determined from the directed graph $G(V,E)$ by the following rule:

$$A_{ij} = \begin{cases} +1, if\ node\ i\ is\ the\ source\ of\ branch\ j \\ -1, if\ node\ i\ is\ the\ sink\ of\ branch\ j \\ 0\ , otherwise \end{cases}$$

If $\underline{u}(t) = [u_1(t), \dots., u_m(t)]^T$ and $\underline{i}(t) = [i_1(t), \dots, i_m(t)]^T$ are the vectors of branch voltages and currents respectively, $\underline{v}(t) = [v_1(t), \dots, v_{n-1}(t)\ ]^{\ T}$ is the vector of the node voltages, then the Kirchhoff's law can be expressed as follows:

Kirchhoff's Voltage Law (KVL):

$$\underline{u}(t) = A^T \underline{v}(t) \qquad (4.1.1)$$

Kirchhoff's Current Law (KCL):

$$A\underline{i}(t) = \underline{0} \qquad (4.1.2)$$

The m elements of the circuit can be partitioned into two groups:

1. The elements whose I-V relationship can be written in the form $i_k(t) = g_k u_k(t) + C_k \frac{du_k(t)}{dt} + s_k(t)$ (capacitances, independent current voltages and resistors whose branch current is to be eliminated are included).
2. The elements whose I-V relationship cannot be written in the above mentioned form or the elements whose branch current is not eliminated.

Let m_1 and m_2 be the number of the elements of the first and the second group respectively. Then, the adjacency matrix, the branch voltage vector and the current voltage vector can be partitioned into these forms:

$$A = [\, A_1 \quad A_2 \,], \qquad A_1 \in R^{n-1,m1}, A_2 \in R^{n-1,m2}$$
$$\underline{u}(t) = [\underline{u_1}(t)^T \ \underline{u_2}(t)^T]$$
$$\underline{i}(t) = [\underline{i_1}(t)^T \ \underline{i_2}(t)^T \,]$$

Then, the I-V relationships of the two element groups can be written in matrix form:

$$\underline{i_1}(t) = G\underline{u_1}(t) + C\frac{d\underline{u_1}(t)}{dt} + \underline{s_1}(t) \tag{4.1.3}$$

, where G the conductance matrix, C the capacitance matrix and $\underline{s_1}$ the independent current source vector. The matrices G and C are diagonal.

$$\underline{u_2}(t) = R\underline{i_2}(t) + L\frac{d\underline{i_2}(t)}{dt} + \underline{s_2}(t) \tag{4.1.4}$$

, where R the resistance matrix (for resistor branches whose current is computed during the simulation), L the inductance matrix and $\underline{s_2}(t)$ the independent voltage source vector. The matrix R is diagonal while the inductance matrix L turns out a *dense* and SPD matrix.

Furthermore, (4.1.1) and (4.1.2) can be written as follows:

$$\underline{u}(t) = A^T\underline{v}(t) \Leftrightarrow \begin{cases} \underline{u_1}(t) = A_1^T\underline{v}(t) \\ \underline{u_2}(t) = A_2^T\underline{v}(t) \end{cases} \tag{4.1.5}$$

$$A_1\underline{i_1}(t) + A_2\underline{i_2}(t) = 0 \tag{4.1.6}$$

Substituting the first equation of (4.1.5) into (4.1.3) and the resulting equation into (4.1.6), we have:

$$A_1GA_1^T\underline{v}(t) + A_1CA_1^T\frac{d\underline{v}(t)}{dt} + A_2\underline{i_2}(t) = -A_1\underline{s_1}(t) \tag{4.1.7}$$

Moreover, substituting the second equation of (4.1.5) into (4.1.3), we get:

$$A_2^T\underline{v}(t) - R\underline{i_2}(t) - L\frac{d\underline{i_2}(t)}{dt} = \underline{s_2}(t) \tag{4.1.8}$$

The combination of (4.1.7) and (4.1.8) yields the following $[(n-1)+m_2] \times [(n-1)+m_2]$ linear system of first order differential equations, which is called MNA (Modified Nodal Analysis) system:

$$\tilde{G}\underline{x} + \tilde{C}\underline{\dot{x}} = \underline{b} \tag{4.1.9}$$

$$\tilde{G} = \begin{bmatrix} A_1GA_1^T & A_2 \\ A_2^T & -R \end{bmatrix}, \tilde{C} = \begin{bmatrix} A_1CA_1^T & 0 \\ 0 & -L \end{bmatrix}, \underline{x} = \begin{bmatrix} \underline{v}(t) \\ \underline{i_2}(t) \end{bmatrix}, \underline{\dot{x}} = \frac{d\underline{x}}{dt}, \underline{b} = \begin{bmatrix} -A_1\underline{s_1}(t) \\ \underline{s_2}(t) \end{bmatrix}$$

The linear system in (4.1.9) can be formulated using element stamps, i.e., the contributions of each element to the matrix equation. Figures 4. 1- 4. 2 depict the element stamps in order to construct matrix $\tilde{G}$ in (4.1.9). In matrix $\tilde{G}$ the capacitance branches are replaced by open circuit while the inductance branches are replaced by short circuit. Figure 4.5 shows the contribution of a capacitance branch into matrix $\tilde{C}$ in (4.1.9). Finally, the diagonal elements of matrix L are the values of self-inductances, and the off-diagonal terms are mutual inductances.

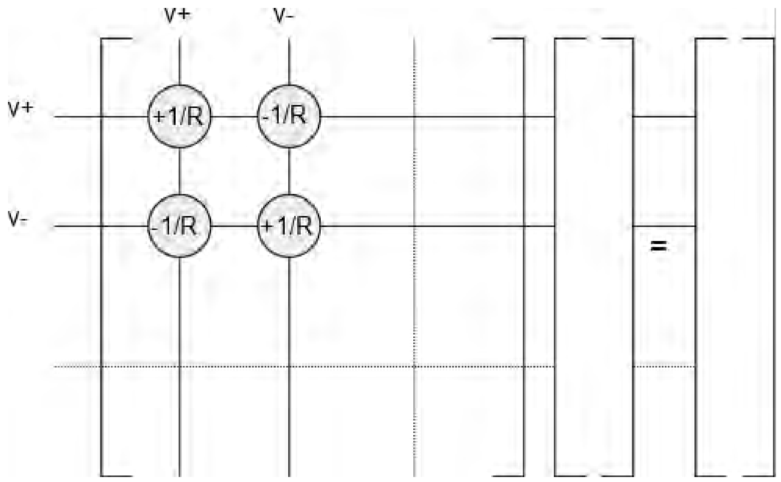

Figure 4.1: Element Stamp for a Resistor in Group 1

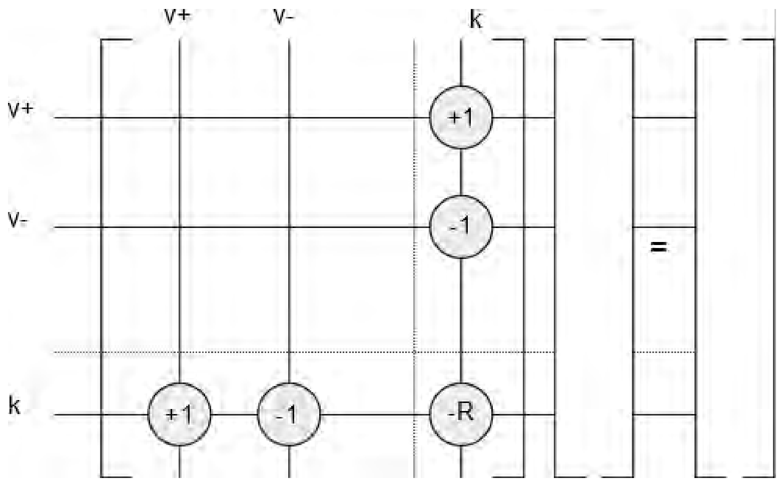

Figure 4.2: Element Stamp for a Resistor in Group 2

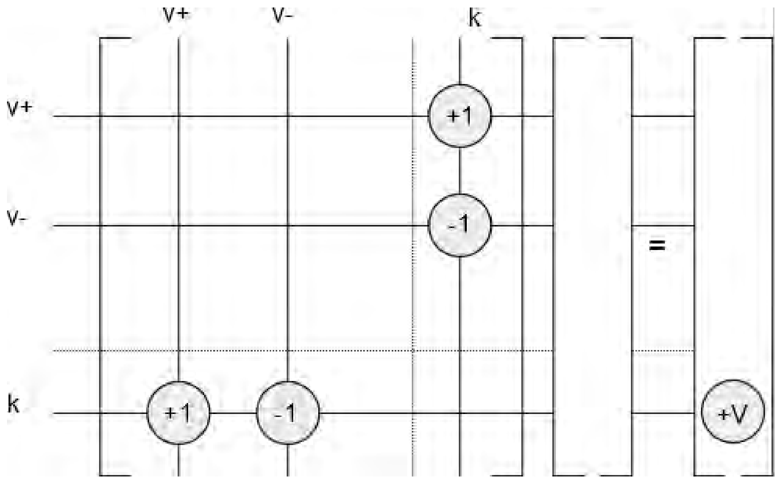

Figure 4.3: Element Stamp for an Independent Voltage Source

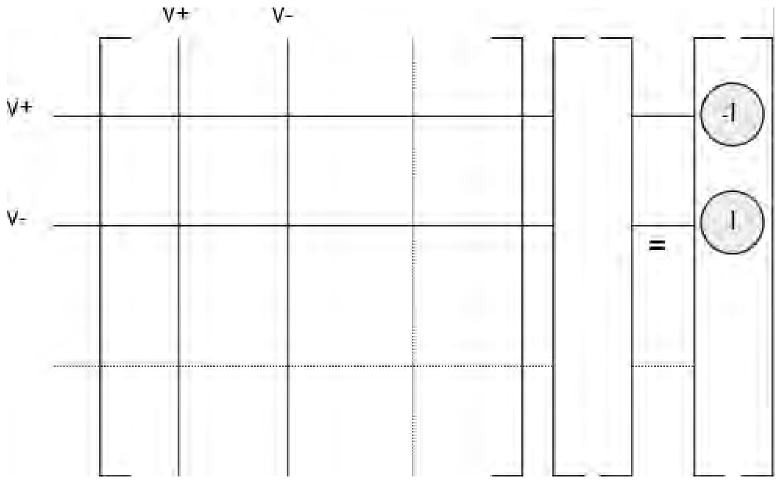

Figure 4.4: Element Stamp for an independent Current Source

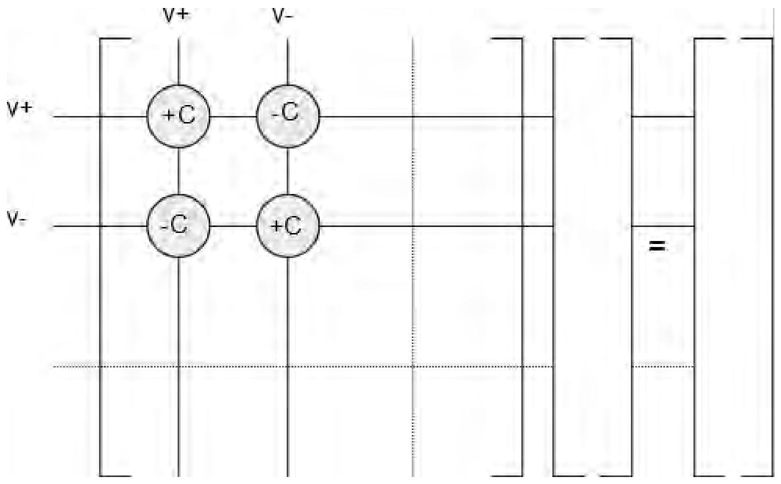

Figure 4.5: Element Stamp for a Capacitance

4.2 Solution of SPD Linear Systems

We remind that a matrix $A \in R^{n,n}$ is SPD (Symmetric Positive Definite) if and only if A is symmetric and $\underline{x}^T A \underline{x} > 0 \ \forall \underline{x} \in R^{n,n} \neq \underline{0}$. Equivalently, a symmetric matrix is SPD if and only if all its eigenvalues are strictly positive. As a consequence, the matrix is nonsingular (it has no zero eigenvalue) and its inverse is also an SPD matrix. Finally, recall that if matrix A is strictly diagonally dominant and all the diagonal elements are positive, then it is also SPD (7.2.3).

A network meets the consistency requirements if the network graph does not contain any current source cutsets and any voltage source loops. Remember that a cutset in a connected graph is a set of edges which, if removed, would cause the graph to become disconnected. It can be proved that, given the consistency constraints, if the network is connected and the group 2 elements do not form a cutset, the matrices $\dot{G} = A_1 G A_1^T$ and $\dot{C} = A_1 C A_1^T$ are SPD. Finally, the inductance matrix L is SPD but not strictly diagonally dominant.

An SPD linear system $Ax = b$ can be solved directly using Cholesky decomposition or iteratively using Preconditioned Conjugate Gradient method. The two methods are described briefly below.

Cholesky Decomposition

An equivalent definition of the SPD property is the following: A matrix $A \in R^{n,n}$ is SPD if and only if there exists a nonsingular, lower triangular matrix $L \in R^{n,n}$ with strictly positive diagonal entries such that $A = LL^T$. This decomposition is unique. Subsequently, the linear system can be solved implementing backward and forward substitution. The algorithm, which computes the L factor is presented below.

```
1. Input: $A \in R^{n,n}$ an SPD matrix.
2. Output: $L \in R^{n,n}$ such that $A = LL^T$ and $L_{ii} > 0,\ i = 1(1)n$.
3. for $k = 1,2,\dots,n$
4.     $L_{kk} = \sqrt{A_{kk} - \sum_{j=1}^{k-1} L_{kj}^2}$
5.     for $i = k+1,\dots,n$
6.         $L_{ik} = \frac{A_{ki} - \sum_{j=1}^{k-1} L_{ij} L_{kj}}{L_{kk}}$
7.     end
8. end
```

Algorithm 5: Cholesky Decomposition

Preconditioned Conjugate Gradient(CG)

The Conjugate Gradient method is based on the minimization of the function $f(\underline{x}) = \frac{1}{2}\underline{x}^T A\, \underline{x} - \underline{b}^T \underline{x}$. The gradient of f equals $\nabla f = A\underline{x} - \underline{b}$, therefore the solution x^* of the linear system $Ax = b$ is the unique minimizer of f. Specifically, in each iteration i a new search direction $\underline{p}^{(i)}$ is discovered such that it is A-orthogonal to the previous search direction $\underline{p}^{(i-1)}$ while the current approximation $\underline{x}^{(i)}$ (and the residual $\underline{r}^{(i)} = b - Ax^{(i)}$ as well) moves in the direction $\underline{x}^{(i)} = \underline{x}^{(i-1)} + \alpha_i \underline{p}^{(i)}$ where a_i such that the function $f(\underline{x}^{(i)})$ is minimized. Finally, preconditioning is used to replace the original system $A\underline{x} - \underline{b} = 0$ with $M^{-1}(A\underline{x} - \underline{b}) = 0$ so that the condition number $\kappa(M^{-1}A)$ gets smaller than $\kappa(A)$. The preconditioned conjugate gradient method takes the following form:

```
1.  Input: $A \in R^{n,n}$ an SPD matrix and $b \in R^n$.
2.  Output: the solution $x^*$ of the linear system $Ax = b$.
3.  $\underline{x}$= initial guess
4.  $\underline{r} = \underline{b} - A\underline{x}$
5.  $iter = 0$
6.  while ( ‖ $\underline{r}$ ‖/‖ $\underline{b}$ ‖> $itol$)
7.          $iter = iter + 1$
8.          Solve $M\underline{z} = \underline{r}$
9.          $rho = \underline{r}^T\underline{z}$
10.         if  $(iter == 1)$
11.                 $\underline{p} = \underline{z}$
12.         else
13.                 $beta = rho/rho1$
14.                 $\underline{p} = \underline{z} + beta * \underline{p}$
15.         end
16.         $rho1 = rho$
17.         $\underline{q} = A\underline{p}$
18.         $alpha = rho/(\underline{p}^T\underline{q})$
19.         $\underline{x} = \underline{x} + alpha * \underline{p}$
20.         $\underline{r} = \underline{r} - alpha * \underline{q}$
21. end
```

Algorithm 6: Preconditioned Conjugate Gradient Method

4.3 Transient Analysis

Transient Analysis computes the response of the system as a function of time in the interval $t \in [t_0, t_1]$. The linear system of (4.1.8) can be solved by discretization of the time interval into m time points, replacing the time derivative operator with a discrete time operator and solving the resulting finite difference equations one time point at a time starting from some initial condition. The particular discrete approximation method is referred to as the integration method. As integration methods Backward-Euler or Trapezoidal Rule can be used. Below the approximation of $\frac{d\underline{x}(t)}{dt}$ in each case is given and it is applied to (4.1.9) so as to get the discretized form of the MNA system.

<u>Backward Euler</u>:

$$\frac{d\underline{x}(t_k)}{dt} \approx \frac{1}{h}\left[\underline{x}(t_k) - \underline{x}(t_{k-1})\right]$$

$$\left(\tilde{G} + \frac{1}{h}\tilde{C}\right)\underline{x}(t_k) = \underline{b}(t_k) + \frac{1}{h}\tilde{C}\underline{x}(t_{k-1}) \quad k = 1,2,\dots,m \tag{4.3.1}$$

Trapezoidal Rule:

$$\frac{1}{2}\left[\frac{d\underline{x}(t_k)}{dt} + \frac{d\underline{x}(t_{k-1})}{dt}\right] \approx \frac{1}{h}\left[\underline{x}(t_k) - \underline{x}(t_{k-1})\right]$$

$$\left(\tilde{G} + \frac{2}{h}\tilde{C}\right)\underline{x}(t_k) = \underline{b}(t_k) + \underline{b}(t_{k-1}) - \left(\tilde{G} - \frac{2}{h}\tilde{C}\right)\underline{x}(t_{k-1}) \quad k = 1,2,..,m \tag{4.3.2}$$

, where $t_k = t_0 + kh$ and $h = \frac{t_1 - t_0}{m}$.

4.4 Nodal Analysis for linear circuits

The MNA approach works only for the full inductance matrix L or ordinary sparse partial inductance approximations, but not for the reluctance matrix $K = L^{-1}$. Furthermore, the introduction of extra current variables can make the system matrix non-positive definite, which is crucially necessary for the efficiency of the direct method (Cholesky decomposition takes only half of multiplications and memory references than the LU decomposition and there is no need for permutation or pivoting) and the fast convergence of the iterative method.

In this section, it is shown that the NA approach is feasible for sparse reluctance matrices and equations (4.3.1) and (4.3.2) are converted in order to make use of the reluctance matrix instead of the inductance matrix. Finally, we eliminate branch currents except those running through inductors and we assume that there are no independent voltage sources in the circuit. Therefore, matrix $\tilde{G}$, vector $\underline{b}$ and vector $\underline{x}$ become respectively:

$$\tilde{G} = \begin{bmatrix} A_1 G A_1^T & A_2 \\ A_2^T & 0 \end{bmatrix},\ \underline{b} = \begin{bmatrix} -A_1 \underline{s_1}(t) \\ 0 \end{bmatrix}, \underline{x} = \begin{bmatrix} \underline{v}(t) \\ \underline{i_L}(t) \end{bmatrix}$$

Backward Euler:

$$\left(\begin{bmatrix} A_1 G A_1^T & A_2 \\ A_2^T & 0 \end{bmatrix} + \frac{1}{h}\begin{bmatrix} A_1 C A_1^T & 0 \\ 0 & -L \end{bmatrix}\right)\begin{bmatrix} \underline{v}(t_k) \\ \underline{i_L}(t_k) \end{bmatrix} = \begin{bmatrix} -A_1 \underline{s_1}(t_k) \\ 0 \end{bmatrix} + \frac{1}{h}\begin{bmatrix} A_1 C A_1^T & 0 \\ 0 & -L \end{bmatrix}\begin{bmatrix} \underline{v}(t_{k-1}) \\ \underline{i_L}(t_{k-1}) \end{bmatrix} \Leftrightarrow$$

$$A_1 G A_1^T \underline{v}(t_k) + A_2 \underline{i_L}(t_k) + \frac{1}{h} A_1 C A_1^T \underline{v}(t_k) = -A_1 \underline{s_1}(t_k) + \frac{1}{h}\, A_1 C A_1^T \underline{v}(t_{k-1}) \tag{4.4.1}$$

$$A_2^T \underline{v}(t_k) - \frac{1}{h} L \underline{i_L}(t_k) = -\frac{1}{h} L \underline{i_L}(t_{k-1}) \Leftrightarrow \underline{i_L}(t_k) = hKA_2^T \underline{v}(t_k) + \underline{i_L}(t_{k-1}) \tag{4.4.2}$$

By substituting (4.4.2) into (4.4.1) we obtain:

$$\left(A_1GA_1^T+\frac{1}{h}A_1CA_1^T+h\,A_2KA_2^T\right)\underline{v}(t_k)=-A_1\underline{s_1}(t_k)+\frac{1}{h}\,A_1CA_1^T\underline{v}(t_{k-1})-A_2\underline{i_L}(t_{k-1})$$

Using the approximate sparse version $\widetilde{K}$ of K, we just get:

$$Y\underline{v}(t_k)=-A_1\underline{s_1}(t_k)+\frac{1}{h}\,A_1CA_1^T\underline{v}(t_{k-1})-A_2\underline{i_L}(t_{k-1}) \tag{4.4.3.a}$$

$$Y=A_1GA_1^T+\frac{1}{h}A_1CA_1^T+h\,A_2\widetilde{K}A_2^T \tag{4.4.3.b}$$

$$\underline{i_L}(t_k)=h\widetilde{K}A_2^T\underline{v}(t_k)+\underline{i_L}(t_{k-1}) \tag{4.4.3.c}$$

Given that K is not only SPD but also strictly diagonally dominant with positive diagonal entries, matrix $\widetilde{K}$ is also strictly diagonally dominant with positive diagonal entries, consequently the SPD property is preserved with the sparse approximate inversion of K. The admittance matrix Y is also SPD since $\forall z$: $z^T\left(A_2\widetilde{K}A_2^T\right)z=(A_2^Tz)\widetilde{K}(A_2z)=w^T\widetilde{K}w>0$ with $w=A_2^Tz$ and the matrices $A_1GA_1^T$, $A_1CA_1^T$ are also SPD. Therefore, the Cholesky Decomposition or the Preconditioned Conjugate Gradient iterative method is applicable to the NA formulation.

Trapezoidal Rule:

$$\left(\begin{bmatrix}A_1GA_1^T & A_2\\ A_2^T & 0\end{bmatrix}+\frac{2}{h}\begin{bmatrix}A_1CA_1^T & 0\\ 0 & -L\end{bmatrix}\right)\begin{bmatrix}\underline{v}(t_k)\\ \underline{i_L}(t_k)\end{bmatrix}=$$

$$\begin{bmatrix}-A_1\underline{s_1}(t_k)\\ 0\end{bmatrix}+\begin{bmatrix}-A_1\underline{s_1}(t_{k-1})\\ 0\end{bmatrix}-\left(\begin{bmatrix}A_1GA_1^T & A_2\\ A_2^T & 0\end{bmatrix}-\frac{2}{h}\begin{bmatrix}A_1CA_1^T & 0\\ 0 & -L\end{bmatrix}\right)\begin{bmatrix}\underline{v}(t_{k-1})\\ \underline{i_L}(t_{k-1})\end{bmatrix}\Leftrightarrow$$

$$\begin{gathered}\left(A_1GA_1^T+\frac{2}{h}A_1CA_1^T\right)\underline{v}(t_k)+A_2\underline{i_L}(t_k)=\\ \left(-A_1GA_1^T+\frac{2}{h}A_1CA_1^T\right)\underline{v}(t_{k-1})-A_2\underline{i_L}(t_{k-1})-A_1\left(\underline{s_1}(t_k)+\underline{s_1}(t_{k-1})\right)\end{gathered} \tag{4.4.4}$$

$$\begin{gathered}A_2^T\underline{v}(t_k)-\frac{2}{h}L\underline{i_L}(t_k)=-A_2^T\underline{v}(t_{k-1})-\frac{2}{h}L\underline{i_L}(t_{k-1})\Leftrightarrow\\ \underline{i_L}(t_k)=\frac{h}{2}KA_2^T\left(\underline{v}(t_k)+\underline{v}(t_{k-1})\right)+\underline{i_L}(t_{k-1})\end{gathered} \tag{4.4.5}$$

By substituting (4.4.5) into (4.4.4) we get:

$$\begin{gathered}\left(A_1GA_1^T+\frac{2}{h}A_1CA_1^T+\frac{h}{2}A_2KA_2^T\right)\underline{v}(t_k)=\\ \left(-A_1GA_1^T+\frac{2}{h}A_1CA_1^T-\frac{h}{2}A_2KA_2^T\right)\underline{v}(t_{k-1})-2A_2\underline{i_L}(t_{k-1})-A_1\left(\underline{s_1}(t_k)+\underline{s_1}(t_{k-1})\right)\end{gathered}$$

Using the sparsified version $\widetilde{K}$ of K, we just get:

$$Y\underline{v}(t_k) = \left(-A_1GA_1^T + \frac{2}{h}A_1CA_1^T - \frac{h}{2}A_2\widetilde{K}A_2^T\right)\underline{v}(t_{k-1}) - 2A_2\underline{i_L}(t_{k-1}) - A_1\left(\underline{s_1}(t_k) + \underline{s_1}(t_{k-1})\right) \quad (4.4.6.a)$$

$$Y = A_1GA_1^T + \frac{2}{h}A_1CA_1^T + \frac{h}{2}A_2\widetilde{K}A_2^T \quad (4.4.6.b)$$

$$\underline{i_L}(t_k) = \frac{h}{2}\widetilde{K}A_2^T\left(\underline{v}(t_k) + \underline{v}(t_{k-1})\right) + \underline{i_L}(t_{k-1}) \quad (4.4.6.c)$$

Assuming that $\widetilde{K}$ is SPD, the admittance matrix Y is also SPD, therefore the Cholesky decomposition or the preconditioned conjugate gradient iterative method is applicable to the NA formulation.

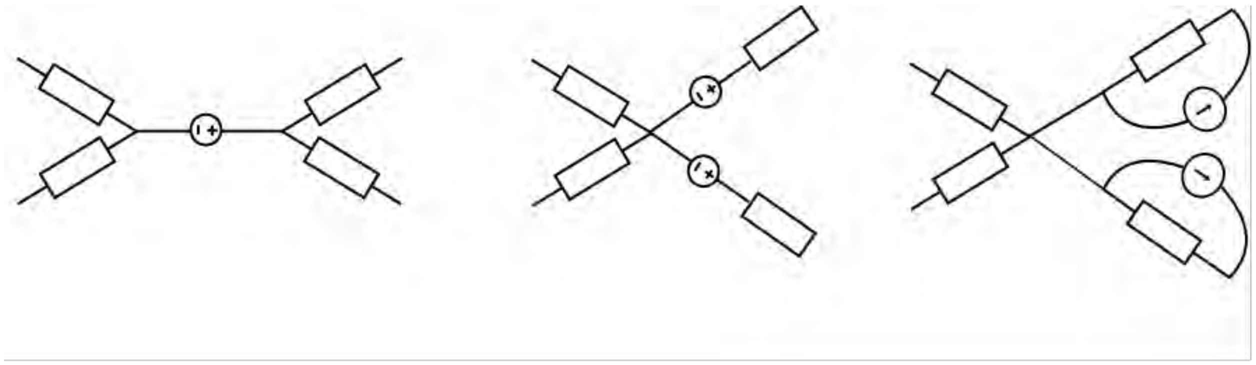

Figure 4.6: Voltage Source Transformation and Norton equivalent Circuit

In case there are independent voltages sources in the circuit, extra current variables should be added. Hence, they are transformed into Norton equivalent circuits, as shown in Figure 4.6. If the voltage source is connected to R or C element, which have Norton equivalent circuit, this transformation can be easily implemented. However, this transformation is inapplicable for L elements. Using frequency domain analysis, the following current-voltage equations are derived:

$$I_1 = \frac{K_{11}}{s}(V_1 - V_S) + \frac{K_{12}}{s}V_2$$
$$I_2 = \frac{K_{12}}{s}(V_1 - V_S) + \frac{K_{22}}{s}V_2$$

These two equations can be rewritten as

$$I_1 = \frac{K_{11}}{s}V_1 + \frac{K_{12}}{s}V_2 - \frac{K_{11}}{s}V_S$$
$$I_2 = \frac{K_{12}}{s}V_1 + \frac{K_{22}}{s}V_2 - \frac{K_{12}}{s}V_S$$

which can be represented as the circuit shown in Figure 4.7. The voltage source is replaced by current sources. Since conductance (G), capacitance (C) and reluctance (K) are all admittances, they share similarities in equivalent circuit transformation. Thus, it can be applied to the NA analysis.

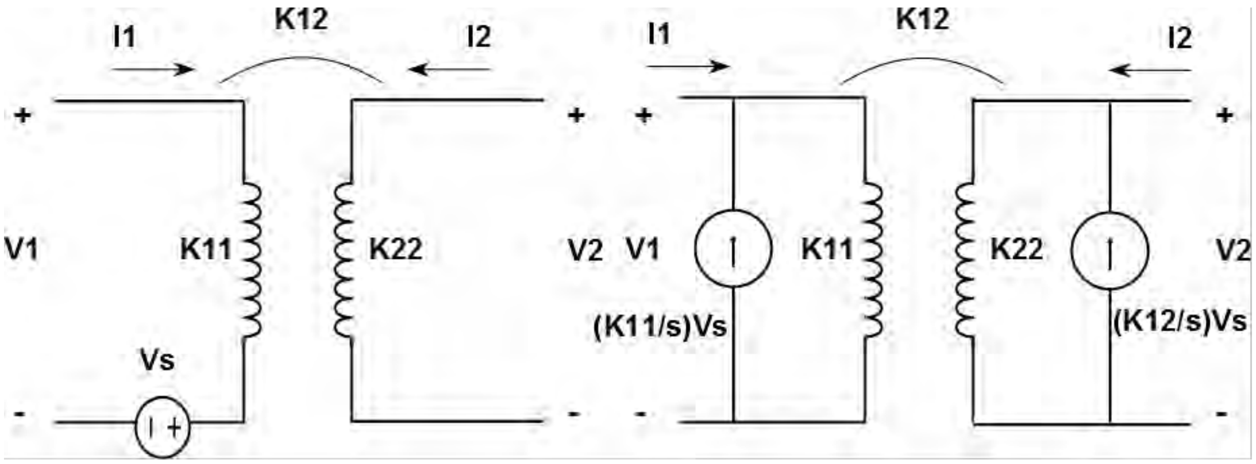

Figure 4.7: Norton equivalent Transformation for Reluctance Element

5

Experimental Results

We demonstrate the sparse approximate inverse methods presented in this thesis on a bus structure with 256 and 1024 signals for different sparsity ratios $\varepsilon = \left(1 - \frac{\#\,nonzeros}{\#entries}\right) \times 100$ while the wires are divided into 8 segments yielding inductance matrices of size 1024×1024, 2048×2048 and 8192×8192 respectively. The first structure consists of two 128-wire blocks. In the second case, a $3D$ interconnect structure consisting of 4 layers with two 128-wire blocks in each layer is considered. The inductance matrices are obtained using FastHenry [24] for conductivity $sigma = 3.77e7$ (aluminum). The wire length is taken $1\,mm$, the cross-section is $1 \times 1\,um$. The separation distance between wires of the same block is $1\,um$ while the separation distance between two bus blocks or layers is considered $2\,um$. The driver resistance of each line is $30\,Ohm$ and the load capacitance is $20\,fF$. A $1V$ $20\,ps$ ramp voltage source input is applied to the first signal of the lowest plane, and the rest are quiet. A time step of $1\,ps$ is taken and the simulation is performed over $700ps$ (or equivalently 700 time steps). The sparsity structure of the inductance matrix L, which is obtained selecting the k largest entries of each column, is considered as the sparsity pattern of the corresponding sparse approximate reluctance matrix $\widetilde{K}$. All experiments were run on an Intel Core i7 at 2.2 GHz with 4 cores and 8 GB main memory. A comprehensive comparison in terms of accuracy and runtime is presented in the next sections.

5.1 Accuracy Comparison

The following figures depict the voltage response at the far end of some selected lines using the sparse approximate inverse methods described in Chapter 3, which are applied to the inductance matrix L of size 2048×2048. Specifically, in Figures 5.1-5.6 results from the K-method as described in section 3.1 are shown for different sparsity ratios ε. We can observe that a less sparse reluctance matrix K improves the accuracy especially on the remote quiet lines since the waveform converges to what is obtained when the full and exact reluctance matrix K is used. Similarly, in Figures 5.7-5.12 the simulation results using the probing technique in order to approximate selected entries of the reluctance matrix are depicted while in Figures 5.13-5.18 the voltage responses are shown when the reluctance matrix $\widetilde{K}$ which minimizes the $\| L\widetilde{K} - I \|_F$ given the prescribed sparsity pattern is used. Finally, the waveforms obtained when the AINV algorithm is applied to the inductance matrix L are given in Figures 5.19-5.25. The AINV algorithm dynamically captures the nonzero elements using a drop tolerance $= 1e-1$, yielding a less sparse reluctance matrix of sparsity ratio $\varepsilon = 84.11\%$. The use of higher drop tolerance leads to AINV's breakdown and a poor and not even sparse approximation of K is obtained.

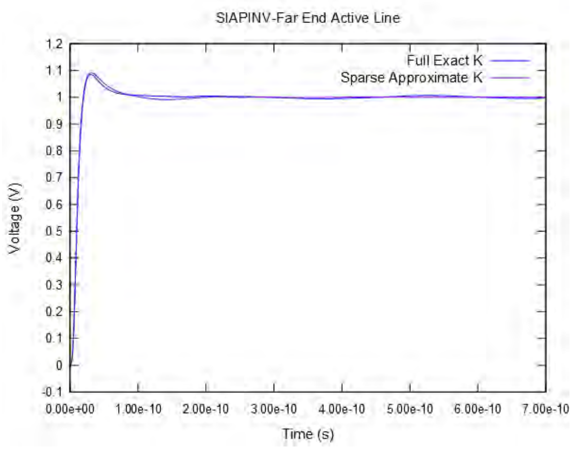

Figure 5.1: 2048X2048-SIAPINV-Voltage Response at far end of Active Line

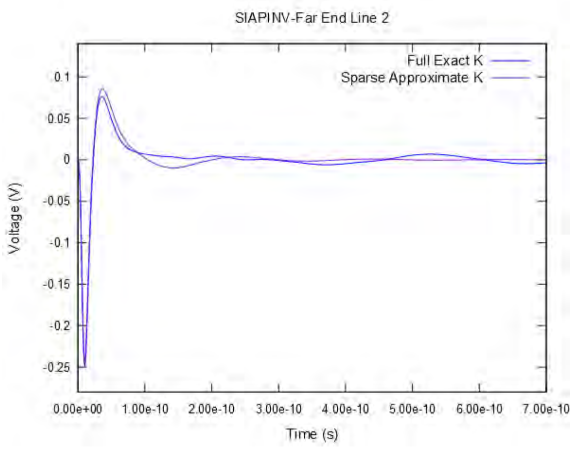

Figure 5.2: 2048X2048-SIAPINV- Voltage Response at far end of Line 2

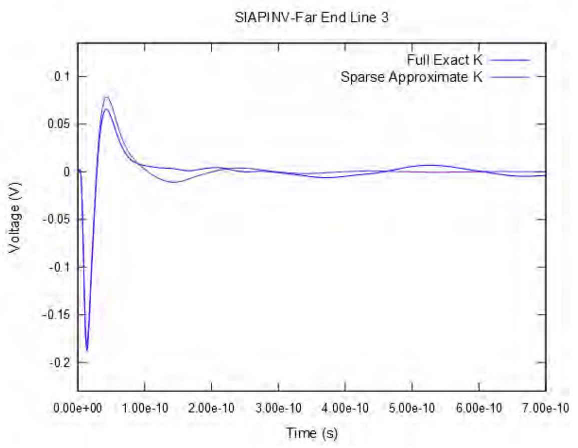

Figure 5.3: 2048X2048-SIAPINV- Voltage Response at far end of Line 3

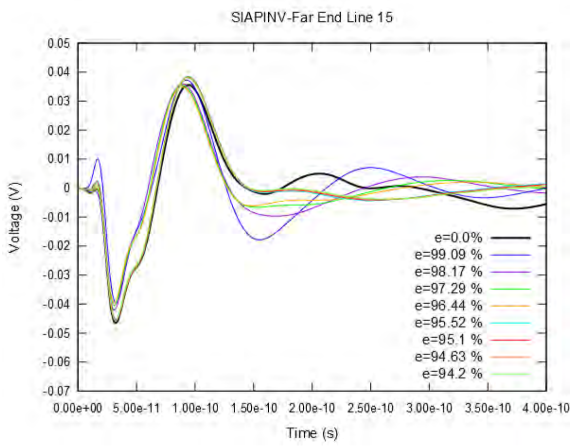

Figure 5.4: 2048X2048-SIAPINV- Voltage Response at far end of Line 15

.

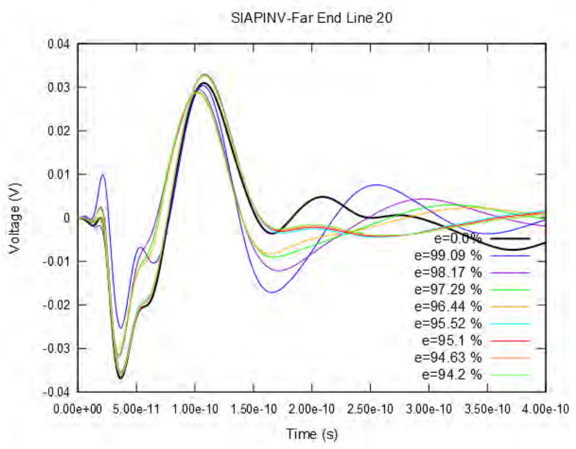

Figure 5.5: 2048X2048-SIAPINV- Voltage Response at far end of Line 20

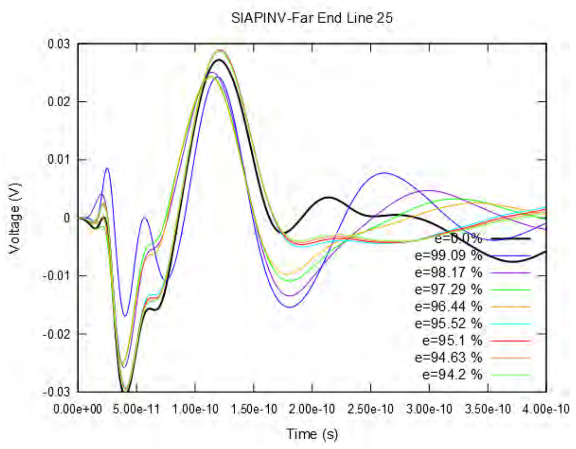

Figure 5.6: 2048X2048-SIAPINV- Voltage Response at far end of Line 25

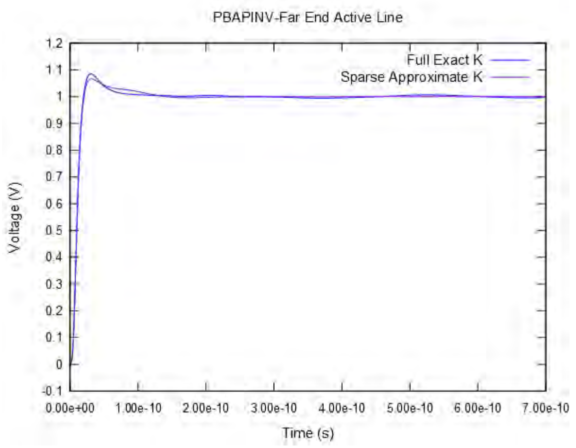

Figure 5.7: 2048X2048- PBAPINV- Voltage Response at far end of Active Line

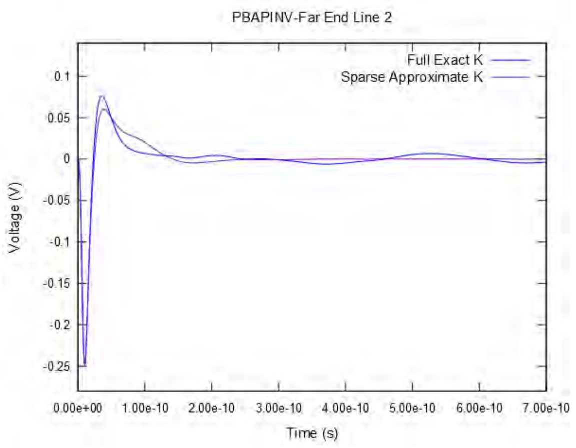

Figure 5.8: 2048X2048-PBAPINV- Voltage Response at far end of Line 2

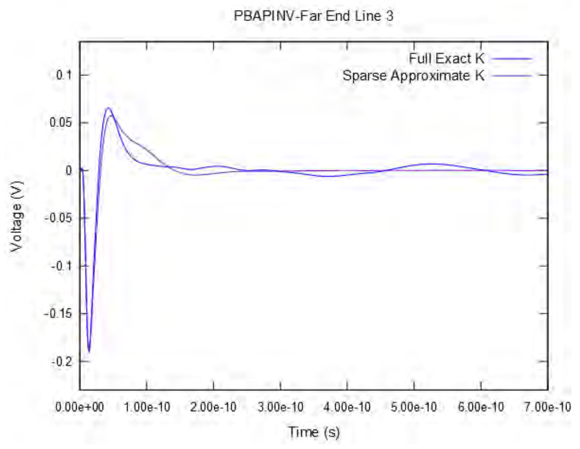

Figure 5.9: 2048X2048-PBAPINV- Voltage Response at far end of Line 3

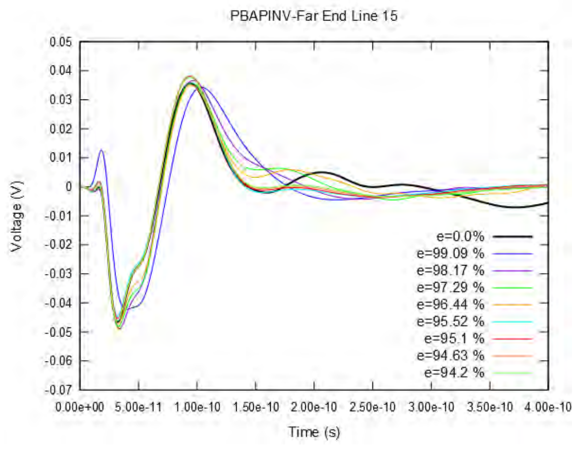

Figure 5.10: 2048X2048-PBAPINV- Voltage Response at far end of Line 15

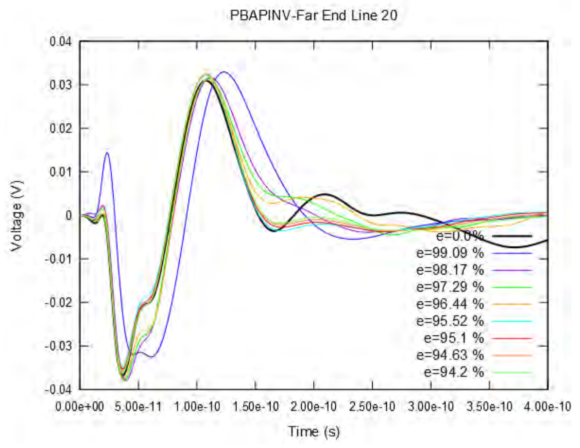

Figure 5.11: 2048X2048-PBAPINV- Voltage Response at far end of Line 20

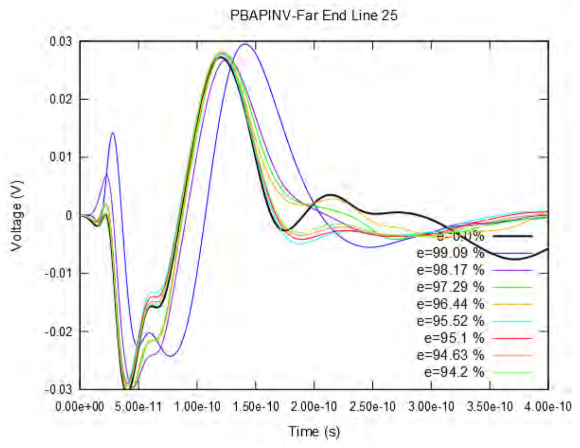

Figure 5.12: 2048X2048-PBAPINV- Voltage Response at far end of Line 25

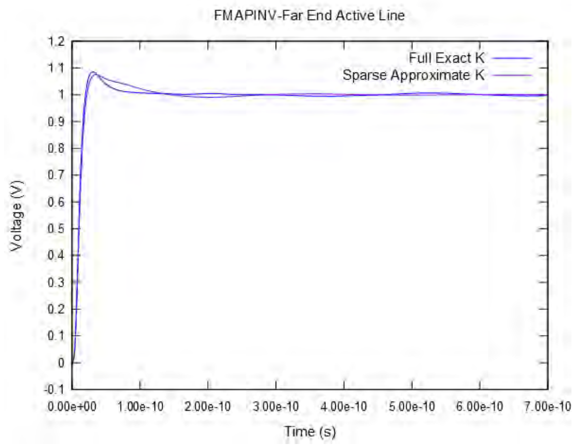

Figure 5.13: 2048X2048-FMAPINV- Voltage Response at far end of Active Line

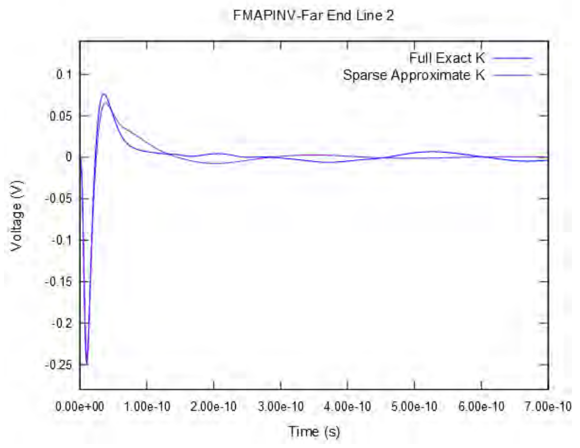

Figure 5.14: 2048X2048-FMAPINV- Voltage Response at far end of Line 2

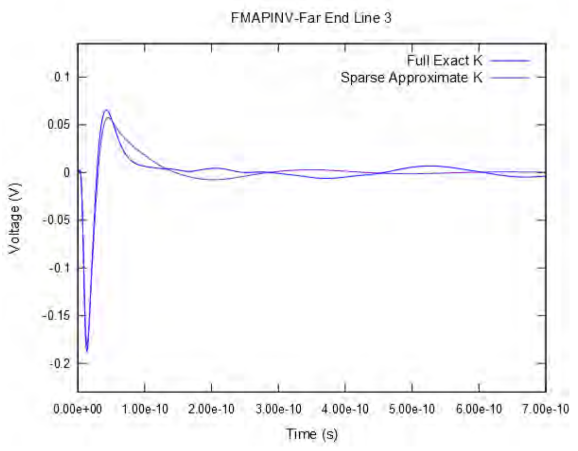

Figure 5.15: 2048X2048-FMAPINV- Voltage Response at far end of Line 3

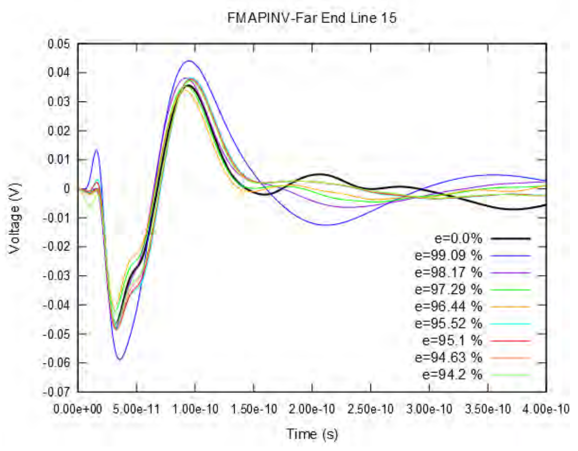

Figure 5.16: 2048X2048-FMAPINV- Voltage Response at far end of Line 15

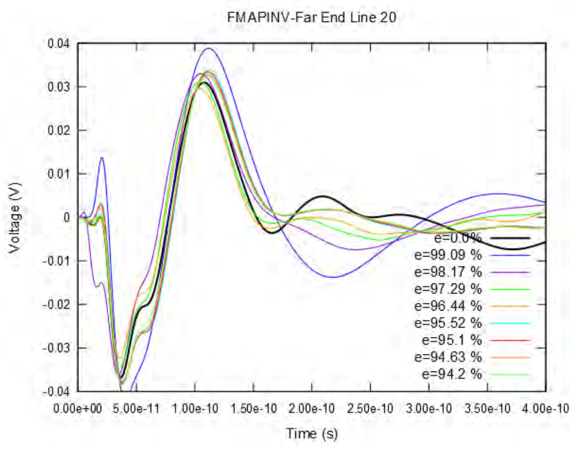

Figure 5.17: 2048X2048-FMAPINV- Voltage Response at far end of Line 20

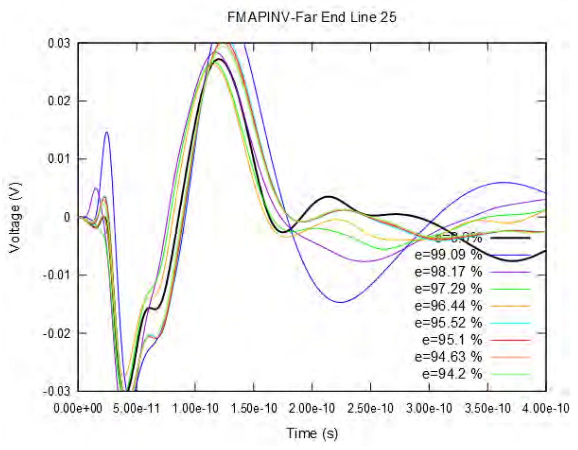

Figure 5.18: 2048X2048-FMAPINV- Voltage Response at far end of Line 25

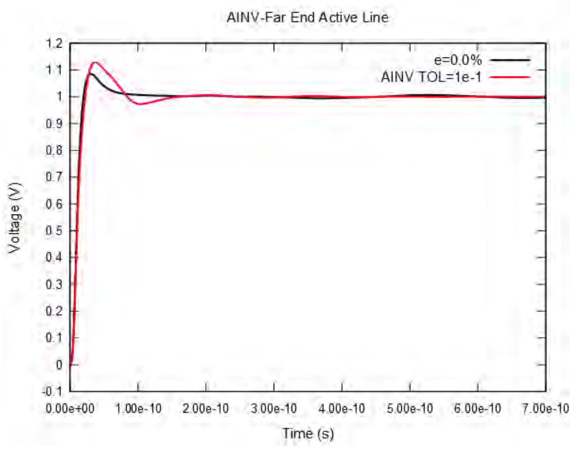

Figure 5.19: 2048X2048-AINV-Voltage Response at far end of Active Line

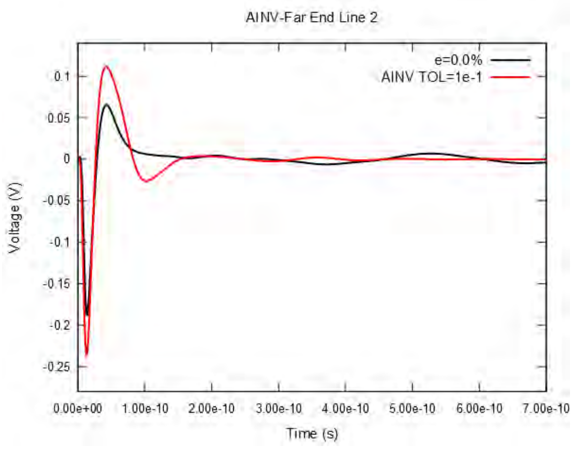

Figure 5.20: 2048X2048-AINV- Voltage Response at far end of Line 2

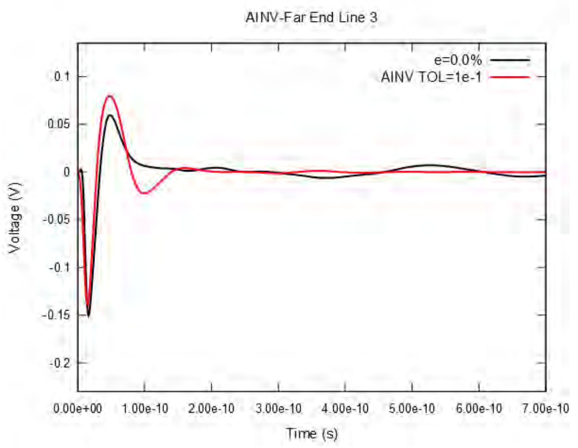

Figure 5.21: 2048X2048-AINV- Voltage Response at far end of Line 3

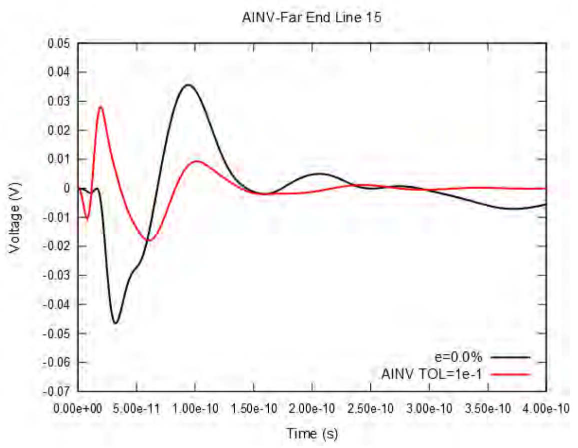

Figure 5.22: 2048X2048-AINV- Voltage Response at far end of Line 15

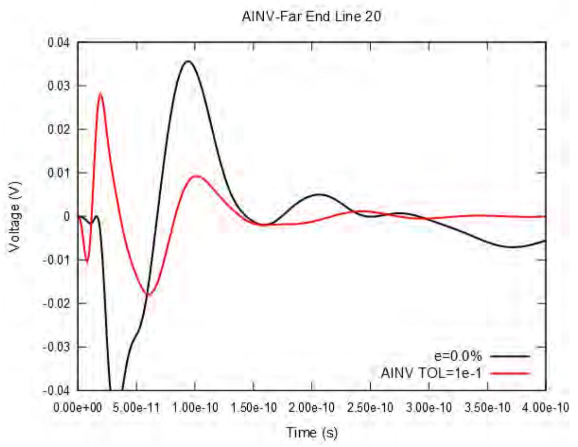

Figure 5.23: 2048X2048-AINV- Voltage Response at far end of Line 20

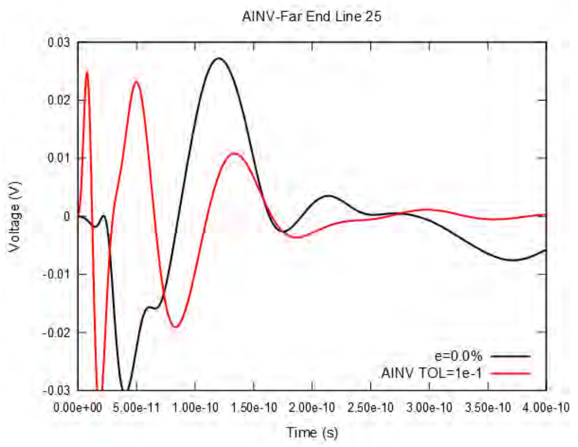

Figure 5.24: 2048X2048-AINV- Voltage Response at far end of Line 25

In the next tables, we also provide numerical results in order to measure the approximation errors involved. The voltage at the far end in i^{th} wire obtained by the full and exact K matrix is denoted by V_i while the i^{th} voltage obtained by the approximation methods is denoted by $\widetilde{V}_i$. Define relative mean square error (RMSE) and average error ratio (AER) for the i^{th} wire as:

$$RMSE = \frac{\sum_t \left|\tilde{V}_i - V_i\right|^2}{\sum_t \left|V_i\right|^2}$$

$$AER = \frac{\sum_t \left|\tilde{V}_i - V_i\right|}{\sum_t \left|V_i\right|}$$

, while for all wires

$$RMSE = \frac{\sum_i \sum_t \left|\tilde{V}_i - V_i\right|^2}{\sum_i \sum_t |V_i|^2}$$
$$AER = \frac{\sum_i \sum_t |\tilde{V}_i - V_i|}{\sum_i \sum_t |V_i|}$$

SIAPINV RMSE	Active Line	Line 2	Line 3	Line 15	Line 20	Line 25	All
ε=9.9089 43e-01	3.036944 e-05	3.576216 e-02	5.570589 e-02	3.605437 e-01	4.790277 e-01	6.059697 e-01	1.105955 e-02
ε=9.8168 18e-01	2.201243 e-05	2.539602 e-02	3.852820 e-02	3.017467 e-01	3.823853 e-01	4.412011 e-01	1.037395 e-02
ε=9.7285 08e-01	2.245933 e-05	2.608806 e-02	3.975907 e-02	2.789697 e-01	3.769884 e-01	4.801728 e-01	1.061554 e-02
ε=9.6440 12e-01	2.300971 e-05	2.681154 e-02	4.096418 e-02	2.762627 e-01	3.649315 e-01	4.627475 e-01	1.086631 e-02
ε=9.5518 83e-01	1.460141 e-05	1.670067 e-02	2.528371 e-02	1.639839 e-01	2.155728 e-01	2.670607 e-01	9.118349 e-03
ε=9.5073 03e-01	1.352611 e-05	1.543163 e-02	2.334371 e-02	1.526012 e-01	1.996600 e-01	2.456598 e-01	8.995759 e-03
ε=9.4632 20e-01	1.265333 e-05	1.440396 e-02	2.176256 e-02	1.427184 e-01	1.860410 e-01	2.274609 e-01	8.894306 e-03
ε=9.4196 32e-01	1.194173 e-05	1.356971 e-02	2.047839 e-02	1.343552 e-01	1.744875 e-01	2.122227 e-01	8.810593 e-03

Table 1: 2048X2048- SIAPINV-RMSE

SIAPINV AER	Active Line	Line 2	Line 3	Line 15	Line 20	Line 25	All
ε=9.9089 43e-01	4.563572 e-03	4.495964 e-01	5.087662 e-01	8.161832 e-01	8.737638 e-01	9.670518 e-01	5.367691 e-01
ε=9.8168 18e-01	4.008922 e-03	3.906328 e-01	4.384428 e-01	7.495390 e-01	7.935443 e-01	8.380581 e-01	5.147417 e-01
ε=9.7285 08e-01	4.010210 e-03	3.922126 e-01	4.417317 e-01	7.423047 e-01	8.051426 e-01	8.765094 e-01	5.238150 e-01
ε=9.6440 12e-01	4.058893 e-03	3.976765 e-01	4.485671 e-01	7.488693 e-01	8.037312 e-01	8.704739 e-01	5.331940 e-01
ε=9.5518 83e-01	3.292822 e-03	3.197420 e-01	3.580709 e-01	5.631057 e-01	6.044620 e-01	6.471088 e-01	4.674041 e-01
ε=9.5073 03e-01	3.162194 e-03	3.063737 e-01	3.428963 e-01	5.432858 e-01	5.812840 e-01	6.204878 e-01	4.613163 e-01
ε=9.4632 20e-01	3.053511 e-03	2.951368 e-01	3.301075 e-01	5.250931 e-01	5.603593 e-01	5.965964 e-01	4.559147 e-01
ε=9.4196 32e-01	2.960593 e-03	2.856870 e-01	3.193152 e-01	5.087999 e-01	5.417205 e-01	5.755338 e-01	4.511952 e-01

Table 2: 2048X2048-SIAPINV-AER

PBAPINV RMSE	Active Line	Line 2	Line 3	Line 15	Line 20	Line 25	All
ε=9.908943e-01	3.632719e-05	4.743120e-02	7.621948e-02	4.487163e-01	7.742496e-01	1.040291e+00	1.619646e-02
ε=9.816818e-01	1.854166e-05	2.248250e-02	3.537868e-02	2.004875e-01	2.426797e-01	3.079974e-01	1.227308e-02
ε=9.728508e-01	1.459244e-05	1.726390e-02	2.676449e-02	1.749815e-01	2.074733e-01	2.310265e-01	9.979212e-03
ε=9.644012e-01	1.153570e-05	1.349770e-02	2.075695e-02	1.390448e-01	1.718290e-01	1.980254e-01	9.013429e-03
ε=9.551883e-01	1.330167e-05	1.528041e-02	2.311502e-02	1.463133e-01	1.897151e-01	2.322801e-01	8.593885e-03
ε=9.507303e-01	1.248727e-05	1.437002e-02	2.179182e-02	1.406837e-01	1.815414e-01	2.208686e-01	8.510533e-03
ε=9.463220e-01	1.135130e-05	1.304557e-02	1.977660e-02	1.285689e-01	1.651975e-01	1.994036e-01	8.345323e-03
ε=9.419632e-01	1.001345e-05	1.147873e-02	1.737454e-02	1.124174e-01	1.438914e-01	1.723368e-01	8.276617e-03

Table 3: 2048X2048-PBAPINV-RMSE

PBAPINV AER	Active Line	Line 2	Line 3	Line 15	Line 20	Line 25	All
ε=9.908943e-01	4.665669e-03	4.748595e-01	5.434199e-01	8.789037e-01	1.026704e+00	1.098721e+00	6.320885e-01
ε=9.816818e-01	3.752458e-03	3.759406e-01	4.293547e-01	6.322776e-01	6.620473e-01	7.133118e-01	5.780999e-01
ε=9.728508e-01	3.438104e-03	3.420509e-01	3.892145e-01	6.029309e-01	6.063024e-01	6.132938e-01	5.316651e-01
ε=9.644012e-01	2.968350e-03	2.940233e-01	3.335979e-01	5.250916e-01	5.362587e-01	5.451274e-01	4.998827e-01
ε=9.551883e-01	3.154733e-03	3.081100e-01	3.456689e-01	5.348186e-01	5.660567e-01	6.026896e-01	4.561597e-01
ε=9.507303e-01	3.026716e-03	2.957211e-01	3.325650e-01	5.237899e-01	5.565792e-01	5.906181e-01	4.539446e-01
ε=9.463220e-01	2.874477e-03	2.808875e-01	3.160333e-01	5.022039e-01	5.302273e-01	5.605201e-01	4.493597e-01
ε=9.419632e-01	2.699713e-03	2.635634e-01	2.961726e-01	4.697004e-01	4.958046e-01	5.212302e-01	4.414794e-01

Table 4: 2048X2048-PBAPINV-AER

FMAPINV RMSE	Active Line	Line 2	Line 3	Line 15	Line 20	Line 25	All
ε=9.908943e-01	1.390027e-04	5.567958e-02	7.790279e-02	5.800737e-01	7.321860e-01	9.300583e-01	1.368296e-02
ε=9.816818e-01	6.208195e-05	2.465078e-02	3.448354e-02	2.456430e-01	3.840173e-01	4.350385e-01	9.971555e-03

ε=9.728508e-01	3.727437 e-05	1.677082 e-02	2.480195 e-02	1.589464 e-01	2.151520 e-01	2.860479 e-01	9.571917 e-03
ε=9.644012e-01	2.567091 e-05	1.341187 e-02	2.083310 e-02	1.414948 e-01	1.756095 e-01	2.168678 e-01	9.675919 e-03
ε=9.551883e-01	3.408929 e-05	1.899437 e-02	2.580210 e-02	1.316269 e-01	1.723002 e-01	2.043942 e-01	8.765922 e-03
ε=9.507303e-01	3.153978 e-05	1.722643 e-02	2.325826 e-02	1.228047 e-01	1.634918 e-01	1.976523 e-01	8.699042 e-03
ε=9.463220e-01	2.947895 e-05	1.611989 e-02	2.168835 e-02	1.146350 e-01	1.549165 e-01	1.900467 e-01	8.641071 e-03
ε=9.419632e-01	2.779084 e-05	1.537354 e-02	2.068458 e-02	1.104444 e-01	1.472017 e-01	1.830657 e-01	8.592041 e-03

Table 5: 2048X2048-FMAPINV-RMSE

FMAPINV AER	Active Line	Line 2	Line 3	Line 15	Line 20	Line 25	All
ε=9.908943e-01	8.135088 e-03	5.622129 e-01	6.138007 e-01	1.076745 e+00	1.125677 e+00	1.228295 e+00	6.259280 e-01
ε=9.816818e-01	6.105304 e-03	4.048609 e-01	4.328292 e-01	7.059326 e-01	8.232973 e-01	8.497204 e-01	5.082938 e-01
ε=9.728508e-01	4.957271 e-03	3.241334 e-01	3.556107 e-01	5.642984 e-01	6.124922 e-01	6.893882 e-01	4.835810 e-01
ε=9.644012e-01	4.207687 e-03	2.815072 e-01	3.176883 e-01	5.430052 e-01	5.624433 e-01	6.038944 e-01	4.810140 e-01
ε=9.551883e-01	4.012481 e-03	3.194234 e-01	3.492766 e-01	5.214196 e-01	5.541736 e-01	5.775397 e-01	4.853048 e-01
ε=9.507303e-01	3.918986 e-03	3.092119 e-01	3.369382 e-01	5.037337 e-01	5.393295 e-01	5.671982 e-01	4.817028 e-01
ε=9.463220e-01	3.835425 e-03	3.019792 e-01	3.280559 e-01	4.840859 e-01	5.227518 e-01	5.541190 e-01	4.780100 e-01
ε=9.419632e-01	3.760577 e-03	2.964306 e-01	3.216818 e-01	4.731816 e-01	5.054494 e-01	5.402189 e-01	4.743792 e-01

Table 6: 2048X2048-FMAPINV-AER

AINV	Active Line	Line 2	Line 3	Line 15	Line 20	Line 25	All
RMSE	3.108489 e-04	1.935032 e-01	3.457808 e-01	8.494478 e-01	1.004666 e+00	1.283734 e+00	1.616782 e-02
AER	9.191743 e-03	7.525569 e-01	8.721943 e-01	9.987274 e-01	1.001242 e+00	1.207149 e+00	6.135792 e-01

Table 7: 2048X2048-AINV-Numerical Results

In the second experiment, we consider a larger inductance matrix of size 8192×8192. In this case, in order to preserve acceptable approximation accuracy, especially on the far "victim" conductors, a larger value of k when determining the sparsity structure of K should be considered. This fact leads to the inversion of sub-matrices of greater size and to the solution of more linear systems in the K −method and probing technique respectively. However, as we will see in the next section, the speedup achieved, compared to the full inversion of the inductance matrix, using the probing technique remains relatively constant and independent of the size of the inductance matrix for a fixed sparsity ratio ε. In contrast, the simulation time required when applying the K-method is strongly correlated to the size of sub-matrices involved rather than the sparisity ratio ε. Consequently, when a large value of k is to be used so as to provide accuracy comparable to what is obtained when the exact reluctance matrix K is used, K-method can become highly time demanding.

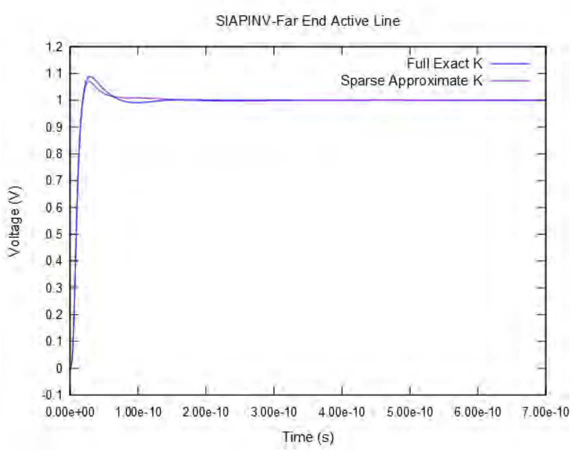

Figure 5.25: 8192X8192-SIAPINV- Voltage Response at far end of Active Line

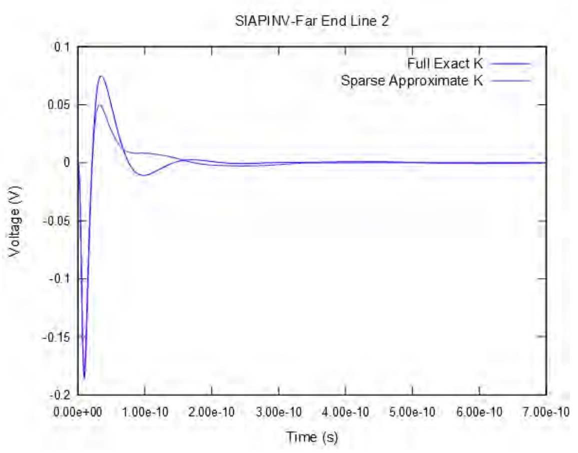

Figure 5.26: 8192X8192-SIAPINV- Voltage Response at far end of Line 2

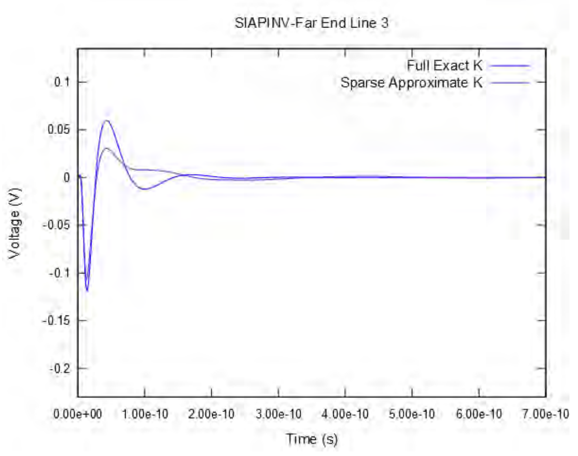

Figure 5.27: 8192X8192-SIAPINV- Voltage Response at far end of Line 3

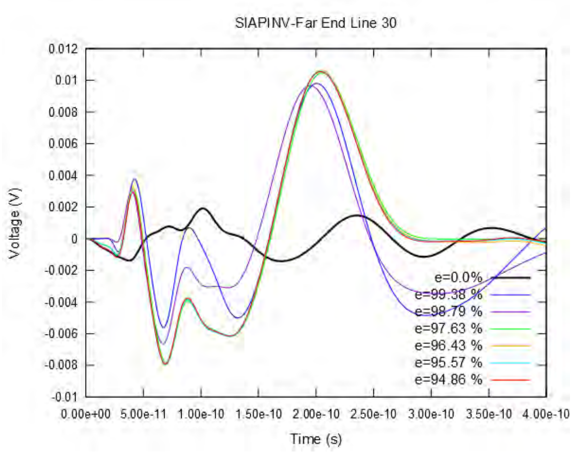

Figure 5.28: 8192X8192-SIAPINV- Voltage Response at far end of Line 30

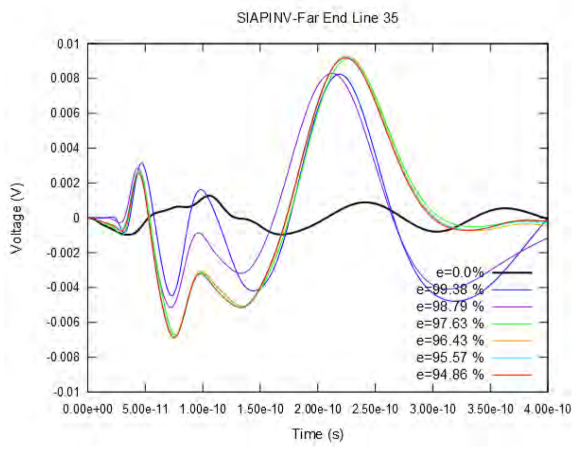

Figure 5.29: 8192X8192-SIAPINV- Voltage Response at far end of Line 35

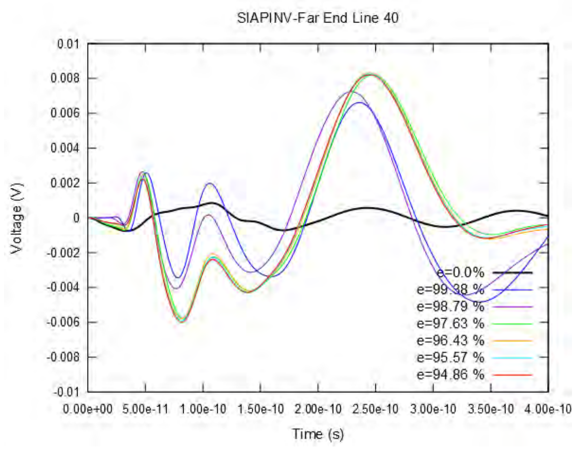

Figure 5.30: 8192X8192-SIAPINV- Voltage Response at far end of Line 40

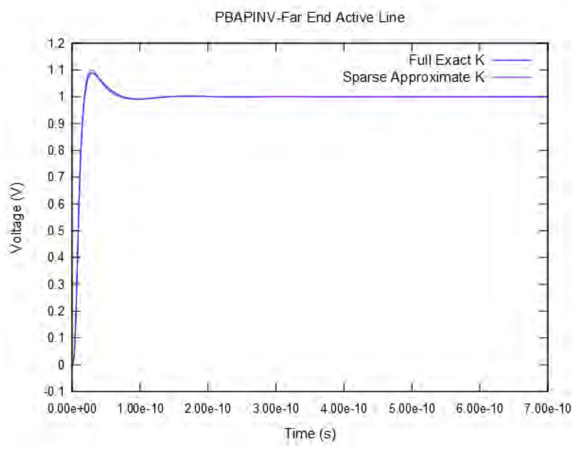

Figure 5.31: 8192X8192-PBAPINV- Voltage Response at far end of Active Line

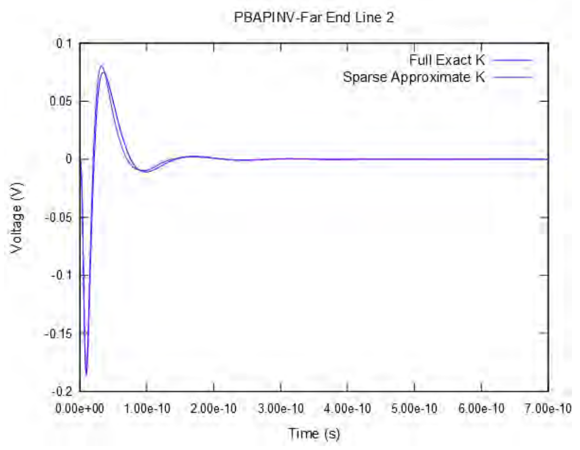

Figure 5.32: 8192X8192-PBAPINV- Voltage Response at far end of Line 2

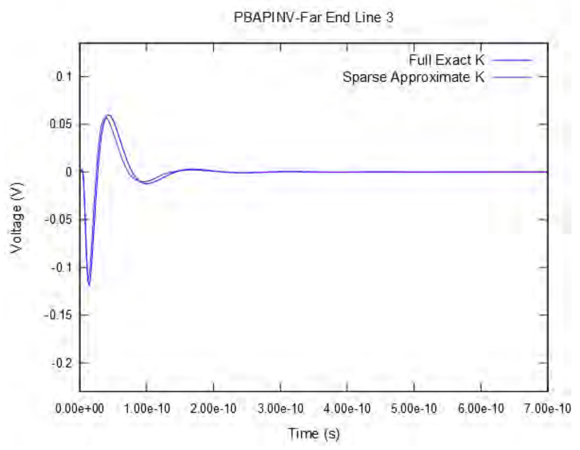

Figure 5.33: 8192X8192-PBAPINV- Voltage Response at far end of Line 3

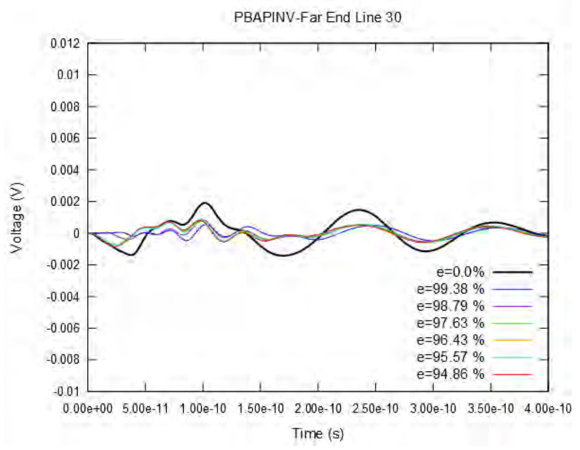

Figure 5.34: 8192X8192-PBAPINV- Voltage Response at far end of Line 30

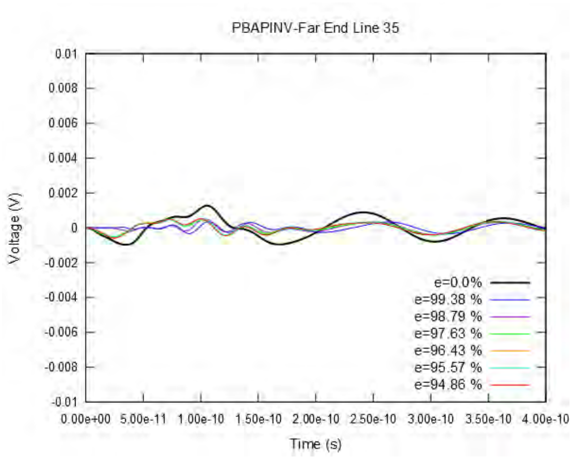

Figure 5.35: 8192X8192-PBAPINV- Voltage Response at far end of Line 35

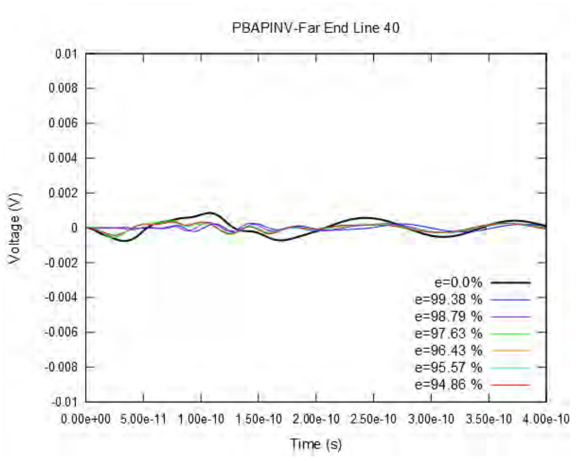

Figure 5.36: 8192X8192-PBAPINV- Voltage Response at far end of Line 40

5.2 Runtime Comparison

Tables 8-11 show runtime information of different sparse approximate inverse methods. Furthermore, we should mention that all the approximate inverse methods except for the AINV demand roughly the same time for the transient analysis as they enjoy the same sparsity into the reluctance matrix.

	ε= 9.908943e-01	ε= 9.816818e-01	ε= 9.728508e-01	ε= 9.644012e-01	ε= 9.551883e-01	ε= 9.507303e-01	ε= 9.463220e-01	ε= 9.419632e-01	ε= 8.410745e-01	ε= 0.000000e-00
SIAPINV	0.090000	0.590000	1.800000	4.130000	7.680000	10.230000	13.050000	16.540000	-	-
PBAPINV	8.080000	8.270000	8.680000	8.920000	9.710000	10.090000	10.140000	10.660000	-	-
FMAPINV	10.490000	-	-	-	-	-	-	-	-	-
AINV	-	-	-	-	-	-	-	-	1.710000	
FULL INV	-	-	-	-	-	-	-	-	-	40.720000

Table 8: 2048X2048- Run time usage for the construction of sparse approximate reluctance matrix $\widetilde{\boldsymbol{K}}$ using different sparse approximate inverse methods

	ε= 9.908943e-01	ε= 9.816818e-01	ε= 9.728508e-01	ε= 9.644012e-01	ε= 9.551883e-01	ε= 9.507303e-01	ε= 9.463220e-01	ε= 9.419632e-01	ε= 0.000000e-00
Transient Analysis Time	3.260000	5.910000	7.220000	8.290000	19.610000	21.940000	22.190000	22.500000	125.230000

Table 9: 2048X2048- Run time usage for Transient Analysis for various sparsity ratios

	ε= 9.937570e-01	ε= 9.879134e-01	ε= 9.762595e-01	ε= 9.642504e-01	ε= 9.556999e-01	ε= 9.485752e-01	ε= 0.000000e-00
SIAPINV	5.670000	63.740000	277.380000	922.140000	1734.230000	-	-
PBAPINV	507.860000	530.800000	565.680000	632.670000	690.560000	718.940000	-
FULL INV	-	-	-	-	-	-	2489.460000

Table 10: 8192X8192- Run time usage for the construction of sparse approximate reluctance matrix $\widetilde{\boldsymbol{K}}$ using different sparse approximate inverse methods

	ε= 9.937570e-01	ε= 9.879134e-01	ε= 9.762595e-01	ε= 9.642504e-01	ε= 9.556999e-01	ε= 9.485752e-01	ε= 0.000000e-00
Transient Analysis Time	82.700000	156.100000	527.910000	924.670000	1051.170000	1185.150000	5461.540000

Table 11: 8192X8192- Transient Analysis Time for various sparsity ratios

6

Conclusions and Future Research

First, we conclude that the use of a sparse reluctance matrix offers increased computational saving in the transient analysis while the sparse approximate inversion methods achieve significant speedup, compared to the full inversion of a matrix, with acceptable approximation accuracy. As already mentioned, the sparse approximate inversion via small inversions is extremely fast when the size of the sub-matrix corresponding to each column is small and the sparse approximate inverse exhibits high sparsity ratio. However, as the number of entries of a column to be captured becomes greater, this approach becomes expensive and inaccurate. This method can be applied to the inductance matrix L to obtain the reluctance matrix $K = L^{-1}$, either at the extraction level (K-method) or at the simulation level. Moreover, there is a trade-off between coupling-window size, or equivalently the sparsity ratio of the sparse reluctance matrix $\widetilde{K}$, and accuracy. As a consequence, when a less sparse reluctance matrix $\widetilde{K}$ should be approximated, other approximate inverse techniques can be applied to obtain an accurate approximation of the K matrix in reasonable time. The approximate inversion, via the minimization of $\parallel L\widetilde{K} - I \parallel_F$, yields satisfactory numerical results while the SPAI algorithm [2] can be used, so that an a priori sparsity pattern is not required. However, it becomes expensive when it is applied directly to the dense inductance matrix L. One solution to this problem would be to sparsify the inductance matrix L as well, which is expected to yield a less accurate approximation. The AINV algorithm is adequately fast and does not require an a priori sparsity pattern. However, since the inductance matrix is not diagonally dominant, there is no guarantee that a sparse and accurate approximate inverse will be produced. The PBAPINV offers high degree of accuracy and a satisfactory speedup compared to the full inversion of the inductance matrix.

All the presented methods are driven by the sparsification of K, owning to better simulation accuracy. However, the stability of these approximate sparsification techniques has not been established since the $\widetilde{K}$ may not be positive definite. As a consequence, further SPD-remedy techniques should be applied to the sparse $\widetilde{K}$ in order for stable circuit simulation to be ensured. The development of methods for generating stable reluctance matrices, which can be applied to irregular geometry cases and incur small errors, constitutes an open research issue.

7

Appendix

7.1 Implementation Code

In this appendix, we provide some selected code segments, written in C, of the simulation code developed in this thesis. The CSPARSE (a Concise Sparse Matrix Package in C) library was used in order to handle sparse matrix structures. We remind that a sparse matrix can be stored either in triplet form, which lists all the nonzero entries in arbitrary order or in compressed-column form. In both cases, the matrix can be represented by three vectors p, i and x. In triplet form, vector p contains column indices, vector i contains row indices and vector x contains the value of the corresponding entry. In compressed column form of a $m \times n$ matrix, row indices of entries in column j are stored in $i[p[j]]$ through $i[p[j+1]-1]$, the corresponding numerical values are stored in the same locations in x. Vector p has size $n+1$, $p[0]=0$, the value of $p[j]$ is such that $p[j+1]-p[j]$ is the number of nonzero entries in column j and $p[n]$ holds the number of nonzero entries in the matrix.

Code snippets 1-4 implement ordinary matrix operations where the one matrix operand is sparse and the other is dense so as to form the linear system involved into the simulation. We also offer the implementation of the partial matrix inversion using the probing technique as described in section 3.4. Specifically, Code Snippet 5 implements the Algorithm 2 where the corresponding graph is returned as a sparse matrix in compressed-column form. Code Snippet 6 implements the Algorithm 3 which colors the graph in order to construct the probing matrix while Code Snippet 7 implements Algorithm 4. Code Snippet 8 contains the interface of a library for the solution of dense linear systems. Code Snippets 9-12 contain the implementation for dynamic sparse matrices. Finally,

Code Snippet 13 implements the incomplete version of Algorithm 1 (AINV algorithm). For the complete simulation code, the implementation of the other approximate inverse methods or the inductance matrices, please contact me at ifaposto@inf.uth.gr or at ifiaposto@gmail.com.

```
/**
 *  Function that scales a sparse matrix, multiplies by a vector, then adds another vector.
 *  @param A the sparse matrix.
 *  @param alpha the scaling factor of A.
 *  @param v the first vector.
 *  @param u the second vector.
 *  @param w the vector of the result,w=alpha*A*v+beta*u.
 *  @return on success 1 on error 0
 */
int ssmxvpsv(cs *A,double alpha,double *v,double *u,double beta,double *w);

int ssmxvpsv(cs *A,double alpha,double *v,double *u,double beta,double *w)
{
		int j,p;
		if (!CS_CSC(A)||!v||!w||!u)
			return 0;
		int n=A->n;
		int m=A->m;
		int *Ap=A->p;
		int *Ai=A->i;
		double *Ax=A->x;



		for(j=0;j<m;j++)
			w[j]=beta*u[j];

		for(j=0;j<n;j++)
			for(p=Ap[j];p<Ap[j+1];p++)
				w[Ai[p]]+=Ax[p]*alpha*v[j];

		return 1;

}
```

Code Snippet 1

```
/**
 * Function that forms the transpose of a matrix.
 * @param A the matrix.
 * @param m the number of rows of A.
 * @param n the number of columns of A.
 * @return the transpose of A or NULL on error.
 */
double **trans (double **A,int m,int n);
/**
 *  Function that scales a sparse matrix, multiplies by a vector, then adds another vector.
 *  @param A the sparse matrix.
 *  @param alpha the scaling factor of A.
 *  @param v,u the vectors.
 *  @param w the vector of the result,w=alpha*A*v+u.
 */
void ssmxvpv(cs *A,double alpha,double *v,double *u,double *w);
/**
 *  Function that multiplies a sparse matrix by a dense matrix, and scales by a constant.
 *  @param A the sparse multiplicand matrix.
 *  @param B the dense multiplier matrix.
 *  @param p the number of columns of B.
 *  @param alpha the scaling factor .
 *  @return the matrix of the result alpha*A*B or NULL on error.
 */
double **smxdmxs (cs *A,double **B,int p,double alpha);

double **smxdmxs (cs *A,double **B,int p,double alpha)

{
	int i,j;
	if(! CS_CSC(A)||!B)
		return NULL;
	int m=A->m;
	int n=A->n;

	double **C=(double **)malloc(sizeof(double *)*m);
	if(!C)
		return NULL;
	for(i=0;i<m;i++){
		C[i]=(double *)malloc(sizeof(double)*p);
		if(!C[i])
			return NULL;
	}

	double *v=(double *)malloc(sizeof(double)*m);
	double **Bt=trans(B,n,p);
	if(!Bt)
		return NULL;

	for(i=0;i<p;i++){
		ssmxvpv(A,alpha,Bt[i],NULL,v);
		for(j=0;j<m;j++)
			C[j][i]=v[j];

	}
	free(v);
	for(i=0;i<p;i++)
		free(Bt[i]);
	free(Bt);
	return C;
}
```

Code Snippet 2

```
/**
 *  Function that multiplies a dense matrix by a sparse matrix, and scales by a constant.
 *  @param A the dense multiplicand matrix.
 *  @param m the number of rows of A.
 *  @param B the sparse multiplier matrix.
 *  @param alpha the scaling factor .
 *  @return the matrix of the result alpha*A*B or NULL on error.
 */
double ** dmxsmxs (double **A,int m,cs *B,double alpha);

double ** dmxsmxs (double **A,int m,cs *B,double alpha)
{
	int p,i,j;
	if(!CS_CSC(B)||!A)
		return NULL;
	int n=B->n;
	int *Bi=B->i;
	int *Bp=B->p;
	double *Bx=B->x;

	double **C=(double **)malloc(sizeof(double *)*m);
	if(!C)
		return NULL;
	for(i=0;i<m;i++){
		C[i]=(double *)calloc(n,sizeof(double));
		if(!C[i]) return NULL;
	}

	for(p=0;p<m;p++)
		for(i=0;i<n;i++)
			for(j=0;j<(Bp[i+1]-Bp[i]);j++)
				C[p][i]+=alpha*A[p][Bi[Bp[i]+j]]*Bx[Bp[i]+j];

	return C;
}
```

Code Snippet 3

```c
/**
 * Function that adds a sparse matrix scaled by a constant with a dense matrix scaled by a
constant.
 * @param A tha sparse matrix.
 * @param B the dense matrix.
 * @param alpha the scaling factor of A.
 * @param beta the scaling factor of B.
 * @param C the matrix of the result C=alpha*A+beta*B.
 * @return 1 if successful and 0 in case of error.
 */
int ssmpsdm(cs *A,double alpha,double**B,double beta,double **C);

int ssmpsdm(cs *A,double alpha,double**B,double beta,double **C)
{
	int i,j;

	if(!CS_CSC(A)||!B||!C)
		return 0;

	int n=A->n;
	int m=A->m;
	int *Ai=A->i;
	int *Ap=A->p;
	double *Ax=A->x;

	for(i=0;i<m;i++)
		for(j=0;j<n;j++)
			C[i][j]=beta*B[i][j];


	for(i=0;i<n;i++){

		for(j=0;j<(Ap[i+1]-Ap[i]);j++)
			C[Ai[Ap[i]+j]][i]+=Ax[Ap[i]+j]*alpha;

	}

	return 1;

}
```

Code Snippet 4

```c
/** Function that computes the modified adjacency graph of a sparse and square matrix.
 * @param csp1 the elements of the inverse to be approximated in compressed-row form.
 * @param csp2 the sparisity pattern of the inverse in compressed-row form
 * @return the adjacency graph in compressed-column form.
 */
extern cs *magcstr(cs *csp1,cs *csp2);

cs *magcstr(cs *csp1,cs *csp2){

	int n1=csp1->n;
	int m1=csp1->m;
	int *p1=csp1->p;
	int *i1=csp1->i;

	int *p2=csp2->p;
	int *i2=csp2->i;

	int *p=(int *)malloc(n1*sizeof(int));
	int *i=(int *)malloc(n1*sizeof(int));

	char **g=(char **)malloc(sizeof(char *)*n1);

	int nzmax;
	int k,l,j;

	for(k=0;k<n1;k++)
		g[k]=(char *)calloc(n1,sizeof(char));

	nzmax=0;
	for(k=0;k<n1;k++){

		 /* preserve element (k,i1[j]) */
		for(j=p1[k];j<p1[k+1];j++){

			for(l=p2[k];l<p2[k+1];l++){

				/* add the edge (i2[l],i1[j]) */
				if(i2[l]!=i1[j]){

					if (!g[i2[l]][i1[j]]){
						g[i2[l]][i1[j]]=1;
						p[nzmax]=i2[l];
						i[nzmax++]=i1[j];
						if(!(nzmax % n1)){

							p=(int *)realloc(p,(nzmax+n1)*sizeof(int));
							i=(int *)realloc(i,(nzmax+n1)*sizeof(int));
						}

					}

					if(!g[i1[j]][i2[l]]){
						g[i1[j]][i2[l]]=1;
						p[nzmax]=i1[j];
						i[nzmax++]=i2[l];
						if(!(nzmax % n1)){

							p=(int *)realloc(p,(nzmax+n1)*sizeof(int));
							i=(int *)realloc(i,(nzmax+n1)*sizeof(int));
						}
					}
```

```
                              }
                      }

              }
      }

      for(k=0;k<n1;k++)
              free(g[k]);
      free(g);

      p=(int *)realloc(p,nzmax*sizeof(int));
      i=(int *)realloc(i,nzmax*sizeof(int));

      cs *t=cs_spalloc(m1,n1,nzmax,0,1);
      memmove(t->p,p,nzmax*sizeof(int));
      free(p);
      memmove(t->i,i,nzmax*sizeof(int));
      free(i);
      t->nz=nzmax;
      cs *cs_g=cs_compress(t);
      cs_spfree(t);

      return(cs_g);

}
```

Code Snippet 5

```
/** Function that colours a graph.
 * @param g the modified adjaceny graph in  compressed-column form.
 * @param c the colour of each node.
 * @return the number of colours used.
 */
extern int gcolor(cs *g,int **c);

int gcolor(cs *g,int **c){
      int n=g->n;
      int *i=g->i;
      int *p=g->p;

      int j,k,l;
      int cn=0;

      *c=(int *)calloc(n,sizeof(int));

      for(j=0;j<n;j++){
              for(k=1;;k++){

                        for(l=p[j];l<p[j+1] && !((*c)[i[l]]==k);l++);

                        if(l==p[j+1]){

                                (*c)[j]=k;
                                cn=(cn<k)? k:cn;
                                break;
                        }
                }
        }
        return cn;
}
```

Code Snippet 6

```
/**
 * Function that approximates entries of the inverse of a matrix,using probing technique.
 * @param Î‘ the matrix.
 * @param csp1 the elements of A^-1 to be approximated in compressed row form.
 * @param csp2 the sparisity pattern of A^-1 in compressed row form.
 * @param options the linear system solution method to be used.
 * @return fills the entries of cps1(the field x of csp1) in compressed column form.
 */
extern void pbapinv(double **A,cs *csp1,cs *csp2,lssm_options *options);

void pbapinv(double **A,cs *csp1,cs *csp2,lssm_options *options){

    cs *g;
       int *c;
       int cn;
       int *Vs;
       double **Xs;
       double **vs;
       int k,j;


       int n=csp1->n;

       /* construct the modified adjacency graph of the inverse */
       g=magcstr(csp1,csp2);

        /* color the modified adjacency graph of the inverse */
       cn=gcolor(g,&c);
       cs_spfree(g);

        /* construct Xs and probing vectors */
        Xs=(double **)malloc(sizeof(double *)*cn);
        vs=(double **)malloc(sizeof(double *)*cn);

        for(j=0;j<cn;j++)
              vs[j]=(double *)calloc(n,sizeof(double));

        /* create the probing vectors */
        Vs=(int *)malloc(sizeof(int)*n);
        for(j=0;j<n;j++){

              Vs[j]=c[j]-1;
              vs[c[j]-1][j]=1;
        }
        free(c);
        /* solve the cn linear systems to create Xs */
       if(options->lssm==ITERATIVE)
              options->method.iter->x0= ( double* ) calloc (n, sizeof(double) );

       ls_state *ls=ls_init(A,options,n);

       for(j=0;j<cn;j++){
              Xs[j] =ls_solve(vs[j],ls,options);
              free(vs[j]);
       }

       if(options->lssm==ITERATIVE)
              free(options->method.iter->x0);
       ls_term(ls);
       free(vs);

       /* extract the elements */
       int *p=csp1->p;
       int *i=csp1->i;
       double *x=csp1->x;

       for(k=0;k<n;k++)
              /* extract element (k,i[j]) */
              for(j=p[k];j<p[k+1];j++)
                     x[j]=Xs[Vs[i[j]]][k];



       for(j=0;j<cn;j++)
              free(Xs[j]);

       free(Xs);
       free(Vs);


}
```

Code Snippet 7

```
typedef struct _iter_options    /*options of the iterative method */
{
       unsigned char iterm;         /* the iterative method to be used */
       double ITOL;                 /* the convergence tolerance*/
       double *x0;                  /* the initial guess */
       preconstruct conp;         /*the function to be used in order to construct the
preconditioner */
       cs_preconstruct cs_conp;   /*the function to be used in order to construct the
preconditioner for sparse matrix */
       presolve solp;             /*the function to be used in order to solve a linear system
                                    with the preconditioner as coefficient matrix.*/
       pretsolve solpt;          /*the function to be used in order to solve a linear system
                                    with trans(M) as coefficient matrix.*/
}
iter_options;
```

```
typedef struct _dir_options  /*options of the direct method */
{
       unsigned char dirm;    /* the direct method to be used */
       double ITOL;           /* the convergence tolerance*/


}
dir_options;

typedef struct _lssm_options /*options of the linear system's solution  method */
{
    unsigned char  lssm;         /*the linear system solution method to be used*/
    union
    {
       iter_options *iter;  /*the options of the iterative method to be used*/
        dir_options *dir;     /*the options of the iterative method to be used*/
       }
       method;
}
lssm_options;

typedef struct _ls_state /* the state of a linear system with dense coefficient matrix */
{
       double **A;      /* the coefficient matrix */
       double **L;      /* the decomposition of A */
       unsigned int *P; /* the permutation matrix of lu decomposition of A */
       double *M;       /* the preconditioner matrix of A */
       int n;           /* the dimension of the linear system */
}
ls_state;

/**
 *  Function that initializes the linear system with dense coefficient matrix A.
 *  According to the options either computes the lu factorization or the
 *  preconditioner matrix of A.
 *  @param A the coefficient matrix.
 *  @param options the method to be used and further options of that method.
 *  @param n the size of the system.
 *  @return the state of the linear system.
 */
extern ls_state *ls_init(double **A,lssm_options *options,int n);

/**
 *  Function that solves a dense linear system specified by state
 *  with rhs b,according to options.
 *  @param b the rhs.
 *  @param state the initialied coeffcient matrix.
 *  @param options the method to be used and further options of that method..
 *  @return the solution.
 */
extern double * ls_solve(double *b,ls_state *state,lssm_options *options);

/**
 *  Function that destroys the state of a dense linear system.
 *  @param state the state of the linear system.
 */
extern void ls_term(ls_state *state);
```

Code Snippet 8

```
/********************************************************************************
 *                                                                              *
 *                       DATA STRUCTURES DEFINITIONS                            *
 *                                                                              *
 ********************************************************************************/


typedef struct csd_row
{
       double x;   /* numerical value of the element */
       int k;      /* row indice */
       struct csd_row *nxt;  /* next row */
}csdr;



typedef struct csd_col
{
       int nzp;  /* number of elements of the column */
       csdr *i;  /* the rows of the column */

} csdc;



typedef struct csd_sparse /* dynamic sparse matrix */
{
       int m;   /* number of rows */
       int n;   /* number of columns */
       csdc *p; /* the columns of the matrix */
       int nz;  /* # of nonzero elements */
} csd;
```

Code Snippet 9

```
/**
 *  Function for allocating the appropriate memory space for a dynamic sparse matrix.
 *  @param m Number of rows.
 *  @param n Number of columns.
 *  @return Pointer to the struct describing the matrix in case of success and NULL otherwise.
 */
extern csd *csd_spalloc(int m, int n);

csd *csd_spalloc(int m, int n) {

       csd *A=(csd *)malloc(sizeof(csd));

       if(!A)
              return NULL;
       A->m=m;
       A->n=n;
       A->nz=0;

       A->p=(csdc *)malloc(sizeof(csdc)*n);
       if(!A->p)
              return NULL;
       int i;
       for(i=0;i<n;i++){
              A->p[i].nzp=0;
              A->p[i].i=NULL;
       }

       return A;
}
```

Code Snippet 10

```
/**
 *  Function for deallocating the allocated memory space for a dynamic sparse matrix .
 *  @param A Pointer to the matrix.
 *  @return NULL.
 */
extern csd *csd_spfree(csd *A);

csd *csd_spfree(csd *A){


       if(!A || !A->p)
              return NULL;
       int n=A->n;

       int i;

       for(i=0;i<n;i++){
           csdr *r;
              while(A->p[i].i){
                     r=A->p[i].i;
                     A->p[i].i=r->nxt;
                     free(r);
              }
       }
        return NULL;
}
```

Code Snippet 11

```
/**
 * Function that converts a dynamic sparse matrix to a static sparse matrix in compressed-
column form.
 * @param A Pointer to the dynamic matrix.
 * @return Pointer to the static sparse matrix in case of success and NULL otherwise.
 */
extern cs *csd_static(csd *A);

cs *csd_static(csd *A){

        if(!A)
              return NULL;
        int m=A->m;
        int n=A->n;

        cs *B=cs_spalloc(m,n,A->nz,1,0);
        int *Bp=B->p;
        int *Bi=B->i;
        double *Bx=B->x;

        csdc *Ap=A->p;
        if(!Ap)
              return NULL;

        int i;
        int nz=0;
        for(i=0;i<n;i++){
               Bp[i]=nz;

              csdr *r;
              for(r=Ap[i].i;r;r=r->nxt){
                     Bx[nz]=r->x;
                     Bi[nz++]=r->k;
              }
        }
        Bp[n]=nz;

        return B;
}
```

Code Snippet 12

```
cs *ainv(double ** A,int n,double tol,double lamda){

       int i,j;

       /* D=diag(p1,p2,..,pn ) */
       cs *D=cs_spalloc(n,n,n,1,0);
       int *Dp=D->p;
       int *Di=D->i;
       double *Dx=D->x;
       for(i=0;i<n;i++){
              Dp[i]=i;
              Di[i]=i;
       }
       Dp[n]=n;
       memset(Dx,0,sizeof(double)*n);

       /* Z=[z1,z2, ,zn] */
       csd *Z=csd_spalloc(n,n);
       csdc *Zp=Z->p;
       /* zi=ei i=0(1)n-1 */
       for(i=0;i<n;i++){
              Z->nz++;
              Zp[i].nzp++;
              Zp[i].i=(csdr *)malloc(sizeof(csdr));
              Zp[i].i->x=1.0;
              Zp[i].i->k=i;
              Zp[i].i->nxt=NULL;
       }

       for(i=0;i<n;i++){

              /* update pivots */
              for(j=i;j<n;j++){

                     /* pj=(Ai,zj) j=i(1)n */
                     csdr *r;
                     Dx[j]=0.0;
                     for(r=Zp[j].i;r;r=r->nxt)
                            Dx[j]+=r->x * A[i][r->k];
              }

              if(Dx[i]<=0){
                     /* ainv breaks down */
                     /* sigma=max{pk},k=i(1)n-1 */
                     double sigma=Dx[i];
                     int l;
                     for(l=i+1;l<n-1;l++)
                            sigma=(sigma < Dx[l]) ? Dx[l]: sigma;

                     /* theta=||zi||oo */
                     double theta=Zp[i].i ? fabs(Zp[i].i->x) :0.0;
                     csdr *ri;
                     for(ri=Zp[i].i;ri;ri=ri->nxt)
                            theta=(theta < fabs(ri->x)) ? fabs(ri->x) :theta;
                     double zeta=lamda*sigma*theta;
```

```
                /* pi=max(sqrt(eps),zeta) */
                Dx[i]=(SQRT_EPS>zeta) ? SQRT_EPS :zeta;
        }

        /* update conjugate directions zj */
        for(j=i+1;j<n;j++){

                /* zj=zj-(pj/pi)*zi j=i+1(1)n*/
                double p=Dx[j]/Dx[i];
                csdr *ri;
                for(ri=Zp[i].i;ri;ri=ri->nxt){

                        int k=ri->k;
                        csdr *rjp,*rj;
                        for(rjp=NULL,rj=Zp[j].i;rj && rj->k<k;rjp=rj,rj=rj->nxt);
                        double z=-p*ri->x+((rj && rj->k==k) ? rj-> x: 0.0);

                        if(rj && rj->k==k){

                                /* element (k,j) already exists in Z */
                                if(fabs(z)<=tol){

                                        /* drop element (k,j) */
                                        if(rjp)
                                                rjp->nxt=rj->nxt;
                                        else
                                                Zp[j].i=rj->nxt;
                                        free(rj);
                                        Z->nz--;
                                        Zp[j].nzp--;
                                }
                                else
                                        /* update value */
                                        rj->x=z;
                        }
                        else{
                                /* element (k,j) does not exist in Z */
                                if(fabs(z)>tol){

                                        /* add element (k,j) */
                                        csdr *r=(csdr *)malloc(sizeof(csdr));
                                        r->x=z;
                                        r->k=k;
                                        r->nxt=rj;
                                        if(rjp)
                                                rjp->nxt=r;
                                        else
                                                Zp[j].i=r;
                                        Z->nz++;
                                        Zp[j].nzp++;
                                }
                        }
                }

        }
```

```
	}

	cs *Z_cs=csd_static(Z);
	csd_spfree(Z);
	for(i=0;i<n;i++)
			Dx[i]=1/Dx[i];
	cs *T=cs_multiply(Z_cs,D);
	cs_spfree(D);
	cs *Z_cs_t=cs_transpose(Z_cs,1);
	cs_spfree(Z_cs);
	cs *B=cs_multiply(T,Z_cs_t);
	cs_spfree(T);
	cs_spfree(Z_cs_t);

	return B;
}
```

Code Snippet 13

7.2 Mathematical Proofs

7.2.1

When A is invertible, then $\begin{vmatrix} A & B \\ C & D \end{vmatrix} = |A||D - CA^{-1}B|$.

Proof:

The block matrix can be factored as

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} A & 0 \\ C & I \end{bmatrix} \begin{bmatrix} I & A^{-1}B \\ 0 & D - CA^{-1}B \end{bmatrix}$$

Hence,

$$\begin{vmatrix} A & B \\ C & D \end{vmatrix} = \begin{vmatrix} A & 0 \\ C & I \end{vmatrix} \begin{vmatrix} I & -A^{-1}B \\ 0 & D - CA^{-1}B \end{vmatrix} =$$
$$(|A||I|)(|I||D - CA^{-1}B|) = |A||D - CA^{-1}B|$$

7.2.2

Consider a pair A, B of $n \times n$ matrices, partitioned as $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$ where A_{11}, B_{11} are $k \times k$ matrices. If A_{11}, A_{22} are nonsingular and $B = A^{-1}$ then

$$\begin{aligned} B_{11} &= (A_{11} - A_{12}A_{22}^{-1}A_{21})^{-1} \\ B_{22} &= (A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1} \\ B_{12} &= -A_{11}^{-1}A_{12}(A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1} \\ B_{21} &= -A_{22}^{-1}A_{21}(A_{11} - A_{12}A_{22}^{-1}A_{21})^{-1} \end{aligned}$$

Proof:

Firstly, we remark that the conditions $|A_{11}| \neq 0, |A_{22}| \neq 0$ are sufficient for the nonsingularity of A, but in general, not necessary. In case of a positive definite A, these conditions are also necessary .If $A = B^{-1}$ then

$$AB = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix} = \begin{bmatrix} I_k & O_{k,n-k} \\ O_{n-k,k} & I_{n-k} \end{bmatrix}$$

Equivalently, we need to solve four matrix equations:

$$A_{11}B_{11} + A_{12}B_{21} = I_k \qquad (i)$$
$$A_{11}B_{12} + A_{12}B_{22} = O_{k,n-k} \qquad (ii)$$
$$A_{21}B_{11} + A_{22}B_{21} = O_{n-k,k} \qquad (iii)$$
$$A_{21}B_{12} + A_{22}B_{22} = I_{n-k} \qquad (iv)$$

It follows from (ii) and (iii) that

$$B_{12} = -A_{11}^{-1}A_{12}B_{22} \qquad (v)$$
$$B_{21} = -A_{22}^{-1}A_{21}B_{11} \qquad (vi)$$

So that (i) and (iv) become

$$(A_{11} - A_{12}A_{22}^{-1}A_{21})B_{11} = I_k$$
$$(A_{22} - A_{21}A_{11}^{-1}A_{12})B_{22} = I_{n-k}$$

Hence,

$$B_{11} = (A_{11} - A_{12}A_{22}^{-1}A_{21})^{-1}$$
$$B_{22} = (A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1}$$

Substituting these solutions in (v) and (vi) it follows that

$$B_{12} = -A_{11}^{-1}A_{12}(A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1}$$
$$B_{21} = -A_{22}^{-1}A_{21}(A_{11} - A_{12}A_{22}^{-1}A_{21})^{-1}$$

7.2.3

If matrix $A \in R^{n,n}$ is strictly diagonally dominant and has positive diagonally elements then it is SPD.

Proof:

This is an immediate consequence of the Gersgorin Circle theorem, by which every eigenvalue λ_k of a square matrix A is located in one of the n disks in the complex plane defined by $\left\{ z: |z - a_{ii}| \leq \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij} \right\}, i = 1(1)n$. Obviously, if the matrix is strictly diagonally dominant with positive diagonal elements, then all Gersgorin disks lie entirely in the positive semi-plane and thus all eigenvalues have positive real parts.

Bibliography

[1] E. Chow and Y. Saad, Parallel approximate inverse preconditioners, PPSC 8, 1997.

[2] M. Grote and T.Huckle, Parallel Preconditioning with sparse approximate inverses, SIAM J. Sci. Comput. 18, 1997.

[3] E. Chow, A Priori Sparsity Patterns For Parallel Sparse Approximate Inverse Preoconditioners, SIAM J. Sci. Comput. 21, 2000.

[4] T. Huckle, Approximate sparsity patterns for the inverse of a matrix and preconditioning, Applied Numerical Mathematics 30, 1999.

[5] M. Benzi, L. Giraud and G. Alleon, Sparse Approximate Inverse Preconditioning For Dense Linear Systems In Computational Electromagnetic, Numerical Algorithms 16, 1997.

[6] M. Ganesh and S. C. Hawkins, Sparse approximate inverse preconditioners for electromagnetic surface scattering simulations, ANZIAM J. 49, 2007.

[7] M. Benzi, C. D. Meyer, M. Tuma, Miroslav and T. Uma, A Sparse Approximate Inverse Preconditioner For The Conjugate Gradient Method, SIAM J. Sci. Comput. 17, 1996.

[8] M. Benzi and M. Tuma, A sparse approximate inverse preconditioner for nonsymmetric linear systems, SIAM J.Sci Comput. 19, 1998.

[9] J. Tang and Y. Saad, A Probing Method for Computing the Diagonal of the Matrix Inverse, Numerical Linear Algebra with Applications Vol. 19, 2010.

[10] C. Bekas, E. Kokiopoulou and Y. Saad, An Estimator for the Diagonal of a Matrix, Applied Numerical Mathematics Vol. 57, 2007.

[11] T. Huckle and A. Kallischko, Frobenius Norm Minimization and Probing for Preconditioning, IJCM 84, 2007.

[12] C. Siefert and E. Sturler, Probing Methods for Saddle-Point Problems, Electronic Transactions on Numerical Analysis 22, 2006.

[13] J. Freericks, M. Rigol, T. El-Ghazawi and Y. Saad, Diagonal of inverse of real symmetric and complex diagonal matrices using Lanczos, Notes by Piere Carrier(CSE/I. Minesota), 2010.

[14] Y. Saad and R. Sidje, Rational approximation to the Fermi-Dirac function with applications in Density Functional Theory, Numerical Algorithms Vol. 56, 2009.

[15] L. Lin, J. LU, L. Ying, R. Car and W. E, Fast algorithm for extracting the diagonal of the inverse matrix with application to the electronic structure analysis of metallic systems, Communications in Mathematical Sciences Vol. 7, 2009.

[16] M. Benzi, Preconditioning techniques for large linear systems: A survey, Journal of Computational Physics, 2002.

[17] F. N. Najim, Circuit Simulation, Wiley & Sons Publications, New Jersey, 2010.

[18] M. W. Beattie and L. T. Pileggi, Inductance 101: Modeling and Extraction, Proc. Design Automation Conf., 2001.

[19] K. Gala, D. Blaauw, J. Wang, V. Zolotov and M. Zhao, Inductance 101: Analysis and Design Issues, Proc. Design Automation Conf., 2001.

[20] M. W. Beattie and L. T. Pileggi, Modeling Magnetic Coupling for On-Chip Interconnect, Proc. Design Automation Conf., 2001.

[21] A. Devgan, H. Ji and W. Dai, How to Efficiently Capture On-Chip Inductance Effects: Introducing a New Circuit Element K, Proc. Int. Conf. on Computer Aided Design, 2000.

[22] A. Devgan, H. Ji and W. Dai, KSIM: a Stable and Efficient RKC Simulator for Capturing On-Chip Inductance Effects, Proc. of the ASP-DAC, 2001.

[23] T-H. Chen, C. Luk and C. C-P. Chen, INDUCTWISE: Inductance-Wise Interconnect Simulator and Extractor, Proc. Int. Conf. on Computer Aided Design, 2002.

[24] M. Kamon, M.J. Tsuk and J.K. white, FASTHENRY: A Multipole-Accelerated 3-D Inductance Extraction Program, Proc. Design Automation Con., 1993.

[25] Y. Tanji, T. Watanabe and H. Asai, Generating Stable and Sparse Reluctance/Inductance Matrix under Insufficient Conditions, Proc. of the ASP-DAC, 2008.

[26] G. Zhong, C.-K. Koh and K.Roy, On-chip Interconnect Modeling by Wire Duplication, Proc. Int. Conf. on Computer Aided Design, 2002.

[27] H. Li, V. Balakrishnan and C.-K. Koh, Compact and Stable Modeling of Partial Inductance and Reluctance Matrices, Proc. of the ASP-DAC, 2005.

[28] H. Li, V. Balakrishnan and C.-K. Koh, Stable and Compact Inductance Modeling of 3-D interconnect structures, Proc. Int. Conf. on Computer Aided Design, 2006.

[29] J. Jain, C.-K. Koh and V. Balakrishnana, Fast Simulation of VLSI Interconnects, Proc. Int. Conf. on Computer Aided Design, 2004.

[30] Hui Zheng, B. Krauter, M. Beattie and L. Pileggi, Window-based susceptance models for large-scale RLC circuit analysis, Proc. Design Automation and Test in Europe, 2002.

[31] M. Beattie and L. Pileggi, Efficient inductance Extraction via Windowing, Proc. Design Automation and Test in Europe, 2001.

[32] Hui Zheng, B. Krauter, M. Beattie and L. Pileggi, Window-based susceptance models for large-scale RLC circuit analyses, Proc. Design Automation and Test in Europe, 2002.

[33] Z. He, M. Celik and L. T. Pileggi, SPIE: Sparse partial inductance extraction, Proc. Design Automation Con., 1997.

[34] G. Zhong, C.-K. Koh, V. Balakrishnan and K.Roy, An Adaptive Window-Based Susceptance Extraction and its Efficient Emplementation, DAC, 2003.