Spring 2017

# Application of nearly linear solvers to electric power system computation

Lisa L. Grant



Department: Electrical and Computer Engineering

# APPLICATION OF NEARLY LINEAR SOLVERS TO ELECTRIC POWER SYSTEM COMPUTATION

by

LISA L. GRANT

A DISSERTATION

Presented to the Graduate Faculty of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

in

ELECTRICAL ENGINEERING

2017

Approved by

Dr. Mariesa Crow, Advisor
Dr. Maggie Cheng
Dr. Mehdi Ferdowsi
Dr. Jonathan Kimball
Dr. Xiaoming He

## PUBLICATION DISSERTATION OPTION

This dissertation has been prepared using the Publication Option. Opening and closing chapters have been added for purposes normal to dissertation writing.

Paper I on pages 3 to 16, "Computationally Efficient Solvers for Power System Applications," was published in *IEEE Power and Energy Conference*.

Paper II on pages 17 to 39, "A Chain Method for Preconditioned Iterative Linear Solvers for Power System Matrices," has been accepted for future publication in *IEEE Transactions on Power Systems*.

Paper III on pages 40 to 59, "Utilization of a Chain Linear Solver for Fast Decoupled Power Flow", will be submitted to *IEEE Transactions on Power Systems*.

# ABSTRACT

To meet the future needs of the electric power system, improvements need to be made in the areas of power system algorithms, simulation, and modeling, specifically to achieve a time frame that is useful to industry. If power system time-domain simulations could run in real-time, then system operators would have situational awareness to implement on-line control and avoid cascading failures, significantly improving power system reliability. Several power system applications rely on the solution of a very large linear system. As the demands on power systems continue to grow, there is a greater computational complexity involved in solving these large linear systems within reasonable time.

This project expands on the current work in fast linear solvers, developed for solving symmetric and diagonally dominant linear systems, in order to produce power system specific methods that can be solved in nearly-linear run times. The work explores a new theoretical method that is based on ideas in graph theory and combinatorics. The technique builds a chain of progressively smaller approximate systems with preconditioners based on the system's low stretch spanning tree. The method is compared to traditional linear solvers and shown to reduce the time and iterations required for an accurate solution, especially as the system size increases. A simulation validation is performed, comparing the solution capabilities of the chain method to LU factorization, which is the standard linear solver for power flow. The chain method was successfully demonstrated to produce accurate solutions for power flow simulation on a number of IEEE test cases, and a discussion on how to further improve the method's speed and accuracy is included.

# ACKNOWLEDGMENTS

I would like to thank my wonderful and patient advisor Dr. Crow for guiding me through my M.S. and PhD. degrees. I thoroughly enjoyed my PhD research project, it has challenged me and made me a better researcher, programmer, and engineer. I also would like to acknowledge the financial support of the National Science Foundation under award EECS 1307458 and the Chancellor's Fellowship of Missouri S&T.

I am thankful for Dr. Mehdi Ferdowsi, Dr. Jonathan Kimball, Dr. Maggie Cheng and Dr. Xiaoming He for serving on my advisory committee. I greatly enjoyed the programming and computer science courses taught by Drs. Cheng and He. The dedication of Drs. Ferdowsi and Kimball to the power group meetings provided a valuable resource for research ideas and practice sessions for presentations.

I would also like to thank my fellow lab members, Maigha and Darshit Shah, for their friendship and support. Our conversations on life, religion, and everything else provided the best breaks from seemingly endless coding.

Most importantly, I would like to thank my father Frank for inspiring me to pursue a PhD, my husband Mark for getting me though this journey, my mom Diane for the free 24/7 babysitting, and my daughter Kaydence for giving me a reason to finish.

# TABLE OF CONTENTS

Page

# LIST OF ILLUSTRATIONS

# LIST OF TABLES

# SECTION

# 1. INTRODUCTION

With new technology innovations and the addition of renewables to the electric power grid, there is a greater computational complexity in solving power system simulations within reasonable time. Better algorithms, modeling, and simulation tools are required to ensure a reliable electric infrastructure for the future. The blackouts that have been experienced more frequently in the last decade demonstrate changes in the way the transmission system is being used. Power system operators need better situational awareness to predict and prevent catastrophic contingencies, which can be achieved by developing simulation and monitoring techniques that approach real-time.

There have been several recent advances in computational methods in other fields that if applied to power systems, could make real-time dynamic simulation a reality. This project explores a new fast linear solver for application to electric power system computation. Contributions of Paper I are:

- Demonstrated the theoretical chain algorithm in practice
- Validated the method for use as a general linear solver
- Compared the chain method to a traditional linear solver

Contributions of Paper II are:

- Analyzed the use of the chain method for power system type matrices

- Demonstrated the scalability of the chain technique as a linear solver for power system matrices of increasing size
- Proved the chain method can produce a nearly linear runtime when compared to the traditional LU factorization technique
- Identified potential areas of further study to explore the chain method's capabilities for solving power system specific problems

Contributions of Paper III are:

- Described how to apply the chain linear solver to power flow applications
- Validated the use of the chain method as a linear solver for fast-decoupled load flow
- Analyzed the characteristics of power system load flow matrices that specifically affect the performance of the chain method
- Demonstrated the potential benefits of using the chain method for contingency analysis

**PAPER**

# I. COMPUTATIONALLY EFFICIENT SOLVERS FOR POWER SYSTEM APPLICATIONS

L. L. Grant, M. L. Crow, and M. X. Cheng

Department of Electrical & Computer Engineering

Missouri University of Science and Technology

Rolla, Missouri 65409–0050

Tel: 573–465–0657

Email: llsnpc@mst.edu

## 1. ABSTRACT

If power system time-domain simulations could run in real-time, then system operators would have situational awareness to implement on-line control and avoid cascading failures, significantly improving power system reliability. Many power system assessment tools, such as power flow and short circuit analysis, require very large sparse matrix computations. As power systems continue to grow, there is a greater computational complexity involved in solving these large linear systems within reasonable time. We are expanding the current work in fast linear solvers to develop power system specific method that show potential for accurate solutions in nearly-linear run times.

Keywords: Matrix preconditioning, graph sparsification, linear system solution, power system simulation

## 2. INTRODUCTION

Growing reliance on technology demands the development of better algorithms and simulation methods for a reliable electric infrastructure with increased renewable penetration. Renewables add variability to the power system network, requiring more frequent calculations and adjustments than traditional fuel sources. Power system matrices, such as the one presented in Fig. 1, tend to be very sparse and can contain thousands to millions of variables. For sparse matrix applications, it is desirable to have an algorithm with a run time that is efficient in terms of the number of non-zero variables in the matrix. A sparse matrix that is ordered and conditioned in such a way as to improve its spectral properties, can allow for a simple iterative method to solve even large systems efficiently in a short amount of time [2]. There is interest in specific types of these problems, where the system is symmetric and diagonally dominant (SDD), similar to the form of power systems [3, 4, 5, 6, 7]. A matrix is SDD if the magnitude of the diagonal entry $a_{ii}$ in row $i$ is greater than or equal to the sum of the magnitudes of all other non-diagonal entries in that row as shown in (1).

$$| a_{ii} | \geq \sum_{j \neq i} | a_{ij} | \quad \text{for all} \quad i \tag{1}$$

A recent breakthrough, presented by Koutis et al., allows these systems to be solved in nearly-linear time and is less complicated to implement than previous methods [3, 4, 5]. The method, designed for imaging applications, contains a combinatorial component and uses graph theoretic algorithms. The goal of this project is to explore the new advances in nearly-linear solvers to analyze how they can be applied to and improved upon for the field of electric power systems. The rest of this paper is organized as follows: Section 3

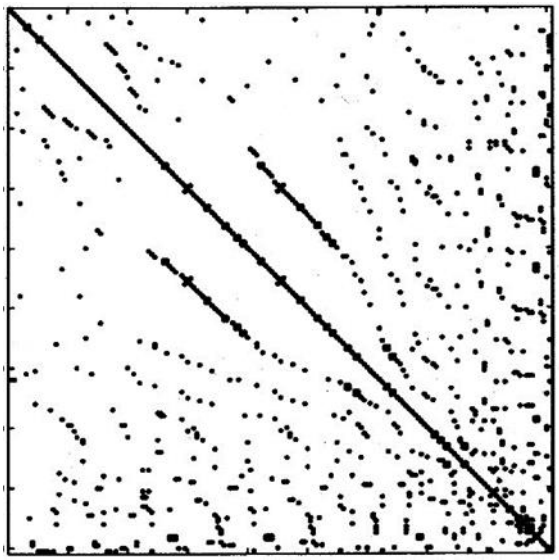

Figure 1. Typical power system matrix structure.

provides background information on the research discoveries leading to the development of this technique, Section 4 includes details on the proposed method, Section 5 presents preliminary results, and the conclusion and future work is discussed in Section 6.

## 3. BACKGROUND

There are several power system analysis tools requiring the solution of a system of linear equations in the form of $Ax = b$, where the vector $x$ represents the unknown state variables, vector $b$ contains the known quantities, and matrix $A$ contains the system component equations and constraints. A typical electric power network problem can have on the order of 25,000 nodes represented in the system matrix $A$. For power system specific applications, the approach to solving these equations is to find a sparse $\tilde{A}$ that provides a very fast solution for $\tilde{A}\tilde{x} = b$ where $\tilde{x}$ can be presented as an initial value for an iterative algorithm to solve the original set of equations $Ax = b$. If $\tilde{x}$ is a good approximation to the

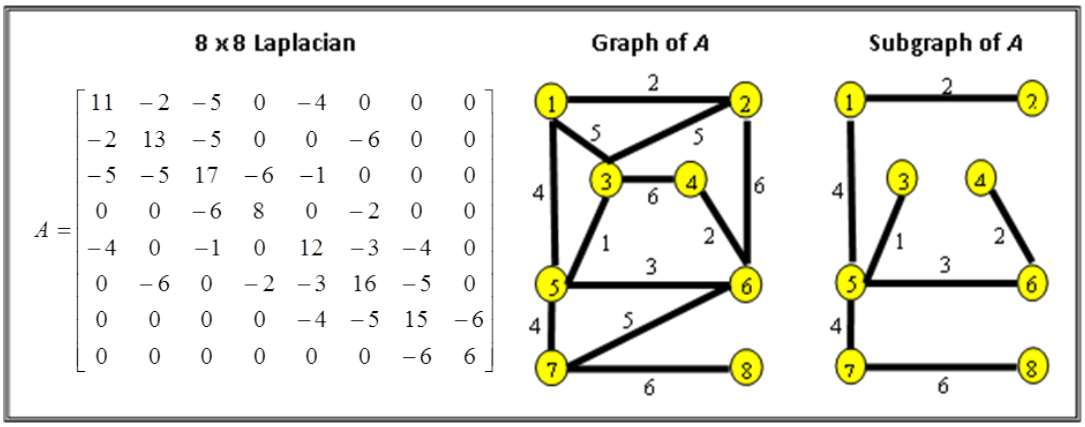


Figure 2. Graph of matrix $A$ with corresponding subgraph $M$.

true solution $x$, then the overall number of iterations can be significantly reduced to improve speed of convergence. Iterative techniques can be improved using preconditioning, where a system is transformed into another system with more favorable spectral properties [8]. In the preconditioned linear system, $M^{-1}Ax = M^{-1}b$, the system matrix $A$ is preconditioned by a non-singular preconditioner matrix $M$. An effective preconditioner matrix $M$ has a low computation and update cost.

New advances in the field of linear solvers were inspired by an idea, proposed in a talk by Pravin Vaidya, to view matrices as undirected graphs and to precondition matrix $A$ by the Laplacian of a subgraph $M$ with substantially fewer edges than $A$ [7]. An example of an $8 \times 8$ matrix with its corresponding graph is presented in Fig. 2. This graph has 8 nodes or vertices $|V|$ denoted by the variable $n$, and 12 weighted edges $|E|$ denoted by $m$. There are several publications on the benefits of using spanning tree subgraphs as preconditioners [9, 10, 11, 12], and Vaidya's ideas led to the development of combinatorial preconditioning, where numerical linearly algebra is merged with spectral graph theory [11, 13].

Spielman and Teng built upon Vaidya's concept by adding off-tree edges back into the spanning tree preconditioner using a random sampling scheme [4]. The graph is viewed as an electrical network and the edges of the input graph are included in the preconditioner using a sampling probability proportional to their effective resistance [5]. The method develops a spectral sparsifier for the system composed of the low-stretch spanning tree of the graph plus a small number of off-tree edges. This method achieved one of the first nearly-linear time algorithms with a convergence rate of $O(m \log^{15} n)$ [14]. Koutis et al. proposed a modification called incremental graph sparsification which shows potential for accurate solutions in $O(m \log^2 n)$ run times [3].

There are two steps to the incremental graph sparsification technique developed in [3]. The first step generates a simplified system for use as the preconditioner for the second step, which performs a Recursive Preconditioned Chebyshev iteration to solve the preconditioned linear system $M^{-1}Ax = M^{-1}b$. Our work extends on this method to develop power system specific methods, which could improve power system reliability by providing accurate solutions in the nearly-linear run time that is claimed by [3]. In the technique, matrix $A$ is represented by a graph $G = (V, E, \omega)$ with each non-zero entry of the matrix representing a connection in the power system. To increase the speed of computation, it is beneficial to reduce the density of $A$. Graph theory has a natural approach to accomplishing this by removing trivial edges in graphs called graph sparsification [3, 15]. The proposed approach utilizes a spectral sparsification scheme, which generates a preconditioner with a provably small condition number between the original graph $G$ and the approximate graph $\tilde{G}$, preserving the spectral properties of the system. The approximate graph $\tilde{G}$, contains a nearly-linear number of edges that $\alpha$-approximates the given graph for a constant $\alpha$. The method explored in this project for fast linear solvers is based on [3] and can be viewed in three components:

1) Developing the low stretch spanning tree.

2) Building a chain of sparse preconditioners.

3) Implementing the actual linear solver.

## 4. METHOD

**4.1. Low Stretch Spanning Tree.** The preconditioner $M$ for the linear system is based on the low-stretch spanning tree of the matrix. To find the low-stretch spanning tree, the system matrix $A$ is represented by a graph with $m$ representing the number of edges in the graph or nonzero entries in the matrix. The concept of stretch was discovered to be critical for constructing a good spanning tree preconditioner by Boman and Hendrickson [9]. In order to have a spanning tree that serves as an effective preconditioner, the off-tree edges must have an average stretch $\delta$ over a spanning tree in order for the spanning tree to be an $O(\delta m)$- approximation of the graph [3]. There exists a unique 'detour' path in the tree between vertices $u$ and $v$, for every edge $e(u, v)$. Stretch is defined as the distortion caused by the detour required by taking the tree path. Eq. (2) is used to compute the stretch of the edges in the tree. The denominator contains the distance between the vertices $(u, v)$ in the graph, and the numerator sums the distances of the edges along the unique path in the tree connecting the vertices $(u, v)$, with distance equaling the inverse of the edge weight, $w' = 1/e$.

$$stretch_T(e) = \frac{\sum_{i=1}^{k} w'(e_i)}{w'(e)} \tag{2}$$

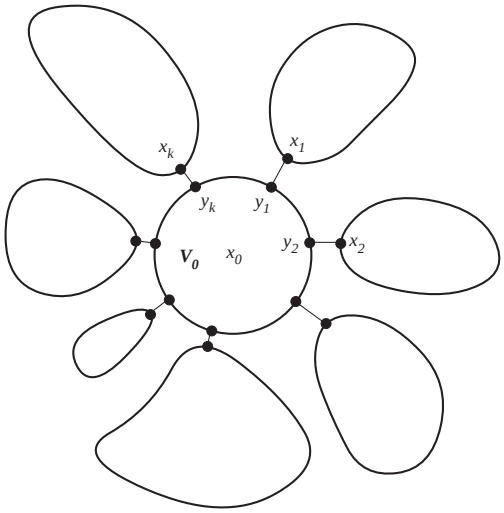


Figure 3. Star-decomposition [1].

The low stretch tree is generated using star-decomposition, described by Elkin et al. in [1]. Star-decomposition recursively performs graph decomposition to partition the graph vertices into clusters, which are connected in a star structure as shown in Fig. 3. A partition of the vertices $V$ is a set of pairwise disjoint subsets $\{V_1, V_2, \ldots, V_k\}$ of the vertices such that their union is $V$ [1]. The vertices of the graph are partitioned into sets by first defining a central ball($V_0$), and then cones $(V_1, \ldots, V_k)$ are grown sequentially from the remaining graph. The ball, which is grown around the central vertex $x_0$, is connected to each outer cone set by a single bridge edge $(x_k, y_k)$. Once all the nodes in the graph are assigned to a set in the star-decomposition, the algorithm is repeated recursively. A new star decomposition is performed on each individual set $V_0$ through $V_k$. The recursion is complete when the size of each set is reduced to at most 2 vertices. A spanning tree is formed containing all of the bridge edges that connect the sets. A trade-off exists between metrics, such as the radius of the graph and the number of edges cut in determining the size of each set, in order to ensure a complete cover of the graph.

**4.2. Chain of Preconditioners.** In the preconditioner phase, a chain of progressively smaller graphs $C = \{A_1, M_1, A_2, M_2, \ldots, A_d\}$ is created based on the low-stretch spanning tree of the matrix. The chain building phase alternates between a sparsification routine to form the preconditioners $M_i$ from the matrices $A_i$ followed by a greedy elimination step to create smaller sub-graphs $A_{i+1}$ from the preconditioners $M_i$ [3, 7]. The sparsification portion of the algorithm generates less dense sub-graphs by removing off-tree edges from the input graph $A_i$. The concept of stretch was defined in the previous sub-section, and it is important to note that if an edge has a stretch equal to 1, then there is no other route between vertices $u$ and $v$ except for the edge joining them. This occurs in the graph depicted in Fig. 2 for the edge between vertices 7 and 8. If the stretch of an edge is small, then there are a significant number of alternative connections between its vertices; therefore, stretch is a measure of the importance of an edge in the graph network. The low-stretch spanning tree is bound by a total stretch of $O(m \log n)$ where $n$ is the number of nodes in the graph and $m$ is the number of edges.

In the incremental sparsify algorithm, the low-stretch tree $T$ is generated using the star-decomposition method defined in the previous sub-section. The weights of the edges of the tree are then scaled up by a factor of $\kappa$, creating a bound on the condition number of the augmented matrix $M^{-1}A$. The condition number, computed in (3), provides a measure of sensitivity to numerical operations and can be used to predict the convergence rate of iterative methods.

$$\kappa(M^{-1}A) = \frac{\lambda_{max}(M^{-1}A)}{\lambda_{min}(M^{-1}A)} \tag{3}$$

A matrix is said to be well-conditioned if its condition number is close to 1 or its eigenvalues are tightly clustered; therefore, the goal is to develop a preconditioner which helps to sufficiently reduce the condition number of the system and is simple to compute [16]. Edges are added to the low-stretch spanning tree to improve the condition number. A version of the graph containing the low-stretch tree with the 'optimal' off-tree edges can satisfy this goal. To achieve this, the incremental sparsification algorithm generates a special graph $\tilde{A}$ from $A$. The scaled version of the tree $T_\kappa$ is added back into the graph, generating $\tilde{A}$, where the edges in the tree path are now heavier by a factor of $\kappa$. If an edge in $\tilde{A}$ is not part of the tree, the weight of that edge remains the same. The stretch of $\tilde{A}$ is now decreased by a factor of $\kappa$ and the total stretch of these edges becomes $t = O(m \log n/\kappa)$. Over-sampling is performed on the off-tree edges with a probability proportional to their stretch over the scaled up tree. The off-tree edges are sampled in a way that adds the most important off-tree edges back into the low-stretch tree preconditioner, based on upper bounds on stretch. Each off-tree edge has a probability $p_e$ of being chosen equal to its stretch, where the total stretch $t = \sum_e p_e$. The sampling scheme assigns each off-tree edge an interval on the unit interval [0, 1], with length corresponding to its probability $p_e$ [3]. A total of $O(t \log n)$ values are sampled at random from the interval [0, 1] and a binary search is performed to find the edge interval corresponding to each random value. The chosen sampled off-tree edges are added to the tree $T$ to form the preconditioner $M$. If an edge is sampled multiple times, it is added as a parallel edge and the weight is scaled accordingly.

In the greedy elimination step, a partial Cholesky factorization is applied to the modified matrix $M_i$. For a symmetric and positive definite matrix $M$, a standard Cholesky factorization can be computed using Gaussian elimination that generates the form $M = LL^T$. In order to reduce the size of $M$ for this application, a partial Cholesky factorization of the

first $k$ variables of $M$ is performed. The partial Cholesky factorization puts $M$ into the form in (4) where $I_k$ is the $k \times k$ identity matrix, and $M_k$ is the Schur complement of $M$ with respect to the elimination of the first $k$ variables [3].

$$M = L \begin{pmatrix} I_k & 0 \\ 0 & M_k \end{pmatrix} L^T \tag{4}$$

The incremental sparsifier $M$ of $A$ is a natural choice for preconditioner, and reducing $M$ using (4) produces the matrix $A_{i+1} = M_k$ for the next chain level [3]. All nodes with degree 1 are removed from the graph $M$, and any remaining degree 2 nodes are replaced by a new single combined edge of weight $w'$, reducing the size of the graph. This process is repeated until the matrix size is reduced to a desired point by certain stopping criteria.

**4.3. Linear Solver.** In the solve phase, the chain of preconditioners is passed into the Recursive Preconditioned Chebyshev algorithm. Recursion is the key to solving preconditioned linear systems quickly. The preconditioner $M$ needs to be a good approximation to $A$ and able to be computed in linear time. Realistically graph preconditioners can't satisfy both requirements since the preconditioner $M$ won't be significantly easier to solve than $A$ [7]. When using a recursive preconditioned method, the preconditioner $M$ is solved approximately. The preconditioned Chebyshev iterative method solves the system $M^{-1}Ax = M^{-1}b$. The condition number of the $A$ matrix can indicate convergence speed for iterative methods like Chebyshev. If the condition number $\kappa(A)$ is large the convergence will be slow, but improvements to the speed can be achieved by proper preconditioning, which reduces the condition number to $\kappa(M, A)$ [17]. Chebyshev iteration is guaranteed to converge after $O(\sqrt{\kappa} \log 1/\epsilon)$ iterations with an accuracy of $x - A^+ b_A \leq \varepsilon A^+ b_A$, where $\kappa = \kappa(A) = \lambda_{max}/\lambda_{min}$ [17]. For the chain of preconditioners, most of the work in the Chebyshev iterations is performed on the chain levels with smaller matrices, levels $i + 1$

Table 1. CONDITION NUMBER REDUCTION AT EACH CHAIN LEVEL

| $Chain\ level\ i$ | $\kappa(A_i)$ | $\kappa(M_i^{-1}A_i)$ |
|---|---|---|
| 1 | 14.292 | 2.798 |
| 2 | 7.245 | 1.349 |
| 3 | 2.36 | 1.098 |

through $i = d$. The partial solutions obtained at the smaller matrix levels are passed on to the larger matrix chain levels. This allows for a great reduction in the number of iterations necessary for convergence at the chain level $i = 1$, where the original matrix $A_1$ is of largest size. Each matrix $M_i$ is a preconditioner for $A_i$, and rather than solving systems in $M_i$ directly, these are reduced to solving a linear system in $A_{i+1}$, performing $O(n)$ additional work. To solve systems on $A_2$, the preconditioned Chebyshev iteration can be recursively applied with a new preconditioner $M_2$, stepping through the progressively smaller graphs in the chain $C$. Table 1 provides an example of the condition number reduction at each chain level for the simulation of a simple 8x8 matrix.

## 5. PRELIMINARY RESULTS

Three power system matrices from the University of Florida Sparse Matrix Collection were used to show the benefit of this method for power system applications [18]. Statistics for the test matrices are presented in Table 2, including: the number of non-zeros in the matrix, condition number of matrix $A_1$, density of matrix $A_1$, and number of edges in the graph of $A_1$. Density is computed by dividing the number of non-zeros in the matrix by the total number of elements in the matrix, $n \times n$.

Table 2. TEST MATRIX STATISTICS

| $Matrix$ | $Nonzeros$ | $Condition, \kappa(A)$ | $Density$ | $Edges, m$ |
|---|---|---|---|---|
| 494 *bus* | 1666 | $2.06E+05$ | 0.68% | 586 |
| 662 *bus* | 2474 | $1.35E+04$ | 0.56% | 906 |
| 1138 *bus* | 4054 | $2.39E+05$ | 0.31% | 1458 |

Table 3. SIMULATION ITERATION RESULTS

| Matrix | Iteration level $i = 1$ | Iteration levels $i = 2$ to $d$ | Preconditioner Chain Size |
|---|---|---|---|
| 494 *bus* | 55 | 11 | 494, 54, 24, 4, 2 |
| 662 *bus* | 37 | 9 | 662, 161, 27, 6, 2 |
| 1138 *bus* | 83 | 28 | 1138, 123, 43, 7, 2 |

Table 3 contains simulation data including a breakdown of the iterations and matrix size for all levels in the chain. In table 3, there are two columns for iterations. The first column contains the number of iterations performed at chain level $i = 1$ where the solution is computed on the original matrix size, and the second iteration column contains the total number of iterations performed at the chain levels $i = 2$ through $i = d$, where solutions are computed on the smaller sub-matrices. For example, the 494-bus system only performs 55 of the total iterations on a system of size $494 \times 494$, the rest of the 11 internal iterations are performed on smaller sub-matrices, which decreases the overall computation time required by the method. The third column labeled Preconditioner Chain Size provides information on the matrix size reduction at each level of the chain, for example, the 494 bus system is reduced to a matrix of size $54 \times 54$ for chain level $i = 2$, and a matrix of size $24 \times 24$ for chain level $i = 3$, etc.

Table 4. COMPARISON OF CHAIN CHEBYSHEV ITERATION TO TRADITIONAL CHEBYSHEV ITERATION

| | **Chain Chebyshev** | | **Traditional Chebyshev** | |
|---|---|---|---|---|
| Matrix | Time (sec) | Total Iterations | Time (sec) | Total Itertions |
| 494 | 0.6907 | 66 | 4.3501 | 502 |
| 662 | 1.6581 | 46 | 2.4991 | 129 |
| 1138 | 8.9947 | 111 | 61.379 | 740 |

Table 4 contains preliminary simulation results, comparing the current development of the Chain of Preconditioners method to a standard Preconditioned Chebyshev iteration without the chain. The error is measured by the norm of the residual $r = b - Ax$ . An error threshold of $e = 0.01$ was used as stopping criteria for the iterative methods. The time and iteration reductions achieved by the Chain method tend to improve as the system matrix size increases.

## 6. CONCLUSIONS

The preliminary results presented in this paper show a significant speed improvement and iteration reduction for the linear solutions of several test matrices and power system networks using the SDD linear solver. This method has the potential to improve power system reliability by significantly decreasing runtime of critical analysis tools required by power system operators to prevent catastrophic contingencies. Preliminary results show this method to be comparable with traditional techniques. Further study and enhancements are planned to improve the speed and accuracy of this method for power system specific applications. There are several benefits that can be achieved by developing nearly linear solvers for large scale power systems including:

- Achieve a time frame useful to industry (real-time)
- Provide on-line control to avoid cascading failures and outages
- Improve power system reliability
- Provide a shift in power system operation from ‘what-if’ scenarios to on-line monitoring with situational awareness
- Assist in Smart Grid development

There are additional areas to be explored to further optimize the algorithm for power system applications. The traditional approach in power systems for dealing with very large sparse matrices is to apply direct methods using reordering. Node reordering helps to minimize the number of computations required by reducing the number of fills or nonzero elements that are generated in the matrix decomposition [19, 20]. Our next step in determining the potential for this tool to improve power system analysis techniques is to benchmark the algorithm against traditional direct sparse methods using reordering.

# II. A CHAIN METHOD FOR PRECONITIONED ITERATIVE LINEAR SOLVERS FOR POWER SYSTEM MATRICES

L. L. Grant, M. L. Crow, and M. X. Cheng

Department of Electrical & Computer Engineering

Missouri University of Science and Technology

Rolla, Missouri 65409–0050

Tel: 573–465–0657

Email: llsnpc@mst.edu

## 1. ABSTRACT

Many power systems applications, such as power flow and short circuit analysis, require very large sparse matrix computations. With the increase in reliance on our electric infrastructure, power systems are continually growing in size, creating greater computational complexity in solving these large linear systems within reasonable time. For sparse matrix applications, it is desirable to have an algorithm with low runtime complexity in terms of the number of non-zeros in the matrix. There have been several recent advances in computational methods in other fields that if applied to power system, could make real-time dynamic simulation a reality. Much work has been done for specific types of these problems where the system is symmetric and diagonally dominant (SDD), similar to the form of power system matrices. This paper details an expansion on the current work in fast linear solvers to develop power system specific methods that show potential for accurate solutions in $O(m \log^2 n)$ run times, where $n$ represents the number of nodes and

*m* represents the number of nonzeros in the power system matrix. This paper presents the simulation validation of a recently developed recursively-solved iterative chain method for sparse matrices using a low stretch spanning tree (LSST) preconditioner.

Keywords: Matrix preconditioning, graph sparsification, linear system solution, power system simulation.

## 2. INTRODUCTION

There are several power system simulations that require the solution of a large number of differential algebraic equations in real-time. In the majority of power system computational problems, the system of equations to be solved typically depends on a (possibly time-varying) matrix solution. The system matrix $A$ in question is usually symmetric and sparse, but with randomly occurring off-diagonal non-zero elements. The most computationally intensive steps in solving numerical power system problems is the solution of the system of linear equations. The majority of currently used power system linear solvers rely on direct methods derived from LU factorization [21]. Most recent approaches propose methods for node ordering schemes to reduce fills, elimination trees, assembly trees, and directed acyclic graphs as developments for more efficient LU factorization. It has been conventional wisdom that direct methods will always outperform iterative methods for sparse systems due to convergence uncertainty of iterative methods. However, as the size of systems under consideration have increased, iterative solvers have become more competitive due to the poor scalability of direct methods. Even the best sparse direct solvers require roughly $O\left(n^1.4\right)$ floating point operations (where $n$ is the size of the matrix) [22]. Recently, power systems researchers have begun to investigate iterative solution methods as advances in preconditioners have begun to produce competitive computational runtimes and accuracy

levels [23]. Chen et. al demonstrated the use of preconditioned iterative methods on DC and transient simulations for large scale power delivery circuits with favorable results using conjugate gradient algorithm [24]. A study was also performed to demonstrate the use of iterative techniques on the Nigerian 330 kv grid [25].

For sparse matrix applications, it is desirable to have an algorithm with a run-time that is efficient in terms of the number of non-zero variables in the matrix. The density of matrix $A$ determines how quickly these equations can be solved. A sparse matrix that is ordered and conditioned in a way that improves its spectral properties, can allow for a simple iterative method to efficiently solve even large systems in a short amount of time [2]. Practical implementation of iterative methods require that the system matrix $A$ be preconditioned by a non-singular preconditioner matrix $M$ that closely approximates the inverse of the system matrix. The preconditioner matrix $M$ should require little computational cost to compute and update and provide a computationally efficient solution to $My = b$ where $y$ is a near solution to $x$ in $Ax = b$. Effective preconditioners arise from incomplete LU factorization, a sparse approximate inverse to $A$, and stationary methods among others. Recent advances have proposed an approach that exploits underlying structural properties to produce a chain of graphs which is used as an input to a recursive preconditioned Chebyshev iteration. This method has been shown to solve symmetric, diagonally dominant (SDD) systems in nearly-linear time [26]. Power system matrices closely mimic SDD matrices and are sparse with randomly occurring off-diagonal non-zero elements [3, 7]. Thus it is desirable to apply these new advances in nearly-linear solvers to the field of electric power systems. In this paper, we extend the developments in [3] and [14] to improve power system computation to nearly-linear run time. Specifically, we propose a computationally efficient preconditioner which reduces the condition number of the system thereby improving convergence and decreasing runtime.

## 3. BACKGROUND

Undirected graphs $G_A = (V, E, \omega)$ are often used to represent matrices. One proposed approach to developing effective preconditioners to $G_A$ is to use the Laplacian of a subgraph $G_M$ that has substantially fewer edges than $G_A$ [7]. Graph theory has a natural approach to reducing the density of $A$ by removing trivial edges [3, 15]. Recent work has focused on using spanning trees as preconditioners, where a spanning tree is a subgraph of $A$ and is a connected graph containing all of the vertices in $A$ with no cycles [9, 10, 11, 12]. These approaches have led to the development of combinatorial preconditioning, where numerical linear algebra is merged with spectral graph theory [11, 13].

One recent approach is to add selected off-tree edges back into the spanning tree preconditioner using a random sampling scheme [4]. This is based on visualizing the graph as an electrical network where the edges are selected for the preconditioner using a sampling probability proportional to their effective resistance [5]. The method develops a spectral sparsifier for the system composed of the low-stretch spanning tree (LSST) of the graph plus a small number of off-tree edges. The LSST is a spanning tree that uses the concept of stretch to determine which edges are kept in the subgraph. This method achieved one of the first nearly-linear time algorithms with a convergence rate of $O(m \log^{15} n)$ [14].

A modification, called incremental graph sparsification, has been proposed, which shows the potential for accurate solutions in $O(m \log^2 n)$ run times [3]. The method was designed for imaging applications and contains a combinatorial component that uses graph theoretic algorithms. There are two steps to the incremental graph sparsification technique developed in [3]. The first step generates a simplified system for use as the preconditioner for the second step, which performs a recursive iterative technique to solve the preconditioned linear system $M^{-1}Ax = M^{-1}b$. The proposed approach generates a preconditioner with a provably small condition number between the original graph $G_A$ and the approximate graph

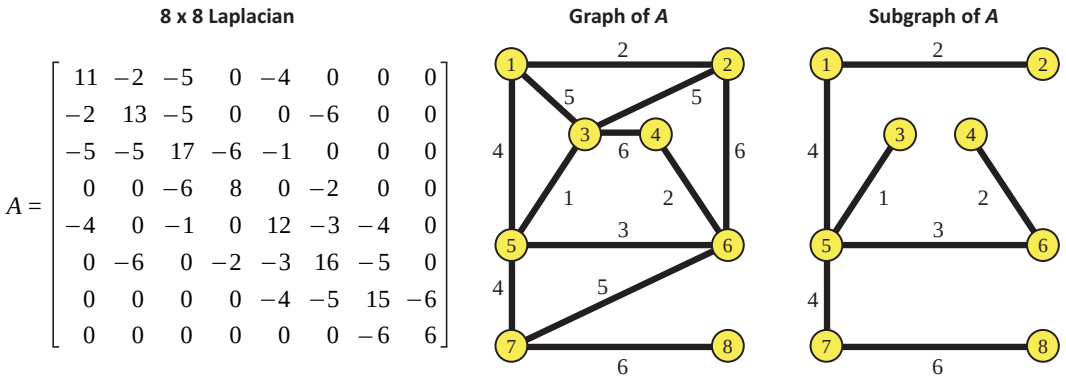


Figure 1. Graph of matrix $A$ with corresponding subgraph $M$

$\widetilde{G_A}$ preserving the spectral properties of the system. The approximate graph $\widetilde{G_A}$ contains a nearly-linear number of edges that $\alpha$-approximates the given graph for a constant $\alpha$. The condition number, given by (1), provides a measure of sensitivity to numerical operations and can be used to predict the convergence rate of iterative methods.

$$\kappa(M^{-1}A) = \frac{\lambda_{max}(M^{-1}A)}{\lambda_{min}(M^{-1}A)} \tag{1}$$

A small matrix example will be used throughout the paper to better convey the graphical interpretations of the proposed approach. An $8 \times 8$ matrix with its corresponding graph is shown in Fig. 1. This graph has 8 nodes or vertices $|V|$ denoted by the variable $n$, and 12 weighted edges $|E|$ denoted by $m$. For simplicity in notation, both the matrix representing the power system and its corresponding graph will be referred to as $A$ and the matrix preconditioner along its corresponding graph will be referred to as $M$.

The method explored in this project for fast linear solvers is based on [3] and can be viewed in three components described in detail in the following subsections:

A) Developing the low stretch spanning tree.

B) Building a chain of sparse preconditioners.

C) Implementing the actual linear solver.

**3.1. Low Stretch Spanning Tree (LSST).** The preconditioner $M$ for the linear system is based on the low-stretch spanning tree of the matrix. To find the LSST, the system matrix $A$ is represented by a graph with $m$ edges, which also correspond to the non-zero entries in the matrix. The size of the tree stretch is critical for constructing a good spanning tree preconditioner [9]. To have a spanning tree that serves as an effective preconditioner, the off-tree edges must have an average stretch $\delta$ over the spanning tree for the spanning tree to be an $O(\delta m)$-approximation of the graph [3]. There exists a unique 'detour' path in the tree between vertices $u$ and $v$ for every edge $e_{(u,v)}$. Stretch is defined as the distortion caused by the detour required by taking the tree path, where the stretch of the edges in the tree is given by:

$$stretch_T(e) = \frac{\sum_{i=1}^{k} w'(e_i)}{w'(e)} \tag{2}$$

The denominator of (2) contains the distance between the vertices $(u, v)$ in the graph and the numerator sums the distances of the edges along the unique path in the tree connecting vertices $(u, v)$, with the distance equaling the inverse of the edge weight, $w' = 1/e$. If the stretch of an edge is small, then there are a significant number of alternative connections between its vertices; therefore, stretch is a measure of the importance of an edge in the graph network. A low-stretch spanning tree is bounded by a total stretch of $O(m \log n)$.

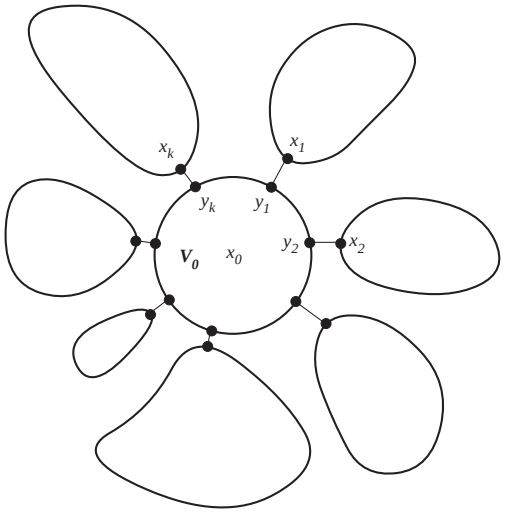


Figure 2. Star-decomposition [1].

The LSST is generated using the star-decomposition approach [1]. Star-decomposition recursively performs a graph decomposition to partition the vertices of the graph into clusters, which are connected in a star structure as shown in Fig. 2. A partition of the vertices $V$ is a set of pairwise disjoint subsets $\{V_1, V_2, \ldots, V_k\}$ of the vertices such that their union is $V$ [1]. The vertices of the graph are partitioned into sets by first defining a central ball $V_0$. Cones $V_1, \ldots, V_k$ are then grown sequentially from the nodes remaining in the graph. The ball, which is grown around the central vertex $x_0$, is connected to each outer cone set by a single bridge edge $(x_k, y_k)$. Once all the nodes in the graph are assigned to a set in the star-decomposition, the algorithm is repeated recursively. A new star decomposition is performed on each individual set $V_0$ through $V_k$. The recursion is complete when the size of each set is reduced to at most 2 vertices. A spanning tree is formed containing all of the bridge edges that connect the sets. A trade-off exists between metrics, such as the radius of the graph and number of edges cut, in determining the size of each set to ensure a complete cover of the graph. The low-stretch spanning tree for the $8 \times 8$ example matrix is shown in

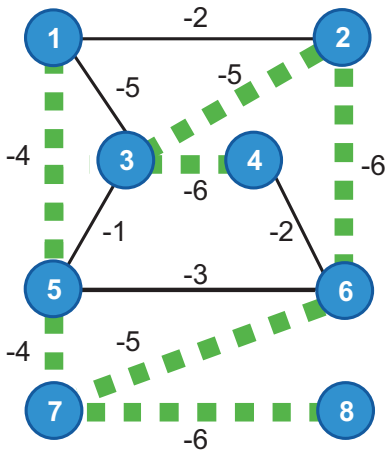


Figure 3. Example of the low stretch spanning tree for an $8 \times 8$ matrix.

Fig. 3 with the tree edges highlighted by the dashed green lines. Edges in the graph that do not belong to the tree are referred to as off-tree edges. To compute the stretch for an off-tree edge $e_{(4,6)}$, the distances of edges connecting nodes 4 and 6 along the tree are summed and then divided by the distance of the edge directly between nodes 4 and 6 as:

$$\begin{aligned} stretch_T(e_{(4,6)}) &= \frac{w'(e_{3,4}) + w'(e_{2,3}) + w'(e_{2,6})}{w'(e_{4,6})} \\ &= \frac{1/6 + 1/5 + 1/6}{1/2} = 1.067 \end{aligned}$$

**3.2. Chain of Preconditioners.** In the preconditioner phase, a chain of progressively smaller graphs $C = \{A_1, M_1, A_2, M_2, \ldots, A_d\}$ is created based on the low-stretch spanning tree of the matrix. The chain building phase alternates between a sparsification routine to form the preconditioners $M_i$ from the matrices $A_i$, followed by a greedy elimination step to create smaller subgraphs $A_{i+1}$ from the preconditioners $M_i$ [3, 7].

The preconditioner $M_i$ needs to be a good approximation to $A_i$ and able to be computed in linear time. The sparsification portion of the algorithm, depicted in Fig. 4, generates less dense subgraphs by adding a subset of off-tree edges from $A_i$ back to the LSST in order to form the preconditioner $M_i$. In sparsification, the number of edges in the graph $A$ needs to be significantly reduced while preserving a good approximation to the original graph. Adding the 'optimal' off-tree edges back to the LSST produces a graph with a better condition number than $A_i$. To achieve this, the incremental sparsification algorithm generates a special graph $\widetilde{A}_i$ from $A_i$. The weights of all tree edges are scaled up by a factor of $\kappa$, creating a bound on the condition number of the augmented preconditioned matrix $M_i^{-1}A_i$. The scaled version of the tree $T_\kappa$ is added back into the graph, generating $\widetilde{A}_i$, where the edges in the tree path are now heavier by a factor of $\kappa$. The weights of the off-tree edges in $\widetilde{A}_i$ remain unchanged. The stretch of the edges in $\widetilde{A}_i$ is now decreased by a factor of $\kappa$ and the total stretch of these edges becomes $t = O(m \log n/\kappa)$.

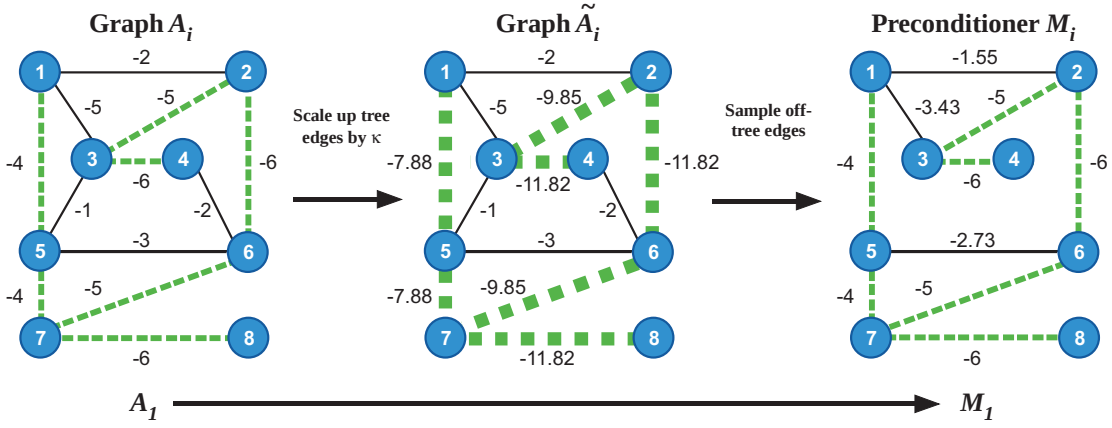


Figure 4. Example of Incremental Sparsify algorithm.

Over-sampling is performed on the off-tree edges of $\widetilde{A}_i$, where each off-tree edge has a probability of being chosen proportional to its scaled stretch value. The off-tree edges are sampled in a way that adds the most important off-tree edges back into the low-stretch

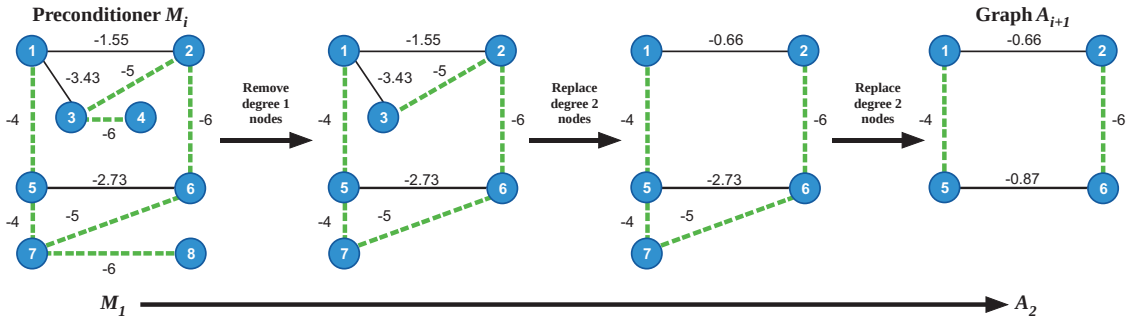


Figure 5. Example of Greedy Elimination.

tree preconditioner, based on upper bounds on stretch. The sampling scheme assigns each off-tree edge an interval on the unit interval [0, 1], with interval length corresponding to its probability $p_e$ [3]. A total of $O(t \log n)$ values are sampled at random from the interval [0, 1], where $t = \sum_e p_e$ indicates the total stretch. A binary search is performed to find the edge interval corresponding to each random value. The chosen sampled off-tree edges are added to the tree $T$ to form the preconditioner $M_i$. If an off-tree edge is sampled multiple times, it is added in parallel and the weight is scaled accordingly. In Fig. 4, the off-tree edges $e_{(1,2)}$, $e_{(1,3)}$, and $e_{(5,6)}$ are added to the tree forming preconditioner $M_i$ with new scaled weight values. The density of the graph is reduced not only by removing edges $e_{(3,5)}$ and $e_{(4,6)}$, but also by the overall reduction in edge weight. This completes the first level $i = 1$ of the chain by producing $A_1$ and $M_1$. The next chain level $i + 1 = 2$ will be initiated by performing greedy elimination on the preconditioner.

The incremental sparsifier $M_i$ is a natural choice for preconditioner, and reducing the size of $M_i$ using greedy elimination produces the matrix $A_{i+1}$ for the next chain level [3]. To reduce the matrix size using greedy elimination, all nodes in $M_i$ with degree 1 are

removed from the graph, and any remaining degree 2 nodes are replaced by a new single edge of combined weight $w'$. The matrix reduction can be achieved by applying a partial Cholesky factorization of the first $k$ variables to the modified matrix $M_i$.

$$M_i = L_i \begin{pmatrix} I & 0 \\ 0 & A_{i+1} \end{pmatrix} L^T \tag{3}$$

The partial Cholesky factorization puts $M_i$ into the form in (3) where $I$ is the $k \times k$ identity matrix, and $A_{i+1}$ is the Schur complement of $M_i$ with respect to the elimination of the first $k$ variables [3]. The new matrix $A_{i+1}$, is a reduced approximation of the original graph $A_i$. Fig. 5 contains an example of reducing the density of $A_i$ by applying greedy elimination to its preconditioner. In the greedy elimination example, the degree 1 nodes 4 and 8 are pruned from the preconditioner $M_i$ along with their connecting edge. Once all degree 1 nodes are removed, the degree 2 nodes get absorbed one-by one starting with node 3 where edges $e_{(1,3)}$ and $e_{(2,3)}$ are combined with their shared mutual edge $e_{(1,2)}$. Node 7 is the last remaining degree 2 node, which is also removed by combining its corresponding edges $e_{(5,7)}$ and $e_{(6,7)}$ with the shared edge $e_{(5,6)}$. The remaining graph $A_{i+1}$ has been reduced from size $8 \times 8$ to size $4 \times 4$ and will be used as a starting point for chain level $i = 2$. The process of generating smaller matrices and their preconditioners using incremental sparsification and greedy elimination is repeated until the matrix size is reduced to a desired point by certain stopping criteria. From both Figs. 4 and 5 it can be observed that the majority of the original low stretch spanning tree edges, depicted by the dashed green lines, are retained as the matrix is reduced, while the off-tree edges are systematically removed by reducing their edge weights. Fig. 7 shows a complete cycle of the chain building process used to produce the reduced matrices $A_i$ and their preconditioners $M_i$ for the example $8 \times 8$ system. For

this illustration, the greedy elimination step has been modified to only remove a few nodes each cycle since this is an abnormally small matrix, instead of eliminating all degree 1 and degree 2 nodes at once.

**3.3. Linear Solver.** In the solve phase, a recursive iterative method is applied to each level $i$ in the chain of preconditioners. Iterative methods can be terminated as soon as they reach a reasonable solution, making them more scalable by requiring less computation than direct methods [4]. Recursion is the key to solving preconditioned linear systems quickly, since the preconditioner $M_i$ can be solved approximately. The preconditioned iterative method solves the system $M^{-1}Ax = M^{-1}b$. The matrices $M_i$ serve as preconditioners for matrices $A_{i-1}$ in the recursive solver [6]. Instead of solving systems in $M_i$ directly, the systems are reduced in $A_i$ and are solved recursively. A product of the form $M^{-1}z$ is equivalent to solving the system $My = c$ [3]. The function $f_{Mi}(z)$ produces approximate solutions to the systems in $M_i$, returning $M^{-1}c$ to the iterative method instead of the preconditioner $M_i$ as input [6]. Details and pseudocode for the recursive solver can be found in [3].

The preconditioning chain $C$ reduces the computation needed to solve the linear system. Most of the work in the iterative methods is performed on the chain levels with smaller matrices, levels $i + 1$ through $i = d$. The recursive method solves the system in $M$ by solving a linear system in $A_2$ [3]. The Chain is evoked to solve systems on $A_2$ by recursively applying the preconditioned iterative method to it using the new preconditioner $M_2$, and so forth through the chain of progressively smaller matrices. The iterative method applied in this paper is the Preconditioned Chebyshev iteration.

The Conjugate Gradient algorithm is suited for linear systems with symmetric positive definite and sparse matrices. It is known as one of the Krylov subspace methods, which generates a sequence of converging solutions, $x_k$, that are an orthogonal projection

of $b$ applied to the Krylov subspace $K_n\langle r_0, A\rangle$, where $r_0 = b - Ax_0$ is the initial residual [2]. A Krylov subspace is a linear combination of vectors, $K_n\langle b, Ab, \ldots, A^{n-1}b\rangle$; the residual is orthogonal to the subspace, along with all previously generated residuals [27, 28]. The solution $x_k$ at each step is found using a linear combination of the previous search directions $p_k$ and the current residual. A graphical description of how the approximate solution moves from $x_k$ to $x_{k+1}$ is provided in Fig. 6 [19].

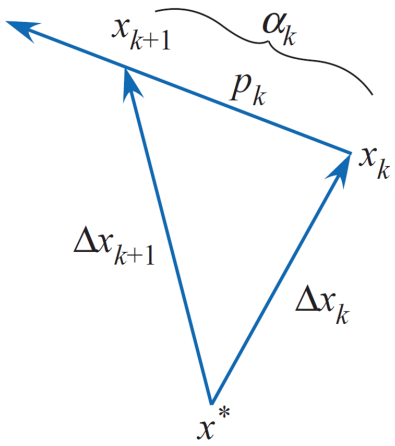


Figure 6. PCG iterative method.

The conjugate relationship among the parameters eliminates the amount of vectors needing to be stored in memory, since they can be derived from each other [27]. One advantage of the PCG method is that only one matrix-vector multiplication is needed per iteration in the computation of the current residue. The greatest advantage of PCG over other methods is that it converges in at most $n$ steps. The goal of iterative techniques is to reach the exact solution in the fewest number of iterations for a fast convergence rate.

The Chebyshev iterative method is similar to the PCG and was developed to avoid the need to compute inner products, which require considerable computation time for large systems. The trade-off of not needing to generate an inner product is that enough must be known about the system in order to generate an ellipse enclosing the spectrum of $A$ [29, 30].

A family of ellipses is defined by scalars $c$ and $d$ with a common center $d > 0$ and foci $d + c$ and $d - c$. The rate of convergence is determined by (4) where $a$ is the length of the $x$-axis of the ellipse. For symmetric positive definite systems, the ellipse enclosing the spectrum of $A$ degenerates to the interval $[\lambda_{min}, \lambda_{max}]$ along the positive $x$-axis, where $\lambda_{min}$ and $\lambda_{max}$ are the largest and smallest eigenvalues of $M^{-1}A$ [16, 30]. Chebyshev has the same upper bound as PCG for symmetric matrices when $c$ and $d$ are computed from $\lambda_{min}$ and $\lambda_{max}$ [2].

$$r = \frac{a + \sqrt{a^2 - c^2}}{d\sqrt{d^2 - c^2}} \tag{4}$$

At each chain level $i$, the Chained Chebyshev method provides a product in the form $M^{-1}z$, which is equivalent to solving the system $My = c$, whereas the traditional Preconditioned Chebyshev iteration must explicitly calculate the solution to $My = c$ [3]. The condition number of the $A_i$ matrix can indicate convergence speed for iterative methods like Chebyshev. If the condition number $\kappa(A)$ is large, then the convergence will be slow. Improvements to the speed can be achieved by proper preconditioning, which reduces the condition number to $\kappa(M_i, A_i)$ [17]. Chebyshev iteration is guaranteed to converge after $O(\sqrt{\kappa} \log 1/\varepsilon)$ iterations with an accuracy of $\|x - A^+b\|_A \leq \varepsilon \|A^+b\|_A$ where the 2-norm condition number of $A$ can be computed by $\kappa = \kappa(A) = \lambda_{max}/\lambda_{min}$ [17]. Table 1 provides an example of the condition number reduction at each chain level for a 117 x 117 matrix. Note that the condition number of the preconditioned system $(M_i^{-1}A_i)$ decreases rapidly to nearly unity, indicating very rapid convergence for the smaller levels in the chain.

Table 1. CONDITION NUMBER REDUCTION AT EACH CHAIN LEVEL

| *Chain level i* | $\kappa(A_i)$ | $\kappa(M_i^{-1}A_i)$ |
|---|---|---|
| 1 | 1846.7 | 143.92 |
| 2 | 225.5 | 65.05 |
| 3 | 119.19 | 41.398 |
| 4 | 57.123 | 1.2163 |

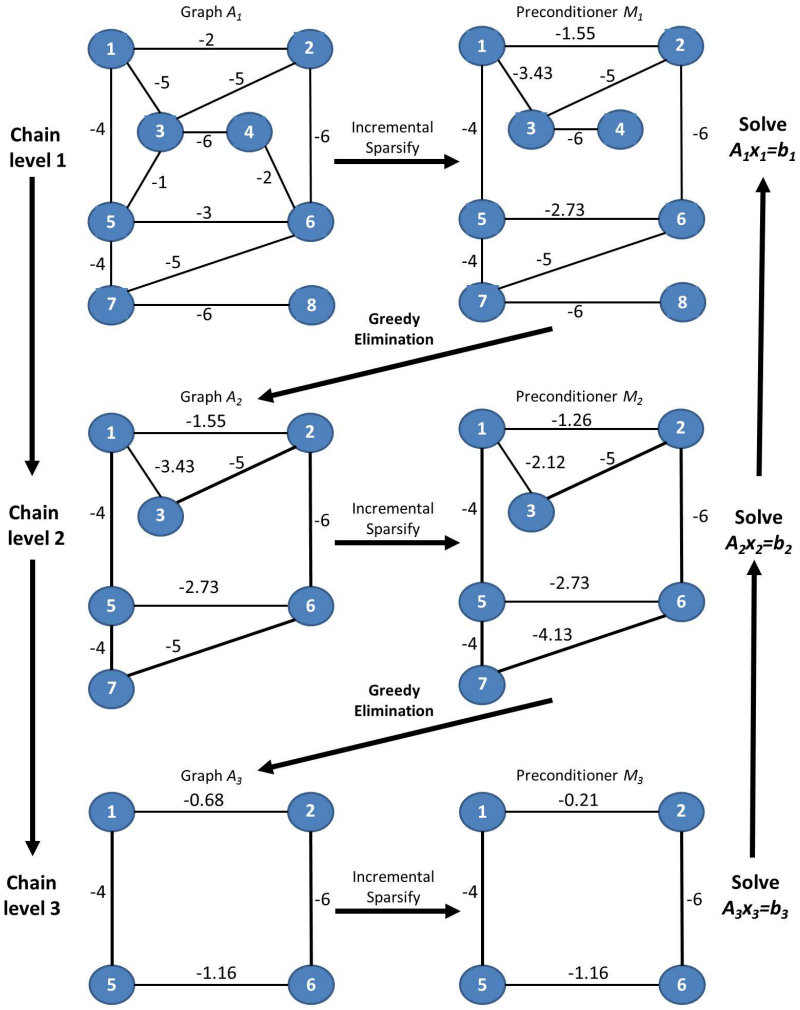


Figure 7. Chain of preconditioners for simple $8 \times 8$ example.

## 4. THE LSST CHAINED PRECONDITIONER

Several matrices were used to demonstrate the capabilities of this method, including power networks from the University of Florida Sparse Matrix Collection and the Matpower library [18] [31].

Statistics for the test matrices are presented in Table 2. The table includes number of edges in graph $A_1$, number of nonzero values in the system matrix, the condition number $\kappa(A_1)$ of the matrix, and the density of the matrix. The density value provides a measure of the sparsity in the matrix and is computed by dividing the number of nonzeros by $n^2$, which is the total number of elements in the matrix. A very small density percentage indicates a very sparse system with a greater proportion of zero values to nonzeros.

Table 2. TEST MATRIX STATISTICS

| Buses ($n$) | Edges ($m$) | Nonzeros | $\kappa(A_1)$ | Density |
|---|---|---|---|---|
| 8 | 12 | 32 | 60.87 | 50% |
| 20 | 26 | 72 | 756.27 | 18% |
| 117 | 175 | 467 | 1,846.67 | 3.41% |
| 131 | 148 | 427 | 203.40 | 2.49% |
| 494 | 586 | 1,666 | $1.08E+05$ | 0.68% |
| 662 | 906 | 2,474 | $1.02E+04$ | 0.56% |
| 1138 | 1,458 | 4,054 | $1.21E+05$ | 0.31% |
| 3374 | 4068 | 11510 | 5.48e+04 | 0.10% |
| 6515 | 8104 | 22723 | 1.37E+05 | 0.05% |
| 11838 | 13630 | 39098 | 6.65E+04 | 0.02% |

Figure 8 shows that the LSST preconditioner reduces the number of iterations required when compared to several popular preconditioning methods [2]. For this study, the following traditional preconditioners were compared to the method described in this paper:

- Jacobi $M = diag(A)$
- Tridiagonal $M = tridiag(a_{i,i-1}, a_{i,i}, a_{i,i+1})$
- SSOR $M = (D + L)D^{-1}(D + L)^T$
- Incomplete LU Factorization (ILU) $M = LL'$
- LSST Preconditioner $M = M_1$

The LSST preconditioner is the level 1 preconditioner matrix $M_1$ derived using the technique described in Sec. II-B1, but the solution is derived using the traditional PCG method without utilizing the chain sparsification.

These results were included to show that the Low Stretch Tree serves as an efficient preconditioner by itself, but as with all preconditioners there is a trade-off between the time required to find the preconditioned matrices $M^{-1}A$ and $M^{-1}b$ and the number of iterations for convergence. For this reason, the proposed Chain Method uses a series of smaller preconditioners and $M^{-1}A$ and $M^{-1}b$ are *never explicitly found*.

**4.1. The Chain Method of Preconditioners.** Fast iterative linear solutions depend on a good choice of preconditioner that significantly reduce the condition number of the preconditioned system. It was shown that the LSST method provides such a matrix. But this is only one aspect of the iterative solver. One of the most often cited drawbacks of preconditioned iterative linear solvers is that the preconditioned system may not be significantly easier to solve than the original system. Therefore the proposed method alleviates this issue by introducing a multilevel hierarchy of chained matrices, which are solved recursively. One of the primary attributes of the Chain Method is the recursive solver. This method starts with the smallest matrix in the chain and uses the solution to predict the solution of the next larger matrix. Furthermore at every chain level, rather than

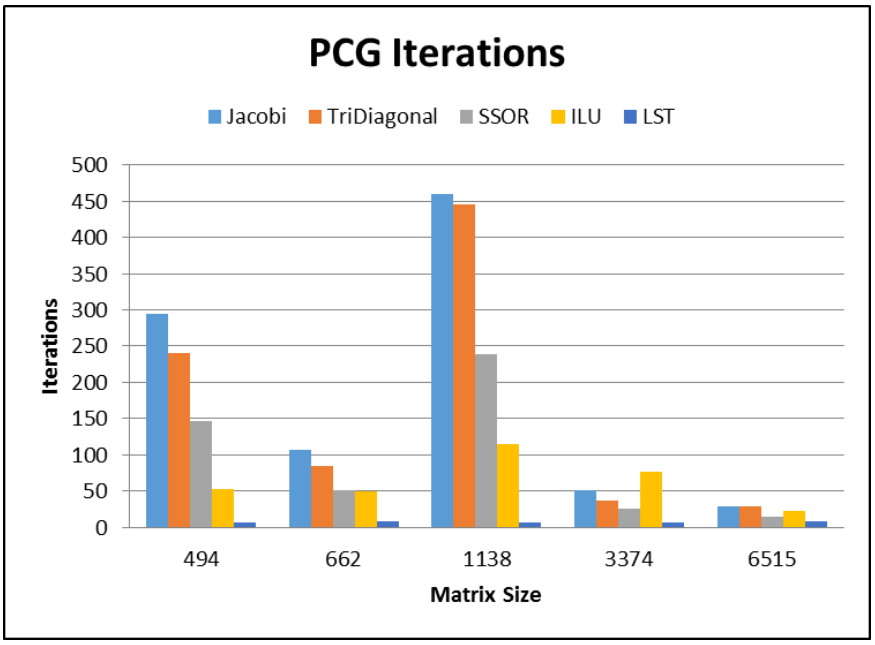


Figure 8. Number of iterations required for different preconditioning techniques using the PCG method.

solving systems directly, recursion is used to produce approximate solutions at each level. Therefore, the proposed Chain Method has several advantages over other iterative linear solvers. The advantages are summarized below:

1. The LSST preconditioner is an excellent preconditioner as evidenced by the significant reduction in condition number of the preconditioned system and the sharp decrease in number of iterations required for convergence (see Fig. 8)

2. The preconditioner $M$ is based on the low-stretch spanning tree and therefore captures physical properties of the underlying system.

3. The recursive solver avoids the requirement to explicitly solve the preconditioned system $M^{-1}A$.

4. The chained method of solving from small to increasingly larger matrices provides an reasonably accurate initial guess to the iteration, thus decreasing the number of iterations at each step.

## 5. RESULTS AND DISCUSSION

The reduction in condition number achieved at key points in the preconditioning chain is presented in Table 3, along with the number of chain levels generated for each test matrix. The size reduction achieved by greedy elimination of the 494 bus system produces 7 chain levels of matrix sizes 494, 162, 68, 34, 18, 7, and 3 respectively. The algorithm parameters are set to roughly reduce the system size by at least half for each level. The second column of Table 3 contains the condition number of the original matrix $\kappa(A_1)$, the third column displays the reduction in condition number achieved by applying the first preconditioner to the matrix $\kappa(M_1^{-1}A_1)$, and the fourth column contains the condition number when the system is reduced to the matrix of smallest size at chain level $i = d$. The table indicates that the smallest chain level produces a condition number close to 1 for all matrices, which is optimal for obtaining an accurate solution very quickly because a condition number close to 1 indicates a well ordered system. Solutions for all matrices at this chain level generally converged with the first few iterations. The preconditioner is able to provide a better condition number even at chain level $i = 1$ by approximately an order of magnitude for the larger test matrices exhibiting the most sparsity.

Table 4 contains simulation results including runtime of each segment of the algorithm (i.e., the set-up (time to find the LSST and build the chain of preconditioners) and perform the PCG iteration). All time values are in seconds and the simulation was run on a Intel i-7 2.5 GHz processor with 12 GB of memory. The error is measured by the 2-norm of the residual $r = b - Ax$. An error threshold of $e = 10^{-5}$ was used as stopping criteria for the

Table 3. SIMULATION REDUCTION IN CONDITION NUMBER AND CHAIN SIZE

| Nodes | $\kappa(A_1)$ | $\kappa(M_1^{-1}A_1)$ | Chain levels |
|---|---|---|---|
| 8 | 60.87 | 2.36 | 2 |
| 20 | 756.27 | 12.66 | 3 |
| 117 | 1,846.67 | 95.15 | 6 |
| 131 | 203.40 | 2.34 | 5 |
| 494 | $1.08E+05$ | 272.44 | 9 |
| 662 | $1.02E+04$ | 740.25 | 8 |
| 1138 | $1.21E+05$ | 1930 | 9 |
| 3374 | 5.48E+04 | 83.12 | 8 |
| 6515 | 1.37E+05 | 147.97 | 11 |
| 11838 | 6.65E+04 | * | 11 |

*matrices too large to calculate

iterative methods. The number of iterations reported in the table is the iterations required to solve chain level 1, where the matrix is of largest size. There is a finite limit imposed on the number of iterations allowed for the smaller chain levels, since only an approximate solution is required at the smaller matrix chain levels. Finding the LSST for each matrix consumes the most time. Fortunately the LSST would only needed to be computed once in situations where only small updates are made to the system matrix without altering the structure of the matrix such as during a Newton-Raphson iteration or during a time-domain simulation. A benefit of this method is the trend presented in the table where the number of iterations required decrease in proportion to the increase in matrix size. For example, only 13 iterations are required to produce a solution for the 3374 bus power system matrix.

Figures 9 and 10 provides a comparison of the runtimes of the chained solver with a time comparison for a sparse LU factorization for the set of matrices. The direct method (LU) is faster than the chain method for small matrices, but appears to start to lag the chain method as the matrix size increases.

Table 4. SIMULATION RESULTS

| **Matrix size** | **117** | **685** | **3374** | **6515** | **13659** | **22902** | **69518** | **149842** |
|---|---|---|---|---|---|---|---|---|
| **Time Set-up (sec)** | 0.0001 | 0.0469 | 0.0469 | 0.1094 | 0.1094 | 0.1250 | 0.0938 | 0.1719 |
| **Time PCG (sec)** | 0.0017 | 0.0019 | 0.0072 | 0.0100 | 0.0171 | 0.0505 | 0.0957 | 0.2361 |
| **Iterations PCG** | 1 | 11 | 13 | 14 | 13 | 10 | 12 | 12 |
| **Time Reorder (sec)** | 0.0004 | 0.0005 | 0.0032 | 0.0031 | 0.0064 | 0.0449 | 0.1202 | 0.3574 |
| **Time LU (sec)** | 0.0004 | 0.0013 | 0.0034 | 0.0062 | 0.0400 | 0.0405 | 0.1138 | 0.2462 |

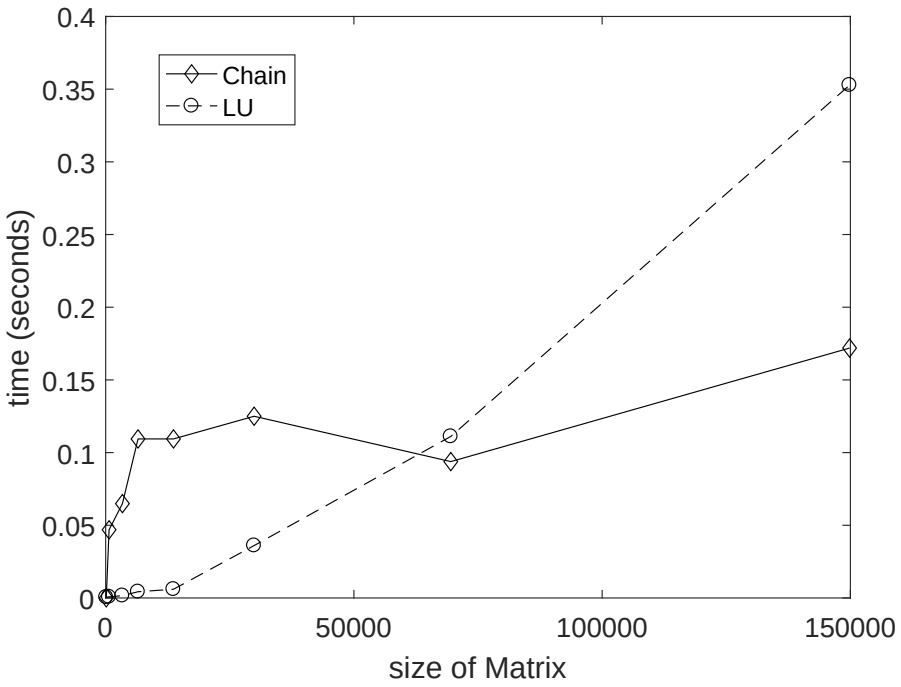


Figure 9. Comparison of the chained solver to sparse LU factorization – Set up

The solve times are the most critical times to the overall simulation runtimes, since these represent the calculations that any nonlinear solver (such as a powerflow or time-domain simulation) must perform repeatedly. The nearly linear solve time of the proposed Chain Method is clearly evident.

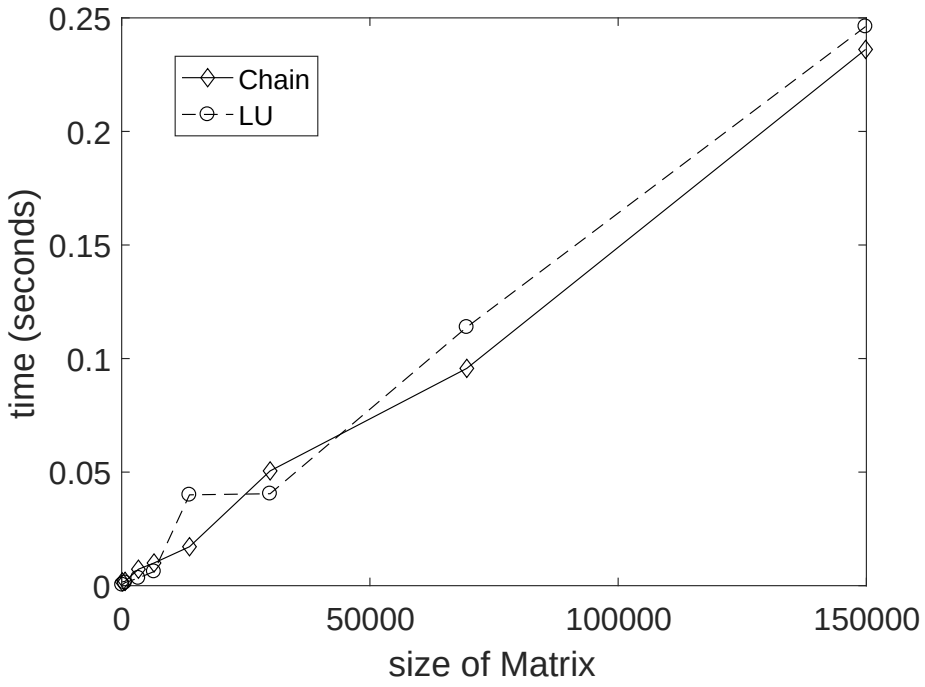


Figure 10. Comparison of the chained solver to sparse LU factorization – Solve

## 6. CONCLUSIONS AND FUTURE WORK

This paper presented algorithmic details on a fast linear solver for large sparse matrices. The focus of this paper is to analyze the algorithm's capabilities for solving systems with structures similar to typical power system matrices. Several matrices were tested to determine any future work required to develop a fast linear solver for power system specific applications. The simulation runtimes displayed a reduction in time and iterations using the chain of preconditioners technique even for larger systems when using common iterative linear solvers. Additional improvements to speed can be achieved by utilizing better computing and coding practices.

Further study and enhancements are planned to improve the speed and accuracy of this method for power system specific applications. The traditional approach in power systems for dealing with very large sparse matrices is to apply direct methods using reordering. Node reordering helps to minimize the number of computations required by reducing the number of fills or non-zero elements that are generated in the matrix decomposition [19, 20]. The authors are currently investigating the degree to which changes in the system matrix $A$ that occur during Newton-Raphson updates or time-domain simulation have on the LSST and subsequent solution times. Current results indicate that small numerical non-symmetry does not significantly impact the proposed method; however the authors are endeavoring to establish an upper bound on the amount of numerical non-symmetry is allowable.

# III. UTILIZATION OF A CHAIN LINEAR SOLVER FOR FAST DECOUPLED POWER FLOW

L. L. Grant, M. L. Crow, and M. X. Cheng

Department of Electrical & Computer Engineering

Missouri University of Science and Technology

Rolla, Missouri 65409–0050

Tel: 573–465–0657

Email: llsnpc@mst.edu

## 1. ABSTRACT

Power system analysis tools, such as power flow, state estimation, and security assessment, involve the solution of very large sparse linear systems. As power systems continue to grow and evolve, these applications require improvements to the speed of numerical algorithms in order to allow system operators the ability to adequately monitor and control systems in a time frame that is useful to industry. Improvements to linear solvers will enhance the overall computational efficiency of applications like power flow, which are needed to run time-critical analysis such as $N$-1 contingency studies and optimal power flow. A new technique, developed for the field of image processing, has shown promise as a linear solver over the traditionally used LU factorization method. This paper applies a novel chain preconditioning technique developed for solving symmetric and diagonally dominant linear systems to improve the speed and convergence of fast decoupled power flow. Several test cases are analyzed and compared to traditional load flow techniques using LU factorization along with a single line outage contingency study.

Keywords: Matrix preconditioning, graph sparsification, linear system solution, power flow, power system simulation.

## 2. INTRODUCTION

The Department of Energy released a report in April of 2011 defining the critical computational needs for the future of the electric power system [32]. Key areas of interest in the report include advancement of algorithms, simulation, and modeling of power systems, achievable in a time frame that is useful to industry. An analysis of the 13 largest power outages in history, reports that the majority of blackouts are caused by unexpected disturbances, such as weather events, control errors, and coordination failures [33]. With load growth in North America at less than 1% per year, these studies show that transmission reliability is more an issue of the need for improved modeling, simulation, and coordination methods, instead of the need for increased generation.

The power flow study is one of the most important power system applications for planing future expansion and determining the best operating points of existing power systems [34]. Power flow serves as the basis for several different power system applications, including power-system planning, operational planning, operation/control, security assessment, optimization, and stability studies [35]. The power flow problem is composed of a system of nonlinear equations that can grow to be very large when modeling full-scale systems such as the North American power grid. Electric power systems are typically solved iteratively by linearizion and the most common technique for solving the power-flow problem is the Newton-Raphson iterative method [36]. The Newton Raphson method can solve linear equations in the form $Ax = b$ (or $[\boldsymbol{J}]\Delta x = -F$ for power flow computation) using LU factorization [36]. The linear solution comprises a significant amount of the overall runtime for a power flow solution. LU based techniques on a full system matrix tend to solve on

the order of $n^3$. Sparse storage techniques, where only non-zero elements of the matrix and their location are saved in computer memory, along with ordering schemes can help to minimize the number of multiplications and divisions required for LU factorization and forward/backward substitution [19]. By taking advantage of these techniques, a sparse system can be solved by LU based methods in the order of $n^2$. For sparse systems, $\alpha = n^3 - n/3$ multiplications and divisions are required for LU factorization and $\beta = n^2$ multiplications and divisions are required for forward/backward substitution [19]. To reduce the number of multiplications and divisions, proper node ordering can be performed. Ordering methods improve computational efficiency of linear equations by reordering the system nodes in a way that the factorization of the Jacobian matrix produces a small amount of non-zero elements and results in the minimum number of fills. A fill occurs during factorization when nonzero elements are generated in the positions of the matrix that originally contained zero elements. Limiting the amount of fill-in helps to reduce the computation time and memory requirements. Different node reordering schemes can have varying effects on memory requirements and solution accuracy. Matrix ordering schemes are not ideal for ill-conditioned systems. The performance of Newton-Raphson power flow method under different ordering schemes was explored in [37] and it was concluded that if the system is ill-conditioned with instances of very high and low impedance, long EHV lines, or series and shunt compensation, it can add numerical complexity to the power flow problem and greatly change the effect of solution accuracy, convergence, and memory when choosing an ordering scheme [37, 35]. Several iterative methods have been developed to improve on the speed of the Newton-LU method [23, 38, 39, 40].

Preconditioning can reduce the number of iterations required for solution convergence by improving the condition number of the preconditioned system $M^{-1}Ax = M^{-1}b$. Recent advances in preconditioning of iterative solution methods have begun to pro-

duce competitive computational run-times and accuracy levels for power system studies [23, 41, 42, 43]. The right preconditioner can improve the spectral properties of an ill-conditioned system and reduce the computational complexity of the method [41, 42, 43]. A new iterative technique developed for solving symmetric and diagonally dominant (SDD) linear systems shows potential for accurate solutions in nearly-linear run-times. This method develops a chain of progressively smaller preconditioners to be applied using a recursive iterative technique. The chain method was shown to outperform LU as a linear solver in [44] for power system matrices up to a 11,838-bus system. This report builds on that work by applying the chain method to the linear solver portions of a power flow algorithm.

## 3. BACKGROUND

Power flow is used to compute the steady-state conditions of an electric power transmission system and involves the solution of a system of nonlinear equations to derive the voltage magnitude and phase angle at each bus, along with the real and reactive power flows. In power flow, the system bus voltages and line flows can be found if the system loads, generation, and network configuration are known [19]. There are two power flow equations at each bus, divided between the active power (1) and reactive power (2) , where $P_i^{inj}$ and $Q_i^{inj}$ are the powers injected at bus $i$.

$$0 = \Delta P_i = P_i^{inj} - V_i \sum_{j=1}^{N} V_j Y_{ij} \cos(\theta_i - \theta_j - \phi_{ij}) \tag{1}$$

$$0 = \Delta Q_i = Q_i^{inj} - V_i \sum_{j=1}^{N} V_j Y_{ij} \sin(\theta_i - \theta_j - \phi_{ij}) \tag{2}$$

The voltage magnitudes at buses $i$ and $j$ are represented by $V_i$ and $V_j$, with phase angles represented by $\theta_i$ and $\theta_j$. The network admittance matrix values are represented by $Y_{ij}\angle\phi_{ij}$. The summations in (1) and (2) are over the number of buses in the system, $N$.

The Newton-Raphson method was developed to solve a set of simultaneous nonlinear equations in an equal number of unknowns, $f(x) = 0$ [35]. The goal of the Newton-Raphson iterative method is to drive the power mismatch, $\Delta P$ and $\Delta Q$ of (1) and (2) to zero, indicating a convergence of the voltage magnitudes and phase angles, which can be used to compute the other unknown components in the system [19]. The linearized Newton-Raphson equation (3) solves the unknown voltage magnitudes and angles to satisfy the power balance equations.

$$\begin{bmatrix} \Delta\theta \\ \Delta|V| \end{bmatrix} = -J^{-1} \begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} \tag{3}$$

The power flow equations are arranged by phase angle, followed by voltage magnitude to produce the Jacobian matrix, $J$ (4). The four sub-matrices of the Jacobian represent the partial derivatives of the power balance equations with respect to the unknown voltages and angles [19].

$$\begin{bmatrix} J_1 & J_2 \\ J_3 & J_4 \end{bmatrix} = \begin{bmatrix} \frac{\partial \Delta P}{\partial \delta} & \frac{\partial \Delta P}{\partial V} \\ \frac{\partial \Delta Q}{\partial \delta} & \frac{\partial \Delta Q}{\partial V} \end{bmatrix} \tag{4}$$

The Jacobian matrix is formed at each iteration of the Newton-Raphson method. Initial values are chosen for voltage magnitudes of the load buses and voltage angles of all buses except for the slack bus. These estimates are used to update the real and reactive power equations and determine the power mismatch at each iteration. Once the estimated Jacobian and power mismatches are known, the unknown voltages and angles can be solved using

LU factorization with forward and backward substitution. The guess for voltage magnitude and angle at each iteration after the initial guess is determined from the linear solution of (3) by (5) and (6).

$$\theta^{k+1} = \theta^k + \Delta\theta \tag{5}$$

$$V^{k+1} = V^k + \Delta V \tag{6}$$

## 4. CHAIN LINEAR SOLVER

The linear solver applied in this paper is based on [3] and an analysis of the chain method technique for power system matrices can be found in [44]. A brief explanation of the method will be discussed here as it pertains to the power flow application specifically. The chain method develops a chain of progressively smaller matrices with preconditioners that approximate the original system in order to reduce computation time and resources for solving the linear system. The method was developed to produce nearly-linear runtimes at $O(m \log^2 n)$ in [3]. Simulations in [44] show the method to produce a nearly-linear solution when compared to the $n^{1.4}$ runtime of LU factorization for power system matrices up to a 11,838-bus system. The preconditioners are based on the low-stretch spanning tree of the matrix when represented by a graph [9]. There are three components of the algorithm: finding the low-stretch spanning tree (LSST) of the system $A$ matrix, forming a chain of preconditioners $C = \{A_1, M_1, A_2, M_2, ..., A_d\}$, and applying the linear solver to the chain of preconditioned matrices.

An $n \times n$ matrix $A$ can be represented by a graph with $m$ edges, which corresponds to the non-zero off-diagonal entries in the matrix. A spanning tree of a graph is a subset where all of the vertices are connected by the minimum possible number of $n - 1$ edges. The low-stretch tree is a unique type of spanning tree where the connecting edges represent the subgraph of lowest stretch. Stretch is defined as the distortion caused by the detour required by taking the tree path, where the stretch of the edges in the tree is given by:

$$stretch_T(e) = \frac{\sum_{i=1}^{k} w'(e_i)}{w'(e)} \tag{7}$$

If the stretch of an edge is small, then there are a significant number of alternative connections between its vertices; therefore, stretch is a measure of the importance of an edge in the graph network.

To form the preconditioning chain $C = \{A_1, M_1, A_2, M_2, ..., A_d\}$, the algorithm alternates between two functions. A sparsification routine called Incremental Sparsify forms the matrix preconditioner $M_1$ by reducing the matrix $A_1$ based on the low-stretch tree and a random sampling of off-tree edges. This completes the first level of the chain. Then a Greedy Elimination function is applied to the preconditioner $M_1$ to reduce the size of the matrix forming $A_2$, while preserving the system structure and important spectral properties. An incremental sparsification is performed on the reduced matrix $A_2$ to create a second preconditioner $M_2$, completing the second level of the chain. The process is repeated until the matrix is reduced to a desired point $A_d$.

For this study, the recursive Chebyshev iterative method was applied to the chain of preconditioners, $C$. A complete iterative solution is found at chain level $i = d$, and this solution is applied at the previous level $i = d - 1$ and so forth until the original system solution is found. This helps to reduce the number of iterations and overall computation

time when the matrix is of original size, especially as the system size increases. Most of the work is performed on much smaller matrices using spectrally well-conditioned approximate systems. A simple example of the stages of the chain method for an $8 \times 8$ matrix is provided in Fig. 1. There are three chain levels in the example.

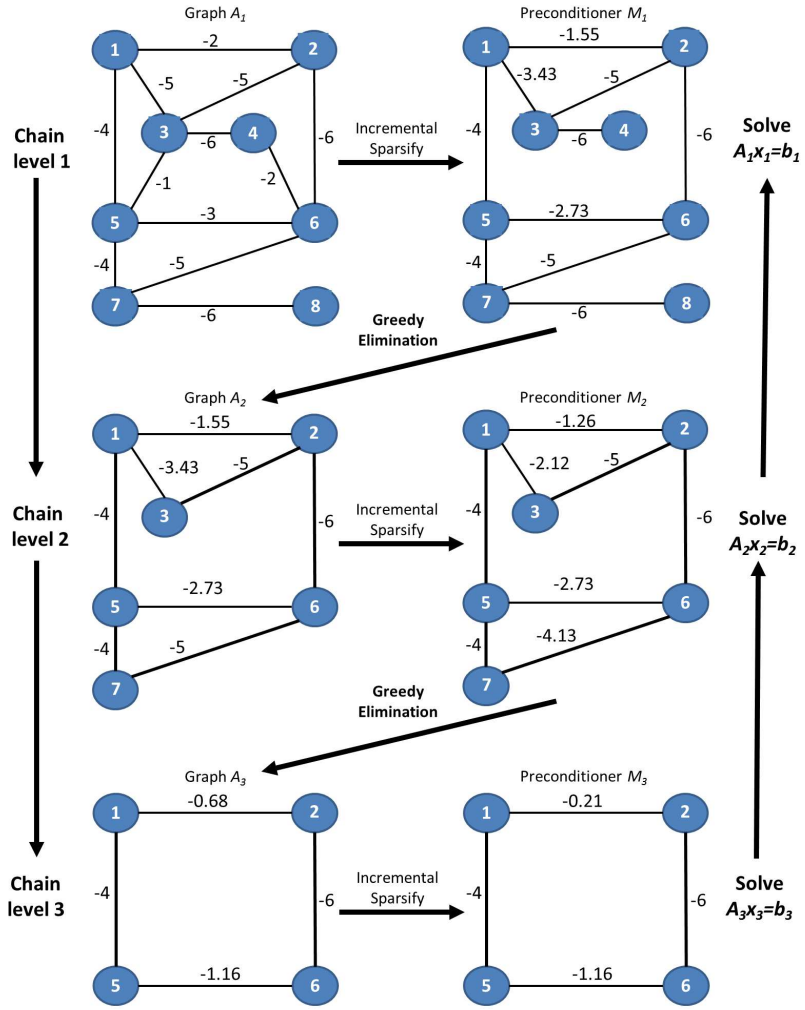


Figure 1. Chain of preconditioners for simple $8 \times 8$ example.

# 5. CHAIN METHOD APPLIED TO POWER FLOW

The Newton method's iteration runtime rises slightly more than linearly with the number of buses in the system, dependent upon the quality of the bus-ordering scheme utilized [35]. Very large systems require significant computational resources to calculate, store, and factorize the Jacobian matrix, which is updated at each iteration of the Newton-Raphson method [19]. The fill-in produced by reordering, and the computation of the Jacobian matrix, are important factors when looking at reducing computation time. An approximation method called Fast Decoupled Load Flow (FDLF) was developed to improve speed by approximating the Jacobian [35, 45]. FDLF is discussed in the following sub-sections along with considerations that are needed when applying the chain method as a linear solver to this technique.

**5.1. Fast Decoupled Power Flow (FDLF).** The Jacobian matrix can be reduced by taking advantage of the power system properties. Due to the nature of electric-power transmission system operating in steady state, there is a strong interdependence between the active power and bus voltage angles, and between the reactive power and voltage magnitudes [35]. The coupling between the off-diagonal components, $P-\theta$ and $Q-V$, is weak leading to very small numerical values in these portions of the Jacobian matrix. The Jacobian sub-matrices $J_2$ and $J_3$ in (4) can be approximated to zero, which allows the Jacobian to be replaced by $J'$ (8) without significantly affecting convergence properties.

$$J' = \begin{bmatrix} \frac{\partial \Delta P}{\partial \delta} & 0 \\ 0 & \frac{\partial \Delta Q}{\partial V} \end{bmatrix} \tag{8}$$

This approximation allows the $\Delta P$ and $\Delta Q$ iterations to be carried out independently by decoupling the power flow equations into (9) and (10) . The decoupling of the equations can reduce the number of iterations and the overall computation can be reduced by stopping the $\Delta P$ or $\Delta Q$ iterative cycle independently if one equation converges before the other.

$$\delta^{k+1} = \delta^k - \left[\frac{\partial \Delta P}{\partial \delta}\right]^{-1} \Delta P \tag{9}$$

$$V^{k+1} = V^k - \left[\frac{\partial \Delta Q}{\partial V}\right]^{-1} \Delta Q \tag{10}$$

The computation speed for power flow can be improved even further with another modification called *fast decoupled power flow*. In standard Newton-Raphson and Decoupled power flow, the Jacobian matrices are updated at each iteration, whereas FDLF holds a modified Jacobian matrix constant to minimize the number of function evaluations and LU factorizations [19]. This greatly reduces the memory storage requirements and computation time needed to achieve a power flow solution. The approximated power flow equations for the FDLF method are (11) and (12).

$$\frac{\Delta P}{V} = B' \cdot \Delta\theta \tag{11}$$

$$\frac{\Delta Q}{V} = B'' \cdot \Delta V \tag{12}$$

The $B'$ and $B''$ matrices derived in (13) and (14) are constant and are close approximations to the Jacobian $J_1$ and $J_4$ matrices at system no-load [35].

$$B'_{ij} = \frac{1}{x_{ij}}; B'_{ii} = -\sum_{j \neq i} B'_{ij} \tag{13}$$

$$B''_{ij} = b_{ij}; B''_{ii} = 2b_i - \sum_{j \neq i} B''_{ij} \tag{14}$$

The $B'$ matrix only contains the diagonal elements of the admittance matrix that are unaffected by shunts and the $B''$ matrix is composed of the shunt susceptance at each bus [19]. Composed of only network or admittance matrix elements, the matrices are simple to compute and only need to be factored once before being held constant at each iteration.

**5.2. Chain Method FDLF.** A few considerations are needed to apply the chain method linear solver to FDLF. The $B'$ and $B''$ matrices are symmetric in both value and structure, unlike the Jacobian matrix; therefore, the FDLF technique is a good candidate for applying the chain method since it was developed specifically for symmetric and diagonally dominant matrices [3]. The majority of the runtime for the chain method involves finding the low-stretch spanning tree of the $B$ matrices and forming the chain of preconditioners. The FDLF only computes the $B$ matrices once before the load flow iteration process occurs; therefore, the time required to compute the LSST and chain of preconditioners will not affect the load flow iterative solution computation time. This property of the FDLF technique is useful for applications where the load flow needs to be solved several times but the overall structure of the system doesn't change.

When the modified Jacobian, $J'$, is split into the phase angle and voltage components to form the $B$ matrices, an islanding situation can occur in the graph of the $\frac{\partial \Delta Q}{\partial V}$ portion of the $B''$ matrix. The PV bus equations are removed from this portion of the load flow to save computation because the voltage magnitudes at the PV buses are fixed and the reactive power is computed from the solved voltages and angles. A vertex or subset of vertices in a

graph is islanded if it is no longer edge-connected to the main graph. To utilize the chain linear solver, the island situation must be corrected in order to find the low-stretch tree of the matrix. The islanded edges can be reconnected to the graph by adding a trivial connecting edge with a small weight value. An edge weight of 0.01 was used because the stretch of the edge has an inverse relationship to the edge's weight value, $w' = 1/e$, which causes the trivial edge to have a high stretch. Edges of high stretch are pruned out of the low stretch tree. This allow the graph to be complete while the influence of the non-essential previously islanded vertex is phased out as the algorithm progresses.

Table 1 contains information on the condition number reduction achieved by using the $B$-matrices in the FDLF technique compared to the Jacobian matrix for the Newton-Raphson method. The condition number of a linear system provides a measure of sensitivity to numerical operations and can be used to predict the convergence rate of iterative methods. A matrix is well conditioned if its condition number is close to 1, indicating that its eigenvalues are tightly clustered, so matrices with a lower condition number will be easier to solve computationally [16]. Splitting the Jacobian into the $B$ matrices provides an initial decrease of the condition number making the FDLF systems individually easier to solve.

The chain step of the proposed algorithm can improve the system by applying several well-conditioned systems with very small condition numbers $\kappa(M_i, A_i) \ll \kappa(J)$ that achieve approximate solutions very quickly, leading to an overall solution at the chain level with the matrix of the original system size. Chebyshev iteration is used to compute the linear solution of the chain method. Since the Chebyshev iteration is guaranteed to converge after $O(\sqrt{\kappa} \log \frac{1}{\epsilon})$ iterations, and the preconditioned matrices at the smaller chain levels have condition numbers very close to 1, only a few iterations are required to solve these approximate systems [17].

Table 1. CONDITION NUMBER REDUCTION ACHIEVED USING FDLF

| No. buses | $\kappa(J)$ | $\kappa(B')$ | $\kappa(B'')$ |
|---|---|---|---|
| 14 | 234.8 | 182.4 | 37.02 |
| 89 | 11,673 | 10,187 | 1740.1 |
| 118 | 5696.5 | 4886.1 | 125.13 |
| 145 | 1.66E+5 | 1.58E+5 | 8.11E+4 |
| 1354 | 5.54E+5 | 4.79E+5 | 2.03E+4 |

## 6. RESULTS AND DISCUSSION

The MATPOWER simulation package developed for MATLAB® was used to generate simulation results. MATPOWER was created for solving power flow and is easy to modify for research purposes [46]. Two tests were performed to determine the functionality of the chain method for solving linear systems in FDLF. The convergence termination tolerance for MATPOWER load flow simulations is $10^{-5}$ per unit P and Q dispatch. The results are discussed in the following subsections.

**6.1. Power Flow.** The Chain FDLF is compared to Newton-Rapshon and regular FDLF using LU factorization. The test cases were taken from the MATPOWER files and include the following: the IEEE 14-, 118-, and 145-bus cases; the 89- and 1354- bus portions of the European transmission system from the PEGASE project [46, 31, 47].

Table 2 contains the time to perform the iterative solution (not including the time to set-up the tree and form the chain). The load flow iteration solve times are the most critical to the overall simulation runtimes, since these represent the calculations that must be performed repeatedly. The runtimes are intended to compare the methods and show that the chain method is capable of producing an adequate solution and follows the same convergence and timing trends as the FDLF. A commercial version of the chain solver would

be significantly faster than the runtime performance presented in table 2 and would better compare the results achieved by the commercial solutions derived using the MATPOWER code for Newton-Raphson and traditional FDLF.

Table 2. COMPARISON OF LOAD FLOW SIMULATION RUNTIMES (SECONDS)

| No. buses | Newton-Raphson | Fast-Decoupled | Chain Fast-Decoupled |
|---|---|---|---|
| 14 | 0.0865 | 0.0146 | 0.0567 |
| 89 | 0.1196 | 0.0163 | 0.1477 |
| 118 | 0.0563 | 0.0127 | 0.1453 |
| 145 | 0.0979 | 0.0148 | 0.4063 |
| 1354 | 0.2737 | 0.0407 | 0.8773 |

Table 3 presents the number of load-flow iterations required for convergence by the three methods. For the 14-, 89-, and 1354-bus systems, the FDLF-Chain method solves in the same number of iterations as the FDLF-LU method. The other two cases require a few more iterations for the Chain method to converge.

Table 3. COMPARISON OF LOAD FLOW SIMULATION ITERATIONS

| No. buses | Newton-Raphson | Fast-Decoupled | Chain Fast-Decoupled |
|---|---|---|---|
| 14 | 2 | 4 | 4 |
| 89 | 5 | 5 | 5 |
| 118 | 3 | 4 | 5 |
| 145 | 3 | 7 | 9 |
| 1354 | 4 | 6 | 6 |

The timings in table 2 show that the Chain method follows a similar trend to the other two methods where the 118-bus system solves a little faster than the 89-bus system even though more iterations are required for the Chain method for that case. The 118-bus case is of interest in regards that the $B''$ matrix reduces to a spanning tree in itself, which can be seen in Fig. 3. Figs. 2 and 3 show the structure of the $B$ matrices for the 118-bus

system with $B$ matrix values represented by ‘x’ symbols and the tree values represented by ‘o’ symbols. Comparing the two figures, the $B'$ matrix has a few off-diagonal values that are not included in the tree, but the $B''$ matrix tree contains all of the $B''$ values. This does not allow for a chain of preconditioners to be formed for the $B''$ matrix. It would be best to solve the 118-bus $B''$ matrix directly since it cannot be further reduced to gain the benefit of the Chain method.

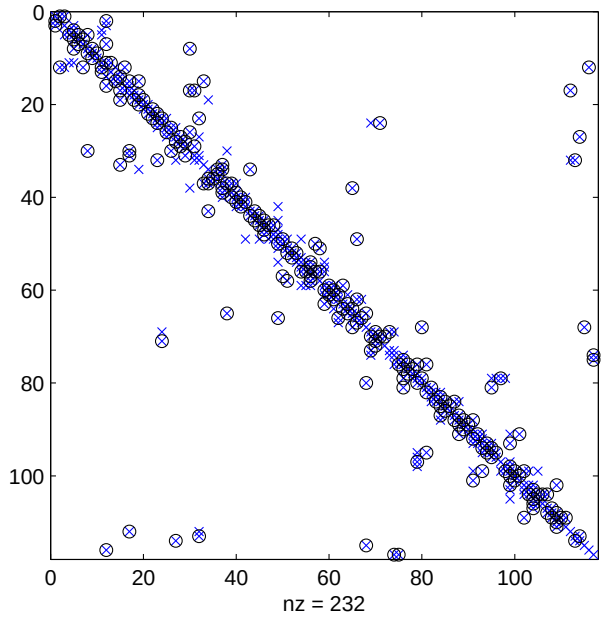


Figure 2. 118-bus $B'$ matrix (x) with tree values (o).

Fig. 4 shows the 118-bus $P$ and $Q$ mismatch convergence for both the FDLF and FDLF-Chain methods. The Chain method tracks the FDLF method’s convergence characteristics for the first few iterations, then the $Q$ mismatch of the Chain method begins to diverge at the 5th iteration. The convergence characteristics of the Chain method are more aligned with the convergence of the FDLF algorithm for the other test cases. This leads to another consideration for choosing systems to solve using the Chain method. Table 4 presents a metric to show the level of diagonal dominance of the Jacobian and $B$ matrices

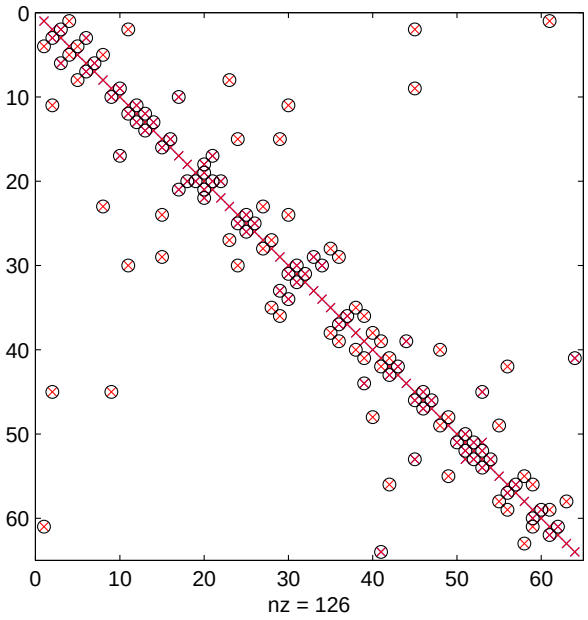


Figure 3. 118-bus $B''$ matrix (x) with tree values (o).

of the test cases. The Chain method was developed to solve SDD systems, so matrices with weak diagonal dominance may not perform as well using this method. A matrix is diagonally dominant if the magnitude of the diagonal entry $a_{ii}$ in row $i$ is greater than or equal to the sum of the magnitudes of all other non-diagonal entries in that row (15).

$$| a_{ii} | \geq \sum_{j \neq i} | a_{ij} | \quad \text{for all} \quad i \tag{15}$$

The metric in Table 4 divides the sum of the matrix off-diagonal weights by the sum of the matrix diagonal to form a percentage. A value of 100% would indicate that the density of the off-diagonal weights is equal to the density of the diagonal; therefore, the matrix is neutrally diagonally dominant. Percentages over 100% indicate a matrix has stronger

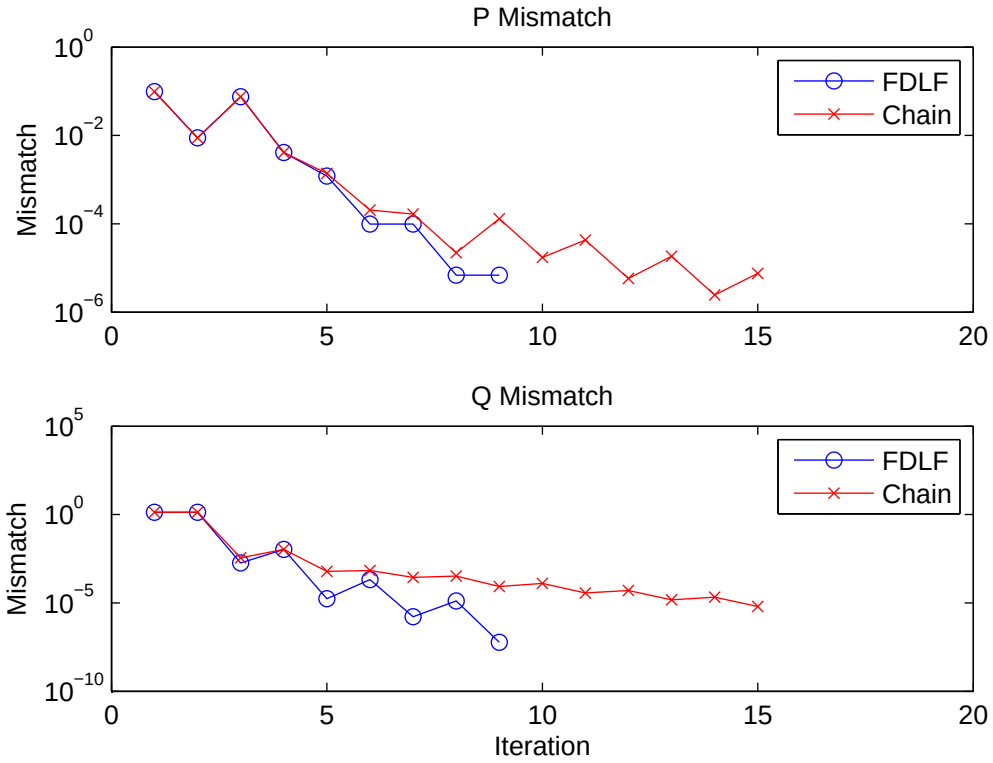


Figure 4. Comparison of P and Q mismatch convergence of the 118-bus system for FDLF and Chain-FDLF methods.

Table 4. DIAGONAL DOMINANCE OF THE TEST CASE MATRICES

| No. buses | Jacobian | $B'$ | $B''$ |
|---|---|---|---|
| 14 | 134% | 92% | 74% |
| 89 | 98% | 100% | 81% |
| 118 | 107% | 99% | 39% |
| 145 | 110% | 101% | 98% |
| 1354 | 133% | 100% | 80% |

diagonal dominance. A low percentage like the 39% for the $B''$ matrix of the 118-bus system indicates the matrix is not diagonally dominant and provides insight to the difficulties the chain method had in solving the system. Matrices with values slightly below 100% may

not be diagonally dominant overall, but several of the rows may still have a strong diagonal dominance. All of the Jacobian matrices are diagonally dominant, but are not symmetric in values.

**6.2. System Topology Changes.** Another study was performed to show where the Chain method has potential to really benefit the power flow application. Power flow simulation is an important component to ensure the security of the power grid. Power systems are designed to run within allowable operating limits during normal real-time operation and under any predicted contingencies. Contingencies can range from a single transmission line outage to several line outages with multiple generators rendered inoperable. To perform a typical contingency study, system operators start with a solved power flow case when the system is under normal steady-state conditions. Then each contingency is checked one-by-one and the power flow is re-solved to see how the outage affects system performance and voltage limits, along with determining what preventative actions are required to keep the system operational. A single study can require the assessment of up to $N-1$ contingencies, which corresponds to running $N-1$ power flow simulations, where $N$ is the number of buses. Contingency studies of very large systems can take several hours to complete even when using a power flow solution method that runs in a few seconds.

Line outages affect the admittance matrix of the system, which is used to compute the $B$ matrices for FDLF. Since the $B$ matrices are used to develop the LSST for the chain method, it is prudent to determine how much the LSST will change if a line outage contingency occurs. The value of the LSST is that it represents the system structure and doesn't necessarily need to be updated if the system weights change slightly as long as the structure remains the same. Table 5 compares data for three different power flow tests on the 89-, 118-, and 1354-bus systems solved using the FDLF-Chain method. The tests include the following:

1. Original test case

2. Single line outage case solved with a tree generated from the system under contingency

3. Single line outage case solved with the tree generated from the original test case

Table 5. SINGLE LINE OUTAGE SIMULATION DATA

| | 89-bus | | 118-bus | | 1354-bus | |
|---|---|---|---|---|---|---|
| Test No. | Time | Iters | Time | Iters | Time | Iters |
| 1 | 0.2984 | 5 | 0.2142 | 6 | 2.6329 | 6 |
| 2 | 0.3083 | 6 | 0.1797 | 6 | 3.3256 | 6 |
| 3 | 0.2775 | 7 | 0.1569 | 6 | 2.9987 | 9 |

The table shows the time required to solve each case and the number of iterations to convergence. In all three cases, the FDLF-Chain method solves the line outage faster using the original tree than using an updated tree, even though it may require a few more iterations. This suggests that the tree does not need to be update to run contingency studies for small system changes, which can save a lot of computation time for critical applications.

## 7. CONCLUSION

The chain method linear solver has been proposed for use in fast decoupled power flow to improve speed of convergence and exploit the properties of symmetry found in power system matrices. Simulation results on several MATPOWER test cases show that the chain linear solver can produce solutions within nearly the same number of iterations and mismatch as the FDLF method using LU factorization. Better coding practices such as implementing parallel computing of the decoupled equations can improve the runtime performance of the chain method to make it comparable to commercial solvers like MATPOWER. The structure of power system test matrices was investigated on how the symmetry, diagonal

dominance, and condition number reduction affect performance of the chain linear solver. The method was shown to have an advantage for contingency studies of a single-line outage by eliminating the need to reconfigure the LSST for small system changes, which can reduce the computation time required for large system security evaluation.

# SECTION

# 2. CONCLUSION

In Paper 1, a new linear solver was proposed for use in power system computation. The method was created to achieve nearly-linear runtimes when solving systems with symmetric and diagonally dominant matrices. The proposed technique recursively solves a chain of preconditioned sparse linear systems using the Chebyshev iterative method. The chain linear solver is compared to traditional Chebyshev iteration in simulations of three actual power system matrices. The simulation results show that the chain method reduces the number of iterations and runtime for all three test matrices when compared to a non-chain approach. Analysis of the chain method is provided to show how it improves the condition number of the linear systems by using a low stretch spanning tree preconditioner.

In Paper 2, the performance of the chain linear solver was demonstrated on several large power system matrices and compared to the runtime potential of LU factorization. More detail is given on the specifics of the method and graphics are provided to show how the method derives the chain of preconditioners for a small matrix example. The low stretch spanning tree preconditioner is compared with four well-established preconditioning techniques and shown to reduce the number of iterations required for the preconditioned conjugate gradient method. The linear solver is simulated on several large power system matrices and compared to sparse LU factorization. The chain method achieved a better runtime convergence trend than the $n^{1.4}$ runtime of LU factorization for up to a 150,000-bus system.

In Paper 3, the chain linear solver was demonstrated for use in the power flow application. The technique was chosen for use as the linear solver in fast decoupled power flow based on the symmetric nature of the power flow method's approximated $B$ matrices. Several cases were tested using MATPOWER programs to compare the FDLF-chain method's capabilities with Newton-Raphson and FDLF using LU factorization. The FDLF-chain method produced similar timing characteristics, though better coding practices are needed to put the method on the same level as the timing achieved by the commercial Newton-Raphson and FDLF solvers. A single-line outage contingency simulation demonstrated a potential advantage of this method for reducing processing time by re-using the original system tree for applications that require several repeated load flow solutions.

# BIBLIOGRAPHY


[1] Michael Elkin, Yuval Emek, Daniel A Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. *SIAM Journal on Computing*, 38(2):608–628, 2008.

[2] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[3] Ioannis Koutis, Gary L Miller, and Richard Peng. Approaching optimality for solving sdd linear systems. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 235–244. IEEE, 2010.

[4] Daniel A Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90. ACM, 2004.

[5] Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

[6] Daniel A Spielman and Shang-Hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM Journal on Matrix Analysis and Applications*, 35(3):835–885, 2014.

[7] Pravin M Vaidya. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. *A talk based on this manuscript*, 2 (3.4):2–4, 1991.

[8] Michele Benzi. Preconditioning techniques for large linear systems: a survey. *Journal of computational Physics*, 182(2):418–477, 2002.

[9] Erik Boman and Bruce Hendrickson. On spanning tree preconditioners. *Manuscript, Sandia National Lab*, 3, 2001.

[10] Daniel A Spielman and Jaeoh Woo. A note on preconditioning by low-stretch spanning trees. *arXiv preprint arXiv:0903.2816*, 2009.

[11] Marshall Bern, John R Gilbert, Bruce Hendrickson, Nhat Nguyen, and Sivan Toledo. Support-graph preconditioners. *SIAM Journal on Matrix Analysis and Applications*, 27(4):930–951, 2006.

[12] SIVAN TOLEDO. Vaidya's preconditioners: Implementation and experimental study. *Electronic Transactions on Numerical Analysis*, 16:30–49, 2003.

[13] Uwe Naumann and Olaf Schenk. *Combinatorial scientific computing*. CRC Press, 2012.

[14] Ioannis Koutis, Gary L Miller, and Richard Peng. A fast solver for a class of linear systems. *Communications of the ACM*, 55(10):99–107, 2012.

[15] Bruce M Maggs, Gary L Miller, Ojas Parekh, R Ravi, and Shan Leung Maverick Woo. Solving symmetric diagonally-dominant systems by preconditioning. In *IN PROCEEDINGS. 38TH ANNUAL SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE*. Citeseer, 2002.

[16] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[17] Ioannis Koutis, Gary L Miller, and David Tolliver. Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. In *International Symposium on Visual Computing*, pages 1067–1078. Springer, 2009.

[18] Timothy A Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1, 2011.

[19] Mariesa L Crow. *Computational methods for electric power systems*. Crc Press, 2015.

[20] Krishna M Sambarapu and S Mark Halpin. Sparse matrix techniques in power systems. In *System Theory, 2007. SSST'07. Thirty-Ninth Southeastern Symposium on*, pages 194–198. IEEE, 2007.

[21] Siddhartha Kumar Khaitan, James D McCalley, and Qiming Chen. Multifrontal solver for online power system time-domain simulation. *IEEE Transactions on Power Systems*, 23(4):1727–1737, 2008.

[22] Fernando L Alvarado. Computational complexity in power systems. *IEEE Transactions on Power Apparatus and Systems*, 95(4):1028–1037, 1976.

[23] Yi-Shan Zhang and Hsiao-Dong Chiang. Fast newton-fgmres solver for large-scale power flow study. *IEEE Transactions on Power Systems*, 25(2):769–776, 2010.

[24] Tsung-Hao Chen and Charlie Chung-Ping Chen. Efficient large-scale power grid analysis based on preconditioned krylov-subspace iterative methods. In *Design Automation Conference, 2001. Proceedings*, pages 559–562. IEEE, 2001.

[25] IA Adejumobi and GA Adepoju. Iterative techniques for load flow study: A comparative study for nigeria 330 kv grid system as a case study. *International Journal of Engineering Advanced Technology*, 3:153–157, 2013.

[26] Ioannis Koutis, Gary L Miller, and Richard Peng. A nearly-m log n time solver for sdd linear systems. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 590–598. IEEE, 2011.

[27] Elena Caraba. *Preconditioned Conjugate Gradient Algorithm*. PhD thesis, Louisiana State University, 2008.

[28] S Ashby, T Barth, T Mantueffel, and P Saylor. A tutorial on iterative methods for linear algebraic systems. *University of Illinois*, 1996.

[29] Thomas A Manteuffel. The tchebychev iteration for nonsymmetric linear systems. *Numerische Mathematik*, 28(3):307–327, 1977.

[30] Steven F Ashby. Chebycode, a fortran implementation of manteuffel's adaptive chebyshev algorithm. *ACM Digital Library*, 1985.

[31] Cédric Josz, Stéphane Fliscounakis, Jean Maeght, and Patrick Panciatici. Ac power flow data in matpower and qcqp format: itesla, rte snapshots, and pegase. *arXiv preprint arXiv:1603.01533*, 2016.

[32] Joseph H Eto and Robert J Thomas. Computational needs for the next generation electric grid. *Department of Energy*, 593, 2011.

[33] M Jacobs. of the largest power outages in history-and what they tell us about the 2003 northeast blackout. *Union of Concerned Scientists: http://blog. ucsusa. org/mike-jacobs/2003-northeast-blackout-and-13-of-the-largest-power-outages-in-history-199 adresinden alınmıştır*, 13.

[34] John J Grainger and William D Stevenson. *Power system analysis*. McGraw-Hill, 1994.

[35] Brian Stott. Review of load-flow calculation methods. *Proceedings of the IEEE*, 62 (7):916–929, 1974.

[36] Xue Li and Fangxing Li. Estimation of the largest eigenvalue in chebyshev preconditioner for parallel conjugate gradient method-based power flow computation. *IET Generation, Transmission & Distribution*, 10(1):123–130, 2016.

[37] Wenbo Li, Xueshan Han, and Bo Zhang. A comparison of power flow by different ordering schemes. In *Electric Utility Deregulation and Restructuring and Power Technologies (DRPT), 2011 4th International Conference on*, pages 742–745. IEEE, 2011.

[38] Hiroyuki Mori, Hideya Tanaka, and Junya Kanno. A preconditioned fast decoupled power flow method for contingency screening. In *Power Industry Computer Application Conference, 1995. Conference Proceedings., 1995 IEEE*, pages 262–268. IEEE, 1995.

[39] MA Pai and H Dag. Iterative solver techniques in large scale power system computation. In *Decision and Control, 1997., Proceedings of the 36th IEEE Conference on*, volume 4, pages 3861–3866. IEEE, 1997.

[40] F De Leon and A Semlyen. Iterative solvers in the newton power flow problem: preconditioners, inexact solutions, and partial jacobian updates. *IEE Proceedings-Generation, Transmission and Distribution*, 149(4):479–484, 2002.

[41] Adam Semlyen. Fundamental concepts of a krylov subspace power flow methodology. *IEEE Transactions on Power Systems*, 11(3):1528–1537, 1996.

[42] Alexander J Flueck and Hsiao-Dong Chiang. Solving the nonlinear power flow equations with an inexact newton method using gmres. *IEEE Transactions on Power Systems*, 13(2):267–273, 1998.

[43] AB Alves, EN Asada, and A Monticelli. Critical evaluation of direct and iterative methods for solving ax= b systems in power flow calculations and contingency analysis. In *Power Industry Computer Applications, 1999. PICA'99. Proceedings of the 21st 1999 IEEE International Conference*, pages 15–21. IEEE, 1999.

[44] Lisa L Grant, Mariesa L Crow, and Maggie X Cheng. A chain method for preconditioned iterative linear solvers for power system matrices. *IEEE Transactions on Power Systems*, 2017.

[45] Brian Stott and Of Alsac. Fast decoupled load flow. *IEEE transactions on power apparatus and systems*, PAS-93(3):859–869, 1974.

[46] Ray Daniel Zimmerman, Carlos Edmundo Murillo-Sánchez, and Robert John Thomas. Matpower: Steady-state operations, planning, and analysis tools for power systems research and education. *IEEE Transactions on power systems*, 26(1):12–19, 2011.

[47] Stéphane Fliscounakis, Patrick Panciatici, Florin Capitanescu, and Louis Wehenkel. Contingency ranking with respect to overloads in very large power systems taking into account uncertainty, preventive, and corrective actions. *IEEE Transactions on Power Systems*, 28(4):4909–4917, 2013.

# VITA

Lisa L. Grant received her B.S. degree in Electrical Engineering from the University of Missouri-Rolla, MO, USA, in 2007 and her M.S. degree in Electrical Engineering from Missouri University of Science and Technology, Rolla, MO, USA, in 2011. In the summers of 2012 and 2013, she was a Graduate Student Intern with Sandia National Laboratories in Albuquerque, New Mexico. She received her PhD. in Electrical Engineering from Missouri University of Science and Technology in May 2017. Her research interests include algorithms and computational methods, smart grid, power system simulation and modeling, optimization, robotics, and computational intelligence.