MULTI-SCALE PROBLEMS: AN IMPROVED NON-INTRUSIVE ALGORITHM THAT ENHANCES FEA PLATFORMS WITH THE GENERALIZED FINITE ELEMENT METHOD; AN IMPROVED PRECONDITIONED CONJUGATE GRADIENT SOLVER FOR HIERARCHICAL GENERALIZED FINITE ELEMENT SYSTEMS OF EQUATIONS

BY

TRAVIS FILLMORE

THESIS

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Civil and Environmental Engineering in the Graduate College of the University of Illinois at Urbana-Champaign, 2016

Urbana, Illinois

Adviser:

Professor C. Armando Duarte
ABSTRACT

The finite element method (FEM) discretizes an object of interest, say a cube, and solves for its displacement and stress under a certain loading. The FEM is used by many commercial softwares. The generalized FEM (GFEM) adds information in the solution process that improves the displacement and stress results, and is not fully available in commercial software, but in third-party software. A GFEM method that transfers small scale information to larger scales is called GFEM global-local (GFEM$_{gl}$). The process of adding GFEM$_{gl}$ functionality to commercial software without modifying the commercial software is called a non-intrusive algorithm.

This thesis presents a new non-intrusive algorithm, the hierarchical non-intrusive algorithm (HNA), that allows the combination of powerful FEM softwares with current and future state-of-the-art GFEM$_{gl}$ software, allowing the user to enjoy the capabilities of each software. The HNA is better than previous non-intrusive methods because it is faster, uses less memory, and is easy to use. In this thesis, the HNA is outlined and its accuracy is verified. It can be used to improve the simulation of vehicles flying at hyper-sonic speeds, greater than five times the speed of sound.

A procedure that reduces the negative effects of machine precision (condition number) in solving GFEM$_{gl}$ systems of equations is called the Stable GFEM$_{gl}$ (SGFEM$_{gl}$). This thesis presents results on the ability of SGFEM$_{gl}$ to not only reduce the condition number, but improve solution accuracy over GFEM$_{gl}$. The reduced conditioning from SGFEM$_{gl}$ systems of equations makes feasible iterative schemes that solve SGFEM$_{gl}$ linear system of equations.

The reduced memory requirements of iterative solvers over direct solvers, combined with improved speed, make larger-scale simulations possible. An iterative solver called the preconditioned conjugate gradient method is investigated within this context, and a preconditioner is proposed for the method. This thesis shows that its proposed iterative solver is faster than previous iterative solvers. The proposed iterative solver is also shown to be faster than a sparse direct solver.
To my wife and our son.
ACKNOWLEDGMENTS

I would like to thank my advisor, Prof. Duarte for his patience and guidance in the research process.

I would also like to acknowledge the support of the Air Force Research Laboratory University Collaborative Center in Structural Sciences for their financial support and collaboration. Specifically Tom Eason, Dr. Patrick O’Hara, and Dr. Joe Hollkamp have offered valuable insight.

I would like to acknowledge the help Haoyang Li, Phillipe Alves, Dr. Julia Plews, and Dr. Piyush Gupta have given me on FEM and GFEM principles.

Finally, I would like to thank my wife and family for their support and love.
# TABLE OF CONTENTS

## LIST OF TABLES

- vii

## LIST OF FIGURES

- viii

## CHAPTER 1 INTRODUCTION

- 1
  - 1.1 Background ....................................................... 1
  - 1.2 Current Approaches and Motivation ................................. 2
  - 1.3 Contributions of this Research .................................. 4

## CHAPTER 2 A HIERARCHICAL NON-INTRUSIVE ALGORITHM COMBINING STANDARD AND GENERALIZED FINITE ELEMENT PLATFORMS

- 6
  - 2.1 Problem Definition .................................................. 6
  - 2.2 GFEM Approximations ................................................ 8
  - 2.3 Implementation of Hierarchical Non-Intrusive Algorithm ......... 13
  - 2.4 Numerical Examples .................................................. 15
  - 2.5 Summary .................................................................... 37

## CHAPTER 3 THE PRECONDITIONED CONJUGATE GRADIENT SOLVER AND ITS APPLICATIONS FOR GFEM

- 41
  - 3.1 Problem Definition ..................................................... 42
  - 3.2 Numerical Examples .................................................... 43
  - 3.3 Summary .................................................................... 63

## CHAPTER 4 CONTRIBUTIONS AND PROSPECTS FOR FUTURE WORK

- 64
  - 4.1 Contributions ........................................................... 64
  - 4.2 Future Directions ....................................................... 64

## APPENDIX A XFEM/GFEM SOFTWARE ........................................... 66

## APPENDIX B ABAQUS COMMANDS TO OUTPUT $K^0$ AND $F^0$ ............ 67

## APPENDIX C AN EXAMPLE HNA INPUT FILE FOR ISET ............................ 68

## APPENDIX D COMPARING MULTIPLE ITERATIONS OF BGS-PCG .................. 71

## APPENDIX E BGS AS AN INEXACT PRECONDITIONER .............................. 74

## APPENDIX F BLOCK GAUSS-SEIDEL PRECONDITIONER EDGE-Crack CONVERGENCE .......................................................... 76
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Difference between solutions from Abaqus-ISET HNA and ISET GFEM&lt;sup&gt;gl&lt;/sup&gt;</td>
<td>27</td>
</tr>
<tr>
<td>2.2</td>
<td>Stress intensity factor $K_I$ and strain energy for edge-crack problem. Enriched global problem results for Abaqus-ISET HNA and ISET GFEM&lt;sup&gt;gl&lt;/sup&gt;</td>
<td>27</td>
</tr>
<tr>
<td>2.3</td>
<td>GFEM&lt;sup&gt;gl&lt;/sup&gt; versus SGFEM&lt;sup&gt;gl&lt;/sup&gt;; stress intensity factor $K_I$ and strain energy for edge-crack problem</td>
<td>28</td>
</tr>
<tr>
<td>2.4</td>
<td>Stress intensity factor $K_I$ for surface-crack problem. Enriched global problem results (HEX8 not Abaqus-ISET HNA)</td>
<td>33</td>
</tr>
<tr>
<td>3.1</td>
<td>BGS-PCG versus Pardiso final slope</td>
<td>61</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>2.1</td>
<td>The linear elastic fracture mechanics problem in $\mathbb{R}^2$ or $\mathbb{R}^3$ over domain $\Omega$</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td>Partition of unity function, enrichment function, and resulting GFEM shape function</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>Domain visualizations for GFEM$^g_l$ initial global, local, and enriched global steps</td>
<td>10</td>
</tr>
<tr>
<td>2.4</td>
<td>GFEM$^g_l$ iterations for 3-D edge-crack problem</td>
<td>12</td>
</tr>
<tr>
<td>2.5</td>
<td>Enriched global domain $\tilde{\Omega}^G$</td>
<td>16</td>
</tr>
<tr>
<td>2.6</td>
<td>2-D edge-crack problem description with TRI3 elements</td>
<td>17</td>
</tr>
<tr>
<td>2.7</td>
<td>TRI3 solution visualizations for ISET GFEM$^g_l$ results and Abaqus-ISET HNA results</td>
<td>17</td>
</tr>
<tr>
<td>2.8</td>
<td>2-D edge-crack problem description with QUAD4 and TRI3 elements</td>
<td>19</td>
</tr>
<tr>
<td>2.9</td>
<td>Standard-Abaqus solution visualizations for Abaqus-ISET HNA results in the QUAD4 region</td>
<td>20</td>
</tr>
<tr>
<td>2.10</td>
<td>3-D domain with edge-crack</td>
<td>21</td>
</tr>
<tr>
<td>2.11</td>
<td>3-D edge-crack problem description with HEX8 elements</td>
<td>22</td>
</tr>
<tr>
<td>2.12</td>
<td>11x11x4 reference enriched global von Mises solution for the edge-crack problem</td>
<td>23</td>
</tr>
<tr>
<td>2.13</td>
<td>3-D edge-crack problem description for TET4, TET10, and HEX20 elements</td>
<td>24</td>
</tr>
<tr>
<td>2.14</td>
<td>3-D edge-crack problem description for TET4, TET10, and HEX20 elements</td>
<td>25</td>
</tr>
<tr>
<td>2.15</td>
<td>Edge-crack HEX20 von Mises results</td>
<td>26</td>
</tr>
<tr>
<td>2.16</td>
<td>Enriched global domain $\tilde{\Omega}^G$</td>
<td>29</td>
</tr>
<tr>
<td>2.17</td>
<td>3-D surface-crack problem description for TET4, TET10, and HEX20 elements</td>
<td>30</td>
</tr>
<tr>
<td>2.18</td>
<td>3-D surface-crack problem description for TET4, TET10, and HEX20 elements</td>
<td>31</td>
</tr>
<tr>
<td>2.19</td>
<td>Surface-crack von Mises solution for each stage of solution</td>
<td>32</td>
</tr>
<tr>
<td>2.20</td>
<td>Mode $I$ stress intensity factor $K_I$ for surface-crack problem</td>
<td>34</td>
</tr>
<tr>
<td>2.21</td>
<td>Mode $I$ stress intensity factor $K_I$ for surface-crack problem finer scale</td>
<td>35</td>
</tr>
<tr>
<td>2.22</td>
<td>Geometry of the hat-stiffened panel</td>
<td>36</td>
</tr>
<tr>
<td>2.23</td>
<td>Boundary conditions of the hat-stiffened panel</td>
<td>36</td>
</tr>
<tr>
<td>2.24</td>
<td>Geometry, discretization, and boundary conditions of the hat-stiffened panel</td>
<td>38</td>
</tr>
<tr>
<td>2.25</td>
<td>Hat-stiffened panel response to boundary conditions</td>
<td>39</td>
</tr>
<tr>
<td>3.1</td>
<td>Layout of 64 $\times$ 64 element mesh and branch function enrichments</td>
<td>43</td>
</tr>
<tr>
<td>3.2</td>
<td>Layout of Five Enrichments Corresponding to the Same Crack Size in 32x32 Mesh</td>
<td>47</td>
</tr>
<tr>
<td>3.3</td>
<td>Comparison of BGS-PCG, BJ-PCG, and BGS for increasing enrichment zone size</td>
<td>48</td>
</tr>
<tr>
<td>3.4</td>
<td>Comparison of BGS-PCG, BJ-PCG, and BGS for decreasing element size $h$</td>
<td>49</td>
</tr>
<tr>
<td>3.5</td>
<td>Iterations to convergence of BGS-PCG for decreasing element size $h$</td>
<td>50</td>
</tr>
<tr>
<td>3.6</td>
<td>Meshes for several refinement schemes</td>
<td>51</td>
</tr>
<tr>
<td>3.7</td>
<td>Condition numbers when nodal values are specified using point Dirichlet boundary conditions</td>
<td>52</td>
</tr>
</tbody>
</table>
3.8 Iterations to $\epsilon = 10^{-5}$ when nodal values are specified using point Dirichlet boundary conditions ........................................... 54
3.9 Iterations to $\epsilon = 10^{-5}$ when nodal values are specified using point Dirichlet boundary conditions ........................................... 55
3.10 3-D edge-crack problem description for surface Dirichlet bc problem ................................................................. 56
3.11 Condition numbers when using face Dirichlet boundary conditions ........................................................................ 57
3.12 Meshes for several refinement schemes .................................................................................................................... 59
3.13 Comparison of BGS-PCG with Pardiso for several refinement schemes ................................................................. 60
3.14 Comparison of BGS-PCG with Pardiso for several refinement schemes ................................................................. 61
3.15 Time of BGS-PCG dominant solver components .................................................................................................... 62

D.1 Number of Block GS iterations vs the Condition Number of $M^{-1}K$ for 32x32 mesh with 12% enrichment zone ............................................. 71
D.2 Number of Block GS iterations vs the Equivalent Convergence of $M^{-1}K$ for 32x32 mesh ............................ 72

F.1 Error Convergence of Increasing Enrichments on 32 × 32 Mesh using BGS-PCG method ........................................................................................................ 76
F.2 Error Convergence of 32 × 32 × 2, 48 × 48 × 2, and 64 × 64 × 2 .............................................................................. 77
CHAPTER 1

INTRODUCTION

1.1 Background

This thesis contains two components, a non-intrusive algorithm (HNA) and an iterative solver. While both improve on the capabilities on the generalized finite element method (GFEM), they do not rely on each other in any way. In each section, the non-intrusive algorithm will be discussed first, and then the iterative solver. Now, several important terms will be introduced.

The finite element method discretizes an object of interest in order to solve for its deformation under a certain loading. This is done well by commercial finite element method (FEM) programs such as Abaqus [9] or NASTRAN [34]; however, they cannot implement every new method that enhances FEM in some way. For instance, the generalized finite element method (GFEM) can enrich the solution process with features about the object that are known a-priori [32] [33] [14]; however the GFEM is not widely or fully implemented in commercial FEM softwares. The softwares that do have GFEM implemented in some way use workarounds such as phantom nodes in Abaqus. Other softwares with some implementation of XFEM/GFEM are Altair Radioss [1], code_ASTER [15], GetFEM++ [18], OOFEM [2], and MXFEM [37]. Nevertheless, the addition of any new feature to a commercial code requires extensive time and effort in modifying the old code.

Rather, non-intrusive algorithms avoid modifying a code to introduce a new feature, but use some other technique. A non-intrusive algorithm treats the commercial code as a black box: information goes in and information comes out. The GFEM with global-local enrichment functions allows the transfer of information from the local scale to the global scale, and is under active development by Duarte et al [10].

The GFEM has been shown to have serious problems related to the condition number of its stiffness matrix. For FEM, as the discretization is refined and the size of an element $h$ decreases, the condition number grows $O(h^{-2})$. It has been shown that for various situations the condition number of GFEM grows $O(h^{-4})$. It has been shown that SGFEM reduces the rate of growth of the condition number and improves the accuracy of the solution (separate effect) [3].
A direct solver solves a system of equations exactly while an iterative solver provides a sequence of approximate solutions that trend toward the exact solution. The speed at which iterative solvers converge to the exact solution usually depends on the condition number. Several papers demonstrate the effectiveness of iterative solvers for SGFEM systems of equations [22, 25, 26].

1.2 Current Approaches and Motivation

**Non-intrusive algorithms** The HNA benefits from the strengths of both the commercial code and the GFEM code playing as a team. From commercial codes it gains many useful features, such as Abaqus's reduced integration hourglass controlled elements that substantially reduce the computational resources needed for calculating element stiffness matrices [8]. The HNA also gives users easy access to features of GFEM, such as meshless fracture modeling. Thus, with HNA not only will investigators avoid “remaking the wheel” by replicating commercial codes, but they will gain additional functionality from GFEM codes.

Alternatively, HNA can be used to couple a research code with a GFEM code. This enables easier collaboration between two research teams. For example, one team may have developed certain non-GFEM capabilities that will combine favorably with a second team’s GFEM capabilities. Providing an elegant tool for investigators is at the heart of this research.

Global-local approaches are often used to facilitate non-intrusive algorithms. Global-local approaches add finer-scale solutions to a coarse global problem. They accomplish this by using the solution from the global problem as boundary conditions for a local problem [35]. A non-intrusive global-local method could use a commercial code to provide the global mesh and need only the solution of the global mesh to compute a finer scale solution, resulting in a non-intrusive implementation. However, poor boundary conditions in the local problem lessens the quality of the solution. Among other global-local methods that address this [17, 46], GFEM\textsuperscript{gl} creates a feedback-looping mechanism by enriching the global problem with the local problem solution, and then using the new solution for new boundary conditions in the local problem [27]. The requirement of these methods to iterative to improve the solution may become computationally burdensome for many iterations, especially if two codes must communicate with each other through the iterations. Regardless, it has been shown that GFEM\textsuperscript{gl} often requires few iterations to converge [27].

Non-intrusive implementations have typically relied on the commercial code outputting the solution quantities. For a previous GFEM non-intrusive algorithm, the authors employed a static condensation-like solution process [20, 21, 40, 41]. The static condensation method is a direct solver that is used in such a way that it becomes a non-intrusive method as well. It and HNA rely on the
The hierarchical arrangement of the system of equations shown in equation (1.1).

\[
Kd = \begin{pmatrix}
K^0 & K^{0,gl} \\
K^{gl,0} & K^{gl}
\end{pmatrix}
\begin{pmatrix}
d^0 \\
d^{gl}
\end{pmatrix}
= \begin{pmatrix}
f^0 \\
f^{gl}
\end{pmatrix} = f. \tag{1.1}
\]

The static condensation non-intrusive algorithm [20, 21, 40, 41] exhibits several efficient computational characteristics. First, the entire matrix \( K \) is not factored, but only the submatrices \( K^0 \) and \( K^{gl} \). If in a transient problem, the enrichment changes to reflect changing phenomena in the problem, the commercial software part of the matrix \( K^0 \) need not be refactored. These features lead to significant computational cost savings, and are also characteristics of HNA.

However, the direct solver aspect of the static condensation non-intrusive algorithm has the inherent drawback of needing to store a dense matrix, which has as many rows as the FEM stiffness matrix and as many columns as the enrichments. This represents a high cost in memory. Further, when this method is used with Abaqus, Abaqus will refactor the FEM stiffness matrix each time a new column of the pseudo-solution is calculated, negating any theoretical advantage [21, 41].

**Iterative Solvers** A common goal for a computational method is to represent problems that are as large as possible as accurately as possible. Iterative methods allow the solution of larger problems by reducing the memory requirements. GFEM has traditionally been shown to be able to reduce the number of degrees of freedom necessary to represent problems with sharp gradients or singularities in systems of equations (like equation (1.1)) (increasing the size of the problems). However, GFEM has also been shown to produce poorly conditioned matrices \( K \) (reducing the accuracy of the problem). The poor conditioning reduces the effectiveness of iterative solvers.

With SGFEM’s introduction and its application to 3-D fracture problems [23], 3-D GFEM\textsuperscript{gl} fracture problems [22], 2-D GFEM material interface problems [25], and 2-D GFEM\textsuperscript{gl} fracture problems [31], the road to iterative solvers for several types of GFEM problems has been unblocked.

We look at solving the SGFEM\textsuperscript{gl} system of equations in equation (1.1) using a re-analysis type method. The goal of re-analysis techniques in general is to be able to change the stiffness matrix or force vector and use a priori knowledge of the original solution to quickly generate a solution to the new problem [28]. Due to knowledge about the solution gained at each step of the SGFEM\textsuperscript{gl} process, any useable method should be able to utilize the Cholesky factorization of the \( K^0 \).

Iterative solvers will be examined here that improve not only the memory cost, but the computational cost. The multigrid method, while most acclaimed for computational efficiency, is not well developed for the discontinuous coefficients resulting from GFEM problems, so the multigrid method is not used [25].
The iterative solvers in [22,25,26] use the powerful concept of factorizing (or solving sub-systems of equations that involve) only $K^0$ or $K^{gl}$. A Block Gauss-Seidel method is proposed in [22,24,25] for different applications. The PCG method with a block factorized “Block Jacobi” preconditioner is proposed in [26].

1.3 Contributions of this Research

The HNA utilizes the ability of FEM codes (commercial or not) to output FEM-stiffness-matrix and load vector files, meaning that HNA stores $K$ in its traditional sparse form, and can use other system-of-equation solvers. The HNA is described in Algorithm 1. The sparse storage requires much less memory than the dense storage in the static condensation method. It also utilizes GFEM$^{gl}$, which requires few iterations than other global-local methods due to poor local problem boundary conditions. Finally, in a transient problem $K^0$ and its factorization remains in memory, allowing the adoption of more powerful methods in finding the solution of equation (1.1).

The HNA also has application to the iterative global-local method [17, 46]. This method calls cycles iteratively between a commercial code and a research code for the local domain. This can be computationaily burdensome because the commercial code would need to do factorization on $K$ each time the commercial code is used. Simply reading the commercial code $K$ and $f$ into the research code means factorization only needs to be done once, or even an iterative solver can be used.

This research seeks to overcome the storage disadvantage of the previous non-intrusive method while maintaining high accuracy in results. Rather than utilize the output solution from FEM software, this research utilizes the output FEM-stiffness-matrix and load vector files. Any solution method can be used, meaning static condensation and its dense memory storage can be avoided.

This thesis shows that when using standard elements over the entire domain, HNA maintains high fidelity to traditional GFEM$^{gl}$ results; thus, we may consider the HNA results to be equivalent to the GFEM$^{gl}$ results. It maintains the same computational advantages for representation of fracture problems as GFEM$^{gl}$. The further step is taken to demonstrate that elements unique to an FEM software can be used in the domain of an accurately solved GFEM problem. The FEM-code elements improve upon the results of GFEM$^{gl}$. Thus, HNA can likely improve upon the computational characteristics of GFEM$^{gl}$.

The ability of SGFEM$^{gl}$ to improve the accuracy of the solution and reduce the condition number of the stiffness matrix from GFEM$^{gl}$ is investigated. It is found that SGFEM$^{gl}$ significantly improves the accuracy of the solution for linear elements. It is also shown that SGFEM$^{gl}$ $K$’s condition
number increases at the same rate as $K^0$'s.

This thesis also proposes a new preconditioner for the preconditioned conjugate gradient method for solving hierarchical systems of equations like equation (1.1). This preconditioner will be shown to be faster than other iterative solvers previously proposed for solving hierarchical systems of equations. It will also be shown to be faster than a direct solver.

**Overview of Thesis** The following two chapters present the HNA and the effects of SGFEM$^gl$ as applied toward the BGS-PCG iterative solver, respectively. Chapter 2 begins by defining the linear elastic fracture problem that will be used in both Chapter 2 and Chapter 3. Chapter 2 then briefly describes the theory behind GFEM and GFEM$^gl$. Section 2.3 describes the HNA in detail. Then Section 2.4 reports the results for a number of numerical examples. It starts with 2-D TRI3 and QUAD4 elements and then moves on to 3-D TET4, TET10, HEX8, and HEX20 elements.

Chapter 3 begins by showing the effects of SGFEM$^gl$ on the accuracy of solution and condition number. It then compares several iterative solvers with BGS-PCG.
CHAPTER 2

A HIERARCHICAL NON-INTRUSIVE ALGORITHM COMBINING STANDARD AND GENERALIZED FINITE ELEMENT PLATFORMS

This chapter begins by defining the fracture problem. It then briefly describes the GFEM formulation, and the process through which GFEM global-local problems are solved. It provides a detailed description of the Hierarchical Non-intrusive Algorithm (HNA), before describing several examples of the uses of HNA. These include 2-D and 3-D fracture problems. It concludes with a 3-D hat-stiffened panel problem with spot welds.

2.1 Problem Definition

The problem of interest in this chapter is the simulation of linear elastic problems in two and three dimensions. Linear elastic fracture mechanics is the vehicle through which this is done.

2.1.1 Linear Elastic Fracture Mechanics

Consider a cracked domain, \( \bar{\Omega} = \Omega \cup \partial \Omega \cup S \) in \( \mathbb{R}^2 \) or \( \mathbb{R}^3 \), as shown in Figure 2.1. The boundary of the domain \( \partial \Omega \) is constrained by \( \partial \Omega = \partial \Omega^u \cup \partial \Omega^\sigma \) with \( \partial \Omega^u \cap \partial \Omega^\sigma = \emptyset \).

The domain \( \Omega \) and boundaries \( \partial \Omega^\sigma \) and \( \partial \Omega^u \) are described respectively by

\[
\text{Equilibrium: } \nabla \cdot \sigma(x) = 0 \forall x \in \Omega, \tag{2.1}
\]

\[
\text{Neumann boundary condition: } \sigma(x) \cdot n = \bar{t}(x) \forall x \in \partial \Omega^\sigma, \tag{2.2}
\]

and

\[
\text{Dirichlet boundary condition: } u(x) = \bar{u}(x) \forall x \in \partial \Omega^u; \tag{2.3}
\]
where $\sigma$ is the Cauchy stress tensor, $n$ is the outward unit normal to $\partial\Omega^\sigma$, $u$ is the unknown displacement that we want to find, and $\mathbf{t}$ and $\mathbf{u}$ are prescribed tractions and displacements respectively. The constitutive and kinematic relations are respectively

$$\sigma(x) = C : \varepsilon(x) \ \forall x \in \bar{\Omega} \quad (2.4)$$

and

$$\varepsilon(x) = \nabla_s u(x) \ \forall x \in \bar{\Omega}; \quad (2.5)$$

where $C$ is Hooke’s tensor, $\varepsilon$ is the linear strain tensor, and $\nabla_s$ is the symmetric part of the gradient operator.

An approximation to the solution $u$ to the crack problem defined by equations (2.1)–(2.5) is sought. Taking the equivalent weak form, applying the Galerkin method with FEM or GFEM, and converting it to matrix form produces

$$Kd = f, \quad (2.6)$$

where $d$ is a vector of coefficients of the shape functions.
The next section reviews the GFEM formulation. A more basic introduction is in [5] and a more in-depth description is in [11]. Each node is taken to have a cloud over which its associated shape functions will be defined. As with the Finite Element Method (FEM), the partition of unity is used for a basic linear approximation of the solution, with the shape functions being denoted \( \phi_\alpha \). However, the GFEM adds additional shape functions defined over the same cloud. This results in a larger approximation space: \( S_{FEM} + S_{ENR} = S_{GFEM} \), where

\[
S_{FEM} = \sum_{\alpha \in I_h} c_\alpha \varphi_\alpha, \; c_\alpha \in \mathbb{R}
\]

and

\[
S_{ENR} = \sum_{\alpha \in I_h \cap I_e} \varphi_\alpha \chi_\alpha; \; \chi_\alpha = \text{span}\{L_{\alpha i}\}_{i=1}^{m_\alpha},
\]

where the enrichment function \( L_{\alpha i} \) is related to the patch space \( \chi_\alpha \) by \( L_{\alpha i} \in \chi_\alpha(\omega) \). Further, the GFEM shape function is \( \phi_{\alpha i}(x) = \varphi_\alpha(x)L_{\alpha i}(x) \). This is illustrated in Figure 2.2.

![Figure 2.2: Partition of unity function, enrichment function, and resulting GFEM shape function](image)
While the nature of the analytical solution is known, suitable functions that resemble the solution can be used to enrich the partition of unity solution. Enrich means that the new function is multiplied by the partition of unity. This creates the more accurate GFEM solution. GFEM is known to be particularly well suited to problems with singularities such as fracture and high gradient loadings.

2.2.1 Global-Local GFEM

This thesis uses the GFEM with global-local enrichments (GFEM$^{gl}$)\cite{27}. First an initial global problem is solved, then a local problem, and finally an enriched global problem. These will be explained in more detail in the following paragraphs. See \cite{13} for a more in depth introduction.

**Initial global problem** In the GFEM$^{gl}$, an initial global problem is first solved with no local features. The domain, $\bar{\Omega} = \Omega \cup \partial \Omega \cup S \subset \mathbb{R}^3$, is illustrated in Figure 2.3a. The boundary is defined as $\partial \Omega = \partial \Omega^u \cup \partial \Omega^\sigma$, with $\partial \Omega^u \cap \partial \Omega^\sigma = \emptyset$. The strong form of the equilibrium equation is given in (equation (2.1)). Let $u^{IG}$ denote an FEM approximation of the solution $u$ to (equation (2.1)). The weak formulation of (equation (2.1)) is given by the principle of virtual work. We find $u^{IG} \in S^{IG}(\Omega)$ such that $\forall w^{IG} \in V^{IG}(\Omega),$

\[
B(u^{IG}, w^{IG}) = F(w^{IG}) (2.7)
\]

where

\[
B(u^{IG}, w^{IG}) = \int_{\Omega} \sigma(u^{IG}) : \varepsilon(w^{IG}) d\Omega,
\]

\[
F(w^{IG}) = \int_{\partial \Omega^\sigma} \bar{t} \cdot w^{IG} d\partial \Omega^\sigma,
\]

$S^{IG} = \{u^{IG}| u^{IG} \in H_1(\Omega) \text{ and } u^{IG} = \bar{u}^{IG} \in \partial \Omega^u \}$,

and

$V^{IG} = \{w^{IG}| w^{IG} \in H_1(\Omega) \text{ and } w^{IG} = 0 \in \partial \Omega^u \}$.

The approximation space for the initial global problem does not include step-function enrichments to represent the crack or branch functions to represent the singularities resulting from the crack. Thus the crack is not actually simulated in the initial-global step. Figure 2.4 shows the initial global solution that does not represent the crack in an example problem.
(a) Initial global domain $\bar{\Omega}$. The crack is not simulated by the mesh or enrichments

(b) local domain $\overline{\Omega}_{loc}$

(c) Enriched subdomain $\Omega^{ENR}$ and nonenriched subdomains $\Omega^{\backslash ENR}$ within the enriched global domain $\bar{\Omega}$

(d) Enriched global domain $\bar{\Omega}$

Figure 2.3: Domain visualizations for GFEM$^d$ initial global, local, and enriched global steps
**Local problem**  The next step in GFEM\(^{gl}\) brings in the local scale with local features. The sub-domain of \(\Omega\), \(\Omega^{loc} = \Omega^{loc} \cup \partial\Omega^{loc} \cup S \subset \mathbb{R}^3\), is illustrated in Figure 2.3b. The boundary is partly composed of the boundary of \(\Omega^{IG}\), and partly composed of a new boundary (with boundary conditions) defined from \(\partial\Omega\). The new boundary is defined by \(\partial\Omega^{loc} \setminus (\partial\Omega^{\sigma} \cup \partial\Omega^{u})\). These new boundary conditions defined from the solution to the initial global problem may be Dirichlet, Neuman, or Spring.

Let \(u^{loc}\) denote a GFEM approximation of the solution \(\hat{u}^{loc}\) to the local version of equation (2.1). The weak formulation of the local version of equation (2.1) is given by the principle of virtual work. We find \(u^{loc} \in S^{loc}(\Omega^{loc})\) such that \(\forall w^{loc} \in V^{loc}(\Omega^{loc}),\)

\[
B(u^{loc}, w^{loc}) = F(w^{loc}). \tag{2.8}
\]

The approximation space here does include step-function enrichments to represent the crack as well as branch functions and a finely discretized mesh around the crack front.

**Enriched global problem**  The final step in GFEM\(^{gl}\) brings the local features to the global scale. The enriched global domain denoted by \(\hat{\Omega} = \Omega \cup \partial\Omega \cup S \subset \mathbb{R}^3\) is shown in Figure 2.3d. Local features are brought to the global scale by enriching all nodes within \(\Omega^{ENR} \subset \Omega^{loc}\), as shown in Figure 2.3c. They are enriched with the computational solution to the local problem. The boundary conditions are the same as the initial global problem.

Let \(u^{EG}\) denote a GFEM approximation of the solution \(\hat{u}^{EG}\) to the enriched global version of equation (2.1). The weak formulation of the enriched global version of equation (2.1) is given by the principle of virtual work. We find \(u^{EG} \in S^{EG}(\Omega^{EG})\) such that \(\forall w \in V^{EG}(\Omega^{EG}),\)

\[
B(u^{EG}, w^{EG}) = F(w^{EG}). \tag{2.9}
\]

The mesh discretization of \(\Omega\) in the enriched global step is identical to the initial global step. However, the approximation space here includes global-local enrichments from the local solution. Because the \(u^{loc}\) represents the crack and the crack front singularities, its corresponding global-local enrichments do as well.

A more concrete example of the steps to GFEM\(^{gl}\) is shown in Figure 2.4. It contains the domain, mesh, and boundary condition information on the left side and solution information on the right side. The domain is a panel the contains an edge-crack on the left edge, as shown with a bolded line. The red arrows indicate Neumann boundary conditions, while the blue arrows indicate point
Dirichlet boundary conditions that prevent rigid body motion. The red dots indicate nodes that are enriched with the local solution $u^{loc}$, while blue dots indicate the nodes that are not enriched.

The initial global problem shows the mesh representing $\Omega^{IG}$ and the Neumann (red arrows) and Dirichlet boundary conditions (blue arrows) for the problem. A maroon box outlines where the local problem is located, $\Omega^{loc}$.

The local problem introduces the crack (dark blue line) and deforms according to the Dirichlet boundary conditions (blue arrows) obtained from the initial global problem. The local problem mesh is refined so that the length of the nearest element is 5% the length of the crack, uses a polynomial order of 3, and uses branch enrichments at the crack front to simulate singularities.

The enrichments produced from the local problem solution are finely tuned to represent the crack. The same mesh used in the initial global problem is used in the enriched global problem.
The enriched global mesh shows the local-problem-solution enriched nodes as red dots and models the opening crack. All of the elements adjacent to the enriched nodes comprise $\Omega^{ENR}$. The rest comprise $\Omega^\setminus{ENR}$.

The arrows pointing from the local problem to the enriched global problem and back indicate that multiple iterations of GFEM$^{gl}$ can be performed, each time using the boundary conditions from the enriched global problem and changing the enrichment shape functions. While multiple iterations would greatly improve the boundary conditions in the local problem, in practice it has been found that GFEM$^{gl}$ converges after only a few iterations.

2.2.2 The Hierarchical Property of GFEM$^{gl}$

Because the GFEM is hierarchical, the stiffness matrix $K^{IG}$ corresponding to the unenriched mesh is used unaltered in $K^{EG}$, called $K$ in the rest of this thesis. Since $K^{IG}$ is unchanged, it is called $K^0$ in equation (1.1) and the part of the stiffness matrix corresponding to the GFEM$^{gl}$ enrichments is termed $K^{gl}$. Finally $K^{0,gl}$ and $K^{gl,0}$ represent the coupling between $K^0$ and $K^{gl}$.

The load vector is assembled in a similar way. Dirichlet boundary conditions are imposed in the GFEM code once $K$ and $f$ are assembled. A suitable solver for GFEM is then used to obtain $d$. The compartmentalization of $K$ and $f$ into components from an FEM software and a GFEM software, means that the enrichment of an FEM problem, turning it into a GFEM problem, can be done non-intrusively with minimal effort.

Since the GFEM$^{gl}$ is used in this thesis it informs our notation in equation (1.1). However, for HNA the nodes in an FEM mesh can be enriched with any GFEM enrichment, derived from a local problem or not. All that it relies on is the hierarchical nature of the matrix as shown in (equation (1.1)).

2.3 Implementation of Hierarchical Non-Intrusive Algorithm

GFEM$^{gl}$ is used primarily in this thesis because it will be solving fracture problems. High refinement is needed near the crack front to capture 3-D effects, even though GFEM enrichment functions greatly help with this. If analytic GFEM functions were used in lieu of GFEM$^{gl}$, then the FEM mesh would need to be highly refined near the crack front. This would limit the flexibility of HNA for this fracture application. HNA applies to GFEM, and possibly to other methods with hierarchical stiffness matrices. Another reason to use GFEM$^{gl}$ is it allows the GFEM modelling of
cracks on non-tetrahedral elements [36].

The HNA has certain practical requirements for the mesh discretizing Ω. Ω_{ENR} is used in the context of the enriched global problem. Ω_{ENR}, or the part of the domain over which the discretization is enriched (see Figure 2.3), must contain “standard” elements used in the GFEM code. This ensures that the GFEM^gl can enrich the K that is produced by the FEM software. The commercial FEM software used in all examples in this thesis is Abaqus. The GFEM code used in this thesis is called the Illinois Scientific and Engineering Toolbox (ISET). Standard elements for ISET in 2-D include TRI3, TRI6, QUAD4, and QUAD8. Standard elements in 3-D include TET4, TET10, HEX8, and HEX20. Outside of this Ω_{ENR}, in Ω^\text{\backslash ENR}, the mesh may contain any element types available in the FEM software.

The problem of solution post-processing must also be addressed. For the GFEM code ISET, shape functions are known over Ω_{ENR}, so displacement and stress can be calculated there. In this thesis, Ω_{ENR} = Ω_{loc}. However, since shape functions are not necessarily known for each element represented in K^0, the displacement and stress cannot always be calculated over Ω^\text{\backslash ENR}. In order to find these post-processing quantities, the user can submit a new job to the FEM software. It will be identical to the original problem definition with additional Dirichlet boundary conditions along the boundary of Ω_{ENR} that constrain the solution to be the same as the GFEM^gl problem.

The hierarchical non-intrusive algorithm is shown in Algorithm 1.

---

**Algorithm 1 Hierarchical nonintrusive algorithm**

1: procedure HNA
2: FEM code compute K^0 and f^0
3: FEM code outputs K^0 and f^0 into files (see Appendix B for Abaqus commands)
4: GFEM code reads K^0 and f^0
5: GFEM code imposes Dirichlet boundary conditions
6: if using GFEM^gl then
7: GFEM code solves initial global problem K^0 u^0 = f^0
8: GFEM code takes local problem boundary conditions from u^0
9: GFEM code solves local problem
10: GFEM code enriches FEM mesh over Ω_{ENR}, producing K^{gl}, K^{0,gl}, and f^{gl}
11: else
12: GFEM code computes step and branch enrichments analytically, producing K^{gl}, K^{0,gl}, and f^{gl}
13: end if
14: GFEM code solves the enriched global problem in equation (1.1)
15: GFEM code post-processes Ω_{ENR} region
16: To visualize Ω_{ENR}, GFEM code outputs Dirichlet boundary conditions around Ω_{ENR}
17: FEM code reads file and post-processes Ω_{\backslash ENR} region
18: end procedure

---
In many cases, the GFEM code could post-process the $\Omega^{ENR}$ region as well as the $\Omega^{\downarrow ENR}$ region, but this requires user discretion.

The bulk of this chapter will be spent verifying that HNA solutions match GFEM$^d$ solutions. Thus HNA is not an approximate way to non-intrusively couple a FEM code and a GFEM code, but an exact one.

2.4 Numerical Examples

In the first section, the non-intrusive algorithm is shown to be as accurate as a standard GFEM$^d$ implementation for many element types when using Abaqus. These include TRI3, TET4, TET10, and HEX20. Additionally, the TRI3 example problem demonstrates that the HNA adds functionality to FEM code and can use functionality unique to FEM code.

The second section verifies the accuracy of the non-intrusive algorithm using both strain energy and stress intensity factors as measures. Again, the TRI3, TET4, TET10, and HEX20 elements are used. HEX8 elements are also used to show the effect of using the Stable GFEM [24] on elements.

Finally, a challenge problem is presented. A panel with hat stiffeners spot welded onto it is loaded. The spot welds are challenging to model due to their size as compared with the overall panel and hat stiffeners size. The challenge problem will show the ability of the non-intrusive algorithm to solve large problems without encountering the memory issues of the static condensation non-intrusive algorithm. It also demonstrates the benefits of combining a FEM commercial code with a GFEM code.

For most examples in this thesis, Abaqus operates as the FEM code in the Algorithm 1 steps lines 2, 3 and 17, while ISET operates as the GFEM code in steps lines 4, 5, 7–10, 12 and 14–16. This is called Abaqus-ISET HNA-ISET is used for the GFEM$^d$, and this is called ISET GFEM$^d$.

2.4.1 2-D Edge-crack Problem

The domain for these elements is $\Omega \subset \mathbb{R}^2$ with boundary $\partial \Omega = \Gamma^u \cup \Gamma^f$, where $\Gamma^u \cap \Gamma^f = \emptyset$. Figure 2.5 shows the domain. The dimensions are $b = 4$, $h = 2$, and $a = 2$. 2-D problems will not use GFEM$^d$, but rather the traditional GFEM for linear elastic fracture mechanics. As was discussed previously, the mesh corresponding to $\Omega^G$, such as that shown in Figure 2.6b, is used in Abaqus. Abaqus produces $\mathbf{K}^0$ and $\mathbf{f}^0$ from it, and then the GFEM code uploads the matrix and vector.
At this point the GFEM code ISET uses branch function enrichments and step function enrichments directly on nodes in the initial global problem [12]. In other words no local problem is solved or its computational solution used; rather analytical functions are used.

This is taken to be a plane strain problem so that $\nu = 0.3$, $E = 1$, and $T = 1$.

**Verification of HNA for TRI3 Element Discretization**  The first non-intrusive test will be performed with TRI3 elements in $\Omega^{ENR}$ and $\Omega^{\setminus ENR}$ using both Abaqus-ISETHNA and ISET GFEM$^gl$. An edge-crack problem is introduced here that will be used throughout this thesis, in 2-D and 3-D, for all elements. The 2-D version is visualized in Figure 2.6.

Figure 2.7 shows the Abaqus-ISET HNA and ISET GFEM$^gl$ solutions. The formulation for the TRI3 element in ISET and Abaqus are identical. Small implementation differences cause differences in the Abaqus-ISET HNA $K^0$ and ISET GFEM$^gl$ $K^0$ that may cause problems in the enriched-global solution. The ISET GFEM$^gl$ solution is shown in the left column while the Abaqus-ISET HNA solution is shown in the right column.

The ISET GFEM$^gl$ and non-intrusive solutions look very similar. This shows that the Abaqus-ISET HNA does not introduce unusual behavior near the crack or elsewhere. A good quantitative measure of their similarity is the strain energy of the deformed plate. This measure will be used throughout this thesis as a quantitative measure of the accuracy of the non-intrusive algorithm.

The ISET GFEM$^gl$ strain energy is $28.2621797446966$ and the Abaqus-ISET HNA strain energy is $28.2621797446956$. There are 13 digits of accuracy, with a relative error of $3.5e - 14$. This is
(a) Initial global domain $\Omega$ and boundary conditions

(b) Initial global mesh

(c) Enriched global domain $\Omega$ and boundary conditions

(d) Enriched global mesh and nodes with branch function enrichments

Figure 2.6: 2-D edge-crack problem description with TRI3 elements

(a) ISET GFEM$^d_l$ Von Mises

(b) Abaqus-ISET HNA Von Mises

Figure 2.7: TRI3 solution visualizations for ISET GFEM$^d_l$ results and Abaqus-ISET HNA results
consistent with the high accuracy of 16 digits or double in the stiffness $K^0$ and load $f^0$ files. The similarity in strain energy leads to the conclusion that the $K^0$ can come from Abaqus for TRI3 elements.

**Selectively Reduced QUAD4, Abaqus-ISET HNA versus ISET GFEM$^{gl}$** Abaqus uses reduced integration QUAD4/HEX8 elements when generating its stiffness matrix, $K^0$ [8]. During the integration step of the assembly process, $\int B^TDBdxdy$, full gaussian quadrature is not done. In order to relate the local coordinates $\xi$ to $x$ we need the jacobian of the transformation $J$: $dxdy = Jd\xi d\eta$. This results in $\int B^TDBJd\xi d\eta$. Now, in 3-D $J$ is not necessarily constant, but in Abaqus’s selective reduced integration, they take the average of $J$. The GFEM generates additional shape functions for the problem which are manifested in $K^{gl}$ and the coupling term $K^{0,gl}$ and its transpose.

However, the GFEM code ISET uses fully integrated QUAD4 elements when calculating the coupling between partition of unity and enrichment shape functions. This means that error exists between the calculated coupling term $K^{FEM,ENR}$ and the true coupling term $\hat{K}^{0,gl}$: $\|K^{0,gl}\| > \|\hat{K}^{0,gl}\|$. This will lead to errors in the solution, but how large is the effect of these errors? This question will be explored in this section and the HEX8 section.

First, it is known that for a system of linear equations $Kd = f$, a change in matrix $\Delta K$ affects the solution according to the equation

$$\frac{\|\Delta d\|}{\|d\|} \leq \text{cond}(K) \frac{\|\Delta K\|}{\|K\|} \quad (2.10)$$

This means that as the $\|\Delta K\|$ increases, the relative error in the solution will also linearly increase at worst.

A crack problem is not considered for the QUAD4, because ISET does not have analytical enrichments implemented for QUAD4 elements. However, a mesh with TRI3 elements near the crack and QUAD4 elements away from it will now be considered.

**Standard-Abaqus Element, Abaqus-ISET HNA versus ISET GFEM$^{gl}$** The fracture problem considered here is similar to the TRI3 considered previously, as shown in Figure 2.8. The domain and material is the same, but the elements are different, as witnessed by a comparison between Figure 2.6 and Figure 2.8. In $\Omega^{ENR}$, QUAD4 elements are used. This allows accurate GFEM enrichment since the problem is enriched only at the TRI3 elements in $\Omega^{ENR}$, where ISET knows the shape functions. The differences between the Abaqus reduced-integration QUAD4 elements and ISET’s standard QUAD4 elements will result in different solutions.
The crack can be seen in the warped solutions in Figure 2.9. This figure shows the solution for Abaqus-ISET HNA and ISET GFEM$^\text{gl}$. The ISET GFEM$^\text{gl}$ solution is shown in the left column while the Abaqus-ISET HNA solution is shown in the right column.

The ISET-GFEM$^\text{gl}$ strain energy is 28.63 and the Abaqus-ISET HNA strain energy is 29.24. Bear in mind that since the elements in $\Omega^{\text{ENR}}$ are not the same between the ISET GFEM$^\text{gl}$ solution and Abaqus-ISET HNA solution, the solutions should not be the same. A reference strain energy is used that has a highly refined TRI3 mesh that has $64 \times 64(\times 2)$ elements, and has a $p = 2$ at every node. This model gives a strain energy of 36.97. This shows that the Abaqus-ISET HNA problem using the reduced integration elements has a better strain energy than the ISET GFEM$^\text{gl}$ problem (20.9% error versus 22.6% error). Also, the SIF for the reference solution is about 7.542 while for the Abaqus-ISET HNA and ISET GFEM$^\text{gl}$ solutions it is 6.573 and 6.534 respectively (12.86% error versus 13.37% error).

Comparison with the previous problem where TRI3 elements were used in $\Omega^{\text{ENR}}$ provides an interesting insight. That problem’s results were identical with this problem’s ISET GFEM$^\text{gl}$ results. It can be concluded that the difference between the results is due to the difference in the two problems, the element type used in $\Omega^{\text{ENR}}$. Thus, the improved accuracy in the current problem is due to improved accuracy from reduced-integration QUAD4 elements. This is a good example of
Abaqus-ISET HNA using good features from both a FEM code (the reduced-integration elements) and the GFEM code (fracture).

**Post-processing of Solution on \( \Omega^{ENR} \) and \( \Omega^{\backslash ENR} \)** This is a good point in the thesis to describe the process of visualizing the FEM-code-specific elements using the solution \( \mathbf{d} \) obtained using Abaqus-ISET HNA. A process is needed because the Abaqus-ISET HNA solution visualized in Figure 2.9 is only nodewise accurate in \( \Omega^{\backslash ENR} \) where the QUAD4 reduced integration elements are. In order to visualize the entire region with QUAD4 elements accurately, we must pass the displacement solution vector to an Abaqus problem as Dirichlet boundary conditions along the border of the QUAD4 elements. Figure 2.9 shows the Dirichlet boundary conditions along this border.

![Figure 2.9: Standard-Abaqus solution visualizations for Abaqus-ISET HNA results in the QUAD4 region](image)

No attempt is made to scale the visualizations to look similar because they are fundamentally different. ISET post-processing of the solution is only accurate in \( \Omega^{ENR} \) and Abaqus post-processing
is only accurate in $\Omega^{ENR}$. Abaqus cannot represent the enrichments or crack in $\Omega^{ENR}$ at all, so the region inside the boundary can be considered a garbage visualization.

In this situation, the visualization from ISET is very accurate due to its nodewise accuracy, making such a procedure seem unnecessary. However, if instead of a QUAD4-type element, we wanted to use an element with a novel (to ISET) number of nodes or coupled formulation, ISET could not visualize the solution in that region. It would rely on this procedure to visualize those results.

2.4.2 Impact of Selectively-Reduced Integration HEX8

The HEX8 element from Abaqus is integrated similarly to the QUAD4 element from Abaqus. Again, this results in $K^{0,gl}$ that is not correct. While some preliminary tests for a beam-type problem yielded decent results, the true test is to solve a fracture problem.

For the case of a full fracture problem, the domain shown in Figure 2.10 is used. The example problem is a 3-D panel with dimensions $b = 4$, $l = 2$, $t = 2$, and $a = 2$. It has $T = 1$, downwards at the bottom of the plate and upwards at the top of the plate, as shown in Figure 2.10.

The loading causes similar behavior between Abaqus’s reduced integration HEX8 elements and ISET’s fully integrated HEX8 elements. An 11x11x4 HEX8 mesh was tested. A crack is introduced onto the center of the left edge of the domain using GFEM$^{gl}$ enrichments from ISET. The initial global domain, mesh, and boundary conditions for the initial global problem, local problem, and enriched global problem are shown in Figure 2.11.
Figure 2.11: 3-D edge-crack problem description with HEX8 elements
The resulting strain energy is compared to a reference solution. The reference GFEM solution has HEX8 polynomial-order for the initial problem, $p = 4$ in the local problem, refinement at the crack front in the local problem such that $h \approx 5\%$ of the crack length, branch functions in the local problem, and enrichment in the enriched global problem only at nodes adjacent to the crack. This means that for the enriched global problem has HEX8 polynomial-order. The reference enriched global solution is shown in Figure 2.12.

![Figure 2.12](image.jpg)

Figure 2.12: 11x11x4 reference enriched global von Mises solution for the edge-crack problem

Several techniques were used to improve the non-intrusive selective-reduced-integration HEX8 element including size of local domain and SGFEM. However, the best result that could be achieved was $22.3\%$ strain energy error for non-intrusive (small local domain using SGFEM), while the standard GFEM implementation was able to achieve $14.4\%$ with the same parameters. Further, the non-intrusive algorithm did not converge as the local problem size increased, making for untrustworthy results.

We conclude that we have little confidence enriching Abaqus HEX8 selectively reduced integration elements for fracture problems. Since this is the closest element Abaqus has to a standard HEX8 element, we must turn to other HEX type elements for enriching $\Omega^{ENR}$. Using Abaqus specific elements is still valid in $\Omega \setminus \Omega^{ENR}$ because they are not enriched and there is no error in $K^{0,gl}$.

2.4.3 Verification of HNA for Edge-Crack Problem using TET4, TET10, and HEX20 elements

A previous paper [21] that uses the static-condensation based non-intrusive method runs several examples in section 7. An identical problem to its panel with a planar edge-crack is considered here. The problem domain and dimensions are described in Figure 2.10. Its dimensions are $b = 2$, $l = 4$, $t = 1$, and $a = 1$. The load $T = 1$. Its material parameters are Young’s modulus $E = 200,000$, and
Poisson’s ratio $\nu = 0.3$. Figure 2.13 shows the domains and boundary conditions for each stage in the GFEM$^g$ solution process.

(a) Initial global domain $\Omega^{IG}$ and boundary conditions

(b) Local domain $\Omega^{loc}$ and boundary conditions

(c) Enriched global domain $\Omega^{EG}$ and boundary conditions

Figure 2.13: 3-D edge-crack problem description for TET4, TET10, and HEX20 elements

Important in this example is demonstration of the ability of the non-intrusive HNA to give accurate solutions for TET4, TET10, and HEX20 elements. These several different elements will be used for the initial global mesh. The initial global, local, and enriched global meshes are shown in Figure 2.14. A problem that uses TET4 elements in its mesh will be called a TET4 problem. The same will be true for other element types.

The local step of GFEM$^g$ simulates the crack. Spring boundary conditions are applied along the edges of the local problem. Mesh refinement is high near the crack front, with the element length adjacent to the crack front being 5% of the crack length, 0.05. The polynomial order of the local problem for all cases is 3. The local solution is used to generate the enrichments in the enriched global problem. These global-local enrichments are the only enrichments in the domain. Figure 2.15 displays the deformed configuration of the panel at each of these steps as well as von Mises values.
Figure 2.14: 3-D edge-crack problem description for TET4, TET10, and HEX20 elements
It comes from a high fidelity HEX20 element mesh.

![Initial global](image1)
![Local](image2)
![Enriched global](image3)

(a) Initial global  
(b) Local  
(c) Enriched global (Von mises\textsuperscript{max} = 194)

**Figure 2.15: Edge-crack HEX20 von Mises results**

**Difference between Abaqus-ISET HNA and ISET GFEM\textsuperscript{gl}** The first step of verification is showing that the Abaqus-ISET HNA solutions match the ISET GFEM\textsuperscript{gl} solutions. The Abaqus-ISET HNA differs from the implementation of GFEM\textsuperscript{gl} in that it obtains $K^0$ and $f^0$ from Abaqus, where GFEM\textsuperscript{gl} assembles them in its code. If the standard GFEM\textsuperscript{gl} method is used, it will be called an ISET GFEM\textsuperscript{gl} problem. The edge-crack problem is solved using TET4, TET10, and HEX20 elements as described in Figure 2.14. Solutions of Abaqus-ISET HNA and ISET GFEM\textsuperscript{gl} are compared.

Accuracy for this example is gauged through the stress intensity factor $K_I$ at the center of the crack front and the strain energy $U$ of the domain. Table 2.1 shows the $U$ and $K_I$ for each case. It
also shows the relative difference in solution between the Abaqus-ISET HNA and the ISET GFEM<sub>gl</sub> solution. The relative difference is calculated as \( \frac{|U_{\text{ISET}} - U_{\text{HNA}}|}{|U_{\text{ISET}}|} \)

Table 2.1: Difference between solutions from Abaqus-ISET HNA and ISET GFEM<sub>gl</sub>

<table>
<thead>
<tr>
<th>Method</th>
<th>Element Type</th>
<th>(K_I)</th>
<th>(K_I) rel. diff.</th>
<th>(U(E - 5))</th>
<th>(U) rel. diff.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISET GFEM&lt;sub&gt;gl&lt;/sub&gt;</td>
<td>TET4</td>
<td>2.229905</td>
<td>n/a</td>
<td>5.108473</td>
<td>n/a</td>
</tr>
<tr>
<td>Abaqus-ISET HNA</td>
<td>TET4</td>
<td>2.229903</td>
<td>8.97E - 7</td>
<td>5.108471</td>
<td>3.32E - 7</td>
</tr>
<tr>
<td>ISET GFEM&lt;sub&gt;gl&lt;/sub&gt;</td>
<td>TET10</td>
<td>2.87537</td>
<td>n/a</td>
<td>5.3423454250874</td>
<td>n/a</td>
</tr>
<tr>
<td>Abaqus-ISET HNA</td>
<td>TET10</td>
<td>2.87537</td>
<td>0</td>
<td>5.3423454250871</td>
<td>4.97E - 14</td>
</tr>
<tr>
<td>ISET GFEM&lt;sub&gt;gl&lt;/sub&gt;</td>
<td>HEX20</td>
<td>3.05214</td>
<td>n/a</td>
<td>5.37984</td>
<td>n/a</td>
</tr>
<tr>
<td>Abaqus-ISET HNA</td>
<td>HEX20</td>
<td>3.05217</td>
<td>7.536E - 6</td>
<td>5.37986</td>
<td>3.604E - 6</td>
</tr>
</tbody>
</table>

The TET4 and HEX20 problems have relative error of about 10<sup>-6</sup>, and TET10 problems have a smaller relative error of 10<sup>-14</sup>. It is concluded that the Abaqus-ISET HNA TET4, TET10, and HEX20 problems produce solutions very similar to their corresponding ISET GFEM<sub>gl</sub> analogs.

Verification with Static Condensation Non-Intrusive Algorithm [21] Results  A more challenging metric to verify the accuracy of HNA TET4, TET10, and HEX20 problems is comparison of HNA solutions with the static condensation non-intrusive algorithm solutions and reference solutions. In order to obtain its reference solution, [21] ran an hp-GFEM problem. The reference strain energy and stress intensity factor are \(U = 5.55154E - 5\) and \(K_I = 3.0796\) respectively. Table 2.2 shows these values as well as their relative error values \(e^r(K_I) = \frac{|K_I^{ref} - K_I|}{|K_I^{ref}|}\) and \(e^r(U) = \frac{|U^{ref} - U|}{|U^{ref}|} \).

Table 2.2: Stress intensity factor \(K_I\) and strain energy for edge-crack problem. Enriched global problem results for Abaqus-ISET HNA and ISET GFEM<sub>gl</sub>

<table>
<thead>
<tr>
<th>Method</th>
<th>Element Type</th>
<th>(K_I)</th>
<th>(U(E - 5))</th>
<th>(e^r(K_I))(%)</th>
<th>(e^r(U))(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>TET4</td>
<td>3.0796</td>
<td>5.55154</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Static Cond.</td>
<td>TET4</td>
<td>2.244</td>
<td>5.114</td>
<td>27.143</td>
<td>7.8729</td>
</tr>
<tr>
<td>Abaqus-ISET HNA</td>
<td>TET4</td>
<td>2.230</td>
<td>5.108</td>
<td>27.59</td>
<td>7.981</td>
</tr>
<tr>
<td>Static Cond.</td>
<td>TET10</td>
<td>2.885</td>
<td>5.348</td>
<td>6.3266</td>
<td>3.6659</td>
</tr>
<tr>
<td>Abaqus-ISET HNA</td>
<td>TET10</td>
<td>2.875</td>
<td>5.342</td>
<td>6.632</td>
<td>3.77</td>
</tr>
<tr>
<td>Abaqus-ISET HNA</td>
<td>HEX20</td>
<td>3.052</td>
<td>5.380</td>
<td>0.891</td>
<td>3.093</td>
</tr>
</tbody>
</table>

The Abaqus-ISET HNA problem results are close to the static condensation problem results. Further, all results are reasonably close to the reference solution. The HEX20 problem has fewer nodes (and dofs) than the TET10 problem, but yields a more accurate solution. This makes the HEX20 a promising element to use in other examples.
Effect of SGFEM\textsuperscript{gl} SGFEM is a simple algorithm that modifies the shape functions to reduce the condition number of the stiffness matrix and may improve the solution accuracy. See [4] and [24] for more information.

This thesis evaluates the effects of SGFEM on a GFEM\textsuperscript{gl} problem, hereafter called SGFEM\textsuperscript{gl}. The effect of using GFEM\textsuperscript{gl} versus SGFEM\textsuperscript{gl} is shown in Table 2.3 for TET4, TET10, HEX8, and HEX20. Problems with HEX8 elements are run using ISET GFEM\textsuperscript{gl}. While the other elements are run using the non-intrusive algorithm, the results were checked against the standard GFEM procedure.

Table 2.3: GFEM\textsuperscript{gl} versus SGFEM\textsuperscript{gl}; stress intensity factor $K_I$ and strain energy for edge-crack problem

<table>
<thead>
<tr>
<th>Case</th>
<th>Method</th>
<th>Element Type</th>
<th>$K_I$</th>
<th>$U(E - 5)$</th>
<th>$\epsilon^e(K_I)(%)$</th>
<th>$\epsilon^e(U)(%)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td>Reference</td>
<td>TET4</td>
<td>3.0796</td>
<td>5.55154</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>GFEM\textsuperscript{gl}</td>
<td>Abaqus-ISET HNA</td>
<td>TET4</td>
<td>2.230</td>
<td>5.108</td>
<td>27.59</td>
<td>7.981</td>
</tr>
<tr>
<td>SGFEM\textsuperscript{gl}</td>
<td>Abaqus-ISET HNA</td>
<td>TET4</td>
<td>2.760</td>
<td>5.310</td>
<td>10.4</td>
<td>4.344</td>
</tr>
<tr>
<td>GFEM\textsuperscript{gl}</td>
<td>Abaqus-ISET HNA</td>
<td>TET10</td>
<td>2.875</td>
<td>5.342</td>
<td>6.632</td>
<td>3.77</td>
</tr>
<tr>
<td>SGFEM\textsuperscript{gl}</td>
<td>Abaqus-ISET HNA</td>
<td>TET10</td>
<td>0.8841</td>
<td>4.201</td>
<td>71.29</td>
<td>24.31</td>
</tr>
<tr>
<td>GFEM\textsuperscript{gl}</td>
<td>ISET GFEM\textsuperscript{gl}</td>
<td>HEX8</td>
<td>2.359</td>
<td>5.191</td>
<td>23.41</td>
<td>6.486</td>
</tr>
<tr>
<td>SGFEM\textsuperscript{gl}</td>
<td>ISET GFEM\textsuperscript{gl}</td>
<td>HEX8</td>
<td>2.891</td>
<td>5.401</td>
<td>6.093</td>
<td>2.712</td>
</tr>
<tr>
<td>SGFEM\textsuperscript{gl}</td>
<td>Abaqus-ISET HNA</td>
<td>HEX20</td>
<td>3.052</td>
<td>5.380</td>
<td>0.891</td>
<td>3.093</td>
</tr>
<tr>
<td>SGFEM\textsuperscript{gl}</td>
<td>Abaqus-ISET HNA</td>
<td>HEX20</td>
<td>1.867</td>
<td>4.778</td>
<td>39.38</td>
<td>13.94</td>
</tr>
</tbody>
</table>

Table 2.3 shows that when a linear (or close to linear) patch space is used, as with TET4 and HEX8 elements, it improves the solution. But for TET10 and HEX20 elements, it worsens the solution. More results will be shown in the next example.

2.4.4 Verification using Small Surface-crack

A second problem considered in Gupta’s Master thesis is a small surface-crack on a panel, which gives more interesting 3-D effects. Each point along the semi-circle will contain unique SIF values, providing a challenging SIF verification problem. While [21] uses the GFEM to solve the example, [42] and [45] use FEM to solve it. The small surface-crack is illustrated in Figure 2.17. Geometry dimensions include $2b = 2h = 2.0$, domain thickness $t = 1.0$, and crack radius $r = 0.2$. Material properties are modulus of elasticity $E = 1$ and Poisson’s ratio is $\nu = 0.25$. Traction magnitude is $T = 1$ and fixed point Dirichlet boundary conditions are applied to prevent rigid body motion.

Again, the HNA must be shown to be accurate for several element types including TET4, TET10, and HEX20 elements. The steps for the GFEM\textsuperscript{gl} procedure, as discussed in Section 2.2.1, are solve
the initial problem, use the solution to produce boundary conditions for the local problem, use the local solution to provide enrichments to the enriched global problem, and solve the enriched global problem. The meshes for each of these steps and the location of the enriched nodes in the enriched global nodes are shown in Figure 2.18.

Figure 2.18 shows that the initial global mesh is $10 \times 11 \times 4$ hexahedral elements. In the cases in which we consider tetrahedral elements, each hexahedral element is divided into six tetrahedrals, resulting in an initial global mesh of $6 \times (10 \times 11 \times 4)$ tetrahedral elements. The stiffness matrices associated with these meshes are created by Abaqus.

Figure 2.18 shows that the local problem is always comprised of TET4 elements that are taken from a $6 \times 5 \times 3$ section of the initial global problem, and then adaptively refined along the front of the crack. The length of the largest element edge along the crack front is 0.00390625, giving an element-to-crack-radius ratio of $0.00390625/0.2 = 0.01953$. Shape functions of order $p = 3$ are used in the local problem. All nodes that lie within the local problem domain (not including the edges) are enriched in the enriched global step. Figure 2.19 shows through coloring the von Mises stress and the deformed configuration of the solution of the initial global problem, local problem, and enriched global problem.

For stress intensity factors, each extraction point that will be considered lie along a half (from
Figure 2.17: 3-D surface-crack problem description for TET4, TET10, and HEX20 elements

(a) Initial global domain $\bar{\Omega}^{IG}$ and boundary conditions

(b) Local domain $\bar{\Omega}^{loc}$ and boundary conditions

(c) Enriched global domain $\bar{\Omega}^{EG}$ and boundary conditions
Figure 2.18: 3-D surface-crack problem description for TET4, TET10, and HEX20 elements
Figure 2.19: Surface-crack von Mises solution for each stage of solution
symmetry) of the edge of the semi-circle crack surface, giving a quarter circle to consider. The x-axis in Figures 2.20 and 2.21 are normalized from $[0, \pi/2]$ to $[0, 1]$. The cut-off function method (CFM) [39] is used to extract SIFs around the quarter-circle. The SIFs are then normalized using $\bar{K}_I = \frac{K_I}{Q \sqrt{\pi r}}$, where $Q = 2.464$ for the circular crack. Reference values used for verification are obtained from Walters et al. [45]. Figure 2.20 shows the SIFs $K_I$ around the quarter circle. Figure 2.20 has similar scale to [21] for easy comparison with those results, while Figure 2.21 has a smaller scale to facilitate comparison of more accurate results.

Several SIF values are compared using the 2-norm of the difference in vectors divided by the 2-norm of the reference solution: $e_r(K_I) = \frac{||K_I^{ref} - K_I^{GFEM}||_2}{||K_I||_2}$. These values are in Table 2.4. All results are for the Abaqus-ISET HNA except for HEX8, for which ISET GFEM$^gl$ is used.

Table 2.4 shows the $e_r(K_I)$ for each of these cases, allowing us to more easily quantify the accuracy of each element and (S)GFEM$^gl$.

<table>
<thead>
<tr>
<th>Case</th>
<th>Method</th>
<th>Element Type</th>
<th>$e_r(K_I)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>GFEM$^gl$</td>
<td>Abaqus-ISET HNA</td>
<td>TET4</td>
<td>15.5%</td>
</tr>
<tr>
<td>SGFEM$^gl$</td>
<td>Abaqus-ISET HNA</td>
<td>TET4</td>
<td>1.44%</td>
</tr>
<tr>
<td>GFEM$^gl$</td>
<td>Abaqus-ISET HNA</td>
<td>TET10</td>
<td>0.60%</td>
</tr>
<tr>
<td>SGFEM$^gl$</td>
<td>ISET GFEM$^gl$</td>
<td>HEX8</td>
<td>16.0%</td>
</tr>
<tr>
<td>SGFEM$^gl$</td>
<td>ISET GFEM$^gl$</td>
<td>HEX8</td>
<td>0.405%</td>
</tr>
<tr>
<td>SGFEM$^gl$</td>
<td>Abaqus-ISET HNA</td>
<td>HEX20</td>
<td>0.506%</td>
</tr>
<tr>
<td>SGFEM$^gl$</td>
<td>Abaqus-ISET HNA</td>
<td>HEX20</td>
<td>10.59%</td>
</tr>
</tbody>
</table>

First, the TET4 GFEM$^gl$ and TET10 GFEM$^gl$ are very similar to corresponding results in [21] (TET4: Table 4.3, Case 3= 14.9%; TET10: Case 9= 0.594%). Thus, the two non-intrusive algorithms have comparable accuracy. Interestingly, the TET4 SGFEM$^gl$ results are much more accurate than TET4 GFEM$^gl$. Thus, SGFEM$^gl$ seems to have a drastic positive effect on the accuracy of SIFs for TET4 elements in the global problem for both the edge-crack and surface-crack examples. However, for the higher polynomial order TET10 element, the opposite is true, SGFEM$^gl$ worsens the result SIF result.

Second, the HEX8 SGFEM$^gl$ has lower error than HEX8 GFEM$^gl$; the improvement seems to be even better than for the TET4 elements. The HEX20 SGFEM$^gl$ has higher error than HEX8 GFEM$^gl$; the increase in error is less for HEX20 than TET10. It is concluded that since SGFEM$^gl$ does not provide the correct solution for TET10 and HEX20 elements when GFEM$^gl$ does provide a good solution, and SGFEM$^gl$ shall no longer be used for TET10 and HEX20 elements. Figure 2.21,
Figure 2.20: Mode I stress intensity factor $K_I$ for surface-crack problem
reflecting this conclusion, shows a closer look at the more accurate methods compared with the reference solution.

![Figure 2.21: Mode I stress intensity factor $K_I$ for surface-crack problem finer scale](image)

**Figure 2.21: Mode I stress intensity factor $K_I$ for surface-crack problem finer scale**

### 2.4.5 Hat-stiffened Panel Spotweld Challenge Problem

The hat-stiffened panel has spotwelds securing the “hat” components to the panel. A picture of hat-stiffened panel that was tested in a lab is shown in Figure 2.22. HNA is used to solve this geometry under certain boundary conditions.

The spot welds can be seen as dark spots on the hat components adjacent to the panel. Spot welds are challenging to model due to their small geometry compared to the panel. GFEM$_{gl}$ is meant to bridge local scales to global ones, so it will be used on this problem. For this example problem, the symmetry of the panel is exploited and it is cut in half along two directions, as shown in Figure 2.23.

Dirichlet boundary conditions enforce the symmetry. Dirichlet boundary conditions clamp the panel along its edge. The panel is pushed upward with a distributed load of magnitude 1. These loading conditions should result in the center of the entire panel, or the top corner of our symmetric domain, having the greatest displacement upward.
Figure 2.22: Geometry of the hat-stiffened panel

Figure 2.23: Boundary conditions of the hat-stiffened panel

(a) HEX20 domain with Dirichlet boundary conditions

(b) Neuman boundary conditions
One common way to model spotwelds is to attach a node in the hat-stiffener mesh to a node in panel mesh at the location of the weld. This creates the initial global mesh, but does not capture local effects well. Rather, GFEMgl accurately represents the geometry of the spotweld in a local mesh, as shown in Figure 2.24.

The entire domain is discretized using HEX20 elements. Abaqus is used to compute $K^0$ and $f^0$ in the Abaqus-ISET HNA. The Figure 2.24 shows this discretization for the enriched global problem as well as the GFEMgl enriched nodes. Once a solution is found for all 44 local domains, the local spot weld behavior is up-scaled to the global domain through GFEMgl enriched nodes.

Results for the enriched global problem are shown in Figure 2.25. The top subfigure shows a few of the von Mises stress on the displaced domain, but it does not show the stress at the spot welds. Thus, a second subfigure shows a slice of the strain that is at the coordinates for the interaction between the hat-stiffener and the panel.

The von Mises stress is fairly constant over the domain, except near the spot welds located at the end of the hat-tiffeners. There the stress reaches its maximum values. Semi-arcs representing the effects of the stress at the edge of the spot weld can be seen vaguely in the top subfigure. The bottom subfigure shows the semi-arc stress at the edges of the spotwelds more clearly. The anomalous high stresses around the spot welds show that the current discretization has not converged to an acceptable level, but higher refinement and polynomial order can be imposed. Nevertheless, the GFEMgl effectively provides local information to the global domain.

The strain energy of the initial global problem with ISET GFEMgl is 0.152887974655 while with the Abaqus-ISET HNA it is 0.152887974656. This is a relative difference of about $1E-12$. The ISET GFEMgl strain energy for the enriched global problem is 0.72149310104086, while with the Abaqus-ISET HNA it is 0.721493379352945. This is a relative difference of about $4E-007$.

2.5 Summary

A new non-intrusive algorithm coupling GFEM software with FEM software has been proposed and evaluated. The results of this thesis are

- Abaqus-ISET HNA results match ISET GFEMgl results closely
- Abaqus-ISET HNA results match verification results closely
- Accuracy decreases when shape functions in Abaqus and ISET are different
- SGFEMgl [24] improves accuracy in TET4, HEX8 elements
Figure 2.24: Geometry, discretization, and boundary conditions of the hat-stiffened panel
Figure 2.25: Hat-stiffened panel response to boundary conditions
• SGFEM reduces accuracy in TET10, HEX20 elements
• The HNA can simulate complex problems (hat-stiffened panel)
CHAPTER 3

THE PRECONDITIONED CONJUGATE GRADIENT SOLVER AND ITS APPLICATIONS FOR GFEM

This chapter focuses on efficiently solving the system of equations resulting in the enriched global-local phase of GFEM$^gl$. GFEM has traditionally been shown to be able to reduce the ndofs necessary to solve problems with sharp gradients or singularities, but it has also been shown to produce poorly conditioned matrices $K$. The poor conditioning discouraged the use of iterative solvers until the conditioning could be improved. The Stable GFEM (SGFEM) [4] uses minimal effort to change the enrichments functions of the problem such that the condition number of the matrix is significantly reduced.

With SGFEM’s introduction and its application to 3-D elastic fracture problems [22, 24] and material interface problems [25], a road to iterative solvers for GFEM systems of equations has been created. Gupta et al. [22] and Kergrene et al. [25] introduce a Block Gauss-Seidel iterative solver while Kim et al. [26] introduces a Block-Jacobi preconditioner for the preconditioned conjugate gradient method.

We look at solving GFEM$^gl$ using a re-analysis type method. The goal of re-analysis techniques in general is to be able to change the stiffness matrix or force vector and use a priori knowledge of the original solution to quickly generate a solution to the new problem [28]. Due to knowledge gained at each step of the $GFEM^gl$ process, any useable method should be able to utilize the Cholesky factorization of the $K^0$.

Previous work has utilized static condensation for solving sharp thermal gradient problems [41] and three-dimensional fracture problems [21]. The static condensation method is both a non-intrusive algorithm and a direct solver. It’s main weakness lies in its high memory costs.

Iterative solvers are examined in this thesis that improve the memory cost and may reduce the tasks cost. These include the Block Gauss-Seidel method, and the preconditioned conjugate gradient method using Block Jacobi and Block Gauss-Seidel preconditioners. The multigrid method is not well developed for the discontinuous coefficients resulting from GFEM problems, so the multigrid method is not used [25].
3.1 Problem Definition

3.1.1 Reduce Computation and Memory Requirements

This research focuses on solving systems of equations assembled with GFEM$^g$. GFEM$^g$ first obtains a global solution, with matrix representation $K^0$. In order to find the solution, it must find the Cholesky factorization of $K^0$ and perform forward and backward substitution. This solution is used to provide boundary conditions to a local problem. The solution to the local problem is used to create enrichments for nodes in the local enrichment zone in the global problem, which is represented as $K$ in equation (1.1) as $K^{gl}$. In a typical direct solver, $K$ is factorized again.

The static condensation direct solver is able to leverage the $d^{FEM} = (K^0)^{-1}f^{FEM}$, reducing the computations necessary to solve the problem over a typical direct solver. However, it also requires the computation of $S^{0,gl} = (K^0)^{-1}K^{0,gl}$, which is a dense matrix the same size as $K^{0,gl}$ [21]. As the problem size gets larger, this memory requirement will dominate system requirements over computations performed.

This chapter seeks to solve $Kd = f$ more efficiently, by lowering the memory requirements and reducing computations. The a priori knowledge of the Cholesky factorization of $K^0$ enable cheaper iterative schemes which require the Cholesky factorization of $K^{gl}$ and use the already factorized $K^0$; specific iterative schemes that utilize this knowledge include the Block Gauss-Seidel method and highly effective preconditioners for the PCG method.

These not only reduce computations, but also memory storage requirements over the static condensation solver, because it does not need to store a large dense matrix, and even the direct solver because it avoids fill-in in the $K^{0,gl}$ regions. Re-analysis methods [28, 38] were considered, but deemed to be no better than the static condensation solver in that they are direct and still require at least the factorization of $K$ to be stored.

**Linear Elastic Fracture Mechanics**  The iterative solvers will be compared using two-dimensional and three-dimensional linear elastic fracture mechanics problems, the same as in Section 2.1.1. In Section 3.2.1, discretizations for the 2-D edge-crack domain that Section 2.4.1 introduces are used to compare the number of iterations to convergence of several iterative solvers. These iterative solvers include the Block Gauss-Seidel method (BGS), the preconditioned conjugate gradient method using a Block Jacobi preconditioner (BJ-PCG), and the preconditioned conjugate gradient method using a Block Gauss-Seidel preconditioner (BGS-PCG).

In Section 3.2.2, discretizations for the 3-D edge-crack domain from Section 2.4.3 are used to
investigate the effect of SGFEM$^{gl}$ on the condition number of $K$. They are also used to compare the speed of the BGS-PCG with a robust direct solver.

3.2 Numerical Examples

3.2.1 2-D Edge-crack

The three different iterative methods are compared using a two-dimensional panel, the same as used in Paragraph 2.4.1. The Young’s modulus is unity, Poission’s ratio is 0.30, and plane strain conditions are assumed. For the 2-D problems, analytical enrichment functions are used to represent the singularity at the crack front. The mesh is created as a uniform and structured mesh of quadrilaterals. Each quadrilateral is then split into two triangular elements. A mesh of 32 rows by 32 columns of quadrilaterals split into two $TRI3$ elements per quadrilateral will be denoted $32 \times 32 \times 2$. Figure 3.1 shows the mesh.

![Figure 3.1: Layout of 64 × 64 element mesh and branch function enrichments (red)](image)

The three iterative solvers that will be compared using this example are the Block Gauss-Seidel, the Block Jacobi PCG, and the Block Gauss-Seidel PCG.
Block Gauss-Seidel Method

The Block Gauss-Seidel iterative method has been used in [22] and [25] to solve SGFEM problems. The Block Gauss-Seidel (Block GS) method uses a priori factorizations well. It requires that the matrix be broken up into four smaller matrices, the ones along the diagonal being square. In the GFEM$^g$, these diagonals are $K^0$ and $K^g$, with the off-diagonals described in equation (1.1). The following algorithm describes the implementation of Block GS:

**Algorithm 2 Block GS algorithm**

1: procedure Block GS($K$, $f$, $d$)
2: for $i = 0$ until convergence do
3: \[ r^g_l \leftarrow f^g_l - K^g_{l,0} d^0 \]
4: \[ d^g_l \leftarrow (K^g_l)^{-1} r^g_l \]
5: \[ r^0 \leftarrow f^0 - K^{0,g}_l d^g_l \]
6: \[ d^0 \leftarrow (K^0)^{-1} r^0 \]
7: end for
8: end procedure

Preconditioned Conjugate Gradient method

The Preconditioned Conjugate Gradient (PCG) method is known as one of the best iterative methods to solve systems of equations. An excellent introduction to PCG can be found in [44]. The method finds new search directions through $A$-orthogonalization of previous search directions. It finds the magnitude of this direction by using the residual, preconditioner, and $K$.

Usually the ease of solving a system of equations is linked with the condition number of the matrix $K$ in $Kd = f$. The higher the condition number, the longer it takes to solve (indirect methods) and the less accurate the solution is. The effectiveness of this method depends on the symmetric positive definite preconditioner $M$. $M$ is usually similar to $K$ but easier to factorize. The lower the condition number of $M^{-1}K$, the faster the convergence of PCG. Thus, the most effective preconditioner will be easy to compute while improving this condition number the most. Objective measures to compare our several preconditioners will be presented. The PCG algorithm as described in [44] is

**Block Jacobi Preconditioner** The Block Jacobi PCG was proposed for use in this context in [26]. The Block Jacobi PCG uses as a preconditioner
Algorithm 3 PCG algorithm

1: procedure PCG(\(K, r, d\))
2: \(i \leftarrow 0\)
3: \(r \leftarrow f - Ku\)
4: \(d \leftarrow M^{-1}r\)
5: \(\delta_{\text{new}} \leftarrow r^Td\)
6: for \(i = 0\) until convergence do
7: \(q \leftarrow Kd\)
8: \(\alpha \leftarrow \frac{\delta_{\text{new}}}{d^Tq}\)
9: \(d \leftarrow d + \alpha d\)
10: if \(i\) is divisible by 50 then \(\triangleright\) This resets \(r\) to exact value.
11: \(r \leftarrow f - Ku\)
12: else \(\triangleright\) \(r\) is typically not evaluated directly to save computations.
13: \(r \leftarrow r - \alpha q\)
14: end if
15: \(s \leftarrow M^{-1}r\)
16: \(\delta_{\text{old}} \leftarrow \delta_{\text{new}}\)
17: \(\delta_{\text{new}} \leftarrow r^Ts\)
18: \(\beta \leftarrow \frac{\delta_{\text{new}}}{\delta_{\text{old}}}\)
19: \(d \leftarrow s + \beta d\)
20: end for
21: end procedure

\[
M = \begin{pmatrix} K^0 & 0 \\ 0 & K^{gl} \end{pmatrix}. \tag{3.1}
\]

The Cholesky factorization of \(M\) can be found for \(K^0\) and \(K^{gl}\) independently, meaning that our a priori knowledge of the Cholesky factorization of \(K^0\) is utilized (reducing the cost for this preconditioner). The SGFEM motivates this preconditioner, since it partially orthogonalizes \(K^0\) against \(K^{gl}\), greatly reducing the magnitude of the entries in \(K^{0,gl}\). We will call this preconditioner the Block Jacobi Preconditioner (BJ-PCG):

Algorithm 4 BJ preconditioner algorithm

1: procedure BLOCK DIAGONAL\((K, d)\)
2: \(s^{gl} \leftarrow (K^{gl})^{-1}d^{gl}\) \(\triangleright\) One iteration
3: \(s^0 \leftarrow (K^0)^{-1}d^0\)
4: end procedure

**Block Gauss-Seidel Preconditioner** The Block Gauss-Seidel PCG uses the Block Gauss-Seidel Algorithm 2 as \(M\). The BGS method is similar computationally to the BJ method. Both solve
two systems of equations using the factorizations of $K^0$ and $K^{gl}$. Since $K^0$ is already calculated in initial global step of GFEM$^{gl}$, they both utilizes a priori knowledge well. The BGS method is different in that it adds additional sparse matrix multiplication of the $K^{0,gl}$ and its transpose, which is relatively cheap, and it can run multiple iterations. One iteration of the BGS, per PCG iteration, is used in this thesis for BGS-PCG.

Comparisons of Solver Algorithms

The main metrics through which we will determine the effectiveness of a solution method are the effects from the enrichment zone size and element size $h$. Any solver for GFEM must be able to efficiently handle increases in enrichment zone size and reduction in element length $h$. The enrichment zone size is important to GFEM methods because they use enrichments to increase the accuracy of the model. As the element size $h$ decreases, the accuracy of the model increases. Thus, both metrics are important to obtaining an accurate GFEM solution.

If a solution method can handle both of these most efficiently, then it is considered the best iterative solver method. We desire an error tolerance (compared to direct solution) of $10^{-5}$. The amount of computational work required to reach this value will be compared across several iterative methods.

The BGS, BJ-PCG, and BGS-PCG have similar costs in number of flops per iteration. The BGS, BJ-PCG, and BGS-PCG require a sparse forward-backward substitution for $K^0$ and $K^{gl}$, costing $O\left((n^0)^2 + (n^{gl})^2\right)$. The BJ-PCG and BGS-PCG also require a sparse matrix vector multiplication that conservatively costs $O\left((n^0)^2 + (n^{gl})^2 + 2 \times n^{0,gl})^2\right)$. The BGS-PCG and BGS require an additional multiplication cost of the coupling terms $2 \times O\left((n^{0,gl})^2\right)$; however it has been found in practice that this cost is much smaller than the block diagonal's multiplication or backward and forward substitution, so we ignore it. As a result, the BJ-PCG and BGS-PCG cost approximately the same per PCG iteration, but each cost about twice as much per iteration as BGS.

The comparable iteration costs allow us to compare each method’s computational speed using their iteration count. If any method takes fewer iterations than the other methods, then it will be considered the faster method. Error $e^i$ in the solution is calculated at each step through comparison of the solutions at an iteration $d^i$ with a precalculated direct solution $\hat{d}$ as $e^i = \frac{|d^i - \hat{d}|}{||d^i||}$. Convergence is defined as error in solution being less than $10^{-5}$.

A 2-D edge-crack example problem identical to that in Section 2.4.1 is considered. The $32 \times 32 \times 2$ mesh was examined with GFEM (sans SGFEM) for each iterative method, but found to be significantly slower ($> 10 \times$) than with SGFEM. Consequently, we have omitted convergence.
results for the GFEM method from this thesis. For a more thorough look at the advantages of SGFEM versus GFEM, see [22] or [25]. While the example run in [22] is very similar to here, the crack configurations are different. This thesis considers an edge-crack while [22] considers a penny-like crack.

**Effect of Enrichment Zone Size** In order to test the effect of enrichment size, the smallest enrichment was created such that the crack was fully formed. Then larger enrichments were taken which represented the same crack. Thus, the effect of enrichment size is isolated. The crack and the outlines of the enrichments are shown in Figure 3.2. The percentages refer to the area of the enrichment normalized by the area of the FEM mesh.

![Figure 3.2: Layout of Five Enrichments Corresponding to the Same Crack Size in 32x32 Mesh](image)

Since the Block Diagonal Preconditioner fails to account for the off-diagonal portions of $K$, it seems reasonable to anticipate that as long as $n_i^{gl} << n^0$ holds, that the BJ-PCG will be accurate. As the size of $K^{gl}$ increases, the size of $K^{0,gl}$ and $K^{gl,0}$ also increase, becoming relatively a larger proportion of the entire matrix $K$. Since the BJ-PCG does not account for these off diagonals, it seems reasonable that convergence worsens considerably. However, this is not the case as shown in
Figure 3.3. This plot shows the ndofs corresponding to enrichment zone size as seen in Figure 3.2 versus the iterations to convergence.

It can be seen clearly that the BGS takes many more iterations than BJ-PCG and BGS-PCG. However, there is negligible detrimental effect of more degrees of freedom for each method. The behavior of BJ-PCG and BGS-PCG are very similar, but with BGS-PCG shifted down about 12 iterations. Also, for the 100% enrichment zone size, BGS-PCG increases only 1 iteration where BJ-PCG increases 2.

It is concluded that for increasing enrichment zone sizes, the BGS-PCG and BJ-PCG are faster than BGS because their iteration count is about $\frac{1}{40}$ the number of iterations for BGS. Thus even when the lower cost of a BGS iteration is considered, the BGS-PCG and BJ-PCG are significantly faster. The BGS does not improve relatively as the enrichment zone size increases. It is also concluded that the BGS-PCG is faster than the BJ-PCG for increasing zone size because it take about half as many iterations. The BJ-PCG does not improve relative to BGS-PCG as the enrichment zone size increases.

**Effect of Mesh Size** In order to measure the effect of decreasing the element size (increasing the problem size), we must hold the enrichment zone and crack size constant. This is done for three mesh sizes, $32 \times 32 \times 2$ elements, $48 \times 48 \times 2$ tri elements, and $64 \times 64 \times 2$ elements. Thinking of this in terms of the raw matrix representation, from $32 \times 32 \times 2$ to $64 \times 64 \times 2$ the size of $K^0$ and $K^{qt}$ quadruples.

The size of the elements in the mesh is decreased so that the number of elements increase from
$32 \times 32 \times 2$ over the domain to $48 \times 48 \times 2$ and $64 \times 64 \times 2$. The size of the enrichment zone is still 12%, as shown in Figure 3.2 and consequently the size of the crack remains constant. Thus we isolate the effect of increasing the number of elements in the mesh, which will add more nodes and more enriched nodes in the 12% enrichment zone size. Results are shown in Figure 3.4.

As the relative size of the mesh gets larger, it appears that the number of iterations follows a path with a power less than one for all methods. The BGS is much slower to converge, even with the $2 \times$ penalty (from matrix-vector multiplication) on BJ-PCG and BGS-PCG. Further, the BGS-PCG has fewer iterations to convergence and has smaller slope than the BJ-PCG method. Thus, the BGS-PCG is the fastest method for reducing element size $h$ and increasing enrichment zone size. This means that as far as preconditioning goes, the BGS-PCG method is superior.

Figure 3.5 shows the convergence of BGS-PCG for even larger problem sizes, going up to a $1024 \times 1024 \times 2$ mesh.

It confirms that the number of iterations increases sublinearly as the problem size increases. This is a good feature for the scalability of the problem. However, it does not seem to reach an asymptotic rate of iterations per ndof.

### 3.2.2 3-D Edge-crack Analysis

A 3-D edge-crack is considered identical to the example in Section 2.4.3. $\mathbf{K}$ is sparse, making matrix multiplication much cheaper for iterative solvers and enabling the use of optimized sparse direct
solvers for comparison purposes. Due to the typically high condition number of GFEM $K$, iterative solvers have had slow convergence and have not been widely used.

In order to more accurately gauge the ability of BGS-PCG to solve the 3-D edge-crack example, several mesh refinement schemes are used, as seen in Figure 3.6. First, a Quasi 2-D mesh refinement scheme keeps one element through thickness $t$ and refines along the in-plane directions ($b$ and $l$) uniformly. Second, the 5-to-2 3-D case case starts with 5 elements along the in-plane directions and 2 elements along the thickness. It maintains the ratio of 5 elements in in-plane directions and 2 elements in thickness as it is refined. Such a refinement scheme is denoted 5-to-2 3-D. Finally, the 1-to-1 3-D case has the same number of elements along all three directions ($b$, $l$, and $t$). Such a refinement scheme is denoted 1-to-1 3-D.

The bandwidth and condition number $\kappa$ of $K^0$ are influenced by these mesh refinement schemes. Consequently, $K$ is influenced as well. Now the effect of SGFEM$^d$ on $\kappa(K)$, $\kappa(K^0)$, and $\kappa(K^d)$ is investigated for each refinement scheme. For $\kappa(K^0)$, as the discretization is refined and the size of an element $h$ decreases, the condition number grows $O(h^{-2})$. It has been shown that for various situations the condition number of GFEM grows $O(h^{-4})$. It has been shown that SGFEM reduces the rate of growth of the condition number to $O(h^{-2})$ [3, 23].

Edge-crack Example with Point Dirichlet Boundary Conditions

However, for the edge-crack example in Section 2.4.3, point Dirichlet boundary conditions are used to prevent rigid body motion. It has been shown that for $K^0$ in which nodal values are assigned, the condition number grows at a rate of $O(h^{-3})$ [6]. The condition number corresponding to $K$ is
Figure 3.6: Meshes for several refinement schemes
called $\kappa$, to $K^0$ is called $\kappa^0$, and to $K^{gl}$ is called $\kappa^{gl}$. The $\kappa(K) \geq \kappa(K^0)$, so it seems unlikely that the SGFEM will lead to a growth in $\kappa(K)$ that is less than $O(h^{-2})$. Figure 3.7 shows condition number $\kappa$ when nodal values are prescribed for the Quasi 2-D and 1-to-1 3-D mesh refinement schemes. These two schemes show the range in condition number growth as a mesh is refined. For both problems, $h$ as taken as the element length in the in-plane directions.

![Figure 3.7: Condition numbers when nodal values are specified using point Dirichlet boundary conditions](image)

The ratio $\frac{\log \kappa_{n+1} - \log \kappa_n}{\log 1/h_{n+1} - \log 1/h_n}$ is considered the rate of condition number change, as in $O(h^{-2})$. As expected, for $K^0$ with no Dirichlet bcs $\kappa = O(h^{-2})$. Also, for both mesh refinement cases with $K^0$ with point Dirichlet bcs, the $\kappa$ and its rate of change is significantly higher. The 1-to-1 3-D case has $\kappa = O(h^{-3})$, matching the estimate in [6]. The Quasi 2-D case is smaller, but it is likely following
the estimate for $\kappa$ for 2-D meshes with prescribed nodal values, which is $\kappa = O((1 + |\ln h|)h^{-2})$.

Interestingly, $\kappa$ is on the same order of magnitude for GFEM$^gl$ and SGFEM$^gl$. This may be explained by the dominating influence that the point Dirichlet bcs have on $\kappa$. It may be that for large enough $1/h$, the rate of growth of GFEM$^gl$ will return to $O(h^{-4})$ as found in [3] for a 1-D GFEM problem.

Another interesting feature that is unaffected by the boundary conditions is $\kappa^{\text{gl}}$. SGFEM$^gl$ noticeably decreases $\kappa^{\text{gl}}$ from GFEM$^gl$. This is surprising given that SGFEM$^gl$ was designed to reduce the condition number by reducing the coupling between $K^0$ and $K^{\text{gl}}$.

What effect does the small decrease in $\kappa$ from GFEM$^gl$ to SGFEM$^gl$ have on iterative solver convergence? For each of the three refinement strategies, and mesh refinements, several iterative solvers are used to solve the system of equations. These include the conjugate gradient method, the BJ-PCG, and BGS-PCG. Figure 3.8 shows the results.

First, notice that the number of iterations of the GFEM$^gl$ are overall significantly higher than that for SGFEM$^gl$. Second, notice that the slope is less for SGFEM$^gl$. This indicates that although $\kappa$ for GFEM$^gl$ and SGFEM$^gl$ are fairly similar, iterative solvers can solve SGFEM$^gl$ much more quickly. SGFEM$^gl$ has the effect of reducing the coupling $K^{0,\text{gl}}$ between $K^0$ and $K^{\text{gl}}$. A larger coupling slows down the conjugate gradient method convergence significantly. This is surprising because a common estimate of the number of iterations $i$ for the conjugate gradient method relies on $\kappa$ and the tolerance $\epsilon$, $\kappa \leq \left[\frac{1}{2}\sqrt{\kappa \ln \frac{2}{\epsilon}}\right]$. For GFEM$^gl$ problems, the coupling seems to have a significant additional effect, beyond the conditioning, on iterations to convergence.

Second, using GFEM$^gl$ rather than SGFEM$^gl$ has a drastic negative effect on the number of iterations to convergence in all cases. Thus, SGFEM$^gl$ is necessary to iteratively solve this class of problems.

Comparison between the Quasi 2-D, intermediate, and 1-to-1 3-D case is difficult because the definition for $h$ is inconsistent. Rather, if the number of equations in the system of equations is used, a comparison can be made. Figure 3.9 shows this comparison.

It can be seen that the CG iterations to convergence for the Quasi 2-D case grows more quickly than for the 5-to-2 3-D case, which in turn grows more quickly than the 1-to-1 3-D case. This is directly an effect of the condition number growth, since for the Quasi 2-D GFEM$^gl$ $\kappa = O(\text{ndofs}^{1.67})$ and for the 1-to-1 3-D GFEM$^gl$ $\kappa = O(\text{ndofs}^{1.06})$.

The Quasi 2-D case has BGS-PCG slopes of 0.495 for GFEM$^gl$ and 0.204 for SGFEM$^gl$. The 5-to-2 3-D case has BGS-PCG slopes of 0.369 for GFEM$^gl$ and 0.205 for SGFEM$^gl$. Finally, the 1-to-1 3-D case has BGS-PCG slopes of 0.471 for GFEM$^gl$ and 0.198 for SGFEM$^gl$. These are very similar values, which indicates that the BGS-PCG seems to cancel some effect of the larger $\kappa$ in the
Figure 3.8: Iterations to $\epsilon = 10^{-5}$ when nodal values are specified using point Dirichlet boundary conditions
Figure 3.9: Iterations to $\epsilon = 10^{-5}$ when nodal values are specified using point Dirichlet boundary conditions
Quasi 2-D case. Indeed, the fact that BGS-PCG solves uses exactly \( K^0 \) in its preconditioner means that it can ignore the primary driver in the growth of \( \kappa, \kappa^0 \).

Edge-crack Example with Face Dirichlet Boundary Conditions

The effect of point Dirichlet bcs on \( \kappa \) can be removed by prescribing face boundary conditions. The bottom of the domain in Section 2.4.3 is clamped, as shown in Figure 3.10.

Figure 3.10: 3-D edge-crack problem description for surface Dirichlet bc problem

The behavior in this problem should be similar to the pure Neuman problem. Because it avoids using point Dirichlet bcs, it is expected that \( \kappa^0 = O(h^{-2}) \). Previous implementations of SGFEM [3, 23] have shown that \( \kappa = O(h^{-2}) \), so it is hoped that SGFEM\( ^{gl} \) will result in \( \kappa = O(h^{-2}) \). The effect of this change on \( \kappa, \kappa^0 \), and \( \kappa^{gl} \) is shown in Figure 3.11.

The \( \kappa^0 \approx O(h^{-2}) \) for the Quasi 2-D case and the 1-to-1 3-D case. The same is true using
Figure 3.11: Condition numbers when using face Dirichlet boundary conditions
SGFEM $\kappa \approx O(h^{-2})$ for the Quasi 2-D case and the 1-to-1 3-D case. However, using GFEM$^d$ the $\kappa < O(h^{-4})$. This may be due to the limited range of $h$ that is considered, and warrants further investigation.

Nevertheless, the effect of the point Dirichlet bcs on $\kappa$ has been removed, and the ability of the BGS-PCG to solve this example problem can be examined. GFEM$^d$ is not considered since it has already been shown to be impractical for this class of problems. For the Quasi 2-D and 1-to-1 3-D cases the BGS-PCG solves this problem in close to or exactly the same number of iterations. This is due to the effect of BGS-PCG solving for $K^0$ directly in its preconditioner. This means that it can ignore the primary driver in the growth of $\kappa$, $\kappa^0$, and consequently it can solve point Dirichlet boundary condition or face Dirichlet boundary condition problems just as quickly. Conversely, if one seeks to improve the BGS-PCG by using an approximate preconditioner for $K^0$, the resulting method would likely suffer when solving problems which use point Dirichlet boundary conditions.

Comparison of BGS-PCG with Pardiso

The sparse direct solver that used for comparison in this thesis is Intel’s Pardiso [30, 43]. While Pardiso does have parallel capabilities, they are not used for comparison with BGS-PCG because it has not been parallelized. The pure Neuman edge-crack problem with point Dirichlet boundary conditions discussed in Section 3.2.2 is used to compare BGS-PCG with Pardiso.

The bandwidth of $K^0$ can vary depending on the number of nodes that are connected through elements. The bandwidth can have a large impact on the speed of Pardiso, so the same 3-D edge-crack problem was meshed in several different schemes to change the bandwidth, as shown in Figure 3.12. The x-axis shows the column number while the y-axis shows the row number. In the figure “nz” refers to the number of nonzero entries. Each nonzero entry is represented in its correct row and column by a small dot.

The Quasi 2-D mesh has only one element through the depth as the other two dimensions are refined. This results in many fewer nodes connected through elements, resulting in a thinner bandwidth in $K$ when assembled. The 5-to-2 3-D mesh is refined in all three directions at the same rate, but will always have a ratio in the $x$, $y$, and $z$ directions of 5 to 5 to 2. The 1-to-1 3-D always has the same number of elements in all three directions. This results in the largest bandwidth. Pardiso uses a fill-in minimizing re-ordering which switches the rows and columns to reduce the bandwidth. The fill-in for the Quasi-2-D and 1-to-1 3-D stiffness matrices before and after the fill-in minimizing re-ordering capability in Matlab (function “symrcm” is used) is shown in Figure 3.12.

These Quasi 2-D, 5-to-2 3-D, and 1-to-1 3-D meshes were refined and their respective problems
Figure 3.12: Meshes for several refinement schemes
were solved until their $K$ had about $2 \times 10^6$ degrees of freedom. Several data points are shown in Figure 3.13, along with the final log-log slope.

![Graphs showing performance comparison between BGS-PCG and Pardiso for different meshes](image)

Figure 3.13: Comparison of BGS-PCG with Pardiso for several refinement schemes

Notice that for smaller ndofs, the BGS-PCG is always faster than Pardiso. This is due to the factorization of $K^0$ occurring in the initial global local step. Thus, BGS-PCG does need to calculate it, and it is very efficient for small matrices.

A quick summary of the largest mesh times and asymptotic slopes is shown in Table 3.1. The thinner bandwidth $K$ from the Quasi 2-D mesh Pardiso takes less time, but has a higher slope. Thus, it is likely that for larger Quasi 2-D problems the BGS-PCG will take less time than Pardiso. The lessening slope of BGS-PCG can be traced to the log-log slope of iterations to convergence versus ndofs decreasing as well.
Table 3.1: BGS-PCG versus Pardiso final slope

<table>
<thead>
<tr>
<th>Case</th>
<th>Final time (s)</th>
<th>Final log-log slope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quasi 2-D Pardiso</td>
<td>120.2</td>
<td>1.32</td>
</tr>
<tr>
<td>Quasi 2-D BGS-PCG</td>
<td>153.6</td>
<td>1.28</td>
</tr>
<tr>
<td>5-to-2 3-D Pardiso</td>
<td>4986</td>
<td>2.0</td>
</tr>
<tr>
<td>5-to-2 3-D BGS-PCG</td>
<td>470</td>
<td>1.58</td>
</tr>
<tr>
<td>1-to-1 3-D Pardiso</td>
<td>7030</td>
<td>2.07</td>
</tr>
<tr>
<td>1-to-1 3-D BGS-PCG</td>
<td>496</td>
<td>1.54</td>
</tr>
</tbody>
</table>

For the 5-to-2 3-D case, Pardiso is 10× slower than BGS-PCG and has a higher slope. Finally, the 1-to-1 3-D case shows the Pardiso about 14× slower than the BGS-PCG. The BGS-PCG is faster than Pardiso for the cases stated; however more can be understood about the effect of bandwidth on BGS-PCG convergence. This is shown in Figure 3.14.

![Comparison of BGS-PCG with Pardiso for several refinement schemes](image)

Figure 3.14: Comparison of BGS-PCG with Pardiso for several refinement schemes

Figure 3.13 shows that Pardiso requires much less time for solving the Quasi 2-D system of equations than for 1-to-1 3-D. This reflects that the reduced bandwidth of the Quasi 2-D makes factorization faster (time is 89, 4899, and 6927, for Quasi 2-D, 5-to-2 3-D, and 1-to-1 3-D respectively). A more minor effect for the Pardiso solver is that a reduced bandwidth reduces the forward-backward substitution time as well (time is 31, 87, and 103, for Quasi 2-D, 5-to-2 3-D, and 1-to-1 3-D respectively).

The effects of the bandwidth on the BGS-PCG are less pronounced, with the final slopes and times being much closer than for Pardiso. The preconditioner and sparse-matrix-vector multiplication steps dominate time spent per iteration and is shown in Figure 3.15.
For the 1-to-1 3-D case, which has the widest bandwidth, the ratio of preconditioner time to matrix-vector multiplication time is higher. One factor that plays into this is the number of BGS-PCG iterations needed for convergence. The Quasi 2-D needs 50, 5-to-2 3-D needs 32, and 1-to-1 3-D needs 28. There are about the same number of dofs along the crack, and the 1-to-1 3-D problem has distorted elements similar to the Quasi 2-D problem, so these can not be sources of the iteration difference. More likely, since $K^0$ is factored directly, the extra iterations are a result of the coupling $K^{0,gl}$ not being reduced by SGFEM$^{gl}$.

The number of iterations for each case (see Figure 3.9) means that we should expect the preconditioner and matrix-vector multiplication to take the longest for Quasi 2-D down to the least for 1-to-1 3-D. While this is true for matrix vector multiplication, it is not the case for the preconditioner. The main cost for the preconditioner is forward-backward substitution. We observed earlier that if the bandwidth is smaller, than the factorization and forward-backward substitution times are also smaller. This in turn greatly reduces the cost of the BGS preconditioner, as confirmed by the Quasi 2-D problems preconditioner step taking 109 seconds (50 calculations) and the the 1-to-1 3-D problem taking 425 seconds (28 calculations).

Interestingly, all three cases have final log-log slopes that seem to converge to a value near 0.20. This only reinforces the concept that bandwidth will continue to cause differences in the cost of BGS-PCG for the three mesh types. The impact of bandwidth on preconditioner step time, which in turn drives the cost of the BGS-PCG, might inspire the use of the incomplete Cholesky factorization to reduce the bandwidth of the factorization. While its effects have not been studied on the BGS-PCG algorithm, a BGS algorithm similar to that proposed in [25] could be used. It would use a
PCG solver preconditioned by incomplete Cholesky factorization to solve the inverse matrix steps (4, 6) of the BGS procedure in Algorithm 2.

3.3 Summary

The BGS-PCG iterative solver has been shown to:

- require fewer iterations to convergence than the BGS and BJ-PCG iterative solvers
- not suffer from conditioning effects of point Dirichlet boundary conditions
- possess asymptotic rates of convergence faster than the direct solver Pardiso
- faster solve times in almost all problems than the direct solver Pardiso

When combined with the benefit of requiring less memory, BGS-PCG has been shown to be an effective replacement for direct solvers for a SGFEM\textsuperscript{gl} problem.
4.1 Contributions

A non-intrusive algorithm was proposed that couples a FEM code with an X/GFEM code. It was shown to be accurate with minimum computational cost, not requiring iterations between the FEM code and GFEM code or extra storage. The GFEM$^h$ hierarchical framework has been shown to be applicable for many simulation types, including thermo-plastic, fracture mechanics, and others. Thus, the HNA allows unique simulations that benefit from both FEM and GFEM codes.

An iterative preconditioned conjugate gradient solver has been proposed for solving GFEM$^h$ hierarchical systems of equations. It has been shown to be faster than several other previously proposed iterative schemes, as well as a sparse direct solver. An effective iterative solver will be critical to solving large-scale GFEM$^h$ problems in the future.

4.2 Future Directions

The HNA proposed in this thesis requires that the element shape functions of enriched nodes be known. Mixed results were encountered when blindly assuming that a node had a classical hat shape function when it did not. A process of accurately enriching a shape function that is not known could be worked out. The non-intrusive algorithm allows features unique to an FEM to be combined with an inherently unique GFEM code quickly and easily. New simulations could combine the features of the two codes and study the capabilities.

The BGS-PCG has been shown to have an effective, albeit expensive, preconditioner. The incomplete Cholesky factorization could make the preconditioner cheaper, but possibly at the cost of an increase in iterations to convergence. This effect could be studied. The BGS-PCG could be parallelized in many regions as well, allowing a better comparison with a parallelized direct solver like Pardiso.
4.2.1 List of Future Directions

In summary, future directions that could be taken are

- Accurately enrich unknown shape functions
- Simulate features unique to FEM and GFEM codes in tandem
- Reduce the cost of the BGS-PCG preconditioner
- Parallelize BGS-PCG
Several XFEM/GFEM codes were mentioned in Section 1.1. Some websites with easily available XFEM/GFEM codes are in the following list. While these have not been tested with HNA they may be good points of entry for XFEM/GFEM.

1. MXFEM [37] http://www.matthewpais.com/2Dcodes
Abaqus can generate and output a stiffness matrix in *.inp files using the keywords "*MATRIX GENERATE, STIFFNESS" and "*MATRIX OUTPUT, STIFFNESS." These keywords must be located within a loading "*STEP" that is not "*STATIC." The matrix will be stored in a file that reflects the name of your job. While multiple keywords can be used in the same "*STEP," it is generally advisable to use different steps to output different things. For example, a second step uses "*MATRIX GENERATE, LOAD" and "*MATRIX OUTPUT, LOAD" to output the load vector. One reason for this is that in order to output the element stiffness matrix, one actually needs a "*STATIC" step type. The syntax is then "*ELEMENT MATRIX OUTPUT, ELSET=Set-1, STIFFNESS=YES." ELSET dictates the specific element the user wants to output. The rest of an .inp file can be understood by creating Abaqus models using their graphical user interface. When you submit a job for analysis, Abaqus will create a .inp file that includes all needed keywords to run that specific example. In ubuntu, an .inp file can be run using the syntax

```
./(executable name) job=(job name) inp=(input file name).
```

APPENDIX B

ABAQUS COMMANDS TO OUTPUT $\mathbf{K}^0$ AND $\mathbf{F}^0$
ISET is run from a file that has a sequence of commands. The following file was used to solve the 3-D edge-crack problem in Section 2.4.3. This is a tcl file, so “#” denote comments and “\” denotes a command continued on the next line.

# read the model
readFile domain_4x4x1_mesh_5x5x2_TET4.grf phfile

# create a linear re-analysis object
createAnalysis linearReAnalysis

#########################################################################
puts "\n\n****** Solve Initial Global Problem *****\n\n"

#Use linear polynomial order for mesh
set p_xyz 1
enrichApprox iso approxOrder $p_xyz

# Use direct method to enforce point Dirichlet BC
#parSet directMethodForPtDirichletBC true

#Use SGFEM
parSet SGFEMSpecialBasisEnrichments on

assemble

solve

#########################################################################
# create a local problem around crack surface
puts "\n\n****** Solve Local Problem *****\n\"

# Define local domain geometry
createLocalProblem probID $ilocprob \
    xyzMin -0.01 1.5 -0.01 \
    xyzMax 1.7 2.5 1.1 \
    numLayers $locLayers userBC "spring_local"

# Create a crack manager for local problem
create crackManager $ilocprob crackMgrID [expr $ilocprob + 10] \
create crackFile "mesh_4x4x1_GFEM_crack.crf"

# Refine local mesh so that the size of the elements bordering
# the crack front is about 5% the crack length
refineMesh crackFronts maxEdgeLen 0.05

# Use branch functions
setOptions useBranchFn true

# Create step and branch functions and set them to CompNods
# of LocalProblem
process

# Use quadratic polynomial order for local mesh
set p_loc 2
enrichApprox iso approxOrder $p_loc

assemble localProb $ilocprob

solve localProb $ilocprob


puts "\n\n****** Solve Enriched Global Problem *****\n\"

69
# Define the part of the global domain to be enriched with the local solution
setCompNodSpBasis inBBox \
xyzMin -0.01 1.5 -0.01 \
xyzMax 1.7 2.5 1.1 \
spBasis $ilocprob

# Enrich the global domain with global-local enrichments
enrichCompNod all hasLocalSol iso order 1

# Assemble and solve the enriched global problem
assemble reAnalysis

solve reAnalysis

puts "\n\n***** Finished Global Problem Enriched w/ Local Problem *****\n\n"
The speed at which each of these matrices converges depends upon the condition number of $M^{-1}K$. Usefully, the factorization $M = LU$, $M^{-1} * K$ and $L^{-1}KU^{-1}$ have the same eigenvalues, allowing us to use $L^{-1}KU^{-1}$ in our calculations [44].

An easy way to compare the effect that multiple BGS iterations have on the PCG method would be to calculate their condition numbers. Looking at how increasing $i_{BGS}$ iterations affect the condition number may give increased insight into how to implement the number iterations that will be optimal. For the Block Diagonal Preconditioner $M = LU$ is part of PCG solution process. However, the Block GS preconditioner requires more effort. The Block GS method solves the linear system $Kd = f$ for $d$. We could solve for many vectors $U = \{d_1, d_2, d_3, ..., d_n\}$ so that we have the system $Kd_i = F$. If we set $F = I$, then we will effectively be solving for $U = M^{-1}$: $KM^{-1} = I$. Figure D.1 displays the effect that increased iterations have on the condition number.

As $i_{BGS}$ iterations increase, the condition number of the matrix reduces dramatically. However, the iterations for each convergence are not equivalent for different numbers of iterations of the Block
GS preconditioner. Indeed, as discussed previously, the iterations for each method are converted into iteration equivalents for comparison. The weak condition for the relationship between the condition number, $\kappa$, and the convergence per iteration $\omega$ is $\frac{\sqrt{\kappa - 1}}{\sqrt{\kappa + 1}}$ [44]. We use this relationship to calculate an $\omega_{equiv}$ to compare the various $i_{BGS}$ iterations:

$$\omega^i = \omega_{equiv}^i$$

$$\omega_{equiv} = \omega_{equiv}^{-\frac{1}{i}}$$

A $32 \times 32$ mesh is used, and the number of nonzero entries calculated in order to make the comparisons shown in Figure D.2.

![Figure D.2: Number of Block GS iterations vs the Equivalent Convergence of $M^{-1}K$ for $32 \times 32$ mesh](image)

For small $i_{BGS}$, $\omega_{equiv}$ decreases dramatically. It then plateaus, and increases again at very large $i_{BGS}$. The $32 \times 32$ mesh for $i_{BGS} = 1$ has theoretical values $\kappa = 349.2$ and $\omega_{equiv} = 0.898$. Also $i_{BGS} = 15$ has $\kappa = 2.9$ and $\omega_{equiv} = 0.905$. If we take the convergence we get $\omega_{equiv} = 0.398$ for $i_{BGS} = 1$ and $\omega_{equiv} = 0.681$ for $i_{BGS} = 15$.

It seems that the best $i_{BGS}$ is 1 or 2. Bear in mind that these plots does not take into account
the beneficial effect of repeated eigenvalues and should be considered only a ceiling for the actual $\omega$ [44].
APPENDIX E

BGS AS AN INEXACT PRECONDITIONER

Inexact Preconditioners  One issue that may arise from using the BGS iterative method as a preconditioner is that it violates the assumption that the preconditioner does not change. Previous work has studied such iterative preconditioners within the context of PCG and termed them flexible, inexact, or variable preconditioners. It has been shown that the particular calculation of $\beta$ in the PCG algorithm above can slow down convergence [19]. Replacing the calculation with the similar $\beta = \frac{\delta_{\text{new}} - r_{\text{old}}}{\delta_{\text{old}}}$ has been shown to accelerate convergence to at least the speed of the steepest descent method [29].

Also, these preconditioners can exhibit poor behavior when the inner iteration is not precise enough [19]. Golub and Ye showed that the convergence deteriorated significantly at a certain stopping criteria, but plateaued below that stopping criteria. They concluded with a heuristic stopping criteria $\eta_0 = \frac{1}{2\sqrt{r}}$ [19]. At the very least this means that we need a stopping criteria based on error level. However, since the condition number changes significantly for increasing Block Gauss-Seidel iterations, the use of $\eta_0$ is inadvisable. We hope to find a computationally efficient equivalent.

The effect of using a constant $n_{BGS}$ for several values is explored in Appendix D. It concludes that $n_{BGS}$ should be 1 or 2. However, several experiments were run attempting to optimize the number BGS iterations in the preconditioner for the $32 \times 32$ problem. Varying the number of iterations did not seem to improve convergence (over a constant number) at all. It was found that many iterations at the beginning of the outer loop would improve convergence, but not subsequent rate of convergence. It was also found that if the number of iterations increased if the outer loop slowed, then the rate of convergence did not increase immediately, but slowly. These results are consistent with the concept of the accuracy of the preconditioner corresponding to its ability to search in orthogonal directions and consequently the convergence rate.

Varying the number of BGS iterations at each PCG step guarantees that the preconditioner will not be constant and that it is inexact. It has been assumed that since the BGS runs iterations that it is also inexact. However, it may be exact for constant BGS iterations, which would support the inability to vary $n_{BGS}$ and improve the solution. In order to facilitate comparison with BGS and
BJ-PCG (which have approximately the same work per iteration), we will set $n_{BGS} = 1$. 
Effect of Enrichment Zone Size  We have an edge-crack and then vary the size of the enrichment from 12% of the domain to all of the domain as shown in Figure 3.2. Figure F.1 shows the results for the varying enrichments.

![Error Convergence of Increasing Enrichments on 32 × 32 Mesh using BGS-PCG method](image)

Figure F.1: Error Convergence of Increasing Enrichments on 32 × 32 Mesh using BGS-PCG method

Figure F.1 shows a small increase in iterations for increased mesh size. Effectively, this means the scalability for larger enrichments is conserved for the Block GS PCG.

Effect of Mesh Size  The length $h$ of the elements in the FEM mesh is decreased, holding the domain of the crack and enrichments 12% constant. Results are shown in Figure F.2.

Only small decreases in convergence are observed for large decreases in $h$. 2.10
Figure F.2: Error Convergence of $32 \times 32 \times 2$, $48 \times 48 \times 2$, and $64 \times 64 \times 2$
REFERENCES


