Augmented Matrix Solvers for Dynamic Linear Systems of Equations

Yu Hong Yeung

Purdue University

Follow this and additional works at: https://docs.lib.purdue.edu/open_access_dissertations

Recommended Citation
https://docs.lib.purdue.edu/open_access_dissertations/1665

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.
AUGMENTED MATRIX SOLVERS FOR DYNAMIC SYSTEM OF EQUATIONS

A Dissertation
Submitted to the Faculty
of
Purdue University
by
Yu Hong Yeung

In Partial Fulfillment of the
Requirements for the Degree
of
Doctor of Philosophy

December 2017
Purdue University
West Lafayette, Indiana
THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Alex Pothen, Chair
    Department of Computer Science
Dr. Ahmed Sameh
    Department of Computer Science
Dr. David Gleich
    Department of Computer Science
Dr. Xavier Tricoche
    Department of Computer Science
Dr. Jessica Crouch
    Department of Computer Science, Old Dominion University

Approved by:
    Dr. Voicu Popescu by Dr. William J. Gorman
        Head of the Department Graduate Program
To my parents and brother,

Wah Yiu Yeung, Kam Hoi Ng, and Yu Hin Yeung
ACKNOWLEDGMENTS

Alex Pothen has been an extraordinary adviser throughout my graduate study. He has always been exceptionally patient, kind and supportive. This dissertation would not be possible without his introduction of augmented matrix system to me, and his tremendous amount of time he devoted to me and this research. I am grateful to his invaluable guidance and generous supports, both financial and intangible, whenever I need them the most. I will always have fond memories of the warm friendship he established with me and all his students.

I would like to thank the members of my thesis committee: Ahmed Sameh, David Gleich, Xavier Tricoche, and Jessica Crouch for taking the time to review my dissertation and giving constructive feedbacks during my defense. In addition to his long travel from Twin Cities to my defense, Ahmed Sameh has also provided me the vast knowledge on numerical linear algebra whenever I encountered difficulties in my research. Jessica Crouch has been working hard with me on coding, providing the data for the experiments and writing the papers.

I am grateful to Mahantesh Halappanavar for being my mentor, paper coauthor and friend during my summer internships at Pacific Northwest National Laboratory, and Zhenyu Huang for introducing me the power-flow problem and providing various supports. I would also like to thank Mallikarjuna Vallem for providing the data for the power-flow experiments.

I also appreciate the helps I got from my fellow colleagues and friends. Ariful Azad, Arif Khan, Mu Wang, Xin Cheng, Ahmed Al-Herz, S. M. Ferdous, Shivaram Gopal, and Ye Chen have provided helpful comments and discussions on research and presentation. Alicia Klinvex and Zhengyi Zhang have also spent considerable time in discussions on research and knowledge on software packages.
I would also like to express my sincere gratitude to my first adviser at Purdue, Daniel Aliaga for opening the doors of research for me. He guided me through my first few years as an inexperienced student and showed me how research is done. He has still been willing to help whenever I need assistance. My fellow colleagues and friends in the Computer Graphics and Visualization Lab have also been very helpful and friendly.

Additionally, I would like to thank William Gorman for helping me with the formatting of this dissertation. Other employees of the Department of Computer Science have also provided great assistance during my study. I am also quite grateful to Gary Chan at the Hong Kong University of Science and Technology for his mentoring on my undergraduate research, and providing the job opportunity during and after bachelor’s degree. A special thank is also given to Justin Gentry to help proofreading my dissertation, and to Basil and Deborah Parker and their family for their enormous kindness and hospitality in Indiana.

Last but not least, I would like to thank my family for their unconditional love and support during this long and arduous process of my doctoral study. My parents have always been caring even I have been far away from them. And my elder brother has always been understanding and taken all the family responsibilities for me.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>ix</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>x</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>xii</td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2 AUGMENTED SYSTEM FORMULATION</td>
<td>7</td>
</tr>
<tr>
<td>2.1 Adding a row and a column</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Removing a row and a column</td>
<td>8</td>
</tr>
<tr>
<td>2.3 Replacing a row and a column</td>
<td>8</td>
</tr>
<tr>
<td>2.4 Replacing multiple rows and columns</td>
<td>9</td>
</tr>
<tr>
<td>3 UNSYMMETRIC COLUMN UPDATES</td>
<td>13</td>
</tr>
<tr>
<td>3.1 Replacing a column</td>
<td>13</td>
</tr>
<tr>
<td>3.2 Replacing multiple columns</td>
<td>14</td>
</tr>
<tr>
<td>3.3 Dimension change</td>
<td>15</td>
</tr>
<tr>
<td>3.4 Solution method</td>
<td>15</td>
</tr>
<tr>
<td>3.5 Exploiting sparsity</td>
<td>17</td>
</tr>
<tr>
<td>3.5.1 Theory</td>
<td>17</td>
</tr>
<tr>
<td>3.5.2 Application in the algorithm</td>
<td>20</td>
</tr>
<tr>
<td>3.6 Complexity analysis</td>
<td>21</td>
</tr>
<tr>
<td>3.7 Preconditioning</td>
<td>24</td>
</tr>
<tr>
<td>3.8 Refactorization</td>
<td>24</td>
</tr>
<tr>
<td>4 AMPS: AUGMENTED METHOD WITH PRINCIPAL SUBMATRIX UPDATE</td>
<td>27</td>
</tr>
<tr>
<td>4.1 Principal submatrix update</td>
<td>27</td>
</tr>
<tr>
<td>4.2 Solution method</td>
<td>28</td>
</tr>
<tr>
<td>4.3 Solution to the modified system</td>
<td>30</td>
</tr>
<tr>
<td>4.4 Relation to the Sherman-Morrison-Woodbury formula</td>
<td>31</td>
</tr>
<tr>
<td>4.5 Forming the Schur complement $S_2$ explicitly</td>
<td>31</td>
</tr>
<tr>
<td>4.6 Complexity analysis</td>
<td>33</td>
</tr>
<tr>
<td>4.7 Comparison with other augmented methods</td>
<td>35</td>
</tr>
<tr>
<td>5 AMPS WITH DIMENSION CHANGE</td>
<td>39</td>
</tr>
<tr>
<td>5.1 Formulation</td>
<td>39</td>
</tr>
<tr>
<td>5.2 Solutions numerical accuracy improvement</td>
<td>43</td>
</tr>
</tbody>
</table>
5.3 Computing the principal submatrix of the inverse ........................................ 43
5.3.1 Memoization ......................................................................................... 44
5.4 Dimension shrinking .................................................................................. 46
5.5 Complexity analysis .................................................................................... 47
5.6 Parallelization ............................................................................................. 48
5.7 Relation to column replacing augmented formulation .................................... 48

6 APPLICATION TO SURGICAL SIMULATION ............................................. 50
6.1 Finite element model ................................................................................... 50
6.2 Linear tetrahedron ...................................................................................... 51
6.3 Barycentric coordinates ............................................................................... 51
6.4 Coordinate transformations .......................................................................... 52
6.5 Displacement interpolation .......................................................................... 54
6.6 The strain field ............................................................................................ 54
6.7 The stress field ........................................................................................... 55
6.8 The element stiffness matrix ....................................................................... 56
6.9 Global stiffness matrix assembly ................................................................. 57
6.10 Boundary conditions .................................................................................. 59
6.11 Topological mesh changes ......................................................................... 63
6.11.1 Element changes .................................................................................. 63
6.11.2 Node changes ....................................................................................... 64
6.12 Unifying the formulation ............................................................................ 66
6.13 Previous work ............................................................................................ 69
6.13.1 Precomputation approaches ................................................................. 69
6.13.2 Iterative solvers .................................................................................... 71
6.13.3 Hybrid solution methods ....................................................................... 73

7 APPLICATION TO $N - x$ CONTINGENCY ANALYSIS OF POWER-FLOW74
7.1 Power flow equation .................................................................................... 75
7.2 “DC” power-flow approximation ................................................................. 77
7.3 Bus classification ........................................................................................ 79
7.4 Contingency analysis ................................................................................... 79
7.5 Previous work ............................................................................................. 80
7.5.1 Contingency selection ........................................................................... 80
7.5.2 Parallel computing ................................................................................. 81

8 EXPERIMENTAL RESULTS ........................................................................ 82
8.1 Implementation ............................................................................................ 82
8.2 Matrix format ............................................................................................... 83
8.3 Target hardware ........................................................................................... 83
8.4 Surgical simulations ..................................................................................... 84
8.4.1 Model meshes ....................................................................................... 84
8.4.2 Experiments ............................................................................................ 88
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.5</td>
<td>$N - x$ contingency analyses</td>
<td>109</td>
</tr>
<tr>
<td>8.5.1</td>
<td>Power systems</td>
<td>109</td>
</tr>
<tr>
<td>8.5.2</td>
<td>Experiments</td>
<td>109</td>
</tr>
<tr>
<td>9</td>
<td>FUTURE WORK</td>
<td>118</td>
</tr>
<tr>
<td>9.1</td>
<td>Further parallelization</td>
<td>118</td>
</tr>
<tr>
<td>9.2</td>
<td>Application to nonlinear systems</td>
<td>119</td>
</tr>
<tr>
<td>10</td>
<td>CONCLUSION</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>REFERENCES</td>
<td>123</td>
</tr>
<tr>
<td></td>
<td>APPENDIX: HOLOMORPHIC EMBEDDING LOAD FLOW METHOD</td>
<td>130</td>
</tr>
<tr>
<td></td>
<td>VITA</td>
<td>137</td>
</tr>
</tbody>
</table>
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Time complexity of the column replacing augmented matrix method</td>
<td>23</td>
</tr>
<tr>
<td>4.1</td>
<td>Time complexity of the AMPS direct and iterative methods</td>
<td>33</td>
</tr>
<tr>
<td>5.1</td>
<td>Time complexity of the AMPS method with dimension changes</td>
<td>47</td>
</tr>
<tr>
<td>6.1</td>
<td>Lookup table $T_{e,i}$ for the mesh shown in Figure 6.2.</td>
<td>58</td>
</tr>
<tr>
<td>7.1</td>
<td>Types of buses and their known and unknown variables</td>
<td>79</td>
</tr>
<tr>
<td>8.1</td>
<td>Maximum number of GMRES iterations vs. mesh sizes</td>
<td>106</td>
</tr>
<tr>
<td>8.2</td>
<td>Comparison of average relative residual norms ($|\hat{K}\hat{a} - \hat{f}|_2/|\hat{f}|_2$) for the cutting experiments</td>
<td>107</td>
</tr>
<tr>
<td>8.3</td>
<td>Comparison of single-core and multi-core average timings</td>
<td>107</td>
</tr>
<tr>
<td>8.4</td>
<td>Comparison of average relative residual norms ($|\hat{A}\hat{x} - \hat{b}|_2/|\hat{b}|_2$) for the power-flow contingency analyses</td>
<td>117</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 A lower triangular matrix $L$ and its corresponding directed graph $G(L)$ with closure$_L$(${6}$) outlined.</td>
<td>22</td>
</tr>
<tr>
<td>3.2 Refactorization process.</td>
<td>25</td>
</tr>
<tr>
<td>4.1 Example of running CHOLMOD on a matrix $A$ with principal submatrix update.</td>
<td>38</td>
</tr>
<tr>
<td>5.1 Example of a modified matrix $\hat{A}$ formed by a principal submatrix update $E$ highlighted in blue to the original matrix $A$ highlighted in grey with dimension change.</td>
<td>39</td>
</tr>
<tr>
<td>6.1 A tetrahedron with a point $P$ inside it. The tetrahedron enclosed by $P$ and the face opposite to node 1 is shaded and its volume is $v_{P1}$.</td>
<td>51</td>
</tr>
<tr>
<td>6.2 2D mesh containing 9 nodes and 8 triangular elements. Individual degrees of freedom of each element at each node are numbered in red.</td>
<td>58</td>
</tr>
<tr>
<td>6.3 A tetrahedron is being cut from top to bottom. Two nodes are created at the intersection of the cutting plane and an edge. The node intersected by the cutting plane is duplicated.</td>
<td>65</td>
</tr>
<tr>
<td>7.1 An example of a power system with a contingency.</td>
<td>76</td>
</tr>
<tr>
<td>8.1 Rendering of the beam mesh with cut.</td>
<td>84</td>
</tr>
<tr>
<td>8.2 Rendering of the brick mesh with cut.</td>
<td>85</td>
</tr>
<tr>
<td>8.3 Rendering of the Stanford bunny mesh with cut on its back.</td>
<td>86</td>
</tr>
<tr>
<td>8.4 Rendering of the eye model are shown with incisions along the corneal limbus, to correct for astigmatism, used in the experiments.</td>
<td>86</td>
</tr>
<tr>
<td>8.5 Rendering of the brain model are shown with incisions on the superior portion of the right frontal lobe used in the experiments.</td>
<td>87</td>
</tr>
<tr>
<td>8.6 $</td>
<td>\text{closure}_L(J\hat{x}_2)</td>
</tr>
<tr>
<td>8.7 Update rates for the deformation experiments of beam mesh of 6,400 nodes.</td>
<td>89</td>
</tr>
<tr>
<td>8.8 Update rates for the deformation experiments of brick mesh of 9,537 nodes.</td>
<td>90</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>8.9</td>
<td>Average update rates and ranges are shown for the deformation experiments of the series of beam meshes.</td>
</tr>
<tr>
<td>8.10</td>
<td>Average update rates and ranges are shown for the deformation experiments of the series of brick meshes.</td>
</tr>
<tr>
<td>8.11</td>
<td>A portion of the tetrahedral brick test mesh. Node A has 13 connected nodes (colored in orange) whereas Node B only has 5 (colored in red).</td>
</tr>
<tr>
<td>8.12</td>
<td>Update rates for the cutting experiments of beam mesh of 6, 400 nodes.</td>
</tr>
<tr>
<td>8.13</td>
<td>Update rates for the cutting experiments of brick mesh of 9, 537 nodes.</td>
</tr>
<tr>
<td>8.14</td>
<td>Average update rates and ranges are shown for the cutting experiments of the series of beam meshes.</td>
</tr>
<tr>
<td>8.15</td>
<td>Average update rates and ranges are shown for the cutting experiments of the series of brick meshes.</td>
</tr>
<tr>
<td>8.16</td>
<td>Update rates for the cutting experiments of Stanford bunny mesh of 20, 133 nodes.</td>
</tr>
<tr>
<td>8.17</td>
<td>Breakdown of the times for the cutting experiments of the Stanford bunny mesh using two augmented matrix methods.</td>
</tr>
<tr>
<td>8.18</td>
<td>Update rates for the cutting experiments of eye mesh of 4, 444 nodes.</td>
</tr>
<tr>
<td>8.19</td>
<td>Update rates for the cutting experiments of eye mesh of 16, 176 nodes.</td>
</tr>
<tr>
<td>8.20</td>
<td>Update rates for the cutting experiments of brain mesh of 23, 736 nodes.</td>
</tr>
<tr>
<td>8.21</td>
<td>Update rates for the cutting experiments of brain mesh of 50, 737 nodes.</td>
</tr>
<tr>
<td>8.22</td>
<td>Eigenspectrum of the eye mesh of 4, 444 nodes.</td>
</tr>
<tr>
<td>8.23</td>
<td>Running times for the contingency analysis experiments</td>
</tr>
<tr>
<td>8.24</td>
<td>Breakdown of the running times of AMPS for the 3, 120-bus Polish system</td>
</tr>
<tr>
<td>8.25</td>
<td>Breakdown of the running times of CHOLMOD and LUSOL for the 3, 120-bus Polish system</td>
</tr>
<tr>
<td>8.26</td>
<td>Breakdown of the running times of AMPS for the 777, 646-bus generated system</td>
</tr>
<tr>
<td>8.27</td>
<td>Breakdown of the running times of CHOLMOD and LUSOL for the 777, 646-bus generated system</td>
</tr>
<tr>
<td>A.1</td>
<td>The ε-table. The boxed values are used as the approximate values of (v_k(1)) and the arrows present the order of evaluations.</td>
</tr>
</tbody>
</table>
ABSTRACT

Yeung, Yu Hong PhD, Purdue University, December 2017. Augmented Matrix Solvers for Dynamic System of Equations. Major Professor: Alex Pothen.

In this dissertation, the problem of updating in real time the solution to a linear system of equations when a sequence of small changes is made to the data is considered. This problem arises in many computational science and engineering applications, and we consider two of them. The first is surgical simulations, where a simulator used to train surgeons needs to provide haptic feedback by updating the system ten to hundred times per second. The second is contingency analysis in the power grids, when their operators need to simulate a large number of scenarios to predict what could happen when elements of the grid fail.

We observe that the changes in both applications result in the matrix’s being modified by a low-rank update within a principal submatrix. In the surgical simulations, the dimension of the matrix changes while it is being updated, since the matrix arises from a finite element mesh that is being cut, and we have to remesh around the cut during the update. In the power grid problem, the size of the matrix remains unchanged.

AMPS, our augmented matrix solver, updates the solution by means of an augmented matrix formulation, in which all changes made to the coefficient matrix are represented by adding rows and columns to the initial matrix. Our approach keeps the initial matrix as a submatrix of the new system of equations and the subsequent updates are accounted for by augmenting the system in blocked form. We characterize the situations when the matrix updates lead to nonsingular systems of equations when the initial matrix is nonsingular. The intactness of the initial matrix allows us to compute its factors only once, and then without refactoring the modified matrix,
we solve the augmented system by means of the Schur complement. A number of approaches, including direct methods, which factor the Schur complement and iterative methods, which use Krylov space solvers, are then available to solve the system.

To accelerate the computation, we utilize the precomputed factors of the initial matrix and the solutions to the original system of equations, exploit the sparsity of the matrices and vectors, and apply memoization and parallelization techniques. By analyzing the time complexity of our solver, we show that the running time of AMPS is $O(m\rho + |L|)$, where $m$ is the dimension of the principal submatrix, $\rho$ is the number of nonzeros in a subset of the columns of the Cholesky factor $L$ that are selected by the nonzeros in the sparse right-hand-side vector, and $|L|$ is the number of nonzeros in $L$. We also show that AMPS has an asymptotically lower time complexity relative to other existing methods.

Experimental results establish the competitiveness of AMPS by demonstrating that it outperforms a state of the art direct solver by two orders of magnitude, and earlier methods that update the matrix factors by 5 to 27 times, in the contingency analysis of a 777K node power-flow system. The computed solutions have accuracy comparable to those computed by a direct solver. Real-time solutions can also be provided using AMPS in the surgical simulation with more than 10 updates per second for a brain model of 50K nodes, and more than 30 updates per second for an eye model of 18K nodes.
1 INTRODUCTION

The problem of computing the solutions to the system of linear equations after a low-rank change in the matrix is a fundamental problem with many applications in the fields of science and engineering, including optimization algorithms, least-squares problems, the analysis of electrical circuits and power systems, and boundary condition and topological changes in finite element methods. We consider a system of equations $Ax = b$, where $A$ is a sparse, symmetric positive definite or indefinite $n \times n$ matrix $b$ is an $n$-vector, and $b$ may or may not be sparse. The change we consider is an update of a principal submatrix of the form

$$
\hat{A} = A - HEH^T,
$$

(1.1)

where $E$ is a symmetric $m \times m$ matrix, and $H$ is an $n \times m$ submatrix of an identity matrix and $m \ll n$. Since both $A$ and $E$ are symmetric, $\hat{A}$ is also symmetric. Note that in some applications the dimension of the matrix does not change when it is updated, i.e., sizes of $A$ and $\hat{A}$ are the same, while in others their dimensions are different. Equation (1.1) only applies for the former case. We will show that with some appropriate modifications, a similar equation can be used to describe those systems where the updated matrix $\hat{A}$ has a different size than $A$. We wish to compute efficiently the solution to the updated system

$$
\hat{A}\hat{x} = \hat{b}.
$$

(1.2)

In many applications, the sizes of the original matrix $A$ and the updated matrix $\hat{A}$ are huge, making the necessary refactorization of the matrix $\hat{A}$ inefficient for every small update to the system. Refactorization is impractical for such applications: Some of
them, such as surgical simulations, require real-time solutions to the updated systems; others where there are a huge number of systems to solve, as in the contingency analysis of power-flow systems, take too long to complete. Iterative methods are often preferred in such situations to avoid the large cost of factorization. However, the convergence rates of iterative methods depend on the condition number and many of these applications are poorly conditioned. In some cases, as shown in our experimental results, some iterative methods even fail to converge. Using preconditioners may solve the convergence problem, but they are often computationally expensive, and sometimes not effective.

In this dissertation, we describe several augmented matrix approaches to the solutions of the updated systems, in which the augmented matrices have a blocked form, in which the (1,1)-block is the original matrix $A$, and the updates to $A$ are represented by the other blocks.

First, we start with describing how to add, remove and replace rows and columns effectively by augmenting the matrix in Chapter 2. Based on these augmenting operations, we then introduce a well-known augmented matrix formulation to compute the solution to the updated systems of equations in Chapter 3, by using a procedure to effectively replace $m$ columns of $A$ by the updated columns in $\hat{A}$, as has been done in the revised simplex method of linear programming [1]. This formulation mathematically solves the system involving a low-rank update $F$ to the original matrix $A$ in the form

$$\hat{A} = A - FH^\top,$$  

(1.3)

where $F$ is an $n \times m$ matrix. We refer to this formulation as the column replacing augmented matrix method. We extend this formulation for the case that the dimension of $\hat{A}$ is different from that of the original matrix $A$ and provide a two-phase solution method that separates the augmented part from the rest of the system by forming the Schur complement. We first solve the augmented part implicitly by using a Krylov subspace solver, and then compute the remainder of the solution by using the precomputed LDL$^\top$ factors of the original matrix $A$ and the solution to the
original system of equations $x$. We also exploit the sparsity of the matrix and the right-hand-side vector $\tilde{b}$, and provide suitable preconditioners to the iterative solver.

In Chapter 4, we refine the augmented matrix formulation by making use of the fact that the update is a principal submatrix of the original matrix $A$, as stated in Equation (1.1). We formulate a symmetric augmented matrix system specific to this type of update and we describe two algorithms to solve this augmented system. In both algorithms, the original matrix $A$ is factored with a direct method. In the first algorithm, the Schur complement system is also solved by a direct method, and in the second algorithm it is solved with a Krylov subspace solver. We maintain symmetry in the augmented system of equations and throughout the second algorithm, whereas in the first algorithm an unsymmetric system is solved to reduce the computation time. Note that our algorithms can handle arbitrary changes to the right-hand-side $\tilde{b}$ in Equation (1.2). However, in many applications such as the power-flow contingency analyses considered in this dissertation, $\tilde{b}$ only changes in the set of $m$ rows where the principal submatrix is updated. Hence we focus on this situation in our experiments.

We describe AMPS, an augmented system solver that solves a linear system of equations equivalent to eq. (1.2), which means that in exact arithmetic solving the augmented system would give us the same solution vector $\tilde{x}$ up to a permutation. Our AMPS algorithms satisfy the following four desiderata:

1. The solution of the augmented system should be computed in a number of operations proportional to the size of the update $m$ rather than the size of the system $n$. This is especially important for large systems when there is a need for a sequence of updates in real-time.

2. The accuracy of the solution to the augmented system should be comparable to that of the direct solution of the modified system.

3. Both the factors of the matrix and the solution of the original system should be utilized in solving the augmented system to avoid redundant computations.
4. Sparsity in the matrices and the vectors should be exploited to accelerate the computations.

We analyze the time complexity of the AMPS algorithms and compare it with that of CHOLMOD, a direct solver that supports updating the Cholesky factors of a matrix $A$ while rows and columns are added to and removed from $A$. We also compare our formulation with other augmented matrix approaches as well as the mathematical equivalence of our augmented formulation with the Sherman-Morrison-Woodbury formula.

In Chapter 5, we further extend the basic AMPS formulation to support applications in which the dimensions of the modified systems are different from that of the original system in Chapter 5. We demonstrate that the same desiderata listed above are met by explicitly describing how the factors and the solutions to the original system can be used in solving the solutions to the modified system. Furthermore, we accelerate the computation time by using the memoization technique as well as by parallelizing the computations. We conclude by showing that this formulation is mathematically equivalent to the column replacing augmented formulation we described earlier in the case when the dimension of the system changes while being updated.

After describing the theory of the AMPS algorithms, we present two applications in two different fields for which our algorithms are well-suited. First, we describe the surgical simulation problem in Chapter 6. This simulation demands in real time tens or hundreds updates per second to a time-varying stiffness matrix, which may be ill-conditioned. We use a physics-based finite element model to realistically reflect the physical behaviors of the models under changes to the boundary conditions and topology in a surgical simulation. We derive the global stiffness matrix that governs the relationship between the nodal displacements and applied forces of the meshes modeling the patient’s organ. Changes in the boundary condition and topology could lead to the addition, deletion and replacement of nodes and elements in the stiffness matrix. Since the number of nodes in the model may change due to these changes,
the dimension of the modified system could be different from the original one. By supporting these types of changes to the stiffness matrix, the surgery can be visualized by updating the displacements of the nodes from the solutions obtained the AMPS algorithm.

Another application of the AMPS algorithm is the $N - x$ contingency analysis of power-flow systems, which we present in Chapter 7. Power grid operators are required by law to constantly assess the stability of the power grid under the assumption that any component in the system could fail. As more and more components are added to the power grid network as renewable energy sources such as solar panels and wind turbines are increasingly popular, it is a complex nonlinear system. As a result, the system is more vulnerable to multiple failures at the same time. Even a small increase in the allowable number of failures in the contingency analysis results in an enormous increase in the number of cases needing to be analyzed. To accommodate such an increase in the number of systems to be solved with a finite processing resource, the computation time for each case needs to be minimized. We show that the AMPS algorithm can be used to solve this problem in a fraction of time required by other existing methods. We first show how each contingency case can be formulated as a principal submatrix update. Since the power-flow equation is a complex nonlinear system of equations governed by the Kirchhoff’s current law, we approximate it by heuristics from the real-world observations to a linear system. With this system, each contingency case of the analysis can be expressed as a principal submatrix update to the base case, which functions normally without any contingencies. The AMPS algorithm can then be applied to the modified system corresponding to each contingency.

Experimental results of the two applications are presented in Chapter 8, which show that our AMPS algorithms outperform other existing direct and iterative methods in vast majority of the problems. The accuracy of the solution using the AMPS algorithms is empirically shown to be comparable with that of the solution $\hat{x}$ obtained by solving eq. (1.2) by a direct method. The results also demonstrate that our AMPS algorithm has the ability to provide real-time updates to the modified linear system,
which makes interactive applications possible, such as surgical simulations that demand tens or hundreds updates per second to large linear systems. For applications with an enormous number of cases to be solved, the speed advantage of our algorithms allows more cases to be analyzed on a given platform with a finite processing power.

Finally we provide several directions for future work in Chapter 9 and conclude the dissertation in Chapter 10.
2 AUGMENTED SYSTEM FORMULATION

It is well known that augmented systems can be used to effectively add and remove rows and columns of matrices [2, 3]. We begin by describing how these operations are accomplished, assuming that both the original matrix and the modifications are symmetric, i.e., the procedures are applied to rows and columns simultaneously. These modifications are not restricted to principal submatrix updates. Also, these modifications might not preserve the nonsingularity of the matrix. Hence after each update, we characterize the conditions that must be satisfied for the updated matrix to be nonsingular when the initial matrix is nonsingular. These results are obtained using the determinantal identity

\[ \det \begin{bmatrix} A & B \\ C & D \end{bmatrix} = \det (A) \det (D - CA^{-1}B), \]

when \( A \) is nonsingular. The goal of these characterizations is to show that our augmented system formulation by itself does not create singular matrices.

2.1 Adding a row and a column

To add a row and a column to \( Ax = b \), we consider the system

\[ \begin{bmatrix} A & \tilde{a} \\ \tilde{a}^\top & \tilde{\alpha} \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{bmatrix} = \begin{bmatrix} \tilde{b} \end{bmatrix}. \]  

(2.1)

If \( A \) is nonsingular and \( \tilde{\alpha} \neq \tilde{a}^\top A^{-1} \tilde{a} \), then the augmented matrix is nonsingular; and if \( A \) is positive definite and \( \tilde{\alpha} > \tilde{a}^\top A^{-1} \tilde{a} \), then the augmented matrix is also positive definite.
2.2 Removing a row and a column

To remove the \( j \)-th row and column from \( Ax = b \), we consider the system

\[
\begin{bmatrix}
A & e_j \\
e_j^T & 0
\end{bmatrix}
\begin{bmatrix}
\tilde{x}_1 \\
\chi
\end{bmatrix} =
\begin{bmatrix}
b \\
\beta
\end{bmatrix}.
\]

(2.2)

The last row \( e_j^T \tilde{x}_1 = 0 \) constrains the \( j \)-th component of \( \tilde{x}_1 \) to be 0, and consequently removes the contribution of the \( j \)-th column of \( A \). This leaves us with one fewer \textit{effective} variable than the number of equations. This is compensated by the additional component \( \chi \) in the solution vector. Consider the \( j \)-th row of the augmented system:

\[ e_j^T A \tilde{x}_1 + \chi = e_j^T b. \]

Since \( \chi \) only appears in the \( j \)-th row of the system, it is constrained to the value \( e_j^T (b - A \tilde{x}_1) \) after all the other components of \( \tilde{x}_1 \) are determined. Its value will be discarded after the system is solved.

Augmentation in this manner makes the matrix symmetric indefinite. If \( A \) is a symmetric positive definite matrix, then we can show that the augmented matrix is nonsingular, since its determinant is equal to \(-\det(A) (A^{-1})_{jj}\), and both terms are positive.

2.3 Replacing a row and a column

Replacing a row and a column can be done by removing the old row and column and adding the new ones. The resulting augmented formulation would be

\[
\begin{bmatrix}
A & \tilde{a}_j & e_j \\
\tilde{a}_j^T & \tilde{a}_{jj} & 0 \\
e_j^T & 0 & 0
\end{bmatrix}
\begin{bmatrix}
\tilde{x}_1 \\
\chi_1 \\
\tilde{x}_2
\end{bmatrix} =
\begin{bmatrix}
b \\
\hat{\beta} \\
\hat{\chi}
\end{bmatrix}.
\]

(2.3)

where \( \tilde{a}_j \) and \( \tilde{a}_{jj} \) are the \( j \)-th column and the \((j,j)\)-th element of \( \tilde{A} \) in 1.2 respectively. Note that the \( j \)-th component of \( \tilde{a}_j \) is then multiplied by the \( j \)-th component of \( \tilde{x}_1 \).
which is constrained to be 0 by the last equation. Hence the $j$-th component of $\hat{a}_j$ can be chosen arbitrarily.

We can calculate the determinant of the augmented matrix as

$$\det(A) \det \begin{bmatrix} \hat{a}_{jj} - \hat{a}_j^T A^{-1} \hat{a}_j & -\hat{a}_j^T A^{-1} e_j \\ -e_j^T A^{-1} \hat{a}_j & -e_j^T A^{-1} e_j \end{bmatrix}.$$ 

Hence if $A$ and the $2 \times 2$ matrix above are both nonsingular, the augmented matrix is also nonsingular.

2.4 Replacing multiple rows and columns

Replacing $m$ rows and columns can be done simultaneously in the same augmented matrix formulation. Suppose the set of indices of the rows and columns to be replaced is $H = \{j_1, j_2, \ldots, j_m\}$. The complete augmented formulation would be

$$\begin{bmatrix} A & J & H \\ J^T & C & 0 \\ H^T & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \tilde{b} \\ H^T \tilde{b} \\ 0 \end{bmatrix},$$

(2.4)

where $H = [e_{j_1}, e_{j_2}, \ldots, e_{j_m}]$ is the submatrix of the identity matrix with the indices of the columns to be replaced, $J = \tilde{A}H = [\tilde{a}_{j_1}, \tilde{a}_{j_2}, \ldots, \tilde{a}_{j_m}]$ are the modified columns of $\tilde{A}$, and $C = H^T \tilde{A}H$ is the diagonal block of the modified matrix $\tilde{A}$ where the changes occur. Note that the submatrix $C$ is $m \times m$, $J$ and $H$ are $n \times m$, and $m \ll n$.

Again, if $A$ is nonsingular, we can express the determinant of the augmented matrix as

$$\det(A) \det \begin{bmatrix} C - J^T A^{-1} J & -J^T A^{-1} H \\ -H^T A^{-1} J & -H^T A^{-1} H \end{bmatrix}.$$ 

If the second matrix above is nonsingular, then the augmented matrix is also nonsingular. (We can choose $J = AH$ as shown in Chapter 4; then the block $2 \times 2$ matrix above is the negation of the Schur complement matrix $S_1$ in the iterative variant of our
AMPS algorithm in Chapter 4.) In other words, the augmented matrix is nonsingular if and only if both $A$ and the Schur complement matrix $S_1$ are nonsingular.

We proceed to refine the system of equations 2.4 further. With a suitable $n \times n$ permutation matrix $P$, we can partition $H$ into an identity matrix and a zero matrix:

$$PH = \begin{bmatrix} I_m \\ 0_{n-m} \end{bmatrix}. \quad (2.5)$$

Applying the same permutation matrix $P$ to $J$, $A$, $\hat{x}_1$ and $b$ yields

$$PJ = \begin{bmatrix} J_1 \\ J_2 \end{bmatrix}, \quad PAP^\top = \begin{bmatrix} A_{11} & A_{12} \\ A_{12}^\top & A_{22} \end{bmatrix}, \quad (2.6a)$$

$$P\hat{x}_1 = \begin{bmatrix} \hat{x}_{11} \\ \hat{x}_{12} \end{bmatrix}, \quad Pb = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}. \quad (2.6b)$$

We can then apply the permutation matrix

$$\hat{P} = \begin{bmatrix} \hat{P} \\ I_m \\ I_m \end{bmatrix} \quad (2.7)$$

to the matrix in 2.4 from both left and right, which yields

$$\begin{bmatrix} I_{11} & A_{12} & J_1 & I \\ A_{12}^\top & A_{22} & J_2 & 0 \\ J_1^\top & J_2^\top & C & 0 \\ I & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \hat{x}_{11} \\ \hat{x}_{12} \\ \hat{x}_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ H^\top b \end{bmatrix}. \quad (2.8)$$

Here $A_{11}$ is the $m \times m$ submatrix being replaced by $C$, $A_{22}$ is the $(n - m) \times (n - m)$ principal submatrix of $A$ that is unchanged, and $A_{12}$ is the $m \times (n - m)$ off-diagonal submatrix of $A$. Note that the third column block effectively replaces the first column block, and by symmetry in the update, the third row block also replaces the first row.
block. Hence, the submatrix \( \begin{bmatrix} J_2 \\ \ell \end{bmatrix} \) must consist of the modified columns in \( \hat{A} \) that correspond to the original columns \( \begin{bmatrix} A_{12}^T \\ A_{11} \end{bmatrix} \) in \( A \).

**Lemma 1** The submatrix \( J_1 \) in 2.8 can be chosen arbitrarily such that the system is always consistent. Moreover, if \( \hat{A} \) in 1.2 is nonsingular, \( \hat{x}_{12} \) and \( \hat{x}_2 \) are independent of the submatrix \( J_1 \).

**Proof** Consider the last row block of 2.8. We have \( \hat{x}_{11} = 0 \). Consequently, the first column block, which then multiplies \( \hat{x}_{11} \), does not contribute to the solution of the system. Moreover, consider the first row block of 2.8:

\[
A_{12}\hat{x}_{12} + J_1\hat{x}_2 + \hat{x}_3 = b_1. \tag{2.9}
\]

Since \( \hat{x}_3 \) only contributes to one row block in the system of equations, its values can be determined uniquely for any values of \( J_1 \). Hence the submatrix \( J_1 \) can be chosen arbitrarily.

Now we can prove the second statement in the lemma. If we consider the second and third row and column blocks of system 2.8, since the last column blocks are zero for these rows, we have, after row and column permutations,

\[
\begin{bmatrix} C & J_2^T \\ \ell_2 & A_{22} \end{bmatrix} \begin{bmatrix} \hat{x}_2 \\ \hat{x}_{12} \end{bmatrix} = \begin{bmatrix} H^T b \\ b_2 \end{bmatrix}. \tag{2.10}
\]

(This system is the updated \( n \times n \) system of equations 1.2 written in its block \( 2 \times 2 \) form.) Hence the vectors \( \hat{x}_{12} \) and \( \hat{x}_2 \) are independent of the submatrix \( J_1 \).

Note that the values of \( \hat{x}_2 \) and \( \hat{x}_3 \) are coupled in 2.9, i.e., we can express one in terms of the other. Therefore, we need only one of them when solving the updated solution \( \hat{x} \) in 1.2.
Since we have applied the permutation to the solution vector in 2.8, we need to unpermute it to obtain the updated solution $\hat{x}$ in 1.2. Hence we obtain

$$P\hat{x} = \begin{bmatrix} \hat{x}_2 \\ \hat{x}_1 \end{bmatrix}.$$  

(2.11)
3 UNSYMMETRIC COLUMN UPDATES

In case of an unsymmetric column update as in Equation (1.3), more compact unsymmetric augmented systems of equations that are, in exact arithmetic, equivalent to the modified system by the procedures of effective column replacements. In this chapter, we will examine the detailed formulation, its associated solution method and acceleration techniques.

3.1 Replacing a column

Similar to Equation (2.3), an unsymmetric augmented system of equations can be formed to replace a column of $A$ by a column vector $\tilde{a}_j$:

$$
\begin{bmatrix}
A & \tilde{a}_j \\
\tilde{c}_j & 0
\end{bmatrix}
\begin{bmatrix}
\tilde{x}_1 \\
\tilde{t}
\end{bmatrix}
= 
\begin{bmatrix}
\tilde{b} \\
0
\end{bmatrix},
$$

(3.1)
in which its determinant is

$$
- \det (A) \det (e_j^T A^{-1} \tilde{a}_j).
$$

Hence, if $A$ is nonsingular and the $j$-th component of $u$ such that $Au = \tilde{a}$ is nonzero. In Section 3.5, we will show that the $j$-th component is nonzero if and only if $j$-th is in the closure of the nonzero pattern of $\tilde{a}$ in the graph of $G(A)$ given that there is no numerical cancellations.

We use the following example to illustrate how this augmented system of equations can be used to compute the same solution as the modified system $\tilde{A}\tilde{x} = \tilde{b}$ with a column being replaced, if exact arithmetic is used, after appropriate permutation of
the solution vector. Suppose that the third column of $A$ is replaced by the column vector $\tilde{a}_3$,

$$
\begin{bmatrix}
A & \tilde{a}_3 \\
0 & 0 & 1 & \cdots & 0
\end{bmatrix}
\begin{bmatrix}
\chi_1 \\
\chi_2 \\
\zeta_3
\end{bmatrix}
= 
\begin{bmatrix}
\beta_1 \\
\beta_2 \\
\beta_3
\end{bmatrix},
$$

(3.2)

where $\zeta_3$ is a placeholder variable at the third component of the solution vector. Notice that multiplying the a vector by the last row of the augmented matrix constrains $\zeta_3$ to be 0, and thus the whole third column of $A$ is multiplied by 0, canceling its effect on the system of equations. As the variable $\tilde{x}_3$ is multiplied by $\tilde{a}_3$, this column acts as a replacement for the third column of $A$. We refer to this operation the effective column replacement procedure.

### 3.2 Replacing multiple columns

The augmentation can be cascaded to replace multiple columns at the same time by applying the effective column replacement procedure to multiple columns. Similar to Equation (2.4), an equivalent non-symmetric augmented formulation can be formed:

$$
\begin{bmatrix}
A & J \\
\hat{H}^T & 0
\end{bmatrix}
\begin{bmatrix}
\vec{x}_1 \\
\vec{x}_2 \\
\vec{0}
\end{bmatrix}
= 
\begin{bmatrix}
\vec{b} \\
0
\end{bmatrix},
$$

(3.3)

We refer to Equation (3.3) as the columns replacing augmented matrix formulation. If $A$ is nonsingular, we can express its determinant as

$$
- \det (A) \det (\hat{H}^T A^{-1} J).
$$

Hence, the augmented matrix is nonsingular if and only if the determinant of the Schur complement matrix $\hat{H}^T A^{-1} J$ is nonsingular given that $A$ is nonsingular.
3.3  Dimension change

In some cases, the dimension of the system is changed after the update. Suppose the size of the modified matrix $\tilde{A}$ is $(n + k) \times (n + k)$ and the number of columns of $A$ being replaced is $m$. We can extend Equation (3.3) for such systems by first replacing $A$ by an augmented matrix $\tilde{A}$ of $A$ by an identity of size $k$:

$$\tilde{A} = \begin{bmatrix} A & \mathbf{I}_k \end{bmatrix}.$$  \hfill (3.4)

We can then apply the columns replacing augmented formulation on $\tilde{A}$ with $\tilde{J} = \tilde{A}\tilde{H}$ of size $(n + k) \times (m + k)$ and

$$\tilde{H} = \begin{bmatrix} H & \mathbf{I}_k \end{bmatrix}.$$  \hfill (3.5)

Mathematically, the resulting augmented formulation has the form

$$\begin{bmatrix} n \{ \begin{bmatrix} A & 0 & \tilde{J} \\ \mathbf{0} & \mathbf{I}_k \end{bmatrix} & \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \tilde{b} \\ \mathbf{0} \end{bmatrix} \end{bmatrix}.$$  \hfill (3.6)

3.4  Solution method

Since we assume that $A$ is nonsingular, it has an LDL$^\top$ factorization if $A$ is symmetric, or an LU factorization if $A$ is nonsymmetric. For simpler notation purposes, we assume hereafter in this chapter $\tilde{A}$ has replaced $A$ and $k = 0$ if the dimension remains unchanged. Equation (3.3) or Equation (3.6) can be reduced into smaller system of equations using the Schur complement with $\tilde{A}$ as the pivot, with a multiplication of $-1$:

$$H^\top \tilde{A}^{-1}J\tilde{x}_2 = H^\top \tilde{A}^{-1}\tilde{b},$$  \hfill (3.7)
where

\[ \widetilde{A}^{-1} = \begin{bmatrix} A^{-1} \\ I_k \end{bmatrix} \]  

(3.8)

Since we have precomputed the factorization of \( A \), any occurrence of the multiplication \( \widetilde{A}y \), for any vector \( y \), can be efficiently computed using the factors of \( A \) via triangular matrix solves with forward and back substitutions for the first \( n \) components of \( y \). Consequently, Equation (3.7) can be solved implicitly using iterative solvers such as GMRES, which involves only sequential matrix-vector multiplications of \( H^\top(\widetilde{A}^{-1}(J\bar{x}_2)) \) in each iteration. If \( m + k \) is small enough, it can also be solved using direct solvers by explicitly forming the matrix in the equation.

Once we have solved for \( \bar{x}_2 \) in Equation (3.7), we can solve for \( \bar{x}_1 \) by premultiplying the first 2 row blocks of Equation (3.6) by \( \widetilde{A}^{-1} \) and rearranging the terms:

\[ \bar{x}_1 = \widetilde{A}^{-1}(\bar{b} - J\bar{x}_2) \]  

(3.9)

Finally, the solution vector \( \bar{x} \) can be obtained by replacing the placeholder variables in \( \bar{x}_1 \) by \( \bar{x}_2 \):

\[ \bar{x}[j] = \begin{cases} \bar{x}_1[j] & \text{if } j \notin \widetilde{H} \\ (\widetilde{H}^\top\bar{x}_2)[j] & \text{if } j \in \widetilde{H} \end{cases} \]  

(3.10)

where \( \widetilde{H} = H \cup \{n + 1, \ldots, n + k\} \) is the set of indices of the columns of \( \widetilde{H} \) correspond to the identity matrix \( I_{n+k} \).

In summary, the augmented system solution method can be broken into four steps:

1. **Calculate the right-hand side of Equation (3.7).**

   We use the precomputed factors of \( A \) to first calculate the vector \( y \), where \( y = \widetilde{A}^{-1}\bar{b} \), via triangular matrix solves, and then calculate the product \( H^\top y \) to arrive at the vector on the right-hand side of Equation (3.7).

2. **Solve Equation (3.7) to find \( \bar{x}_2 \).**

   Since the right-hand side of Equation (3.7) is known from Step 1, an iterative
solver can be used to successively improve estimates of $\tilde{x}_2$ by a sequence of matrix-vector multiplications to compute the product $H^T(\tilde{A}^{-1}(J\tilde{x}_2))$ in each iteration. As in Step 1, performing the second multiplication involving $\tilde{A}^{-1}$ requires triangular matrix solves with forward and back substitution. Alternatively, explicit formation of the matrix and direct solvers can be used if the size of the augmentation is small enough.

3. Substitute $\tilde{x}_2$ into Equation (3.9), then solve for $\tilde{x}_1$.

This step requires the use of the factors of $\tilde{A}$ a final time to perform the multiplication of $\tilde{A}^{-1}$ with the known vector $(b - J\tilde{x}_2)$.

4. Combine $\tilde{x}_1$ and $\tilde{x}_2$ to form $\tilde{x}$.

Finally, we replace the placeholders in $\tilde{x}_1$ by the corresponding values of $\tilde{x}_2$ and append the rest of $\tilde{x}_2$ to the end to form $\tilde{x}$.

3.5 Exploiting sparsity

In this section we explain how we can solve the Schur complement equations efficiently. For easy explanation, we make use of definitions and theorems in graph theory stated as follows.

3.5.1 Theory

**Definition 2** An $n \times n$ sparse matrix $A$ can be represented by a directed graph $G(A)$ whose vertices are the integers $1, \ldots, n$ and whose edges are

$$\{(i, j) : i \neq j, \text{ and } A_{ij} \neq 0\}.$$

This set of indices is called the structure of $A$.

**Definition 3** An elimination tree of a Cholesky factor $L$ is the transitive reduction of the directed graph $G(L)$. [4]
Definition 4  The structure of a vector \( \mathbf{x} \) with \( n \) components is

\[
\text{struct}(x) := \{ i : x_i \neq 0 \},
\]

which can be interpreted as a set of vertices of the directed graph of any \( n \times n \) matrix. In this paper without introducing ambiguity, for a vector \( x \), \( \text{closure}_A(x) \) refers to \( \text{closure}_A(\text{struct}(x)) \).

Definition 5  Given a directed graph \( G(A) \) and a subset of its vertices denoted by \( V \), we say \( V \) is closed with respect to \( A \) if there is no edge of \( G(A) \) that joins a vertex not in \( V \) to a vertex in \( V \); that is, \( v_j \in V \) and \( A_{ij} \neq 0 \) implies \( v_i \in V \). The closure of \( V \) with respect to \( A \) is the smallest closed set containing \( V \),

\[
\text{closure}_A(V) := \bigcap \{ U : V \subseteq U, \text{ and } U \text{ is closed} \},
\]

which is the set of vertices of \( G(A) \) from which there are directed paths in \( G(A) \) to vertices \( V \).

Theorem 6  Let the structures of \( A \) and \( b \) be given. Whatever the values of the nonzeros in \( A \) and \( b \), if \( A \) is nonsingular then

\[
\text{struct}(A^{-1}b) \subseteq \text{closure}_A(b).
\]

The proof of Theorem 6 can be found in [5].

Theorem 7  Suppose we need only some of the components of the solution vector \( x \) of the system \( Ax = b \). Denote the needed components by \( \pi \). If \( A \) is nonsingular, then the set of components in \( b \) needed is \( \text{closure}_{A^T}(\pi) \).
Proof Let values be given for which $A$ is nonsingular. Renumber the vertices of $G(A^\top)$ so that $\text{closure}_{A^\top}(\bar{x}) = \{1, 2, \ldots, k\}$ for some $k \leq n$. Then $Ax = b$ can be partitioned as
\[
\begin{bmatrix}
B & D \\
C & E
\end{bmatrix}
\begin{bmatrix}
y \\
z
\end{bmatrix}
= 
\begin{bmatrix}
d \\
e
\end{bmatrix},
\]
where $B$ is $k \times k$. By the definition of $\text{closure}_{A^\top}(\bar{x})$, there is no edge $(i, j)$ with $i \in \text{closure}_{A^\top}(\bar{x})$ and $j \notin \text{closure}_{A^\top}(\bar{x})$. Therefore $D = 0$. Then $By = d$. Since $A$ is nonsingular, $B$ is also nonsingular. Thus $x$ can be computed by solving only $By = d$, which implies only $\text{closure}_{A^\top}(\bar{x})$ is needed to compute the components in $\bar{x}$.

Theorem 8 Let $A = LL^\top$ be a Cholesky factorization and $V$ be a subset of vertices in $G(L)$. If $r$ is the root of the elimination tree $T$ of $L$, then
\[
\text{closure}_L(V) = \bigcup_{v \in V} \{r \stackrel{T}{\Rightarrow} v\},
\]
where $r \stackrel{T}{\Rightarrow} v$ is the path from $r$ to $v$ in $T$, including all intermediate vertices along the path.

Proof

(i) $\bigcup_{v \in V} \{r \stackrel{T}{\Rightarrow} v\} \subseteq \text{closure}_L(V)$:
For any edge between a node $v$ and its parent $u$ in $T$, there is an edge $(u, v)$ in $G(L)$. By definition if $v \in \text{closure}_L(V)$, then $u \in \text{closure}_L(V)$. Since $V \in \text{closure}_L(V)$, all ancestors of $V$ must be in $\text{closure}_L(V)$.

(ii) $\text{closure}_L(V) \subseteq \bigcup_{v \in V} \{r \stackrel{T}{\Rightarrow} v\}$:
If a node $u \notin \bigcup_{v \in V} \{r \stackrel{T}{\Rightarrow} v\}$, there must be a path from a node $w \in \bigcup_{v \in V} \{r \stackrel{T}{\Rightarrow} v\}$ to $u$. Hence there is also a directed path from $w$ to $u$ in $G(L)$. Since $L$ is lower triangular, there is no cycle in $G(L)$. Hence there is no directed path from $u$ to $w$ and $u \notin \text{closure}_L(V)$.
3.5.2 Application in the algorithm

To solve the Schur complement equations efficiently, we carefully exploit the sparsity in both the vectors and the matrices. Recall from Equation (3.7) the right-hand side of the equation is obtained by computing

\[ z = \tilde{H}y = \begin{bmatrix} H^TA^{-1}\hat{b} \\ 0 \end{bmatrix} \]  

(3.11)

and

\[ H^TA^{-1}\hat{b} = H^T(LDL^T)^{-1}\hat{b} = \underbrace{HL^{-T}D^{-1}L^{-1}\hat{b}}_{(2)} \]  

(3.12)

By applying Theorem 6, \( \text{struct}(L^{-1}\hat{b}) \subseteq \text{closure}_L(\hat{b}) \). We observe that if the vector \( \hat{b} \) is sparse, only the submatrix of \( L \) corresponding to \( \text{closure}_L(\hat{b}) \) is useful for the computation.

Suppose we have computed the vector \( y \) in Equation (3.12). The first \( n \) components of it is then projected by the sparse matrix \( H^T \). We notice that \( H^T \) is a submatrix of an identity matrix by construction and its number of rows is less than its number of columns, which means that only a \( m \) columns of \( H^T \) are nonzero and each contains only a 1 in one row. Consequently \( H^T \) selects only a few components of \( y \) to form \( z \), which are the only components needed in the computation, while others will be multiplied by 0. Let the subset of components of \( y \) needed be denoted by \( \tilde{y} \). By applying Theorem 7, only the submatrix of \( L^T \) corresponding to \( \text{closure}_L(\tilde{y}) \) is needed for computing the vector \( z \).

Similarly we can speed up the computations in Steps 2 and 3. In step 2 during each GMRES iteration the solution guess \( \tilde{x}_2 \) is projected by a sparse matrix \( \tilde{J} \) to a larger space. The product vector \( w = J\tilde{x}_2 \) in the equation \( y = \tilde{A}^{-1}w \) is sparse. Hence, only the submatrix of \( L \) corresponding to \( \text{closure}_L(\tilde{J}\tilde{x}_2) \) is useful. The solution vector \( y \) is also projected by the matrix \( H^T \) as in Step 1. Hence, during the backward substitution only a few columns of \( L^T \) are needed for the computation.
Finally in Step 3, since both $\hat{b}$ and $\hat{J}\tilde{x}_2$ are sparse, the difference vector $\hat{b} - \hat{J}\tilde{x}_2$ is also sparse. Hence the forward substitution can be sped up by considering only those needed rows of $L$. However, since the solution vector $\tilde{x}_1$ is not projected by a sparse matrix, the whole matrix $L^T$ is needed in the backward substitution.

Since in both Steps 1 and 3 the triangle solves are only done once, in practice we modify the solves with 2 additional inputs to indicate the sparsity of the right-hand side vector and the components we need in the solution vector. On the other hand, an iterative solver such as GMRES takes a number of iterations in Step 2. To reduce the overhead of indirect indexing, it is beneficial to explicitly form in advance the submatrices by copying the needed rows and columns in $L$ and $L^T$.

### 3.6 Complexity analysis

The time complexity of the computations of (1) and (2) in Equation (3.12) depends on the size of $\text{closure}_L(\hat{b})$ and $\text{closure}_L(\hat{y})$. By Theorem 8 to find the sizes of the closures we need only follow the path from the root of the elimination tree to the nodes in $\text{struct}(\hat{b})$ or $\hat{y}$. It is obvious that if those nodes are close to the root, the sizes would be small constants, independent of $n$. On the other hand, if they are the leaves of the elimination tree, the sizes would be close to $n$, and the cost of the triangular solve step is linear in the number of nonzeros in $L$, denoted by $|L|$. In general, the cost lies in between these two extremes. In practice, the closures are small relative to $|L|$.

Figure 3.1 shows an example of a lower triangular matrix $L$ and its directed graph $G(L)$ with $\text{closure}_L(\{6\})$ outlined. We can see that if the vector $b$ in the equation $Lx = b$ has only one nonzero at component $b_6$, then the solution vector $x$ has only nonzero at components $\{x_6, x_7, x_8, x_{10}\}$. Similarly, if we only need to compute the component $x_6$ in the solution of $L^T x = b$, then we only need to use the components $\{b_6, b_7, b_8, b_{10}\}$ in the vector $b$ and the corresponding submatrix of $L^T$. 
Key parts of the complexity analysis hinge on the sparsity of the matrix, the complexity of the LDL$^T$ factorization, and the sparsity of the factors. The number of non-zeros in each column of the matrix is dependent on the application problem. For instance, the sparsity of the stiffness matrix of a 3D mesh depends on the connectivity between its vertices. Since in a well-formed mesh the number of edges incident on a single node is limited by geometric considerations, the number of non-zeros per column can be bounded by a constant that is independent of the total number of degrees of freedom in the model. Similar observation can be done on the power flow problem as the sparsity of the matrix reflects the connectivity of the power grid. In real world, a bus is only connected to a handful of other buses. Thus, the admittance matrix is very sparse in general. Due to this assumption and the use of sparse matrix data structures in our work, the complexity of all the steps in the augmented algorithm that update or otherwise operate on the matrix columns is dependent only on the number of columns affected, not the number of rows in the matrix.

The derivation of complexity bounds for the LDL$^T$ factorization can be found in [6]. Their work shows that using efficient sparse matrix algorithms, an LDL$^T$ factorization of an $n \times n$ stiffness matrix for a 3D finite element model can be accomplished with $O(n^2)$ operations; the resulting lower triangular matrix $L$ will contain
$O(n^{4/3})$ non-zeros if the mesh elements have good aspect ratios. Similar upper bound can be applied to the power flow problem as well. Therefore, without imposing any restrictions on the force vector, we can say that the triangular solves needed in Steps 1 and 3 of the solution algorithm described in Section 3.4 can be completed in $O(n^{4/3})$ operations.

Considering the sparsity analysis in the previous section raises the question of whether sparsity could provide a basis for a tighter complexity bound. In solution Step 1, the complexity depends on $|\text{closure}_L(\tilde{b})|$ and $|\text{closure}_L(\tilde{y})|$. Theorem 8 from the Appendix informs us that the sizes of the closures depend on the length of the path from the root of the elimination tree to the nodes in $\text{struct}(\tilde{b})$ and $\text{struct}(\tilde{y})$. If those nodes are close to the root, the sizes would be small constants, independent of $n$. On the other hand, if they are leaves of the elimination tree, the sizes would be close to $n$, and the cost of the triangular solve step would be linear in the number of nonzeros in $L$, $O(n^{4/3})$. In general, the cost lies in between these two extremes, and the upper bound of $O(n^{4/3})$ is not tight.

<table>
<thead>
<tr>
<th>Computation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization: $t = t_0$</td>
<td>$O(n^2)$ for 3D meshes; $O(n^{3/2})$ for 2D meshes</td>
</tr>
<tr>
<td>1 Compute LDL$^\top$ factorization of $A$</td>
<td>$O(m + k)$</td>
</tr>
<tr>
<td>2 Compute $\bar{J}$ and $\bar{H}$</td>
<td>$O(m + k)$</td>
</tr>
<tr>
<td>3 Solution Step 1</td>
<td>$O(</td>
</tr>
<tr>
<td>4 Solution Step 2</td>
<td>$O(</td>
</tr>
<tr>
<td>5 Solution Step 3</td>
<td>$O(</td>
</tr>
</tbody>
</table>

The complexity for each step in the algorithm, including the precomputation phase and the real-time update loop, is detailed in Table 3.1. Summing the complexity of each of the real-time update steps and simplifying the expression to retain only the
dominant terms results in a complexity bound for an update iteration of $O(n^{4/3} \cdot n_{iter})$, where $n_{iter}$ is the number of GMRES iterations needed for convergence. Note that $n_{iter}$ is influenced by $m + k$, the number of columns updated, rather than $n$, the dimension of the matrix.

3.7 Preconditioning

The GMRES iteration in Step 2 can be preconditioned to reduce the number of iterations. One possible preconditioner is a matrix $M$ that approximates $\tilde{H} \tilde{A}^{-1} \tilde{J}$ in Equation (3.7). However, there are two drawbacks of this approach: neither forming the matrix $\tilde{H} \tilde{A}^{-1} \tilde{J}$ nor minimizing $\|M(\tilde{H} \tilde{A}^{-1} \tilde{J}) - I\|$ is computationally cheap. Also, $M$ has to be recomputed whenever there is a change to the mesh. Another possible preconditioner is a matrix product $\tilde{H} S \tilde{J}$ such that $S$ approximates $\tilde{A}^{-1}$. In this case, $S$ only needs to be computed once and can be reused in later time steps, even after any updates. In the surgical simulation application, we use two approximations of $S$: the inverse of $D$ in the pre-computed LDL$^\top$ factors of $A$ (for all meshes), and a tridiagonal sparse approximate inverse (SPAI) [7] of the matrix $A$ (for Stanford Bunny, brain and eye meshes).

3.8 Refactorization

The augmented matrix approach produces a solution for the time-varying linear system more quickly than a full refactorization would allow, so long as $m$, the number of modified columns, is sufficiently small. As changes to the matrix accumulate across a growing number of columns, the augmented method begins to slow down because the size of the $\tilde{x}_2$ vector also increases. To maintain fast solution speeds for an interactive application, we prevent $m$ from growing indefinitely by periodically recomputing a full LDL$^\top$ factorization of $A$ in a process that runs concurrently with the simulation loop. Figure 3.2 shows the changes in the equations before and after refactorization. Let $m'$ and $k'$ denote the number of columns modified and dimension change after the
refactorization was initiated. When freshly computed factors are used to replace the original factors, the size $m+k+m'+k'$ is reduced to $m'+k'$. The rate at which matrix changes accumulate will vary widely and depend both on the nature of a simulation and how quickly and aggressively a user manipulates a model. Even considering a single user and a single simulation, the growth rate of $m+k$ will vary unevenly across time as an interactive task progresses through moments of updates. Since the speed of our solution method is dependent on $m+k+m'+k'$, the update rate it provides is affected by the speed of updates to the matrix. The best way to address this issue will be depend on the application, but in some contexts it is reasonable to limit the rate at which cutting can occur in order to maintain a desired update rate. Variability in the update rate arising from the refactorization process can be smoothed by buffering the computed solutions.

![Diagram](image)

**Figure 3.2.:** Refactorization process

To provide some context for how refactorization will impact simulation speed, the timing results in the surgical simulation experiments in Chapter 8 indicate the simulation step at which the cumulative time for mesh updates equals the time for matrix factorization, assuming one newly cut node per update and beginning with an empty list of mesh modifications. In practice, the list of recent mesh modifications will be non-empty when a refactorization step completes, there will be update steps that involve changes to multiple nodes, and many update steps will not involve any topological mesh changes. Depending on these factors, actual simulation update rates
could be faster or slower when refactorization concludes than the times shown in the graphs. It is also feasible to run multiple refactorization processes concurrently, so that mesh changes get incorporated into the factors as quickly as possible and the size of $m + k$ is kept to a minimum. If multiple processors are available, this is one way to maximize the update rate since it is not necessary for one factorization process to complete before another one begins.
4 AMPS: AUGMENTED METHOD WITH PRINCIPAL SUBMATRIX UPDATE

In this chapter, we focus on problems where the updates to the matrix is a principal submatrix. This kind of updates appears in problems where the connectivity of vertices in a graph changes. Indeed both applications discussed in Chapters 6 and 7, namely surgical simulations and power grid contingency analyses fall into this category.

4.1 Principal submatrix update

We now refine the techniques described in the previous chapter to design an augmented matrix approach to update the solution when $A$ is modified by a principal submatrix update as in Equation (1.1). In this case, all the changes are captured in the submatrix $C$ in Equation (2.4), and we can deduce that $C = H^T \tilde{A}H = H^T AH - E$. Therefore, the submatrix $J_2$ in Equation (2.8) remains unchanged from the original system and thus $J_2 = A_{12}^T$. As proven in the previous subsection, $J_1$ in Equation (2.8) can be chosen arbitrarily. With the choice of $J_1 = A_{11}$, we can show that

\[
J = P^T \begin{bmatrix} J_1 \\ J_2 \end{bmatrix} = P^T \begin{bmatrix} A_{11} \\ A_{12}^T \end{bmatrix} = AH. \tag{4.1}
\]

The last equation follows from

\[
AH = P^T \begin{bmatrix} A_{11} & A_{12} \\ A_{12}^T & A_{22} \end{bmatrix} PH = P^T \begin{bmatrix} A_{11} & A_{12} \\ A_{12}^T & A_{22} \end{bmatrix} \begin{bmatrix} I_m \\ 0_{n-m} \end{bmatrix} = P^T \begin{bmatrix} A_{11} \\ A_{12}^T \end{bmatrix}. \tag{4.2}
\]
We can thus write Equation (2.4) as
\[
\begin{bmatrix}
  A & AH & H \\
  H^\top A & C & 0 \\
  H^\top & 0 & 0
\end{bmatrix}
\begin{bmatrix}
  \mathbf{f}_1 \\
  \mathbf{f}_2 \\
  \mathbf{f}_3
\end{bmatrix}
= 
\begin{bmatrix}
  b \\
  H^\top b
\end{bmatrix}.
\]
(4.3)

Here is an example in which the principal submatrix at the 3rd and 5th rows and columns are modified. The augmented system Equation (4.3) would be
\[
\begin{bmatrix}
  \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} & \alpha_{35} & \cdots \\
  \alpha_{51} & \alpha_{52} & \alpha_{53} & \alpha_{54} & \alpha_{55} & \cdots \\
  0 & 0 & 1 & 0 & 0 & \cdots \\
  0 & 0 & 0 & 0 & 1 & \cdots \\
\end{bmatrix}
\begin{bmatrix}
  \alpha_{13} & \alpha_{15} & 0 & 0 \\
  \alpha_{23} & \alpha_{25} & 0 & 0 \\
  \alpha_{33} & \alpha_{35} & 1 & 0 \\
  \alpha_{43} & \alpha_{45} & 0 & 0 \\
  \alpha_{53} & \alpha_{55} & 0 & 1 \\
  \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\end{bmatrix}
\begin{bmatrix}
  \mathbf{x}_1 \\
  \mathbf{x}_2 \\
  \mathbf{z}_3 \\
  \mathbf{z}_4 \\
  \mathbf{z}_5 \\
\end{bmatrix}
= 
\begin{bmatrix}
  \mathbf{f}_1 \\
  \mathbf{f}_2 \\
  \mathbf{z}_3 \\
  \mathbf{z}_4 \\
  \mathbf{z}_5 \\
\end{bmatrix},
\]
(4.4)
in which the \( \zeta \) terms are constrained to be 0, the \( \chi \) terms are the permuted solutions to Equation (1.2), and the \( \delta \) terms are the values of \( \hat{x}_3 \) in Equation (4.3).

### 4.2 Solution method

In this section, we describe our algorithms to solve the system Equation (2.4). Suppose we have computed the \( LDL^\top \) factorization of \( A \) when solving the original system \( Ax = b \). Here, \( L \) is a unit lower triangular matrix and \( D \) is a diagonal matrix or block diagonal matrix with \( 1 \times 1 \) or \( 2 \times 2 \) blocks if \( A \) is indefinite. A fill-reducing ordering and an ordering to maintain numerical stability are usually used during the factorization, and thus a permuted matrix of \( A \) is factored, i.e., \( \hat{P}^\top \hat{A} \hat{P} = LDL^\top \).
for some permutation matrix \( \hat{P} \). We assume that hereafter the permuted system \( \hat{P}^\top A \hat{P} \hat{P}^\top x = \hat{P}^\top b \) has replaced the original system. Solutions to the original system \( Ax = b \) can then be obtained by applying the inverse permutation \( \hat{P} \). For simplicity, we will not explicitly write the permutation matrices \( \hat{P} \). We can solve Equation (4.3) in two ways.

**Iterative method** With \( A = LDL^\top \) as a block pivot, Equation (4.3) can be reduced to a smaller system involving the symmetric matrix \( S_1 \), the Schur complement of \( A \), with a multiplication by \(-1\):

\[
\begin{bmatrix}
E & I \\
H^\top A^{-1}H & S_1
\end{bmatrix}
\begin{bmatrix}
\tilde{x}_2 \\
\tilde{x}_3
\end{bmatrix} = 
\begin{bmatrix}
H^\top (b - \hat{b}) \\
H^\top A^{-1}b
\end{bmatrix},
\]  
(4.5)

where \( E = H^\top AH - C \), which is the same \( E \) as in Equation (1.1). This can be shown by premultiplying and postmultiplying Equation (1.1) by \( H^\top \) and \( H \) respectively:

\[
C \equiv H^\top \hat{A} H = H^\top AH - H^\top HEH^\top H = H^\top AH - E.
\]  
(4.6)

We can solve Equation (4.5) by an iterative method such as GMRES or MINRES. Matrix-vector products with \( S_1 \) need a partial solve with \( A = LDL^\top \) involving only the rows and columns selected by \( H \) and \( H^\top \), and products with \( E \), in each iteration. Note that \( H^\top \) in the right-hand-side vector selects the components from the difference vector \( \hat{b} - b \) and the solution \( x \) of the original system \( Ax = b \) if the changes in the right-hand-side vector are only in the changed rows of \( A \).

**Direct method** Alternatively, the \((2,1)\)-block of \( S_1 \) can be used as a block pivot with another Schur complement:

\[
\begin{bmatrix}
E H^\top A^{-1}H - I \\
S_2
\end{bmatrix} \tilde{x}_4 = EH^\top A^{-1}b - H^\top (b - \hat{b}).
\]  
(4.7)
We can then solve this equation for $\hat{x}_3$ using a direct solver with an LU factorization of $S_2$, which can be constructed efficiently as described later in Section 4.5. Note that $S_2$ is unsymmetric although both the augmented matrix in Equation (4.3) and $S_1$ in Equation (4.5) are symmetric. This is because we have chosen an off-diagonal block pivot in forming $S_2$, to avoid the computation of the inverse of either $E$ or $H^\top A^{-1}H$ on the diagonal block of $S_1$.

4.3 Solution to the modified system

It turns out that we only need to compute $\hat{x}_3$ to obtain the full solution vector $\hat{x}$ to the modified system Equation (1.2). This can be done by making the following observation. Premultiplying the first row block of Equation (4.3) by $A^{-1}$, and rearranging terms yields

$$\hat{x}_1 = A^{-1}b - H\hat{x}_2 - A^{-1}H\hat{x}_3.$$  \hspace{1cm} (4.8)

From Equation (2.11), we have

$$P\hat{x} = \begin{bmatrix} \hat{x}_2 \\ \hat{x}_2 \\ \hat{x}_2 \\ 0 \end{bmatrix} = \begin{bmatrix} \hat{x}_{11} \\ \hat{x}_2 \\ \hat{x}_2 \\ 0 \end{bmatrix} = P\hat{x}_1 + PH\hat{x}_2,$$  \hspace{1cm} (4.9)

by using Equations (2.5) and (2.6b), and the fact that $\hat{x}_{11} = 0$. Premultiplying both sides by $P^\top$ yields

$$\hat{x} = \hat{x}_1 + H\hat{x}_2.$$  \hspace{1cm} (4.10)

Substituting Equation (4.8) into Equation (4.10), we have

$$\hat{x} = A^{-1}b - A^{-1}H\hat{x}_3,$$  \hspace{1cm} (4.11)

in which the first term is the solution to the original system.
4.4 Relation to the Sherman-Morrison-Woodbury formula

The solution $\hat{x}$ in Equation (1.2) obtained by AMPS using the direct approach can be expressed in a single equation by substituting $\hat{x}_3$ in Equation (4.7) into Equation (4.11):

$$\hat{x} = A^{-1}b - A^{-1}H (EH^T A^{-1}H - I)^{-1} \left[ EH^T A^{-1}b - H^T (b - \hat{b}) \right].$$

(4.12)

In the case when the right-hand-side of Equation (1.2) does not change from the original system, i.e. $\hat{b} = b$, Equation (4.12) becomes

$$\hat{x} = \left[ A^{-1} - A^{-1}H (EH^T A^{-1}H - I)^{-1} EH^T A^{-1} \right] b.$$

(4.13)

Using the Sherman-Morrison Woodbury formula [8]

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U \left( C^{-1} + VA^{-1}U \right)^{-1} VA^{-1},$$

(4.14)

the inverse of $\hat{A}$ in Equation (1.1) can be expressed as

$$\hat{A}^{-1} = [\hat{A} + (H)(-I)(EH^T)]^{-1}$$

$$= A^{-1} - A^{-1}H (-I + EH^T A^{-1}H)^{-1} EH^T A^{-1},$$

(4.15)

which when multiplied by $b$ is identical to Equation (4.13).

4.5 Forming the Schur complement $S_2$ explicitly

In the previous section we have described the algorithm to solve the modified system using the augmented formulation. We now discuss how we form the matrix $W \equiv EH^T A^{-1}H$ in the Schur complement $S_2$ in Equation (4.7) by using partial triangular solves.
From the factorization of $A$, the matrix $H^\top A^{-1}H$ can be expressed as the product $H^\top L^{-\top} D^{-1} L^{-1} H$. Then $W^\top$ can be expressed as

$$W^\top = H^\top L^{-\top} D^{-1} L^{-1} HE.$$  \hfill (4.16)

Recall that $E$ is symmetric. Let $X \equiv L^{-1}HE$. Premultiplying both sides of this equation by $L$, we have

$$LX = HE \equiv \tilde{E}.$$  \hfill (4.17)

Observe that the right-hand-side of Equation (4.17) is a matrix $\tilde{E}$ mapping the $i^{th}$ row of $E$ to the $j_i^{th}$ row of $\tilde{E}$ with the rest of $\tilde{E}$ filled with 0. For instance, if the set of indices of updates is $S = \{3, 5\}$, then Equation (4.17) would be

$$LX = HE = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ \vdots \end{bmatrix} E = \begin{bmatrix} 0 & 0 \\ \epsilon_{11} & \epsilon_{12} \\ 0 & 0 \\ \epsilon_{21} & \epsilon_{22} \\ \vdots & \vdots \end{bmatrix}.$$  \hfill (4.18)

Since both $L$ and $\tilde{E}$ are sparse, we can use partial forward substitution to solve for $X$.

Let $Y := L^{-1}H$. We can again use partial forward substitution by exploiting the sparsity in $H$ and $L$ to compute $Y$, as discussed in the next subsection. Once we have the matrices $X$ and $Y$, we can compute $W^\top$ as follows:

$$W^\top = H^\top L^{-\top} D^{-1} X = Y^\top D^{-1} X.$$  \hfill (4.19)
4.6 Complexity analysis

The time complexity of principal submatrix updates using the symmetric augmented formulation can be summarized in Table 4.1. Direct method refers to the approach of solving for $\tilde{x}_3$ directly using Equation (4.7), and iterative method refers to the approach of applying an iterative method to Equation (4.5). Recall that $n$ is the size of the original matrix $A$, $m$ is the size of the principal submatrix update $C$, while $t$ denotes the number of iterations that the iterative method takes to converge.

<table>
<thead>
<tr>
<th>Computation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amortized initialization:</td>
<td></td>
</tr>
<tr>
<td>1 Compute $\text{LDL}^T$ factorization of $A$</td>
<td>$O(n^{3/2})$ for planar networks</td>
</tr>
<tr>
<td>2 Compute $x = A^{-1}b$</td>
<td>$O(</td>
</tr>
<tr>
<td>Real-time update steps:</td>
<td></td>
</tr>
<tr>
<td>1 Obtain the submatrix $B$</td>
<td>$O(</td>
</tr>
<tr>
<td>2 Compute $E = H^TAH - B$</td>
<td>$O(</td>
</tr>
<tr>
<td>3 Compute $W^T = H^TA^{-1}HE$</td>
<td>$O(m \cdot \rho)$</td>
</tr>
<tr>
<td>(a) Form $\tilde{E} = HE$</td>
<td>$O(</td>
</tr>
<tr>
<td>(b) Solve $LX = \tilde{E}$</td>
<td>$O(m \cdot \rho)$</td>
</tr>
<tr>
<td>(c) Solve $L^T\tilde{W}^T = D^{-1}X$</td>
<td>$O(m \cdot \rho)$</td>
</tr>
<tr>
<td>(d) Form $W^T = H\tilde{W}^T$</td>
<td>$O(m^2)$</td>
</tr>
<tr>
<td>4 Form $W - I$</td>
<td>$O(m)$</td>
</tr>
<tr>
<td>5 Form R.H.S. of Equations (4.5) and (4.7)</td>
<td>$O(m)$</td>
</tr>
<tr>
<td>6 Solve for $x_3$</td>
<td>$O(m^3)$</td>
</tr>
<tr>
<td>7 Solve $\tilde{x} = x - A^{-1}H\tilde{x}_3$</td>
<td>$O(</td>
</tr>
</tbody>
</table>

Table 4.1.: Time complexity of the AMPS direct and iterative methods

The overall time complexity of the direct method is dominated by either Step 3 (computing $W^T$) or Step 7 (solving for $\tilde{x}$), i.e., $O(m \cdot \rho + |L|)$. For the iterative method, the time complexity is dominated by either Step 6 (solving for $\tilde{x}_3$) or Step 7 (solving for $\tilde{x}$), i.e., $O(t \cdot \rho + |L|)$. Hence the AMPS algorithms have the time complexities

$$O(m \cdot \rho + |L|) \text{ (direct)} \quad \text{and} \quad O(t \cdot \rho + |L|) \text{ (iterative).}$$

(4.20)
In comparison, for CHOLMOD [9], the time complexity for updating the Cholesky factor of the matrix, when row and column changes are made, is

$$O \left( \sum_{j \in S} \left( \sum_{L_{jk} \neq 0} \left( |L_{sjk}| + \sum_{k \in P_j} (L_{sk}) \right) \right) \right) \quad (4.21)$$

where $L$ is the original Cholesky factor, $L'$ is the modified Cholesky factor and $P_j$ is the path from node $j$ to the root of the elimination tree of $L$. (Note that we have to add the cost $|L|$ to compute the solution by solving the triangular system of equations.)

Consider the two inner sums in the expression for the complexity. The first inner sum computes the total number of operations of Steps 1–4 in both Algorithms 1 (Row Addition) and 2 (Row Deletion) in CHOLMOD. If we denote $T_j$ as the set of nodes $k < j$ in $G(\hat{A})$ that have an edge incident on node $j$, then this sum is equivalent to the number of outgoing edges of the closure of $T_j$ in $G(\overline{L})$ up to node $j$. The second inner sum computes the number of operations needed for Step 5 (rank-1 update/downdate) in Algorithms 1 and 2 of CHOLMOD. This sum is equivalent to the closure of $\{j\}$ in the updated graph $G(\overline{L})$. Combining the two summation terms, we can express the time complexity of CHOLMOD in terms of the closures:

$$O \sum_{j \in S} \text{closure}_{\overline{L}}(T_j) \leq O \left( m \cdot \max_j \text{closure}_{\overline{L}}(T_j) \right). \quad (4.22)$$

We make two observations when comparing the AMPS algorithms with CHOLMOD. First, in general, the AMPS algorithms do not introduce new fill-in elements in the Cholesky factor whereas fill-ins are possibly introduced in CHOLMOD. However, this happens when the update introduces a new nonzero entry in row/column $j$ of $\hat{A}$. In our application to the contingency analysis for power flow, we only remove connections between buses. Hence running CHOLMOD neither introduces fill-ins to the factor nor changes the elimination tree. Second, since the nodes in $T_j$ are numbered less than $j$, the closure of $T_j$ is always larger than the closure
of \{j\}, whether or not the updated factor \( \bar{L} \) is different from \( L \). In the case that row \( j \) of \( \bar{L} \) is relatively dense due to fill-in, the first inner sum in Equation (4.21) may be the dominant term. On the other hand, the AMPS algorithms only need the closure from node \( j \) in \( G(L) \).

Figure 4.1 shows an example running CHOLMOD on a matrix \( A \) with connection between nodes 2 and 6 removed. It can be observed that CHOLMOD needs a traversal through nodes 1, 2, 5, 6 and 8 to compute the closures needed. On the other hand, our AMPS methods only need a traversal through nodes 2, 6 and 8 to compute the closures they need.

4.7 Comparison with other augmented methods

Several algorithms have been proposed to solve a modified system of linear equations using augmented matrices. Gill et al. [3] used augmented matrices and a factorization approach to update basis matrices in the simplex algorithm for linear programming, motivated by the work of Bisschop and Meeraus [1, 2]. In their method, the matrix was factored in a block-LU form as

\[
\begin{bmatrix}
A & \tilde{A}H \\
\tilde{H}^T & \tilde{C}
\end{bmatrix} =
\begin{bmatrix}
L & \tilde{Y} \\
\tilde{Y}^T & \tilde{D}
\end{bmatrix}
\begin{bmatrix}
U & \tilde{Y} \\
I & I
\end{bmatrix}.
\]  

(4.23)

Here the matrix \( L \) is unit-lower triangular and the matrix \( U \) is upper triangular. The matrices \( \tilde{Y} \) and \( \tilde{D} \) are \( n \times m \) submatrices of the block factors and \( \tilde{D} \) is the Schur complement of \( A \).

Maes [10] and Wong [11] used a similar approach to implement active-set QP solvers with symmetric augmentation to solve the Karush-Kuhn-Tucker (KKT) matrices arising from Hessian updates and factored in a block-LU form as

\[
\begin{bmatrix}
A & V \\
V^T & \tilde{C}
\end{bmatrix} =
\begin{bmatrix}
L & \tilde{Y} \\
\tilde{Y}^T & \tilde{I}
\end{bmatrix}
\begin{bmatrix}
U & \tilde{S} \\
I & \tilde{S}
\end{bmatrix}.
\]  

(4.24)
Here the submatrices $Y$ and $Z$ are $n \times 2m$ submatrices, doubling the size of $\tilde{Y}$ and $\tilde{Z}$ in Equation (4.23). These submatrices were updated using sparse triangular solves and $\tilde{S}$ was updated using a dense LU-type factorization. Comparing the augmented matrix in Equation (4.24) with Equation (2.4), we have

$$V = \begin{bmatrix} AH & H \end{bmatrix} Q^\top \quad \text{and} \quad \tilde{C} = Q \begin{bmatrix} C & 0 \\ 0 & 0 \end{bmatrix} Q^\top,$$

for some permutation matrix $Q$.

To take advantage of symmetry, Maes and Wong factored the augmented matrix in a symmetric block-LBL$^\top$ form

$$\begin{bmatrix} A & V \\ V^\top & \tilde{C} \end{bmatrix} = \begin{bmatrix} L & I \\ \tilde{C} & D \end{bmatrix} \begin{bmatrix} Z^\top & I \\ \tilde{Z} & I \end{bmatrix}.$$  

The major differences between our methods and the KKT matrix block-LU/block-LBL$^\top$ update method are as follows. We exploited the explicit forms of the submatrices $Z$ and $\tilde{D}$ in the factorization when the update is a principal submatrix. Specifically, if we factor the augmented matrix in Equation (4.3) as in Equation (4.26), we have

$$\begin{bmatrix} A & AH & H \\ H^\top A & C & 0 \\ H^\top & 0 & 0 \end{bmatrix} = \hat{L}\hat{D}\hat{L}^\top,$$

where

$$\hat{L} = \begin{bmatrix} L \\ H^\top L & I \\ H^\top L^\top D^{-1} & 0 & I \end{bmatrix} \quad \text{and} \quad \hat{D} = \begin{bmatrix} D \\ 0 & -S_1 \end{bmatrix}.$$  

Here $\hat{L}$ is a lower triangular matrix, $\hat{D}$ is a matrix in which its $(1, 1)$-block is (block) diagonal and the rest is the negation of the Schur complement $S_1$ in Equation (4.5).
Combining the results from Equation (4.25) and Equation (4.28), we obtain the relationship between the factors in Equation (4.26) and those in our method:

\[ Z^\top = Q \begin{bmatrix} H^\top L \\ H^\top L^{-\top} D^{-1} \end{bmatrix} \quad \text{and} \quad \tilde{D} = Q(-S_1)Q^\top. \]  

(4.29)

Hence, we do not need to construct \( Z \) and \( \tilde{D} \) as in the Gill et al., Maes, and Wong algorithms. We also make use of the structure of the factors in computing the solution, whereas Maes updated the factors by treating the augmentation submatrix \( V \) as sparse and \( \tilde{C} \) as dense. Finally, we compute the solution to the modified system by explicitly using the solution to the original system.
Figure 4.1.: Example of running CHOLMOD on a matrix $A$ with principal submatrix update on rows/columns 2 and 6. Red vertices are those columns of the factor involved in the first 4 steps. Blue vertices are those columns involved in the rank-1 update/downdate step.
5 AMPS WITH DIMENSION CHANGE

In this chapter, we expend the AMPS algorithm discussed in the last chapter for problems that the size of the system changes while being updated with a principal submatrix $E$, in which a submatrix of $E$ is also a submatrix of the original matrix $A$. Figure 5.1 shows an example of the original matrix $A$, the modified matrix $\tilde{A}$ and the corresponding principal submatrix update $E$.

![Figure 5.1: Example of a modified matrix $\tilde{A}$ formed by a principal submatrix update $E$ highlighted in blue to the original matrix $A$ highlighted in grey with dimension change.](image)

5.1 Formulation

We consider the difference between the original $n \times n$ matrix $A$ and the modified $(n + k) \times (n + k)$ matrix $\tilde{A}$ after updates at time $t$. If we denote by $\tilde{A}$ the matrix formed by augmenting $A$ with the identity matrix $I_k$, i.e.

$$\tilde{A} = \begin{bmatrix} A \\ I_k \end{bmatrix}, \quad (5.1)$$
we can observe that \( \tilde{A} \) can be expressed as the result of a principal submatrix update to \( \tilde{A} \):

\[
\tilde{A} = \tilde{A} - \tilde{H} \tilde{E} \tilde{H}^\top,
\]

(5.2)

where \( \tilde{H} \) is the matrix formed by augmenting \( H \) with the identity matrix \( I_k \), i.e.

\[
\tilde{H} = \begin{bmatrix} H \\ I_k \end{bmatrix},
\]

(5.3)

where \( H \) is the \( m \) columns of the identity matrix \( I_n \) which indices are the rows and columns of \( A \) to be updated; and \( \tilde{E} \) is the sum of \((m+k) \times (m+k)\) principal submatrix update \( E \) and an identity of size \( k \) at the \((2,2)\)-block, i.e.

\[
\tilde{E} = E + \begin{bmatrix} 0_m \\ I_k \end{bmatrix},
\]

(5.4)

where \( E_{11} \) is an \( m \times m \) update to \( A \) and \( \begin{bmatrix} E_{12} \\ E_{22} \end{bmatrix} \) is the new columns of \( \tilde{x} \) corresponds to the newly added vertices after the cutting. We have shown in Chapter 4 that principal submatrix update in the form of Equation (5.2) can be solved using augmented matrix formulation

\[
\begin{bmatrix} \tilde{A} & \tilde{A} \tilde{H} \\ \tilde{H}^\top \tilde{A} & \tilde{H}^\top \tilde{A} \tilde{H} - \tilde{E} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \\ \tilde{x}_3 \end{bmatrix} = \begin{bmatrix} \tilde{b} \\ 0 \end{bmatrix},
\]

(5.5)

where \( \tilde{b} \) is the \((n+k)\)-vector of the modified system. With \( \tilde{A} \) as the pivot, Equation (5.5) can be reduced to a smaller system involving the symmetric matrix \( S_1 \), the Schur complement of \( \tilde{A} \), with a multiplication of \(-1\):

\[
\begin{bmatrix} \tilde{E} & I \\ \tilde{H}^\top \tilde{A}^{-1} \tilde{H} \end{bmatrix} \begin{bmatrix} \tilde{x}_2 \\ \tilde{x}_3 \end{bmatrix} = \begin{bmatrix} \tilde{H}^\top \tilde{A}^{-1} \tilde{b} \\ \tilde{H}^\top \tilde{A}^{-1} \tilde{b} \end{bmatrix},
\]

(5.6)
in which
\[
\tilde{A}^{-1} = \begin{bmatrix} A^{-1} \\ I_k \end{bmatrix}
\]  \hfill (5.7)

Equation (5.6) can be further reduced with another Schur complement using the (1, 2)-block of \( S_1 \) as the block pivot:
\[
\begin{pmatrix} I - \tilde{H}^\top \tilde{A}^{-1} \tilde{H} E \end{pmatrix} \begin{pmatrix} \tilde{x}_2 \\ \tilde{s}_2 \end{pmatrix} = \tilde{H}^\top \tilde{A}^{-1} \tilde{b} - \tilde{H}^\top \tilde{A}^{-1} \tilde{H} \tilde{H}^\top (\tilde{b} - \tilde{b}) = \tilde{H}^\top \tilde{A}^{-1} \tilde{b}.
\]  \hfill (5.8)

If \( \tilde{b} \) only differs from \( b \) at the newly added components, i.e.
\[
\tilde{b} - b = \begin{bmatrix} 0_n \\ b \end{bmatrix}
\]  \hfill (5.9)
then the right-hand-side vector of Equation (5.8) can be simplified to
\[
\tilde{H}^\top \tilde{A}^{-1} \tilde{b} = \begin{bmatrix} H^\top a \\ b \end{bmatrix}
\]  \hfill (5.10)

where \( x \) is the solution to the original system \( Ax = b \) and \( \bar{b} \) is the right-hand side of the newly expanded indices.

After solving for \( \tilde{x}_2 \) using a direct solver on Equation (5.8), we can solve for \( \tilde{x} \) in the modified system \( \tilde{A} \tilde{x} = \tilde{b} \) directly using the following observation. Premultiplying the first row block of Equation (5.5) by \( \tilde{A}^{-1} \) and rearranging the terms yields
\[
\tilde{x}_1 = \tilde{A}^{-1} \tilde{b} - \tilde{H} \tilde{x}_2 - \tilde{A}^{-1} \tilde{H} \tilde{x}_3.
\]  \hfill (5.11)

In addition, rearranging the terms in first row block of Equation (5.6) yeilds
\[
\tilde{x}_3 = \tilde{H}^\top (\tilde{b} - \tilde{b}) - \tilde{E} \tilde{x}_2.
\]  \hfill (5.12)
Substituting Equation (5.12) into Equation (5.11) yields

$$
\tilde{x}_1 = \tilde{A}^{-1} \left[ \begin{bmatrix} \tilde{y} - \tilde{H} \tilde{H}^\top (\tilde{b} - \tilde{b}) \\ \tilde{K}^{-1} \tilde{H} \tilde{E} - \tilde{H} \end{bmatrix} \right] + \left( \begin{bmatrix} \tilde{K}^{-1} \tilde{H} \tilde{E} - \tilde{H} \end{bmatrix} \right) \tilde{x}_2. 
$$

(5.13)

Again, if \( \tilde{b} \) satisfies the condition of Equation (5.9), Equation (5.13) can be simplified to

$$
\tilde{x}_1 = \begin{bmatrix} x \\ \tilde{b} \end{bmatrix} + \left( \begin{bmatrix} \tilde{K}^{-1} \tilde{H} \tilde{E} - \tilde{H} \end{bmatrix} \right) \tilde{x}_2. 
$$

(5.14)

Furthermore, substituting Equation (5.14) into Equation (4.10) in Chapter 4 yields

$$
\hat{x} = \begin{bmatrix} x \\ \tilde{b} \end{bmatrix} + \tilde{A}^{-1} \tilde{H} \tilde{E} \tilde{x}_2. 
$$

(5.15)

Alternatively, one can use the (2,1)-block in Equation (5.6) as the block pivot for the Schur complement and get

$$
\begin{bmatrix} \tilde{E} \tilde{H}^\top \tilde{A}^{-1} \tilde{H} - I \end{bmatrix} \tilde{x}_3 = \tilde{H}^\top (\tilde{b} - \tilde{b}) + \tilde{E} \tilde{H}^\top \tilde{A}^{-1} \tilde{b},
$$

$$
= \begin{bmatrix} 0 \\ \tilde{b} \end{bmatrix} + \begin{bmatrix} E_{11}^\top \\ K_{12}^\top \end{bmatrix} \tilde{H}^\top x,
$$

(5.16)

assuming that the condition in Equation (5.9) is satisfied. Upon \( \tilde{x}_3 \) is solved using Equation (5.16), the solution \( \hat{x} \) can be solved similarly as follows:

$$
\hat{x} = \begin{bmatrix} x \\ \tilde{b} \end{bmatrix} - \tilde{A}^{-1} \tilde{H} \tilde{x}_3. 
$$

(5.17)
5.2 Solutions numerical accuracy improvement

It turns out that we can improve the numerical accuracy of the solutions by substituting \( \tilde{x}_2 \) into \( \tilde{x} \) directly as follows. From the third row block of Equation (5.5), we have

\[
\tilde{H}^T \tilde{x}_1 = 0.
\]  

(5.18)

Premultiplying Equation 4.10 by \( \tilde{H}^T \) yields

\[
\tilde{H}^T \tilde{x} = \tilde{H}^T \tilde{x}_1 + \tilde{H}^T \tilde{H} \tilde{x}_2 = \tilde{x}_2.
\]  

(5.19)

Note that the components of \( \tilde{x} \) picked out by \( \tilde{H}^T \) are indeed \( \tilde{x}_2 \), which are arithmetically identical to the same components computed using Equation (5.15) but with higher accuracy. If we denote \( \mathbb{H} \) as the set of indices which the rows and columns of \( A \) are updated including the newly added ones, combining the two equations, we have

\[
\tilde{x}[i] = \begin{cases} 
(\tilde{H} \tilde{x}_2)[i] & \text{for } i \in \mathbb{H}, \\
\left( \tilde{x} - \tilde{A}^{-1} \tilde{E} \tilde{x}_2 \right)[i] & \text{for } i \notin \mathbb{H}.
\end{cases}
\]  

(5.20)

Skipping the computations of those components in \( \tilde{x} \) that are in \( \mathbb{H} \) also improves the performance of the algorithm.

5.3 Computing the principal submatrix of the inverse

Our augmented algorithm involves solving Equations (5.8) and (5.20). Unlike the column replacing augmented formulation described in Chapter 3 both equations are
solved using a direct solver. The Schur complement matrix $S_2$ in Equation (5.8) can be expressed in block matrix form using Equations (5.3), (5.4), (5.7) and (5.10) to

\[
\begin{pmatrix}
I_m \\
0_k
\end{pmatrix}
\begin{bmatrix}
H^T A^{-1} H \\
I_k
\end{bmatrix}
E
\begin{bmatrix}
H^T x \\
b
\end{bmatrix}
= 
\begin{bmatrix}
H^T x \\
b
\end{bmatrix}.
\]  
(5.21)

Solving Equation (5.21) involves computing the principal submatrix of the inverse $A^{-1}(H) \equiv H^T A^{-1} H$. Assuming that $K = LDL^T$ is a factorization of $A$, we have

\[
A^{-1}(H) = H^T L^{-T} D^{-1} L^{-1} H.
\]  
(5.22)

If we denote $V \equiv L^{-1} H$, then $A^{-1}(H) = V^T D^{-1} V$, which can be computed by first solving for $V$ using forward substitution, then scaling $V$ to obtain $U \equiv D^{-1} V$ and finally premultiplying $U$ by $V^T$. The computation of the rest of the matrix in Equation (5.21) is straightforward.

5.3.1 Memoization

For an efficient computation of the principal submatrix of the inverse $K_t^{-1}(H)$ at time $t$, we observe that since the vertices removed during the cutting are accumulating and $H$ is the submatrix of the identity corresponding to the replaced rows and columns in $A$, the matrix $H_{t-1}$ at the previous time $t-1$ is a submatrix of the first $m_{t-1}$ columns of matrix $H_t$ at time $t$, i.e.

\[
H_t = \begin{bmatrix} H_{t-1} & H_{\Delta t} \end{bmatrix},
\]  
(5.23)

where $H_{\Delta t}$ is the ($m_t - m_{t-1}$) columns of the identity matrix corresponding to the newly removed columns at timestep $t$. Consequently, the matrix $V_{t-1}$ is also the first $m_{t-1}$ columns of the $V_t$ since each column of $V_t$ is independently solved, i.e.

\[
V_t = \begin{bmatrix} V_{t-1} & V_{\Delta t} \end{bmatrix}
\]  
(5.24)
where $V_{\Delta t} = L^{-1}H_{\Delta t}$, which are the only columns of $V_t$ needed to be computed. Furthermore, the top-left $(m_t-1 \times m_t-1)$ submatrix of $A_t^{-1}(H)$ is identical to $A_{t-1}^{-1}(H)$ because

$$A_t^{-1}(H) = V_t^T D^{-1} V_t = \begin{bmatrix} V_{t-1}^T & | & V_t^T \end{bmatrix} D^{-1} \begin{bmatrix} V_{t-1} \mid V_t^T \end{bmatrix}$$

$$= \begin{bmatrix} V_{t-1}^T D^{-1} V_{t-1} & V_{t-1}^T D^{-1} V_{\Delta t} \\ V_{\Delta t}^T D^{-1} V_{t-1} & V_{\Delta t}^T D^{-1} V_{\Delta t} \end{bmatrix}$$

$$= \begin{bmatrix} A_{t-1}^{-1}(H) & V_{t-1}^T D^{-1} V_{\Delta t} \\ V_{\Delta t}^T D^{-1} V_{t-1} & V_{\Delta t}^T D^{-1} V_{\Delta t} \end{bmatrix}. \quad (5.25)$$

Furthermore, it can be observed from Equation (5.25) that $A_t^{-1}(H)$ is also symmetric and only the lower or upper triangular part needs to be computed and stored, and subsequent updates can be done sequentially by trapezoidal augmentations to $\text{tril}(A_t^{-1}(H))$:

$$\begin{cases} \text{tril}(A_t^{-1}(H)) = \text{tril}(A_{t-1}^{-1}(H)) & \text{for } i \in [m_t-1+1, m_t] ; j \leq i. \quad (5.27) \end{cases}$$
It is obvious that Equation (5.27) can be easily parallelizable for all $i$’s and $j$’s since they are independent from each other.

5.4 Dimension shrinking

In the case of the imposition of Dirichlet boundary conditions, the dimension of the system is shrunk instead of expanded. An augmented matrix system similar to Equation (5.5) is equivalent to the modified system of equations:

$$
\begin{bmatrix}
  A & H \\
  H^T & 0
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2
\end{bmatrix}
= 
\begin{bmatrix}
  \hat{b} \\
  0
\end{bmatrix}
$$

where $x_2 = -H^T f$ is the variables imposed by the new boundary conditions and $\hat{b}$ is the new right-hand side vector. Similar to Equation (5.5), we can reduce Equation (5.28) to a smaller system using $A$ as the pivot:

$$
H^T A^{-1} H x_2 = H^T A^{-1} \hat{b}.
$$

Note that the matrix on the left-hand side is the principal submatrix of the inverse $A^{-1}(H)$, which can be efficiently computed as described in previous subsections. The right-hand side can be computed using $V \equiv L^{-1} H$ as

$$
H^T A^{-1} \hat{b} = V^T D^{-1} L^{-1} \hat{b}.
$$

Upon $x_2$ is computed, $x_1$ can be computed using the first row block of Equation (5.28) as

$$
x_1 = A^{-1} \left( \bar{f} - H^T x_2 \right)
= L^{-T} D^{-1} L^{-1} \left( \bar{f} - H^T x_2 \right)
= L^{-T} \left( \bar{g} - D^{-1} V x_2 \right).
$$
in which \( g \) is already computed in Equation (5.30) and can be reused.

5.5 Complexity analysis

The time complexity of principal submatrix updates using the symmetric augmented formulation can be summarized in Table 5.1. Both per cut and total update times are provided. In the table, variables with subscript \( t \) are the values at time \( t \), those with subscript \( \Delta t \) are the newly added values at time \( t \), whereas those without any subscript are their maxima over all \( t \). Recall that \( n \) is the size of the original matrix \( A \), \( m \) is the size of the principal submatrix update \( H \), \( A \) is the dimension change. In addition, \( \mathcal{H} \) is the set of indices of the nonzero rows of \( H \), \( |L| \) is the number of nonzeros in \( L \), \( c \) is the total number of cuts, and \( v = \max_j |V_{s,j}| \) is the maximum number of nonzeros in any column of \( V \), which is equivalent to the maximum closure size of any vertex in the graph of \( G(L) \). In Section 3.5 we gave a detailed discussion on the concepts of closure and the relations between sparse matrix computations and its corresponding graph. We also discussed about the theorems that are used to prove the upper bounds of the complexity of the AMPS algorithms. The overall time com-

<table>
<thead>
<tr>
<th>Computation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amortized initialization: Compute ( \text{LDL}^T ) factorization of ( A )</td>
<td>( O(n^2) ) for 3D meshes; ( O(n^{3/2}) ) for 2D meshes</td>
</tr>
<tr>
<td>Compute ( a = A^{-1} f )</td>
<td>( O(</td>
</tr>
<tr>
<td>Real-time update steps: Solve for ( V_{\Delta t} )</td>
<td>( O \left( \sum_{\mathcal{H}} \text{closure}_L(h) \right) ) per step</td>
</tr>
<tr>
<td>Compute ( \text{tril}(K_{\Delta t}^{-H}) )</td>
<td>( O((m_{t-1} + 1 + m_t) m_{\Delta t} \cdot v_t) ) total</td>
</tr>
<tr>
<td>Form ( S_2 )</td>
<td>( O(m_t^2(m_t + k_t) + m_t) )</td>
</tr>
<tr>
<td>Form R.H.S. of Equation 5.10</td>
<td>( O(m_t) )</td>
</tr>
<tr>
<td>Solve for ( \bar{x}_2 )</td>
<td>( O((m_t + k_t)^3) )</td>
</tr>
<tr>
<td>Solve for ( \bar{x} )</td>
<td>( O(m_t \cdot v_t +</td>
</tr>
</tbody>
</table>
plexity of the algorithm is dominated by either Step 2 (computing tril \( A_{\Delta t}^{-1}(H) \)) or Step 6 (solving for \( \tilde{x} \)). The AMPS algorithm has an overall time complexity

\[
O \left( m^2 v + c \cdot |L| \right)
\]  

(5.32)

5.6 Parallelization

We can observe that Steps 1–4 in Table 5.1 are easily parallelizable from the facts that in Step 1 each columns of \( V_{\Delta t} \) are independently solved, both Steps 2 and 3 involve matrix-matrix multiplications, and in Step 4 the R.H.S. of Equation (5.10) is formed by mapping. The parallelization of Step 5 and 6 is non-trivial, which is out of scope of this paper. The parallel time complexity of the algorithm for \( p \) processors is

\[
O \left( \frac{m^2 v + c \cdot |L|}{p} \right)
\]  

(5.33)

5.7 Relation to column replacing augmented formulation

In Chapter 3 we presented a hybrid asymmetric augmented algorithm to solve for the solutions to \( \hat{A}\tilde{x} = \hat{b} \) where \( \hat{A} \) of size \((n + k) \times (n + k)\) is identical to the enlarged original matrix \( A \) of the same size but a few columns, denoted by an \((n + k) \times (m + k)\) submatrix \( \tilde{J} \) of \( \hat{A} \). If the changes in those columns are only in the rows of the same indices as the columns, i.e., it is a principal submatrix update, \( \tilde{J} \) can then be expressed in terms of \( \hat{A} \) as

\[
\tilde{J} = \hat{A}\tilde{H}.
\]  

(5.34)

Substituting Equation (5.2) into Equation (5.34) yields

\[
\tilde{J} = \left( \hat{A} - \tilde{H}\tilde{E}\tilde{H}^\top \right)\tilde{H} = \hat{A}\tilde{H} - \tilde{H}\tilde{E}.
\]  

(5.35)
Substituting Equation (5.35) into Equations (3.7) and (3.9) yields

\[
( I - \tilde{H}^\top \tilde{A}^{-1} \tilde{H} \tilde{E} ) \tilde{b} = \tilde{H}^\top \tilde{A}^{-1} \tilde{b} \quad \text{and} \quad (5.36a)
\]

\[
\tilde{x}_1 = \tilde{A}^{-1} \tilde{b} - \tilde{H} \tilde{x}_2 + \tilde{A}^{-1} \tilde{H} \tilde{E} \tilde{x}_2, \quad (5.36b)
\]

in which the first equation is identical to Equation (5.8). Substituting Equation (5.12) into Equation (5.36b) yields

\[
\tilde{x}_1 = \tilde{A}^{-1} \tilde{b} - \tilde{H} \tilde{x}_2 + \tilde{A}^{-1} \tilde{H} \left[ \tilde{H}^\top \left( \tilde{l} - \tilde{b} \right) - \tilde{x}_3 \right] \quad (5.37)
\]

which is identical to Equation (5.11) if the condition in Equation (5.9) is satisfied. Hence, the two augmented formulations are mathematically equivalent.
6 APPLICATION TO SURGICAL SIMULATION

This chapter outlines the standard method of constructing a system of linear equations which mathematically represents the interactions between nodal displacements and forces in a finite element model and updating these equations to account for the imposition of boundary conditions and mesh modifications, including cutting. The traditional formulation of the finite element equations are then incorporated into the augmented matrix formulations presented in Chapters 3 and 5 to provide efficient solution methods for enabling arbitrary mesh deformations and cutting procedures to the meshes. This makes real-time surgical simulations feasible.

6.1 Finite element model

A finite element model is a discrete approximation of a continuous displacement field defined over the volume of a body. A collection of space filling elements forms the mesh that defines the spatial domain of the body, and each element approximates the displacement over the portion of the volume it occupies. The continuous displacement field inside an element is approximated by the interpolation of all the displacement vectors computed at each node of the element. The size of the vectors equals to the dimension of the space and each node has its own displacement vector. Therefore, in an elastostatic model the number of degrees of freedom (DoF) of an element is simply the product of the number of nodes of the element and the dimension of the space. For example, a quadrilateral in 2-space has $4 \times 2 = 8$ DoF whereas a linear tetrahedron in 3-space has $4 \times 3 = 12$ DoF. To illustrate how the system of linear equations is constructed, we use linear tetrahedra in 3-space as the constructing elements. However, the method itself is not constrained to tetrahedra only.
6.2 Linear tetrahedron

A linear tetrahedron is the simplest solid element in 3-space. The word linear means its edges are straight lines. Its geometry is defined by 4 nodes with coordinates
\[ x_i, y_i, z_i; \quad i = 1, 2, 3, 4. \]

Throughout this section we will use the abbreviations for node coordinate differences:
\[ x_{ij} = x_i - x_j, \quad y_{ij} = y_i - y_j, \quad z_{ij} = z_i - z_j; \quad i, j = 1, 2, 3, 4. \]

6.3 Barycentric coordinates

The position of a point \( P \) in a tetrahedron can be expressed by its Cartesian coordinates \( \{ x, y, z \} \) or by its barycentric coordinates \( \{ \rho_1, \rho_2, \rho_3, \rho_4 \} \). Each component \( \rho_i \) is the ratio of the volume of the tetrahedron enclosed by \( P \) and the face opposite to the \( i \)th node to the volume of the whole tetrahedron as shown in Figure 6.1, i.e.
\[ \rho_i = \frac{v_{P_i}}{v}. \]

Figure 6.1.: A tetrahedron with a point \( P \) inside it. The tetrahedron enclosed by \( P \) and the face opposite to node 1 is shaded and its volume is \( v_{P_1} \).
Since the volumes of the tetrahedra enclosed by $P$ and all 4 faces sum up to the total volume of the whole tetrahedron, we can show that

$$\sum_i \rho_i = \sum_i \frac{v_{pi}}{v} = 1. \quad (6.1)$$

And the point $P(x, y, z)$ can be expressed as a weighted sum of the nodes using barycentric coordinates:

$$x = \sum_i \rho_i x_i \tag{6.2}$$
$$y = \sum_i \rho_i y_i$$
$$z = \sum_i \rho_i z_i$$

6.4 Coordinate transformations

Quantities that are intrinsically linked to the geometry of the tetrahedron (e.g. shape functions) are best expressed in barycentric coordinates, whereas quantities such as displacement, strain and stress components are expressed in Cartesian coordinates $\{x, y, z\}$. Therefore, we need a transformation between two coordinate systems. Combining Equations (6.1) and (6.2), we can form

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix} \begin{bmatrix} \rho_1 \\ \rho_2 \\ \rho_3 \\ \rho_4 \end{bmatrix}.$$
The above $4 \times 4$ matrix is called the Jacobian matrix and it serves the transformation from barycentric coordinates to Cartesian coordinates. For the reverse transformation, we can use the inverse of the Jacobian matrix, explicitly

\[
\begin{bmatrix}
\rho_1 \\
\rho_2 \\
\rho_3 \\
\rho_4
\end{bmatrix} = \frac{1}{6v}
\begin{bmatrix}
6v_{01} & 0 & 0 & 0 \\
6v_{02} & a_1 & b_1 & c_1 \\
6v_{03} & a_2 & b_2 & c_2 \\
6v_{04} & a_3 & b_3 & c_3
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix},
\]

(6.3)

where the first column entries are given by

\[
6v_{01} = x_2(y_3z_4 - y_4z_3) + x_3(y_4z_2 - y_2z_4) + x_4(y_2z_3 - y_3z_2),
\]

\[
6v_{02} = x_3(y_1z_4 - y_4z_1) + x_4(y_3z_1 - y_1z_3) + x_1(y_4z_3 - y_3z_4),
\]

\[
6v_{03} = x_4(y_1z_2 - y_2z_1) + x_1(y_2z_4 - y_4z_2) + x_2(y_4z_1 - y_1z_4),
\]

\[
6v_{04} = x_1(y_3z_2 - y_2z_3) + x_2(y_1z_3 - y_3z_1) + x_3(y_2z_1 - y_1z_2),
\]

and $v$ is the volume of the tetrahedron, which is given by

\[
\begin{vmatrix}
x_{14} & x_{24} & x_{34} \\
y_{14} & y_{24} & y_{34} \\
z_{14} & z_{24} & z_{34}
\end{vmatrix} = 6v.
\]

To simplify the expression hereafter it is convenient to represent the entries of the rightmost 3 columns of Equation (6.3) in a more compact notation:

\[
\begin{bmatrix}
\rho_1 \\
\rho_2 \\
\rho_3 \\
\rho_4
\end{bmatrix} = \frac{1}{6v}
\begin{bmatrix}
6v_{01} & a_1 & b_1 & c_1 \\
6v_{02} & a_2 & b_2 & c_2 \\
6v_{03} & a_3 & b_3 & c_3 \\
6v_{04} & a_4 & b_4 & c_4
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix},
\]

(6.4)
Taking partial derivatives to Equation (6.4) we have

\[
\frac{\partial \rho_i}{\partial x} = \frac{a_i}{6v}, \quad \frac{\partial \rho_i}{\partial y} = \frac{b_i}{6v}, \quad \frac{\partial \rho_i}{\partial z} = \frac{c_i}{6v}; \quad i = 1, 2, 3, 4. \tag{6.5}
\]

6.5 Displacement interpolation

The displacement field over the tetrahedron is defined by the three components \(u_x, u_y, u_z\), which can be linearly interpolated from the values at the nodes. Arranging it in matrix form yields

\[
u = \begin{bmatrix}
\psi_x \\
\psi_y \\
\psi_z
\end{bmatrix} = \begin{bmatrix}
\psi_{x1} & u_{x2} & u_{x3} & u_{x4} \\
\psi_{y1} & u_{y2} & u_{y3} & u_{y4} \\
\psi_{z1} & u_{z2} & u_{z3} & u_{z4}
\end{bmatrix} \begin{bmatrix}
\rho_1 \\
\rho_2 \\
\rho_3 \\
\rho_4
\end{bmatrix}.	ag{6.6}
\]

6.6 The strain field

The element strain field is strongly connected to the displacements by the strain-displacement equations, which in matrix notation read

\[
e = \begin{bmatrix}
e_{xx} \\
e_{yy} \\
e_{zz} \\
2e_{xy} \\
2e_{yz} \\
2e_{zx}
\end{bmatrix} = \begin{bmatrix}
\frac{\partial}{\partial x} & 0 & 0 \\
0 & \frac{\partial}{\partial y} & 0 \\
0 & 0 & \frac{\partial}{\partial z} \\
\frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \\
\frac{\partial}{\partial z} & \frac{\partial}{\partial x} & \frac{\partial}{\partial y}
\end{bmatrix} \begin{bmatrix}
\psi_x \\
u_y \\
u_z
\end{bmatrix} = Du,
\]
where \( e_{ii} \) are called true strains and \( 2e_{ij} \) are called shear strains. Combining this with Equation (6.6) with the partial differentiation relations in Equation (6.5) we obtain the matrix relation between strains and nodal displacements arranged node-wise,

\[
e = Bu^e,
\]

where

\[
B = \frac{1}{6\nu} \begin{bmatrix}
a_1 & 0 & 0 & a_2 & 0 & 0 & a_3 & 0 & 0 & a_4 & 0 & 0 \\
b_1 & 0 & 0 & b_2 & 0 & 0 & b_3 & 0 & 0 & b_4 & 0 & 0 \\
0 & c_1 & 0 & 0 & c_2 & 0 & 0 & c_3 & 0 & 0 & c_4 & 0 \\
b_1 & a_1 & 0 & b_2 & a_2 & 0 & b_3 & a_3 & 0 & b_4 & a_4 & 0 \\
0 & c_1 & b_1 & 0 & c_2 & b_2 & 0 & c_3 & b_3 & 0 & c_4 & b_4 \\
c_1 & 0 & a_1 & c_2 & 0 & a_2 & c_3 & 0 & a_3 & c_4 & 0 & a_4 \\
\end{bmatrix},
\]

and

\[
u^e = \begin{bmatrix} u_{x1} & u_{y1} & u_{z1} & u_{x2} & u_{y2} & u_{z2} & \cdots & u_{x4} \end{bmatrix}^T.
\]

6.7 The stress field

If the material is linearly elastic and no initial strains are considered, the constitutive equation between the stress field and the strain field can be compactly expressed as

\[
\sigma = Ee
\]
where $E$ is a $6 \times 6$ symmetric matrix which is called the elasticity matrix. If the material is isotropic, with elastic modulus $\epsilon$ and Poisson’s ratio $\nu$, the elasticity matrix is then

$$E = \frac{\epsilon}{(1 + \nu)(1 - 2\nu)} \begin{bmatrix} -\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1 - \nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1 - \nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} - \nu & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} - \nu & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} - \nu \end{bmatrix}.$$  

6.8 The element stiffness matrix

Introducing $e = Bu^e$ and $\sigma = Ee$ into the total potential energy functional restricted to the element volume and rendering the resulting algebraic form stationary with respect to the nodal displacements $u^e$ we get the element stiffness matrix

$$K^e = \int_{\Omega^e} B^\top EB \, d\Omega^e,$$

where $\Omega^e$ is the domain of the tetrahedron. Assuming that the elastic moduli $\epsilon$ is constant over the tetrahedron, the foregoing integrand is constant because the matrix $B$ in Equation (6.8) is constant. Since $v = \int_{\Omega^e} d\Omega^e$ we have

$$K^e = vB^\top EB,$$

where $K^e$ is of size $12 \times 12$.

The relationship between forces and displacements within an element is then represented by the system of linear equations

$$K^e u^e = f^e \quad (6.10)$$
where \( u^e \) is defined in Equation (6.9) and \( f^e \) is a \( 12 \times 1 \) vector containing forces applied to the nodes of the element.

6.9 Global stiffness matrix assembly

The number of global degrees of freedom, \( n_G \), is the product of the number of nodes in the whole mesh, \( |V^G| \) and the dimension of the space, \( D \). The elastic response of the entire mesh is captured by the system of linear equations

\[
K^G u^G = f^G, \tag{6.11}
\]

where \( K^G \) is the sparse \( n_G \times n_G \) global stiffness matrix whose nonzero components are computed by appropriately summing up components of the element stiffness matrices. \( u^G \) and \( f^G \) are vectors of size \( n_G \times 1 \) containing nodal displacements and forces respectively.

To carry out the summations needed to construct \( K^G \) each node of individual element has to be assigned a globally unique index. As each node has \( D \) degrees of freedom the same as the dimension of the space, each index corresponds to \( D \) rows and columns in \( K^G \). A two-dimensional lookup table, \( T_{e,d} \), which maps from the \( e \)-th element’s \( d \)-th degree of freedom to the global degree of freedom, is constructed to help the assembly of the global stiffness matrix from individual element stiffness matrix. Figure 6.2 shows a simple 2D triangular mesh and Table 6.1 shows its lookup table. Notice that a node shared by multiple elements always maps to the same \( D \) degrees of freedom and they are repeated in all the rows of the elements sharing the same node.
Figure 6.2: 2D mesh containing 9 nodes and 8 triangular elements. Individual degrees of freedom of each element at each node are numbered in red.

Table 6.1: Lookup table $T_{e,i}$ for the mesh shown in Figure 6.2.

<table>
<thead>
<tr>
<th>$d$ : Element Degree of Freedom</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>7</td>
<td>8</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>10</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>10</td>
<td>7</td>
<td>8</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>10</td>
<td>15</td>
<td>16</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>17</td>
<td>18</td>
<td>11</td>
<td>12</td>
<td>15</td>
<td>16</td>
</tr>
</tbody>
</table>

If $\mathcal{E}$ is the set of all elements in the mesh, the global stiffness matrix $K^G$ in Equation (6.11) can be formed by assembling individual element stiffness matrices $K^e$ in Equation (6.10) as follows.

$$K^G_{i,j} = \begin{cases} 
\sum_{e \in \mathcal{E}'} K^e_{d_i,d_j}, & \mathcal{E}' = \{ e \mid e \in \mathcal{E}; \exists d_i, d_j : T_{e,d_i} = i \text{ and } T_{e,d_j} = j \} \\
0, & \mathcal{E}' = \emptyset 
\end{cases} \quad (6.12)$$
6.10 Boundary conditions

The matrix in Equation (6.12) is weighted Jacobian and hence singular. Thus, in order to solve the system for an arbitrary force vector, the system of equations has to be constrained by boundary conditions to make it nonsingular. Essential, or Dirichlet, boundary conditions, which represent the known node displacements, can be used to reduce the system of equations and produce a nonsingular global stiffness matrix. For a 3D model, applying essential boundary conditions to a minimum of 3 nodes is sufficient for ensuring the system of equations to have unique solutions for an arbitrary force vector. On the other hand, natural, or Neumann, boundary conditions represent known forces applied to nodes of the mesh. These two types of boundary conditions are mutually exclusive.

We represent each boundary condition by a tuple \( \langle d, v \rangle \) where \( d \) is the index of the global degree of freedom and \( v \) is the value of the corresponding vector component. Collectively the sets of essential and natural boundary conditions are denoted as \( \mathcal{D} \) and \( \mathcal{N} \) respectively. Upon applying the boundary conditions, Equation (6.11) can be permuted and partitioned as follows.

\[
\begin{bmatrix}
K_{DD} & K_{DN} \\
K_{DN}^T & K_{NN}
\end{bmatrix}
\begin{bmatrix}
u_D \\
\mu_N
\end{bmatrix}
=
\begin{bmatrix}
f_D \\
f_N
\end{bmatrix}
\]

(6.13)

where

\( u_D \) are the known displacements,
\( u_N \) are the unknown displacements,
\( f_D \) are the unknown forces, and
\( f_N \) are the known forces.
Multiplying the second row of Equation (6.13) and rearranging it yields

\[ K_{NN}u_N = f_N - K_{ND}u_D. \]  

(6.14)

Now the unknown displacements \( u_N \) can be found by solving this equation. Substituting the solutions \( u_N \) back to Equation (6.13) gives the values of the unknown forces. Upon solving for \( u_N \) and \( f_N \) both the displacement and force values are known for all nodes. Equation (6.14) is called the reduced system of equations, which is simplified in the following compact notation:

\[ Ku = f, \]  

(6.15)

where

\[ K = K_{NN}, \]
\[ u = u_N, \]  
and
\[ f = f_N - K_{ND}u_D. \]

6.10.1 Boundary condition changes

New boundary conditions may be imposed over time. If a node \( v \) that has been experiencing an essential boundary condition now experiences a natural boundary condition, by Equation (6.15) the reduced system is now expanded. Equation (6.15) is then expanded and the right-hand side is changed. Specifically if \( D_v \) is the set of indices of the original essential boundary condition of \( v \), the modified system of equation is then

\[
\begin{bmatrix}
K & K_{ND_v} \\
K_{DN_v}^T & K_{D_vD_v}
\end{bmatrix}
\begin{bmatrix}
u \\
\bar{f}_{D_v}
\end{bmatrix} =
\begin{bmatrix}
f_N - K_{ND}u_D \\
\bar{f}_{D_v} - K_{D_vD}u_D
\end{bmatrix} =
\begin{bmatrix}
\tilde{f}_N \\
\tilde{f}_{D_v}
\end{bmatrix}.
\]  

(6.16)
where
\[ \hat{D} = D \setminus D_v. \] (6.17)

Schur complement method can be applied to Equation (6.16) to reduce it to
\[ \left( K_{D_vD_v} - K_{ND_v}^T K^{-1} K_{ND_v} \right) \begin{pmatrix} u_{D_v} \\ f_{D_v} \end{pmatrix} = \begin{pmatrix} \hat{f}_D \\ -K_{ND_v}^T K^{-1} \hat{f}_N \end{pmatrix}, \] (6.18)
in which \( u_{D_v} \) can be solved. Since \( K_{ND_v} \) is sparse in general and the LDL^T factors of \( K \) is known, the matrix \( K^{-1}(K_{ND_v}) = K_{ND_v}^T K^{-1} K_{ND_v} \) can then be computed in a similar way as the principal submatrix of the inverse of \( K \) as described in Section 5.3 via partial forward substitution. Specifically if we denote \( Z = L^{-1} K_{ND_v} \), Equation (6.18) would become
\[ \left( K_{D_vD_v} - Z^T D^{-1} Z \right) \begin{pmatrix} u_{D_v} \\ f_{D_v} \end{pmatrix} = \begin{pmatrix} \hat{f}_D \\ -Z^T D^{-1} L^{-1} \hat{f}_N \end{pmatrix}. \] (6.19)

Once \( u_{D_v} \) is computed, \( u \) can be solved by the first row block of Equation (6.16):
\[ Ku = \hat{f}_N - K_{ND_v} u_{D_v}. \] (6.20)

Alternatively, since \( K_{ND_v} \) is sparse in general, Equation (6.16) can be treated as a principal submatrix update to Equation (6.15). Hence, the AMPS algorithm with dimension change as described in Chapter 5 can be applied.

On the other hand, if a node which has been experiencing a natural boundary condition is now experienced an essential boundary condition, the reduced system is shrink. This happens when some nodes are moved and held in new positions by external forces. An augmented system in the form of Equation (3.3) can be formed to account for the shrink reduced system. In the standard formulation this is accomplished by deleting the associated rows and columns from the stiffness matrix. In the augmented matrix formulation, degrees of freedom are removed via steps that
resemble the effective column replacement procedure in Equation (3.2). The following example illustrates removal of the third degree of freedom from the augmented system.

\[
\begin{bmatrix}
K \\
\vdots \\
e_3^T \\
\end{bmatrix}
\begin{bmatrix}
\psi_1 \\
\psi_2 \\
\zeta_3 \\
v_4 \\
\vdots \\
\psi_3 \\
\end{bmatrix}
= \begin{bmatrix}
\psi_1 \\
\psi_2 \\
0 \\
0 \\
0 \\
0 \\
\end{bmatrix}
- \begin{bmatrix}
v_3Ke_3 \\
\end{bmatrix}.
\]  
(6.21)

As seen from Equation (3.2), the last row constrains \( \zeta_3 \) to be 0. Performing the multiplication of the third row yields

\[\kappa_{31}v_1 + \kappa_{32}v_2 + \kappa_{33}\zeta_3 + \kappa_{34}v_4 + \cdots - \psi_3 = -\kappa_{33}v_3.\]

Substituting \( \zeta_3 = 0 \) and rearranging the terms would get back the third row of the standard formulation. Similarly, performing the multiplication of the \( i \)-th row other than the third row yields

\[\kappa_{31}v_1 + \kappa_{32}v_2 + \kappa_{33}\zeta_3 + \kappa_{34}v_4 + \cdots = \psi_3 - \kappa_{33}v_3,\]

which is identical to the \( i \)-th row of the standard formulation after substitution of \( \zeta_3 = 0 \) and rearrangements of terms.

Here \( \psi_3 \) is moved from the right-hand-side vector to the solution vector since the force applied to the third degree of freedom becomes unknown after the imposition of Dirichlet boundary condition.

Comparing the matrices in Equations (3.3) and (6.21), we can easily observe that in the case of boundary condition changes, we can set \( J = H \). If \( D \) is the set of degrees
of freedom with boundary condition changes, then the solution and the right-hand side vectors of Equation (3.3) can be expressed as

\[
\hat{x}_1[i] = \begin{cases} 
\varphi_i & \text{if } i \notin \mathcal{D} \\
\zeta_i & \text{if } i \in \mathcal{D} 
\end{cases},
\]

\[
(6.22)
\]

\[
\hat{x}_2[i] = -\psi_{d_i}, \quad d_i \in \mathcal{D}.
\]

\[
(6.23)
\]

and

\[
\hat{b}[i] = \begin{cases} 
-\sum_{j \in \mathcal{D}} k_{ij}v_j & \text{if } i \in \mathcal{D} \\
v_i - \sum_{j \in \mathcal{D}} k_{ij}v_j & \text{if } i \notin \mathcal{D} 
\end{cases}.
\]

\[
(6.24)
\]

6.11 Topological mesh changes

All topological mesh changes can be accomplished using four operations: node deletion, node creation, element deletion, and element creation. The global stiffness matrix \( K^G \) in Equation (6.11) has to be modified to account for the topological changes. Node changes are reflected by the changes of the corresponding rows and columns as well as the size of \( K^G \) whereas element changes are reflected by the changes of entry values in \( K^G \). These changes may occur at the same time. For example, deleting an element may make a node to be isolated from the mesh and hence needs to be deleted.

6.11.1 Element changes

Recall from Equation (6.12) that the global stiffness matrix \( K^G \) is assembled by summing up individual element stiffness matrices. Hence the addition of any element \( e_a \) can be done by adding another element stiffness matrix \( K^{e_a} \) to the global stiffness matrix. Similarly the deletion of any element \( e_d \) can be done by subtracting the
corresponding element stiffness matrix $K^{ed}$ from the global stiffness matrix to cancel out its contribution. These operations are summarized by the following updates to $K^G$.

Element creation:

$$K^G[i,j] \leftarrow K^G[i,j] + \begin{cases} K^{ea}[d_i,d_j], & \text{if } \exists d_i,d_j : T[e_a,d_i] = i \text{ and } T[e_a,d_j] = j \\ 0, & \text{otherwise} \end{cases}$$ \hspace{1cm} (6.25)

Element deletion:

$$K^G[i,j] \leftarrow K^G[i,j] - \begin{cases} K^{ea}[d_i,d_j], & \text{if } \exists d_i,d_j : T[e_d,d_i] = i \text{ and } T[e_d,d_j] = j \\ 0, & \text{otherwise} \end{cases}$$ \hspace{1cm} (6.26)

Here $T[e,d]$ is the components in the lookup table that maps from the $e$-th element’s $d$-th degree of freedom to the global degree of freedom.

Notice that in both cases, a matrix of a smaller dimension is added to or subtracted from the global stiffness matrix $K^G$. This corresponds to a principal submatrix update to $K^G$ and consequently to $K$, the matrix in the reduced system 6.15.

6.11.2 Node changes

When a node is deleted from the mesh, the global degrees of freedom associated to the node, denoted by the set $\mathbb{X}_d$, are dropped from the finite element system of equations. To maintain the integrity of the model, all elements that have an edge connected to a node must be deleted from the mesh before the node is deleted. The changes in the global stiffness matrix reflecting the deletions of all the elements connected to the node are described in the previous subsection. After all those elements are deleted, since the node is fully disconnected from any element, all the rows and columns of $K^G$ from Equation (6.26) with indices of $\mathbb{X}_d$ must be 0, which can then be removed from the global stiffness matrix. Alternatively, to maintain the size as well
as the nonsingular property of the matrix, one can also replace the diagonal entries of $X_d$ by 1, which separates those rows and columns from the rest of the matrix while keeping the whole matrix nonsingular.

Node deletion:

$$K^G[i, j] \begin{cases} 1, & \text{if } j \in X_d \text{ and } i = j \\ K^G[i, j], & \text{otherwise} \end{cases} \quad (6.27)$$

where $K^G[i, j]$ on the right-hand-side is the resulting matrix from Equation (6.26).

![Diagram](image.png)

Figure 6.3.: A tetrahedron is being cut from top to bottom. Two nodes are created at the intersection of the cutting plane and an edge. The node intersected by the cutting plane is duplicated.

Nodes are created when new elements are added to the mesh or when the mesh is cut. When the mesh is cut, the cut may intersect at a node, on an edge or on a face. For the first case, a node is duplicated and all edges on one side of the cutting plane are disconnected from the original node and reconnected to the duplicated node. For the second case, two nodes are created, the intersecting edge is broken into two segments and the end nodes of the original edge are connected to the two new nodes separately. The third case can be considered as intersecting on two edges simultaneously. Figure 6.3 shows the first two cases when a tetrahedron is being cut. In any case all elements connected to the nodes and edges being cut are deleted and new elements are created to connect the newly created nodes and the nodes of the
original element. The changes in the matrix are therefore the same as described in the previous subsection.

Often some DoFs of the nodes being cut may remain unchanged after element deletion followed by element addition because the particular connection stays the same. For instance, suppose the duplicate of node $A$ in Figure 6.3 is behind the cutting plane and the original node $A$ is in front. Then node $A$ is still connected to nodes $B$ and $C$ in the same way as before the cut. Hence, the components of $K^G$ corresponds to those connections are not modified.

Since node additions can be represented by a sequence of element deletions and addition, and node deletions can be represented by removing the connections between those removed nodes and the mesh, these operations can also be done by performing principal submatrix updates to the global stiffness matrix $K^G$.

6.12 Unifying the formulation

Since topological mesh changes are essentially principal submatrix updates to the stiffness matrix $K^G$, they can also be expressed as column replacements to $K^G$. Notice also the similar structure of Equations (3.3) and (6.21), demonstrating that both topological changes and imposition of Dirichlet boundary conditions can be accomplished using a unified augmentation procedure. Next we provide the complete algorithm for formulating the augmented system that supports both replacement and expansion affecting multiple matrix columns.

Starting from Equation (3.6), we define the matrices $\tilde{J}$ and $\tilde{H}$ of sizes $(n + k) \times (m + k)$ as

\[
\tilde{J}[*, j] = \begin{cases} 
\hat{K}[*, \mathcal{L}_j] & \text{if } \mathcal{L}_j \notin \mathcal{D}, \text{ and} \\
\hat{I}_{n+k}[*] & \text{if } \mathcal{L}_j \in \mathcal{D}
\end{cases} \\
\tilde{H}[*, j] = \hat{I}_{n+k}[*], \quad \text{(6.28)}
\]

\[
\hat{H}[*, j] = I_{n+k}[*], \quad \text{(6.29)}
\]
Here $\mathcal{D}$ is the set of degrees of freedom constrained by Dirichlet boundary conditions, and $\mathcal{L}$ is an accessory data structure that maps the indices of columns and rows in $\tilde{J}$ and $\tilde{H}$ to the indices of columns in $K$ to be replaced, i.e. the $i$-th column of $\tilde{J}$ replaces the $\mathcal{L}_i$-th column of $K$. Hence, the $i$-th column of $\tilde{J}$ contains a copy of the $\mathcal{L}_i$-th column of $\tilde{K}$.

Augmented displacement and force vectors must have sizes and degree of freedom orderings consistent with the augmented stiffness matrix. The augmented displacement vector can be partitioned into two parts as shown in Equation (3.6) and is defined as

$$\tilde{x}_1[i] = \begin{cases} 
q[i] & \text{if } i \notin \mathcal{L}, \\
z_i & \text{if } i \in \mathcal{L}.
\end{cases}$$  \hfill (6.30)

$$\tilde{x}_2[i] = \begin{cases} 
q[\mathcal{L}_i] & \text{if } i \notin \mathcal{D}, \\
-f[\mathcal{L}_i] & \text{if } i \in \mathcal{D}.
\end{cases}$$  \hfill (6.31)

As in Equations (3.3) and (6.21), the $\zeta$ terms are constrained to have a value of zero, the $u$ terms represent unknown nodal displacements, and the $f$ terms represent the unknown nodal forces when a new Dirichlet boundary condition is imposed.

The augmented force vector is also partitioned into two parts: $\tilde{b}$ of length $(n+k)$, and a zero vector of length $m+k$. Some components of $\tilde{b}$ have terms subtracted to account for imposition of new Dirichlet boundary conditions. Here

$$\tilde{b}[i] = \begin{cases} 
q[i] - \sum_{j \in \mathcal{D}} K[i, j]u[j] & \text{if } i \notin \mathcal{D}, \\
- \sum_{j \in \mathcal{D}} K[i, j]u[j] & \text{if } i \in \mathcal{D}.
\end{cases}$$  \hfill (6.32)

The augmentation procedure with column replacements can be summarized by the following five steps.

1. Construct the accessory data structures $\mathcal{L}$ and $\mathcal{D}$. 
2. Form matrices $\tilde{J}$ and $\tilde{H}$ using Equations (6.28) and (6.29). Append $\tilde{J}$ to the right side of the stiffness matrix, $K$, and append $\tilde{H}$ to its bottom as shown in Equation (3.6).

3. Form the right-hand-side vector $\hat{b}$ using Equation (6.32).

4. Solve the augmented system of equations as described in Chapter 3.

5. After computing the solution, copy terms in $\hat{x}_2$ to the appropriate positions in the nodal displacement and force vectors as indicated by Equations (6.30) and (6.31), discarding the $\zeta_\mathcal{L}$ terms.

Alternatively, as shown in previous sections, any topological mesh changes can be expressed as principal submatrix updates to the reduced global stiffness matrix $\tilde{K}$. The resulting reduced global stiffness matrix $\tilde{K}$ is then

$$\tilde{K} = \tilde{K} - \tilde{H}\tilde{E}\tilde{H}^\top,$$  

(6.33)

where $\tilde{K}$ is the original reduced stiffness matrix $K$ augmented by $I_k$ as defined in Equation (5.1), $\tilde{H}$ is defined in Equation (5.3) where $H$ is the $m$ columns of the identity matrix $I_n$ corresponding to the modified rows and column of $K$, and $\tilde{E}$ is sum of the principal submatrix update $E$ and an identity of size $k$ at the $(2,2)$-block as defined in Equation (5.4) where $E$ is

$$E = \tilde{H}^\top(\tilde{K} - \tilde{K})\tilde{H}.$$  

(6.34)

The right-hand side vector $\hat{b}$ is the same as in Equation (6.32). The imposition of Dirichlet boundary conditions can also be solved by the method described in Section 5.4 with the new right-hand side vector $\hat{b}$.

The augmentation procedure with AMPS can be summarized by the following four steps.

1. Construct the accessory data structures $\mathcal{L}$ and $\mathcal{D}$. 

2. Form matrices $\tilde{E}$ and $\tilde{H}$ using Equations (6.29) and (6.34).

3. Form the right-hand-side vector $\hat{b}$ using Equation (6.32).

4. Solve the augmented system of equations as described in Chapter 5 using $K$, $\tilde{E}$, $\tilde{H}$ and $\hat{b}$.

6.13 Previous work

Beginning with Terzopoulos, et al. [12] [13], physics-based deformable models have been used for animation and simulation. By the mid-1990’s, a variety of work specific to surgery simulation began to appear [14] [15]. This section reviews existing approaches for computing physics-based deformation solutions, with a particular focus on methods that involve finite element analysis and cutting. Methods are categorized according to whether they use a direct solution approach with precomputation, an iterative solver, or a combination of both.

6.13.1 Precomputation approaches

Precomputation strategies accelerate the solution step of a simulation by shifting the bulk of the computational burden to a preprocessing stage. The bottleneck in a finite element simulation is the solution of a system of linear equations, $Ku = f$, where $K$ is the stiffness matrix, $a$ is a vector of nodal displacements, and $f$ is a vector of nodal forces. Precomputation methods such as [16] minimize the time required to calculate the solution vector by inverting $K$ before a simulation begins so that $u$ can be directly computed via the multiplication $u = K^{-1}f$ during the simulation. Since $K^{-1}$ is dense, condensation methods such as [15,17–19] further reduce computation time by producing from the full inverse matrix a smaller dimension one that contains only the equations necessary to compute a solution for a small subset of the nodes, such as a set of surface nodes. The inability to compute a solution for nodes not included in the preselected subset poses a problem for applications that involve cutting.
The Sherman-Morrison-Woodbury update formula has been used to address this by allowing selected degrees of freedom to be added back into a condensed stiffness matrix as they are needed. This approach was suggested by James and Pai [20], and later was used in needle insertion simulation [21,22], and in a cutting simulation [16]. As shown in Section 4.4, our AMPS algorithm is equivalent to the Sherman-Morrison-Woodbury update formula as used in these work. However, our algorithm does not need to precompute and store the potentially denser matrices $A^{-1}$ or $A^{-1}B$ for some matrix $B$. Moreover, we focus on the principal submatrix update but not the whole columns unlike other aforementioned methods. Finally, our AMPS algorithm also supports any dimension changes that arise from boundary condition and topological changes.

A variation on the precomputed inverse approach sharing some similarities with our work is the precomputed stiffness matrix factorization described by Turkiyyah [23] which updated a Cholesky factorization to accommodate the addition and modification of discontinuous basis functions along a cutting path. While our work updates the solutions to the modified system, it does not update the Cholesky factorization of the stiffness matrix. Our methods also support any local mesh modification, whether from remeshing or addition of basis functions. As a result, our methods’ update process substantially differs from Turkiyyah’s work.

Solution techniques that rely on the superposition principle such as [24], [25], and [26] precompute and record the set of node displacements that result from a constraint being applied to a single node. This computation is repeated for every node that might be subject to a constraint, and all the results are stored. At runtime nodal displacements are computed as a linear combination of the stored results. For some applications this approach can closely approximate the ideal solution, but without modification it cannot handle changes in mesh connectivity.

Precomputed Green’s functions have also been used by Nikitin et al. [27], and James and Pai [28] to quickly compute deformation solutions for subsets of mesh nodes. Similarly, the banded matrix method proposed by Berkley et al. [29] prioritizes
and rearranges the rows and columns of a stiffness matrix based on node type, then factors the stiffness matrix in such a way that a fast update is provided only for the highest priority nodes. In both cases, solutions for internal nodes are generally not computed.

A limitation generally shared by precomputation approaches is that results produced in the precomputation phase are invalidated when the topology of the mesh changes, so cutting and remeshing require special consideration. Constraint removal is a precomputation approach that requires cutting paths to be known a priori. Lindblad and Turkiyyah [30] and Sela et al. [31] have demonstrated how duplicate nodes along a cutting path can be constrained to move together until they are cut, at which time the constraint is removed to open up a predefined cut.

Discontinuous basis functions provide a more flexible cutting scheme that has been used in concert with precomputation. This approach was originally introduced in the engineering literature as a way of studying crack formation [32] and more recently has been applied to the problem of cutting in surgical simulations [23,33]. Unpredictable and arbitrary cutting paths are accommodated through the addition of new degrees of freedom that use discontinuous interpolation functions to account for mid-element breaks in nodal influence. As shown in [30], an update procedure similar to the Sherman-Morrison-Woodbury update can be employed to update a precomputed inverse stiffness matrix to account for the new degrees of freedom. Because the complexity of Sherman-Morrison-Woodbury update is cubic with respect to the number matrix rows and columns changed, this works well only when the modifications are very limited.

6.13.2 Iterative solvers

Iterative solvers do not share the same limitations as precomputation methods because all of the calculations needed to produce a solution occur at runtime. Thus iterative solvers can be successfully applied when stiffness matrix updates are caused
by topological mesh changes. However, using an iterative solver does require that attention be paid to issues of convergence and stability.

Conjugate gradient solvers have been frequently used with finite element simulations [34–36]. The popularity and relatively straightforward implementation of the conjugate gradient algorithm make the method a good benchmark for comparisons with alternative solution methods. Conjugate gradient implementations that take advantage of sparse matrix-vector multiplication have been used for interactive applications and can be accelerated with parallel [37] and GPU implementations [38]. However, the simulation community continues to seek solution methods that outperform conjugate gradient, as real-time performance on higher resolution models promises improved realism.

Some of the most recent work in interactive finite element simulation has explored the use of multigrid solution methods [39–42]. Multigrid methods are among the most efficient iterative solution approaches and accelerate convergence by reducing error at multiple spatial resolutions. However, they also have higher fixed overhead costs than methods such as conjugate gradient. A GPU implementation of multigrid by Dick, et al. [43] has demonstrated further speed improvement. The need for a multi-resolution mesh makes multigrid naturally suited for structured meshes, with hexahedral grids typically being used. Although hexahedral grids do not lend themselves to smooth cutting surfaces, recent work by Zhu et al. [40] has demonstrated a way to incorporate cutting into a simulation with a multigrid solver.

A final category of iterative solvers is explicit integration methods. Explained in detail in [44], it was originally suggested for use in surgery simulation by Bro-Nielsen [17] and also implemented in the software suite described in [45]. Explicit integration has been successfully used in real-time simulation with dynamic and non-linear finite element models [46], and has been used in simulations involving surgery [47] and cutting [48]. Care must be taken in selecting the time step size for explicit integration because it can be numerically unstable if the time steps are too large.
6.13.3 Hybrid solution methods

Some simulations have been implemented using hybrid approaches that use two or more solution methods. Typically, some portion of a model is designated as susceptible to cuts and deformation while the remainder is subject only to deformation. The strategy is to apply a fast precomputation approach to the portion of a model that cannot be cut and apply a slower method that supports cutting to the remainder. For example, Wu and Heng combine the use of condensation and conjugate gradient solvers [49, 50] while Cotin et al. combine the use of a linear superposition method with explicit integration applied selectively to the dynamic, cuttable portion of a mesh [51]. Kocak et al. [52] provided further support for this approach by describing a framework for building a consistent finite element simulation when different regions of the mesh are solved at different update rates.
7 APPLICATION TO N − X CONTINGENCY ANALYSIS OF POWER-FLOW

In this chapter we present another application of the AMPS algorithms, the $N - x$ contingency analysis of power flow problem, which evaluates the stability of a power system by simulating the changes in the system under the circumstance of the failures of $x$ out of $N$ components, which can be generators, transmission lines or loads. Currently each analysis case is performed independently. However, since the number of cases increases exponentially with $x$, this becomes computationally impractical to solve all cases in a reasonable amount of time [53]. For example, if $N = 3000$ and $x = 5$, there are in total $C_{3000,5} \approx 2.018 \times 10^{15}$ cases. These numbers are typical in cascading failure analysis for a regional power grid. Due to such computational constraints, only a very limited number of $N - 1$ and $N - 2$ cases selected based on heuristics are analyzed, and the analysis is incomplete, introducing risks in power grid reliability. Bienstock discusses a mixed-integer programming approach to this problem [54], which restricts the size of the problems they can solve to a few hundred buses. A typical example is the Energy Management System at Bonneville Power Administration (BPA), one of the well-maintained systems. It runs 500 contingency cases in a time interval of five minutes. In order to analyze more cases, we need to reduce the solution time for contingency analysis.

In the following sections we will derive the simplified “DC” power-flow problem, which is an approximation to the system of non-linear complex equations of the AC power-flow problem, followed by a brief description of the $N - x$ contingency analysis. Finally we illustrate how the AMPS algorithms can be used to solve the contingency analysis problems efficiently.
7.1 Power flow equation

Based on the Kirchhoff's current law, for AC circuits in low frequencies where the wavelengths of electromagnetic radiation are very large compared to the circuits, which is the case for real-world power grid systems, the algebraic sum of currents in a network of conductors meeting at a point is zero, i.e.

$$\sum_l I_{kl} = 0 \quad \forall k \in B,$$

(7.1)

where $I_{kl}$ is the complex current flowing from bus $k$ to bus $l$, and $B$ is the set of buses in the power network. The net current at a bus $k$ is the difference between the constant-power injected by either the generators or the loads and the net flow of currents between the bus $k$ and its neighboring buses in the power grid. In other word, these two terms are equal to each other, i.e.

$$\sum_l Y_{kl}v_l = \frac{s^*_k}{v^*_k} \quad \forall k \in B,$$

(7.2)

where $Y_{kl}$ is the generalized admittance between buses $k$ and $l$, $v[k]$ and $s[k]$ are the voltage and the power at bus $k$ respectively. All these quantities are complex numbers. They can be expressed as real and imaginary parts or magnitude and angle parts as follows:

$$Y_{kl} = G_{kl} + jB_{kl},$$

(7.3)

$$s_k = p_k + jq_k, \quad \text{and}$$

(7.4)

$$v_k = |v_k|e^{j\theta_k},$$

(7.5)
where \( j = \sqrt{-1} \) is the pure imaginary number, \( p_k \) and \( q_k \) are the active and reactive powers at bus \( k \), and \(|v_k|\) and \( \theta_k \) are the magnitude and phase shift of the voltage at bus \( k \), and \( G_{kl} \) and \( B_{kl} \) are defined as

\[
G_{kl} + jB_{kl} = \begin{cases} 
-\hat{G}_{kl} - j\hat{B}_{kl} & \text{if } k \neq l, \\
\hat{G}_{kk} + j\hat{B}_{kk} + \sum_{l \neq k} \left( \hat{G}_{kl} + j\hat{B}_{kl} \right) & \text{if } k = l,
\end{cases}
\]  

(7.6)

where \( \hat{G}_{kl} \) and \( \hat{B}_{kl} \) are the conductance and susceptance of the transmission line between buses \( k \) and \( l \) whereas \( \hat{G}_{kk} \) and \( \hat{B}_{kk} \) are the shunt conductance and susceptance between bus \( k \).

![Diagram of the power system](image)

(a) Diagram of the power system

![Sparsity pattern of the admittance matrix](image)

(b) Sparsity pattern of the admittance matrix

Figure 7.1.: An example of a power system. The link in blue between bus 5 and bus 7 is removed in a contingency analysis case. The values of the blue dots in the admittance matrix are changed and those of the blue circles become zero.
The right-hand side contains the constant-power injections, which are the nonlinear part. The admittance matrix $Y$ captures the effects of all linear devices, typically includes line admittance, transformers, line susceptance, bus shunt admittance, constant-impedance loads, and constant-current injections. It also captures the connectivity of the network itself since $Y_{kl}$ is nonzero if and only if the buses $k$ and $l$ are connected. Figure 7.1 shows an example of a power system and the corresponding sparsity pattern of its admittance matrix $Y$.

7.2 “DC” power-flow approximation

As we observe that Equation (7.2) is a complex nonlinear equation, it is computationally expensive to obtain a solution to it. In order to accelerate the contingency analyses, we approximate the power-flow equation by several heuristics from real-world observations. We start with substituting Equations (7.3) to (7.5) into Equation (7.2) and rearranging the terms:

$$
\sum_l \left( G_{kl} + jB_{kl} \right) |v_k||v_l|e^{j(\theta_l-\theta_k)} = p_k - jq_k \quad \forall k \in B. \tag{7.7}
$$

**Observation 1** The resistance $R[k,l]$ of transmission lines is significantly less than the reactance $X[k,l]$. Hence, we have

$$
Y_{kl} = \frac{R_{kl}}{R_{kl}^2 + X_{kl}^2} - j \frac{X_{kl}}{R_{kl}^2 + X_{kl}^2} \\
\approx -j \frac{X_{kl}}{R_{kl}^2 + X_{kl}^2} \\
= jB_{kl}, \tag{7.8}
$$

since the first term is close to 0.
Observation 2 For most typical operating conditions, the difference in angles of the voltage phasors at two buses $k$ and $l$ connected by a transmission line, which is $\theta_l - \theta_k$. Hence, we have

$$e^{j(\theta_l - \theta_k)} = \cos(\theta_l - \theta_k) + j \sin(\theta_l - \theta_k) \approx 1 + j(\theta_l - \theta_k).$$  \hspace{1cm} (7.9)

Substituting Equations (7.8) and (7.9) into Equation (7.7) and separating the real and imaginary parts yields

$$\sum_l \hat{B}_{kl} |v_k||v_l| (\theta_k - \theta_l) = p_k, \text{ and}$$  \hspace{1cm} (7.10)

$$\sum_l \hat{B}_{kl} |v_k||v_l| = -q_k.$$  \hspace{1cm} (7.11)

We focus on Equation (7.10) since $p_k \gg q_k$ in real-world circuits.

Observation 3 In the per-unit system, the numerical values of voltage magnitudes $|v_k|$ and $|v_l|$ are close to 1.0. Hence, Equation (7.10) would become

$$\sum_{l \neq k} \hat{B}_{kl} (\theta_k - \theta_l) = p_k.$$  \hspace{1cm} (7.12)

Rearranging terms in Equation (7.12) yields

$$\sum_{l \neq k} B_{kl}\theta_k - \sum_{l \neq k} \hat{B}_{kl}\theta_l = p_k,$$  \hspace{1cm} (7.13)

which can be expressed in matrix form:

$$B'd = p,$$  \hspace{1cm} (7.14)
where \( d \) and \( p \) are \((n + 1)\)-vectors, and \( B' \) is a matrix of size \((n + 1) \times (n + 1)\) and is defined as

\[
B'_{kl} = \begin{cases} 
\sum_{l \neq k} B_{kl} & \text{if } l = k, \\
B_{kl} & \text{if } l \neq k.
\end{cases}
\tag{7.15}
\]

7.3 Bus classification

There are usually three types of buses in a power flow system: the load bus (PQ bus), denoted by \( Q \), the voltage-controlled bus (PV bus), denoted by \( V \), and the slack bus, which serves as a boundary condition for the system. Each type of buses has different known and unknown variables, which is summarized in the following table.

Table 7.1: Types of buses and their known and unknown variables

<table>
<thead>
<tr>
<th>Bus type</th>
<th>Index set</th>
<th>Known variables</th>
<th>Unknown variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slack</td>
<td>( {0} )</td>
<td>(</td>
<td>v</td>
</tr>
<tr>
<td>Load (PQ)</td>
<td>( Q )</td>
<td>( p, q )</td>
<td>(</td>
</tr>
<tr>
<td>Voltage-controlled (PV)</td>
<td>( V )</td>
<td>( p,</td>
<td>v</td>
</tr>
</tbody>
</table>

Notice that \( B' \) in Equation (7.14) is weighted Laplacian and hence singular. In order to solve for \( d \), we use the slack bus in the equation as the boundary condition and remove the corresponding row and column. In the “DC” power-flow approximation 7.14, we only focus on the variables \( p \) and \( d \). Since \( p \) is known and \( d \) is unknown in both load and voltage-controlled buses, the “DC” power-flow equations are identical for both buses. We combine them in a single linear system of equations of size \( n \times n \) where \( n = |Q| + |V| \).

7.4 Contingency analysis

We observe from Equations (7.6) and (7.15) that \( B' \) has the same sparsity pattern as \( Y \). If, in a contingency case, the link between buses \( k \) and \( \mathbf{k} \) is disconnected, the
off-diagonal components \( B'[i,j] \) and \( B'[j,i] \) become zeros and the values of diagonal components \( B'[i,i] \) and \( B'[j,j] \) are also updated. In the example shown in Figure 7.1, the link in blue between bus 5 and bus 7 is removed and thus the values of \( B'[5,5], B'[5,7], B'[7,5] \) and \( B'[7,7] \) are changed to \( \hat{B}'[5,5], 0, 0 \) and \( \hat{B}'[7,7] \) respectively and the rest of the matrix \( B' \) remains the same. Notice that this corresponds to a principal submatrix update \( E \) of size \( m \times m \) to the matrix \( B' \) and \( m \leq 2x \) if we remove \( x \) components from the power system. Mathematically,

\[
\hat{B}' = B' - HEH^\top,
\]  

(7.16)

where \( H \) is the \( m \) columns of an identity matrix \( I_n \) in the set \( \mathcal{H} \) of buses connected by the \( x \) removed components and

\[
E[i,j] = \begin{cases} 
    B[H_i,H_j] & \text{if } i \neq j, \\
    \sum_{k \neq H_i} B[H_i,k] & \text{if } i = j.
\end{cases} 
\]  

(7.17)

7.5 Previous work

To tackle the computational complexity of contingency analysis, significant research has been devoted to contingency selection and more recently, on parallel computing.

7.5.1 Contingency selection

Contingency selection aims to choose a subset of the contingency cases with the assumption that the unchosen cases would not cause outages or blackouts. Many methods have been developed for contingency selection, including heuristic and historical knowledge, performance-index-based ranking [55, 56], concentric relaxation [57], sparse vector methods [58], partial refactorization method [59], quadratized sensitivity analysis [60], minimum graph cut [61, 62], and betweenness centrality [63]. Contin-
gency selection effectively reduced the number of contingency cases to be analyzed. But none of the selection methods is perfect, resulting in risks of missing important cases.

7.5.2 Parallel computing

Parallel computing work tackles this issue from a different angle, with the aim to reduce the total wall-clock time through parallel processing. Contingency cases are relatively independent of one another, so contingency analysis is inherently a parallel process. Distributed client-server scheme [64], distributed networking [65], parallel virtual machines (PVM) [66], master-slave models [67], and dynamic load balancing [53] have been applied to the parallel processing of contingency analysis, focusing on the efficient use of computing resources. The best result reported in literature is 67 seconds to solve one million $N - 2$ contingency cases using 10,240 cores, with a speedup of $7877 \times$ [68]. $N - x$ contingency analysis would involve solving significantly more cases.
8 EXPERIMENTAL RESULTS

The column replacing augmented matrix solution method and AMPS algorithm with dimension changes were evaluated through the application to surgical simulations demonstrated with finite element deformation and cutting experiments with five model types, whereas the AMP algorithm without dimension changes is evaluated through $N - x$ contingency analyses of two real-world and a synthesized power-flow systems. This chapter provides relevant implementation details and presents experimental data. For the surgical simulation experiments, comparisons between the column replacing augmented matrix solution method and AMPS algorithm with dimension changes as well as both non-preconditioned and Jacobi preconditioned conjugate gradient (CG) solvers were provided. ILU0 and ILUT preconditioners for CG were tested, but the reduction in number of iterations did not compensate for the increased computation complexity per iteration. For the $N - x$ contingency analysis experiments, the AMPS algorithms with direct and iterative options were compared with the PARDISO direct solver [69–71] on the modified systems, and the CHOLMOD direct solver that updates the factors of $A$ according to $\hat{A}$.

8.1 Implementation

The AMPS algorithms can be implemented on any platform with a suitable linear algebra library for sparse and dense matrix and vector operations, including matrix-matrix and matrix-vector multiplications for symmetric and unsymmetric dense matrices, matrix-vector multiplication for sparse matrices, as well as triangular solves for sparse matrices and dense vectors. Currently, our implementation relies on the dense and sparse BLAS subroutines as well as the iterative solvers such as GMRES and CG in the Intel Math Kernel Library (MKL) [72]. However, with a suitable
library substitution, it can also run on non-Intel platforms. The precomputed $\text{LDL}^T$ factorization of the matrices were computed using OBLIO, a sparse direct solver library [73]. The PARDISO direct solver used for comparison purposes was from MKL. The CHOLMOD solver applied to Equation (1.2) was from the SparseSuite package. The remainder of the codes were written in C++.

8.2 Matrix format

Matrices with $n$ rows or columns were stored in sparse matrix format to reduce both the storage space and access time, whereas augmentation matrices with much fewer than $n$ rows or columns were stored in dense matrix format for fast computations. Since the closure of a set of indices in the graph of a triangular matrix can be found effectively column by column, and OBLIO uses supernodes in matrix factorization, all sparse matrices were stored in compressed sparse column (CSC) format for efficient column access. The matrix $H$ and its transpose were represented as an array of indices and their multiplications with other matrices were done by index mappings.

8.3 Target hardware

Since our implementation uses MKL, we primarily focus on Intel architecture. Our results were obtained on the following architectures:

1. a Linux desktop computer with four 8-core Intel Xeon E5-2670 processors running at 2.6GHz with 20 MB L3 cache and 256 GB RAM. We refer to this architecture as machine 1.

2. a node of Linux cluster with 2 sockets with a 16-core Intel Xeon E5-2698 v3 (“Haswell”) processor running at 2.3 GHz with 40 MB L3 cache and 128 GB DDR4 RAM at 2133 MHz. We refer to this architecture as machine 2.
The surgical simulation experiments were conducted on both machines 1 and 2 whereas the $N - x$ contingency analysis experiments were conducted on machine 1. All data represent an average timing from 20 runs.

8.4 Surgical simulations

8.4.1 Model meshes

Five types of solid tetrahedral meshes were used in the surgical simulation experiments.

1. *Elongated Beam:* A group of five elongated rectangular solids with varying lengths were generated. Nodes were placed at regularly spaced grid points on a $5 \times 5 \times h$ grid, where $h$ ranged from 4 to 1024. Each block mesh was anchored at one end of the solid. All elements had good aspect ratios and were arranged in a regular pattern. However, models with greater degrees of elongation produced more poorly conditioned systems of equations, as fixation at only one end meant that longer structures were less stable. Thus experiments with this group of meshes illuminates the way solver performance varies with
stiffness matrix conditioning. The estimated condition numbers of the beam mesh stiffness matrices range from $1.14 \times 10^3$ to $3.29 \times 10^{12}$.

![Figure 8.2: Rendering of the brick mesh with cut.](image)

2. **Brick:** A group of five rectangular brick solids with varying mesh resolutions were generated. Each of the models had the same compact physical dimension of $1 \times 1 \times 2$. An initial good-quality mesh was uniformly subdivided to produce meshes of increasingly fine resolution. These meshes allowed us to examine solver performance relative to node count for fixed model geometry. Similar to the beam meshes, zero-displacement boundary conditions were applied to one face of the block. The estimated condition numbers of the brick mesh stiffness matrices range from $2.19 \times 10^3$ to $1.18 \times 10^5$.

3. **Stanford bunny:** A 20,133 node mesh of the Stanford bunny [74] is used to demonstrate solver performance on an irregular mesh. Zero-displacement boundary conditions are applied to nodes on the bottom of the bunny’s feet. The bunny mesh stiffness matrix has an estimated condition number of $6.17 \times 10^7$. 
Figure 8.3.: Rendering of the Stanford bunny mesh with cut on its back.

Figure 8.4.: Rendering of the eye model are shown with incisions along the corneal limbus, to correct for astigmatism, used in the experiments.

4. *Eye:* Two resolutions of a human eye model [75] with a clear corneal cataract incision were used to demonstrate applicability to surgery simulation. The models contained 4,444 and 16,176 nodes, and zero displacement boundary conditions
were applied to the posterior portion of the globe. The eye mesh stiffness matrices have estimated condition numbers of $2.66 \times 10^6$ and $1.62 \times 10^7$ respectively.

![Diagram of brain model with incisions](image)

Figure 8.5: Rendering of the brain model are shown with incisions on the superior portion of the right frontal lobe used in the experiments.

5. **Brain:** Two resolutions of a human brain model (contributed by INRIA to the AIM@SHAPE Shape Repository) were used to demonstrate applicability to surgery simulation on an organ of complicated structure. The models contained 23,734 and 50,737 nodes, and zero displacement boundary conditions were applied to the interior portion of the brain. The small brain mesh stiffness matrix has estimated condition number of $4.64 \times 10^7$. The condition estimation failed for the large brain mesh due to insufficient memory.

On average, the nodes in the brick meshes have a higher degree of connectivity than those in the elongated beam meshes. This is due to a greater proportion of surface nodes in the beam models versus interior nodes in the brick models. The increased connectivity leads to a higher percentage of non-zeros in the stiffness matrix factors and larger sizes for the closures referenced in Table 3.1. These differences have a significant impact on the relative performance of the solution methods. Figure 8.6 compares $|\text{closure}_L(J\hat{x}_2)|$ for the different test meshes during the cutting
Figure 8.6.: $|\text{closure}_L(J\hat{x}_2)|$ vs. mesh node count.

experiments. The set closure$_L(J\hat{x}_2)$ is the largest of the closures referenced in the complexity analysis in Table 3.1, and is a measure of the size of the triangular system to be solved. As expected, brick meshes have larger closures than the other two meshes.

8.4.2 Experiments

Performance was examined through two types of experiments: deformation of intact meshes, and deformation of meshes undergoing cutting.

Deformation of Intact Meshes

Figures 8.7 and 8.8 shows how solution time varied with the number of constrained nodes for instances of the beam and brick meshes. AMPS outperformed both augmented matrix and CG methods, and maintained a high update rate for both meshes throughout. The augmented method came in the second place, maintaining update
Figure 8.7.: Update rates for the deformation experiments of beam mesh of 6,400 nodes.
Figure 8.8.: Update rates for the deformation experiments of brick mesh of 9,537 nodes.
Figure 8.9: Average update rates and ranges are shown for the deformation experiments of the series of beam meshes.
Figure 8.10: Average update rates and ranges are shown for the deformation experiments of the series of brick meshes.
Figure 8.11.: A portion of the tetrahedral brick test mesh. Node $A$ has 13 connected nodes (colored in orange) whereas Node $B$ only has 5 (colored in red).

rates around 200 Hz. CG performed the worst, providing updates in the range of 1.6–33 Hz for the first 19 cutting steps, and experienced a zig-zag pattern afterwards caused by the connectivity pattern of nodes in the tetrahedral brick mesh. Periodically, nodes with a higher degree of connectivity were cut. These cutting steps required a larger number of changes to the stiffness matrix and resulted in periodically slower CG solution times. The connectivity pattern is illustrated in Figure 8.11. This pattern also appeared in the results of the cutting experiments of the beam and brick meshes, as well as the eye mesh as they have a structural pattern in the ellipsoidal shapes.

For brick meshes, AMPS vastly outperformed the other methods again, and maintained relatively constant update rates. The augmented method outperformed CG as constraints were applied to the first dozen nodes, but performance dropped as the number of constrained nodes increased, eventually resulting in similar update rates between the augmented method and CG.

Figures 8.9 and 8.10 are log-log plots that shows how solution times varied for different sizes of beam and brick meshes. The lines show the trend of the average
times for various methods and the shaded areas are the ranges of the solution times. These graphs show that AMPS ran faster than both the augmented and CG methods for the beam meshes except for the very smallest instance that had only 100 nodes. It can also be observed that CG has the largest ranges among all methods especially for the larger beam meshes. This means that the CG solution times fluctuated a lot while the deformation progressed. For the brick meshes, AMPS also outperformed both the augmented and CG methods with smaller solution time ranges than the other methods.

Deformation of Meshes Undergoing Cutting

In this group of experiments we made an advancing planar cut into the volume of each mesh. As a cut progressed, a duplicate of each node along the cut path was added to the mesh, and connectivity was modified so that elements on opposite sides of the cut became separated. The newly added node necessitates a dimensional expansion of the system and the remeshing process of the duplicated node and all its neighboring nodes results in a principal submatrix update to the stiffness matrix. Opposing force vectors were applied to selected surface nodes to pull the cut faces apart. Figures 8.1 to 8.3 shows the three test meshes at the initial stages of cutting.

While the other methods behaved differently for the cutting and deformation experiments for the beam and brick meshes, AMPS performed similarly between the two experiments as shown in Figures 8.12 and 8.13 compared to Figures 8.7 and 8.8. AMPS outperformed both the augmented and CG in the beam cutting experiments, providing updates in the range 167–479 Hz. Augmented method provided 0.3– 209 Hz whereas CG needed more than 1 second for most of the cutting steps except for the first one, and failed to converge to any solution after the 18th step. On the other hand, AMPS performed on par with CG for the brick mesh cutting experiment while the augmented methods underperformed, as shown in Figure 8.13.
Figure 8.12. Update rates for the cutting experiments of beam mesh of 6,400 nodes.
Figure 8.13.: Update rates for the cutting experiments of brick mesh of 9,537 nodes.
Figure 8.14.: Average update rates and ranges are shown for the cutting experiments of the series of beam meshes.
Figure 8.15: Average update rates and ranges are shown for the cutting experiments of the series of brick meshes.
Figure 8.16: Update rates for the cutting experiments of Stanford bunny mesh of 20,133 nodes.
Figure 8.17.: Breakdown of the times for the cutting experiments of the Stanford bunny mesh using two augmented matrix methods.
Figure 8.18: Update rates for the cutting experiments of eye mesh of 4,444 nodes.
Figure 8.19.: Update rates for the cutting experiments of eye mesh of 16,176 nodes.
Figure 8.20.: Update rates for the cutting experiments of brain mesh of 23,736 nodes.
Figure 8.21.: Update rates for the cutting experiments of brain mesh of 50,737 nodes.
Figures 8.14 and 8.15 shows that the beam vs. brick performance trend held over a variety of mesh sizes. AMPS provided the fastest updates when cutting a beam mesh, maintaining an update rate over 50 Hz even with a relatively large cut in a 25,600 node mesh. Particularly for the larger beam meshes, CG was often unable to provide any solution. AMPS provided an on par performance with CG when cutting a brick mesh while the SPAI preconditioned augmented method performed the worst.

There are two primary reasons for the disparity between the beam mesh and brick mesh results. First, the beam meshes have a higher percentage of surface nodes, resulting in sparser matrix factors and smaller closure sizes, as shown in Figure 8.6. Smaller closures result in faster execution of the augmented solution steps, particularly the GMRES iterations when solving for the augmented part of the column replacing augmented method and computing the principal submatrix of the inverse of the original matrix in the AMPS algorithm. Thus we see that the structure of a mesh is an important factor in determining whether the augmented method will be a particularly efficient solution method for a given problem. In general, the augmented method is particularly attractive for meshes that have greater amounts of surface area relative to their volume.

The second reason for the wide disparity in results is that the brick meshes had particularly well-conditioned stiffness matrices while the beam meshes had more poorly conditioned stiffness matrices. Iterative methods can converge very slowly or fail to converge at all when systems are not sufficiently well-conditioned. In contrast, the direct solution approach provided by the augmented factors is more robust in poorly conditioned scenarios. We conclude that the augmented method is particularly appropriate when a problem would benefit from the robustness of a direct solution approach but also needs the flexibility to update the system due to cutting or other changes.

Results from the bunny mesh cutting experiment are shown in Figure 8.16. Here again we find that AMPS performed the best, with an update rate ranged from 42 Hz to 168 Hz, compared to the 0.3–6.8 Hz update rate by preconditioned CG and the 5.1–53.9 Hz update rate by the nonpreconditioned augmented method. Figure 8.17(a)
Table 8.1.: Maximum number of GMRES iterations of SPAI preconditioned augmented method for the cutting experiments of the beam and brick meshes of various sizes.

<table>
<thead>
<tr>
<th>Size</th>
<th>100</th>
<th>400</th>
<th>1,600</th>
<th>6,400</th>
<th>25,600</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max GMRES iterations</td>
<td>26</td>
<td>56</td>
<td>66</td>
<td>79</td>
<td>74</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Size</th>
<th>225</th>
<th>1,377</th>
<th>4,225</th>
<th>9,537</th>
<th>18,081</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max GMRES iterations</td>
<td>39</td>
<td>40</td>
<td>57</td>
<td>65</td>
<td>50</td>
</tr>
</tbody>
</table>

shows that the bulk of the computation time of the SPAI preconditioned column replacing augmented method is spent in the GMRES iteration of Step 2 in the bunny mesh cutting experiment. The dominance of the GMRES iterations in the distribution of computing time is also a feature of the experiments with beam and brick meshes. However, Table 8.1 demonstrates that the number of GMRES iterations needed for convergence does not grow with model size. The diminishing trend of the performance is more visible for this mesh as the size of the cut and complexity of the attendant remeshing grows. This can be explained by the increasing time spent on solving for \( \tilde{a}_2 \) as the size of the augmentation increases as shown in Figure 8.17(b). The most computationally expensive steps were the computation of the principal submatrix of the inverse and the triangular solve for the final solution \( \tilde{a} \). The valleys in the area plot are due to the fact that at some cuts no additional neighboring vertices were included in \( \mathbb{H}_{\Delta t} \) and thus \( \text{tril}(K_{\Delta t}^{-1}(H)) \) is empty and need not be computed.

Results from the eye mesh cutting experiments are shown in Figures 8.18 and 8.19, and those from the brain mesh cutting experiments are shown in Figures 8.20 and 8.21. Here we show that AMPS vastly outperformed both the augmented and CG methods with and without preconditioning. The numbers of updates per second achieved on both the eye and brain meshes make interactive stimulation feasible.

Since AMPS uses direct solver in both augmented part and the whole solutions, the solution accuracy of AMPS is only affected by the numerical errors. Hence, AMPS not
Table 8.2.: Comparison of average relative residual norms ($\|\hat{K}\hat{a} - \hat{f}\|_2 / \|\hat{f}\|_2$) for the cutting experiments

<table>
<thead>
<tr>
<th>Mesh</th>
<th>Size</th>
<th>AMPS</th>
<th>SPAI precond. aug.</th>
<th>Jacobi precond.</th>
<th>CG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beam</td>
<td>25,600</td>
<td>$7 \times 10^{-11}$</td>
<td>$1 \times 10^{-4}$</td>
<td>failed to converge</td>
<td></td>
</tr>
<tr>
<td>Brick</td>
<td>18,081</td>
<td>$3 \times 10^{-14}$</td>
<td>$5 \times 10^{-5}$</td>
<td>$5 \times 10^{-5}$</td>
<td></td>
</tr>
<tr>
<td>Bunny</td>
<td>20,133</td>
<td>$1 \times 10^{-10}$</td>
<td>$4 \times 10^{-5}$</td>
<td>$5 \times 10^{-5}$</td>
<td></td>
</tr>
<tr>
<td>Brain</td>
<td>50,737</td>
<td>$9 \times 10^{-14}$</td>
<td>$7 \times 10^{-5}$</td>
<td>$1 \times 10^{-5}$</td>
<td></td>
</tr>
<tr>
<td>Eye</td>
<td>16,176</td>
<td>$4 \times 10^{-14}$</td>
<td>$7 \times 10^{-4}$</td>
<td>$1 \times 10^{-5}$</td>
<td></td>
</tr>
</tbody>
</table>

only provided faster update times than both the augmented matrix and CG methods, but also with higher accuracy. Table 8.2 compares the relative error of the computed solutions of the tested methods. It can be observed that the solutions computed by AMPS are much more accurate than the others, whereas the SPAI preconditioned column replacing augmented method produced similar solution accuracy as CG.

Figure 8.22 shows the eigenspectrum of the eye mesh of 4,444 nodes. The smallest and the largest eigenvalues are clearly seen in the logarithmic and normal scales respectively.

Table 8.3.: Comparison of single-core and multi-core average timings

<table>
<thead>
<tr>
<th>Mesh</th>
<th>Size</th>
<th>Single-core</th>
<th>Multi-core</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beam</td>
<td>25,600</td>
<td>$1.85 \times 10^{-2}$</td>
<td>$1.75 \times 10^{-2}$</td>
<td>1.05</td>
</tr>
<tr>
<td>Brick</td>
<td>18,081</td>
<td>$1.61 \times 10^{-1}$</td>
<td>$5.53 \times 10^{-2}$</td>
<td>2.92</td>
</tr>
<tr>
<td>Bunny</td>
<td>20,133</td>
<td>$4.12 \times 10^{-2}$</td>
<td>$1.17 \times 10^{-2}$</td>
<td>3.51</td>
</tr>
<tr>
<td>Brain</td>
<td>50,737</td>
<td>$1.97 \times 10^{-1}$</td>
<td>$8.94 \times 10^{-2}$</td>
<td>2.2</td>
</tr>
<tr>
<td>Eye</td>
<td>16,176</td>
<td>$3.77 \times 10^{-2}$</td>
<td>$2.54 \times 10^{-2}$</td>
<td>1.48</td>
</tr>
</tbody>
</table>

Table 8.3 shows the average timings of the cutting experiments using single and multiple cores on Machine 2 and their relative speedups. It can be observed that although the computation times were reduced using multiple cores compared to those obtained using single core, the relative speedups were much lower than the number of cores (32). There are two reasons for the mediocre speedups. First, the sizes of the updates were not large enough for significant speed improvement using parallelism.
Figure 8.22: Eigenspectrum of the eye mesh of 4,444 nodes.
Second, one of the time dominating step, the triangular solve for the solution vector, as shown in Figure 8.17(b), was running in serial.

8.5 $N - x$ contingency analyses

8.5.1 Power systems

The AMPS algorithms was evaluated through a series of $N - k$ contingency analyses of two real-world power systems, the 3,120-bus Polish system from the MATPOWER repository [76] and the 14,090-bus WECC system; and a 777,646-bus generated system, which is based on the case2736sp system from the MATPOWER repository and the IEEE 123 bus distribution feeder [77]. The distribution feeder is balanced by equivalencing the load on each phase and extending the unbalanced laterals. Several distribution feeders are added at appropriate locations in the transmission case to create this system.

The estimated condition numbers of the admittance matrices $B$ calculated by using MATLAB’s condest function are $1.2 \times 10^6$ for the 3,120-bus Polish system, $2.1 \times 10^7$ for the 14,090-bus WECC system, and $9.9 \times 10^8$ for the 777,646-bus generated system. The estimated eigenvalues with the largest magnitude calculated by using MATLAB’s eigs function are $5.0 \times 10^{-2}$ for the Polish system, $2.3 \times 10^{-3}$ for the WECC system, and $1.4 \times 10^{-4}$ for the generated system.

8.5.2 Experiments

In our $N - k$ contingency analysis experiments we remove $k$ out of $N$ connections in the power grid and form the modified system Equation (1.2). This corresponds to a principal submatrix update as described in Equation (5.2), where $H$ is formed by the columns of the identity matrix corresponding to the end-points of the removed connections, and $m \leq 2k$. 
We compare the solution of the augmented system using an iterative solver on Equation (4.5) and using Equation (4.7) by means of the LU factorization of the Schur complement matrix $S_2$. Note that since $B$ in Equation (7.14) is a weighted Laplacian, the update matrix $E$ at the $(1,1)$-block of Equation (4.5) is singular, and thus the whole matrix is symmetric and indefinite. We have used the MINRES method and the generalized minimum residual (GMRES) method to solve these indefinite systems. For MINRES, the average solve time per iteration is faster than the GMRES method, but since it converged slowly and needed more iterations than GMRES, the total solve time was higher than the latter. Hence we report times obtained from GMRES. We also compare our augmented system with PARDISO, LUSOL and CHOLMOD being applied to Equation (1.2).

The LDL$^T$ factorization times using Oblio were 0.0306 seconds for the 3,120-bus Polish system, 0.156 seconds for the 14,070-bus WECC system, and 2.53 seconds for the 777,646-bus generated system. In comparison, the average factorization times using PARDISO were 0.00735 seconds for the 3,120-bus Polish system, 0.039 seconds for the 14,070-bus WECC system, and 2.37 seconds for the 777,646-bus generated system. Although Oblio did not perform as well as PARDISO on the larger problems, it provides the ability to extract the factors, which is essential for closure computation and the sparsity-exploiting triangular solves. In Figure 8.23, we plot the time to compute the updated solution when up to 20 edges are removed from the grid. The augmented methods outperform PARDISO, LUSOL and CHOLMOD on all three power grids. The time taken by PARDISO for the 777,646-bus generated system is not plotted in Figure 8.23(c) to better differentiate the relative performance of our methods with LUSOL and CHOLMOD. For this large grid, PARDISO took approximately 2.4 seconds for solving each modified system, which is two orders of magnitude ($149 - 186$ times) slower than our augmented iterative method. In comparison, CHOLMOD computed the solutions $1.47 - 5.09$ times slower than AMPS whereas LUSOL computed the solutions $4.9 - 27.8$ times slower.
Figure 8.23: Running times for the contingency analysis experiments
Figure 8.24.: Breakdown of the running times of AMPS for the 3,120-bus Polish system
Figure 8.25.: Breakdown of the running times of CHOLMOD and LUSOL for the 3,120-bus Polish system
Figure 8.26.: Breakdown of the running times of AMPS for the 777,646-bus generated system
Figure 8.27.: Breakdown of the running times of CHOLMOD and LUSOL for the 777,646-bus generated system
Figures 8.24 and 8.26 show the breakdown of the total time used in solving the updated systems using AMPS direct and iterative methods on the 3,120-bus Polish system and the 777,646-bus generated system. Here Preprocessing is the step of computing the closure of the modified rows and columns in the graph of $G(L)$, and extracting the necessary submatrix of $L$ for solving for $\hat{x}_3$ in Equations (4.5) and (4.7). Augmentation refers to the step of solving Equation (4.5) for the iterative method and Equation (4.7) for direct method. Matrix Formation corresponds to the step of forming $W$ as described in Section 4.5. Solution is the step of computing the solution to the modified system in Equation (4.11).

We also observe that the AMPS methods scale much better than both CHOLMOD and LUSOL as the number of edges removed (size of the updates) increases. However, the numbers of fill-ins, if any, introduced by CHOLMOD or LUSOL are insignificant, as we can see from Figures 8.25 and 8.27 that only the factor update times increases when the number of edges removed increases.

It can be seen that for a large system the time is dominated by the augmentation part in Equation (4.5) (Step 6 in Table 4.1) for the iterative method, or by the matrix formation of the reduced system in Equation (4.7) (Step 3 in Table 4.1) in the direct method. Hence the product of $m$ or $t$ (the number of steps of the iterative solver) with $O(\rho)$ is the dominant term. On the other hand, for a large system the time is dominated by the computation of $\hat{x}$ in Equation (4.11) (Step 7 in Table 4.1), which has $O(|L|)$ time complexity.

The experimental results also indicate that the augmented solution methods do not lead to difficulties with solution accuracy. Table 8.4 summarizes the average relative residual norms for the solutions computed by AMPS methods, PARDISO and CHOLMOD.
Table 8.4.: Comparison of average relative residual norms (\( \| \hat{A}\hat{x} - \hat{b} \|_2 / \| \hat{b} \|_2 \)) for the power-flow contingency analyses

<table>
<thead>
<tr>
<th>Problem</th>
<th>AMPS (direct)</th>
<th>AMPS (GMRES)</th>
<th>PARDISO</th>
<th>CHOLMOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>3,120-bus Polish system</td>
<td>(2 \times 10^{-13})</td>
<td>(3 \times 10^{-13})</td>
<td>(2 \times 10^{-13})</td>
<td>(2 \times 10^{-13})</td>
</tr>
<tr>
<td>14,070-bus WECC system</td>
<td>(4 \times 10^{-13})</td>
<td>(4 \times 10^{-13})</td>
<td>(4 \times 10^{-13})</td>
<td>(5 \times 10^{-13})</td>
</tr>
<tr>
<td>777,646-bus system</td>
<td>(6 \times 10^{-12})</td>
<td>(6 \times 10^{-12})</td>
<td>(6 \times 10^{-12})</td>
<td>(5 \times 10^{-12})</td>
</tr>
</tbody>
</table>
9 FUTURE WORK

In this chapter, we explore the further improvement to our augmented solver in several directions.

9.1 Further parallelization

As we have observed from the experimental results of the large power-flow problem as shown in ??, the time used in the augmented part of the system accounted only for a small fraction of the total running time. The majority of it was used in solving for the remaining part of the system, which involves triangular solves using the precomputed factors of the original matrix. To further reduce the total running time, we must improve the time for all the triangular solves involved in the algorithm. Since we have analyzed the sparsity pattern in the matrix during the factorization of the original matrix in the initialization stage, the elimination forest of the matrix can be used to determine the dependency between each two rows or columns. One simple approach would be to compute all the elements corresponding to the same level from the roots of the elimination forest in parallel because by definition they do not have an edge between them and hence they are independent from each other. This approach could accelerate the solution time via GPU implementations as investigated by several recent publications [36, 43, 78–80]. Alternatively, the product form of the inverse can be used in parallel to solve the triangular system [81]. For extremely large systems, distributed memory architecture implementation needs to be examined.
9.2 Application to nonlinear systems

As mentioned in earlier chapters, the problems to which this solver is applicable are limited to be linear. To further explore the applicability of this method to a wider range of problems in the future work, we need to develop a solver for dynamic nonlinear systems. Such solver would the potential to solve many of the existing nonlinear problems in the fields of science and engineering. One particular application to investigate is surgery simulation, where there is evidence that viscoelastic and hyperelastic material models are often appropriate for soft-tissue modeling [82–84]. Non-linear material models can require stiffness matrix updates at each time step, even without cutting. However, in the case of tool-tissue interaction, acceptable non-linear accuracy might possibly be achieved by only updating the stiffness of a subset of the most deformed elements or those closest to the contact area. This raises the interesting possibility of using the augmented matrix method for fast updates of non-linear materials.

Another nonlinear system that is worth investigating is the full AC power-flow contingency analysis without the “DC” approximation. A recent method called the holomorphic embedding load flow method (HELM) proposed by Trias [85] has the potential to be integrated with our AMPS algorithm for solving the AC power-flow contingency analysis in an augmented matrix formulation. HELM solves the power-flow problem by embedding Equation (7.2) into an analytic function. The unknown variables are then expressed as a power series of the embedding complex variable \( s \) and their coefficients are solved at a particular value of \( s \). Usually this value of \( s \) is chosen to be zero for simplicity. Then by making use of the continuation property of the analytic function, the coefficients of the power series are constant for all values of \( s \). Hence, we could obtain the values of the unknown variables by setting \( s \) to a desired value. The appendix provides a detailed formulation of this method and how our AMPS algorithm can be combined with this method to solve the AC power-flow contingency analysis problem.
Problems of solving in real time a linear system of equations that experiences a sequence of small changes within a principal submatrix arise in many computational science and engineering applications, and we presented two of them. In the medical field, surgical simulations provide a realistic training to surgeons for understanding the effects of the operations to the human organs. The simulator demands updating the system ten to hundred times per second to provide real-time haptic feedback, while enabling the flexibility of undergoing arbitrary cutting and deformation on the mesh. The stiffness matrix of the model may be ill-conditioned, making it more challenging to provide accurate solutions needed. In the power engineering field, the power grid operators are required by law to constantly analyze the stability of the power grid under the assumption that multiple components may go down at the same time, which may cause a cascading effect in the system and result in a catastrophic blackout. This becomes particularly challenging as an increasing number of renewable energy sources such as solar panels and wind turbines connected to the power grid makes it more vulnerable to multiple failures in a short period of time. The increase in the allowable failed components in the system skyrockets the number of contingency analysis cases exponentially even when the number of failures remains in the single digits.

The modified systems resulting from the principal submatrix updates can be solved in several ways. The first option is direct solvers. While providing accurate solution for ill-conditioned systems, the factorizations of large matrices are computationally expensive. Any changes to the matrix invalidate its factorization, and therefore refactorization is required. This makes them impractical for real-time applications even when the solvers are parallelized, and it is also undesirable for applications with many scenarios since many updates will be needed. Iterative solvers provide an alternative
to direct solvers by providing faster running times with the trade-off of less accurate solutions. While the running times are improved, they may still fall short under the high demand of interactive responses. The convergence rates may be slow for ill-conditioned problems. Some problems such as the power grid contingency analysis may even fail to converge due to the ill-conditioning. An effective preconditioner may be needed but computing and applying it may be too expensive. Solvers updating the factors of the initial matrix have been proposed. Examples are CHOLMOD and LUSOL. While they provide faster solutions to linear systems with low-rank updates, they do not scale well when the size of the updates increases.

To address the challenge of providing real-time solutions to ill-conditioned systems, we have developed AMPS, an augmented matrix solver for dynamic linear systems with small changes within a principal submatrix. All changes made to the coefficient matrix are represented by adding rows and columns in blocked form while keeping the initial matrix intact as a submatrix of the new system. This allows us keep the initial factors static and reuse the previous solutions to reduce computations. Symmetry is preserved as much as possible for symmetric matrices. Our solver is also capable of solving systems with varying dimensions as these arise from changes in the boundary conditions and topology. We further accelerate the computation by exploiting the sparsity of the matrices and vectors, as well as applying memoization and parallelization techniques. Preconditioning options were also briefly discussed in the case when iterative solver is chosen to solve the augmented part of the system. We showed that the running time of AMPS is $O(m\rho + |L|)$, where $m$ is the dimension of the principal submatrix, $\rho$ is the number of nonzeros in a subset of the columns of the Cholesky factor $L$ that are selected by the nonzeros in the sparse right-hand-side vector, and $|L|$ is the number of nonzeros in $L$. We also show that AMPS has an asymptotically lower time complexity relative to other existing methods.

Our experimental results demonstrated the feasibility of using augmented matrices to provide fast principal submatrix updates to large linear systems of equations in both applications of surgical simulation and power-flow contingency analysis. Our
AMPS algorithm vastly outperformed iterative solvers such as CG, with or without preconditioning, in a vast majority of the experiments of the surgical simulations. It also outperformed parallel direct solvers such as PARIDSO by two orders of magnitude while providing solutions with accuracy comparable to it; it also was 5 to 27 times faster than earlier methods that update the matrix factors, in the contingency analysis of a 777K node power-flow system. AMPS also maintained a better scalability than other existing solvers for handling low-rank updates while the changes accumulate. Real-time solutions can also be provided using AMPS in the surgical simulation with more than 10 updates per second for a brain model of 50K nodes, and more than 30 updates per second for an eye model of 18K nodes. Our solver has been experimentally shown to offer advantages both in speed and reliability, and has the potential to be applied on other problems in which the underlying matrix is subject to principal submatrix updates.
REFERENCES
REFERENCES


APPENDIX
APPENDIX: HOLOMORPHIC EMBEDDING LOAD FLOW METHOD

In this appendix, we present the holomorphic embedding load flow method following the canonical embedding proposed by Trias in [86]:

\[
\sum_{l} \rho^{(tr)}_{kl} v_l(s) = \begin{cases} 
 sY^{(sh)}_k v_k(s) + s \frac{p_k}{v_k^*(s^*)} - j \frac{q_k(s)}{v_k^*(s^*)} & \forall k \in \mathcal{V} \\
 -sY^{(sh)}_k v_k(s) + s \frac{s_k^*}{v_k^*(s^*)} & \forall k \in \mathcal{Q}
\end{cases}
\]  

(A.1)

with the embedding of the voltage control constraint of the PV buses to be

\[
v_k(s)v_k^*(s) = 1 + s \left( |v_k^{(sp)}|^2 - 1 \right) \quad \forall k \in \mathcal{V},
\]  

(A.2)

where \( |v_k^{(sp)}| \) is the voltage modulus setpoint at bus \( k \), with the assumption that the slack bus has a unit voltage. For this embedding, at \( s = 0 \), \( q_k(s) \) are required to be zero for the PV buses and \( v_k(s) \) all become equal to 1. Now we express the voltages \( v_k(s) \) in Equation (A.1) as a power series of \( s \),

\[
v_k(s) = \sum_{i=0}^{\infty} \bar{f}_k[i] s^i.
\]  

(A.3)

By setting \( s = 0 \) in Equation (A.3), we obtain \( v_k[0] = V_0 \) for all buses. Similarly, we express \( q_k(s) \) as a power series of \( s \),

\[
q_k(s) = \sum_{i=0}^{\infty} \bar{g}_k[i] s^i \quad \forall k \in \mathcal{V}.
\]  

(A.4)
We also express the reciprocal of \( v_k(s) \), \( w_k(s) = 1/v_k(s) \), as a separate function of \( s \) as

\[
    w_k(s) = \sum_{i=0}^{\infty} w_k[i] s^i, \quad (A.5)
\]

in which the coefficients \( w_k[i] \) can be computed by using the fact that the product of Equations (A.3) and (A.5) is equal to 1.

\[
    1 = \left( \sum_{i=0}^{\infty} v_k[i] s^i \right) \left( \sum_{i=0}^{\infty} w_k[i] s^i \right) = \sum_{i=0}^{\infty} \sum_{x=0}^{i} \left( f_k[x] w_k[i - x] s^i. \quad (A.6) \right.
\]

By comparing the coefficients of \( s^i \) on both sides of Equation (A.6), we obtain

\[
    w_k[i] = \begin{cases} 
        1 & \text{for } i = 0 \\
        \sum_{x=1}^{i} f_k[x] w_k[i - x] & \text{for } i > 0
    \end{cases}, \quad (A.7)
\]

by using the assumption that \( v_k[0] = 1 \) for all \( k \). With the power series of \( w_k(s) \), we could rewrite the PQ case of Equation (A.1) as

\[
    \sum_{l} Y_{kl}^{(tr)} \sum_{i=0}^{\infty} f_k[i] s^i = -Y_k^{(sh)} \sum_{i=0}^{\infty} v_k[i] s^{i+1} + s_k^{*} \sum_{i=0}^{\infty} w_k^{*}[i] s^{i+1} \quad \forall k \in Q. \quad (A.8)
\]

By comparing the coefficients of \( s^i \) on both sides, we obtain for all \( k \in Q \)

\[
    \sum_{l} Y_{kl}^{(tr)} v_k[i] = \begin{cases} 
        \emptyset & \text{for } i = 0 \\
        Y_k^{(sh)} v_k[i - 1] + s_k^{*} w_k^{*}[i - 1] & \text{for } i > 0
    \end{cases}, \quad (A.9)
\]

in which the right-hand side is denoted as \( \sigma_k[i - 1] \).
Now we consider the case for the PV buses. We could also rewrite such case in Equation (A.1) using Equations (A.3) to (A.5) as

$$
\sum_l Y_{kl}^{(tr)} \sum_{i=0}^{\infty} Y_{k[i]} s^i = -Y_{k}^{(sh)} \sum_{i=0}^{\infty} v_k[i] s^{i+1} + p_k \sum_{i=0}^{\infty} w_k^*[i] s^{i+1} - j \sum_{i=0}^{\infty} q_k[i] s^i \left( \sum_{i=0}^{\infty} w_k^*[i] s^i \right) \forall k \in \mathcal{V}. \quad (A.10)
$$

By consider the coefficients of $s^i$ on both sides of Equation (A.10), we obtain for all $k \in \mathcal{V}$,

$$
\sum_l Y_{kl}^{(tr)} v_k[i] = \begin{cases} 
  j q_k[0] & \text{for } i = 0 \\
  Y_k^{(sh)} v_k[i - 1] + p_k w_k^*[i - 1] - j \sum_{x=0}^{i} q_k[x] w_k^*[i - x] & \text{for } i > 0 
\end{cases}.
$$

(A.11)

For the case of $i = 0$ of Equation (A.11), since $\sum_l Y_{kl} = 0$ and $v_k[0] = 1$ for all $k$, we have $q_k[0] = 0$. For the case of $i > 0$, every coefficients on the right-hand side are computed for smaller $i$ except $q_k[i]$. We can rewrite this case of Equation (A.11) as

$$
\sum_l Y_{kl}^{(tr)} v_k[i] = \zeta_k[i - 1] - j q_k[i], \quad (A.12)
$$

where

$$
\zeta_k[i - 1] = -Y_k^{(sh)} v_k[i - 1] + p_k w_k^*[i - 1] - j \sum_{x=1}^{i-1} q_k[x] w_k^*[i - x]. \quad (A.13)
$$

Note that $q_k[i]$ only appears in the imaginary part of Equation (A.12). We can compute the real and imaginary parts separately. By expressing $Y_{kl}^{(tr)} = G_{kl} + j B_{kl}$,
where $v_k[i] = v_k^R[i] + jv_k^I[i]$ and $\zeta_k[i - 1] = \zeta_k^R[i - 1] + j\zeta_k^I[i - 1]$, we can rewrite the equation as

$$\sum_l \left[ G_{kl} - B_{kl} \right] \begin{bmatrix} v_l^R[i] \\ v_l^I[i] \end{bmatrix} = \zeta_k^R[i - 1], \quad \text{and}$$

$$q_k[i] = \zeta_k^I[i - 1] - \sum_l \left[ B_{kl} G_{kl} \right] \begin{bmatrix} v_l^R[i] \\ v_l^I[i] \end{bmatrix}$$

(A.15)

in which $q_k[i]$ can be computed once after $v_l^R[i]$ and $v_l^I[i]$ are computed, and used to obtain $\zeta_k[i]$ for larger $i$ in Equation (A.13).

Since the voltage magnitudes of the PV buses are fixed, we can compute $v_k^R[i]$ in Equation (A.14) by substituting Equation (A.3) into Equation (A.2),

$$\sum_{i=0}^\infty v_k[i]s^i = \sum_{i=0}^\infty \left( v_k^I[i]s^i \right) = 1 + s \left( |v_k^{(sp)}|^2 - 1 \right) \quad \forall k \in \mathcal{V},$$

(A.16)

which can be rewritten as

$$\sum_{i=0}^\infty \sum_{x=0}^i f_k[x]v_k^*[i - x]s^i = 1 + s \left( |v_k^{(sp)}|^2 - 1 \right) \quad \forall k \in \mathcal{V},$$

(A.17)

in which $v_k^R[i]$ for all $k \in \mathcal{V}$ can be computed in ascending order of $i$ as

$$v_k^R[i] = \begin{cases} v_k^{(sp)}|^2 - 1 & \text{for } i = 0 \\ \frac{v_k^{(sp)}|^2 - 1}{2} \sum_{x=1}^{i-1} v_k[x]v_k^*[i - x] & \text{for } i = 1 \\ -\frac{1}{2} \sum_{x=1}^{i-1} v_k[x]v_k^*[i - x] & \text{for } i > 1 \end{cases}$$

(A.18)
By separating the real and imaginary parts of Equation (A.9) and combining them with Equation (A.14), we obtain

\[
\begin{bmatrix}
G_{QQ} & -B_{QQ} & G_{QV} & -B_{QV} \\
B_{QQ} & G_{QQ} & B_{QV} & G_{QV} \\
G_{QQ}^T & -B_{QQ}^T & G_{VV} & -B_{VV} \\
\end{bmatrix}
\begin{bmatrix}
v_Q^R[i] \\
v_Q^I[i] \\
v_V^R[i] \\
v_V^I[i] \\
\end{bmatrix}
= \begin{bmatrix}
\sigma^R[i - 1] \\
\sigma^I[i - 1] \\
\end{bmatrix}, \quad (A.19)
\]

where \(G_{QQ} + jB_{QQ}, G_{VV} + jB_{VV}\) and \(G_{VQ} + jB_{VQ}\) are the transmission admittance matrices among the PQ buses, the PV buses and between the two types of buses, respectively; and \(v_Q^R, v_Q^I, v_V^R, v_V^I\) are the real and imaginary parts of the voltages of the PQ and the PV buses respectively. Since \(v_k^R[i]\) can be computed using Equation (A.18), the third column block can be rearranged to the right-hand side of Equation (A.19), which can then be made symmetric by exchanging the first and second row block,

\[
\begin{bmatrix}
B_{QQ} & G_{QQ} & G_{QV} \\
G_{QQ} & -B_{QQ} & -B_{QV} \\
G_{QQ}^T & -B_{QQ}^T & -B_{VV} \\
\end{bmatrix}
\begin{bmatrix}
v_Q^R[i] \\
v_Q^I[i] \\
v_V^R[i] \\
\end{bmatrix}
= \begin{bmatrix}
\sigma^R[i - 1] \\
\sigma^I[i - 1] \\
\end{bmatrix}
\begin{bmatrix}
G_{QQ} \\
G_{QQ} \\
G_{VV} \\
\end{bmatrix}
\begin{bmatrix}
v_Q^R[i] \\
v_Q^I[i] \\
v_V^R[i] \\
\end{bmatrix}, \quad (A.20)
\]

in which the right-hand side is computed using Equations (A.9), (A.13) and (A.18).

The assumption that the slack bus has a unit voltage setting can be lifted by embedding the slack bus by the analytic function

\[
v_0(s) = 1 + s \left(V^{(sl)} - 1\right), \quad (A.21)
\]
where $V^{(sl)}$ is the fixed voltage of the slack bus. The case for $i = 1$ of Equation (A.20) is then modified to include this new embedding as

\[
\begin{bmatrix}
B_Q & G_Q & G_QV \\
G_Q & -B_Q & -B_QV \\
G_Q^T & -B_QV & -B_VV
\end{bmatrix}
\begin{bmatrix}
\sigma_Q[1] \\
\sigma_V[1] \\
\sigma_V[0]
\end{bmatrix}
= \begin{bmatrix}
B_Q & G_Q \\
G_Q & -B_Q \\
G_V & -B_V
\end{bmatrix}
\begin{bmatrix}
\sigma_Q[0] \\
\sigma_V[0] \\
\sigma_V[0]
\end{bmatrix}
- \begin{bmatrix}
B_Q & G_Q \\
G_Q & -B_Q \\
G_V & -B_V
\end{bmatrix}
\begin{bmatrix}
\Re[v^{(sl)}] - 1 \\
\Im[v^{(sl)}]
\end{bmatrix}
\] (A.22)

where $Y_Q = G_Q + jB_Q$ and $Y_V = G_V + jB_V$ are the transmission admittance matrices between the PQ buses and the slack bus, and between the PV buses and the slack bus respectively. Note that since the matrix in Equations (A.20) and (A.22) are constant, we can use our AMPS algorithms to solve for AC power-flow contingency cases.

Once $v_k[i]$ are computed, $v_k(s)$ can be obtained at any values of $s$ where $v_k(s)$ is analytic. In particular, we are interested in the values at $s = 1$ as Equation (A.1) would become Equation (7.2). However, the convergence of this power series at $s = 1$ may be slow. We need to use the Padé approximants to compute $v_k(1)$. An efficient algorithm to compute the values of the Padé approximants would be using the Wynn’s Epsilon method, which transforms the series into a two dimensional array called the $\epsilon$-table.

Figure A.1 shows the $\epsilon$-table for the power series of $v_k(1)$. The first column $\epsilon_{(-1,r)}^k$ are defined to be zero. The second column are set to the summation

\[
\epsilon_{(0,r)}^k = \sum_{x=0}^{r} v_k[x].
\] (A.23)
Figure A.1.: The $\epsilon$-table. The boxed values are used as the approximate values of $v_k(1)$ and the arrows present the order of evaluations.

The other columns are subsequently computed by

$$
\epsilon^k_{(i,r)} = \epsilon^k_{(i-2,r)} + \frac{1}{\epsilon^k_{(i-1,r+1)} - \epsilon^k_{(i-1,r)}},
$$

(A.24)

in which the even columns of the $\epsilon$-table are equal to the values of the columns of the lower triangular part of the Padé table, i.e.

$$
\epsilon^k_{(2t,r)} = [(t + r)/t]v_k(1).
$$

(A.25)

Hence, once $v_k[i]$ is computed, all values in $\mathcal{G}_t$ can be computed in the $\epsilon$-table from left to right using only $\mathcal{G}_{t-1}$. Since only the even columns in the table are equal to the rows of the Padé approximants, the latest computed values of the even columns are then used as the approximation to $v_k(1)$, i.e.

$$
v_k(1)[i] = \epsilon^k_{(2[i/2], mod(i, 2))},
$$

(A.26)

which are the diagonals and subdiagonals of the Padé table. These values are boxed in Figure A.1 and the arrows show the order of evaluations of $v_k(1)[i]$. 
VITA
VITA

Yu Hong Yeung was born in Hong Kong in 1983. In May 2006, he received his bachelor’s degree in computer engineering with first class honors from the Hong Kong University of Science and Technology. He also completed the minor program in mathematics at the same time. He then went on to earn his master’s degree in computer science from Purdue University in May 2012 and his PhD degree from the same department in December 2017.