DISSERTATION

COMPUTATIONAL FEASIBILITY OF SIMULTANEOUS ANALYSIS AND DESIGN IN INTERIOR POINT TOPOLOGY OPTIMIZATION

Submitted by

Justin O'Connor

Department of Mathematics

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2023

Doctoral Committee:

Advisor: Wolfgang Bangerth

Patrick Shipman
James Liu
Chris Weinberger

ABSTRACT

COMPUTATIONAL FEASIBILITY OF SIMULTANEOUS ANALYSIS AND DESIGN IN INTERIOR POINT TOPOLOGY OPTIMIZATION

Topology optimization is a class of algorithms designed to optimize a design or structure to accomplish some goal. It is part of a process of computer generated design that allows engineers to design better products faster.

One such algorithm that has piqued the imagination of developers is called Simultaneous Analysis and Design (SAND), especially in the context of Interior Point Optimization (IPO). This method is known to generate extremely optimal designs, and is good at avoiding local minima. However, this method is not used in practice, due to its computational cost.

This thesis examines the SAND IPO method, and develops an effective algorithm to generate a design using it. I begin by discussing nonlinear optimization algorithms, selecting pieces that work together for this problem, to generate a cohesive algorithm for the whole process.

Inside this developed algorithm, as with most nonlinear optimization algorithms, the most expensive part is a linear solve. In my case, it is a linear solve of a block system. I develop and implement a multi-tier preconditioning approach to solve this system in a reasonable amount of time.

Finally, I present a large topology optimization problem presented in three dimensions that has been solved using IPO and SAND, demonstrating the usability of the implemented algorithm.

ACKNOWLEDGEMENTS

Words cannot express my gratitude to my wife, Kelly, for her love and support, and for all the help she provided through this whole process.

I'm extremely grateful to my advisor, Wolfgang, for helping me find a research project I enjoyed, guiding me through the research process, and encouraging me even through debugging my code.

Thank you to my parents and brother for pushing me to do hard things, and for trying their hardest to proofread this thesis.

Many thanks to the students, teachers and administration at St. John Vianney Catholic School, where I rediscovered my love for mathematics.

Finally, thank you to all my friends, including Adam, Mats, Brian, Seth, Jayme, Lexie, Johnna, Amie, Vlad, and everyone who has encouraged me through this adventure, and have kept me motivated and looking towards the finish line.

DEDICATION

I would like to dedicate this thesis to my Heavenly Mother

Holy Mary, Holy Mother of God, Holy Virgin of virgins, Mother of Christ, Mother of the Church, Mother of Mercy, Mother of Divine Grace, Mother of Hope, Mother most pure, Mother most chaste, Mother inviolate, Mother undefiled, Mother most amiable, Mother admirable, Mother of good counsel, Mother of our Creator, Mother of our Savior, Virgin most prudent, Virgin most venerable, Virgin most renowned, Virgin most powerful, Virgin most merciful, Virgin most faithful, Mirror of justice, Seat of wisdom, Cause of our joy, Spiritual vessel, Vessel of honor, Singular vessel of devotion, Mystical rose, Tower of David, Tower of ivory, House of gold, Ark of the covenant, Gate of heaven, Morning star, Health of the sick, Refuge of sinners, Solace of migrants, Comfort of the afflicted, Help of Christians, Queen of Angels, Queen of Patriarchs, Queen of Prophets, Queen of Apostles, Queen of Martyrs, Queen of Confessors, Queen of Virgins, Queen of all Saints, Queen conceived without original sin, Queen assumed into Heaven, Queen of the most Holy Rosary, Queen of families, Queen of peace.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

1.1 What is Topology Optimization?

Structural optimization is a method used to answer the question "what is the best structure for a task"? There are several ways to accomplish this, namely size optimization, SO, and topology optimization [1]. A visual description of the differences between these methods is shown in Figure 1.1. Shape optimization reshapes voids and material in a specified domain, and size optimization chooses how large specific components of a structure should be. By the 1980s, the clearly preferred theoretical method was topology optimization [3], which allows for creation and removal of holes in the design of the structure. However, this was largely limited by computational ability at the time. Topology Optimization provides an opportunity to find very versatile designs that are mathematically optimal. In today's structural optimization landscape, shape optimization, when used, is typically used as a post-processing method to smooth out voxels after beginning with topology optimization [4].

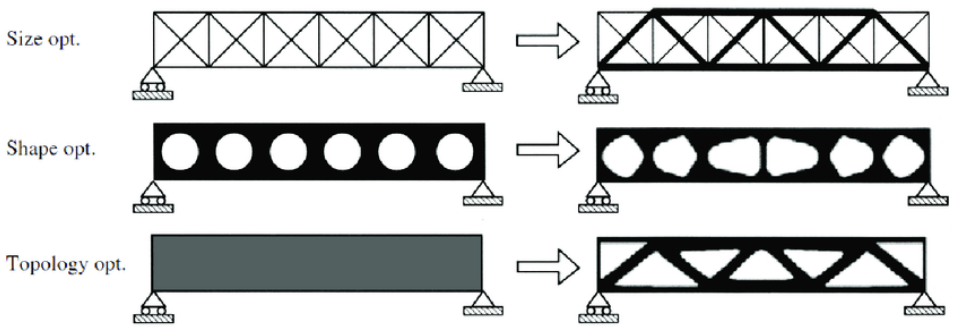


Figure 1.1: Topology optimization compared to other methods of structural optimization. Figure taken from [1].

1.2 History of Topology Optimization

Topology optimization as a class of algorithms was first considered in 1904, but became more well formulated in 1977 as "Optimal Layout Theory" [5], although this was used exclusively on systems of beams. Early implementations were greatly limited by computational load, and through the 1980s, analytical solutions to problems on plates were found, largely with a focus on microstructures [6]. It was not until the late 1980s that computed solutions began to be calculated using finite elements [7]. The results were limited to fairly low resolution 2- or 2.5-dimensional designs.

When this algorithm became more computationally feasible, it was revolutionary to the world of structural engineering. Topology optimization is capable of generating organic-looking shapes, allowing for lightweight and strong structures that can be used in cars, spacecraft, and several other applications. However, manufacturing could not keep up with the designs at the time, limiting the utility of the field.

It was in the years around the turn of the millennia that the utility of topology optimization outside of purely structural design was realized. Topology optimization began to be used in such applications as compliant mechanism design [8], and optimization of fluids in Stokes flow [9]. These generalizations of topology optimization to more multi-physics problems with diverse objectives, even including heat dissipation [10], continues to this day.

In the last decade, 3D-printing and additive manufacturing have allowed manufacturers to make more complex designs more efficiently [11], and topology optimization has shifted focus to be easily utilized for additive manufacturing [12]. With this advancement in manufacturing, the topology optimization field has also grown rapidly.

1.3 General Problem Description

Topology optimization of elastic media can be used to optimize a structural design in many ways. Mass of a structure can be minimized with any number of constraints, such as a constraint on displacement given a load, or even a constraint on the minimum eigenfrequency.

An alternate objective would be to minimize the compliance of the structure, which is a way of measuring weakness. In this case, mass can be used as a constraint. This formulation creates many simple problems commonly used in the literature, and so this formulation will be the primary focus of this thesis.

In a compliance minimization problem, given a domain where the structure could exist, the "design domain", Ω the goal is minimizing the maximum strain placed on a structure by selecting a region E where material is placed. A maximum volume for the design $V_{\max}$ is imposed, as otherwise the strongest design would always be the entire domain filled with material.

A formulation not discussed further in this thesis is stress minimization. This problem is known to be difficult to solve, and more computationally complex than the previously mentioned formulations. This is largely because in some cases, stress can be decreased by removing material, whereas compliance is strictly decreased when material is added. However, the creation of compliance minimization comes, in a way, from a simplification of this stress formulation in the l_∞ norm

Given a vector-valued displacement field $\mathbf{u}$, the stress $\sigma(\mathbf{u})$ can be calculated. In an ideal world, the maximum stress would be minimized, making a structure with no weak points. The stress must offset any internal force $\mathbf{f}$ in the design. This is governed by the linear elasticity equation, which will be further discussed in Section 2.4.

In other words,

$$\begin{aligned} &\text{minimize} \|\sigma(\mathbf{u})\|_\infty \\ &\text{subject to} \|E\| \leq V_{\max}, \\ &\text{and } \nabla \cdot \sigma + \mathbf{f} = 0. \end{aligned} \tag{1.1}$$

A brief discussion of the above equations is now necessary. We want to minimize the maximum value of this stress over the entire structure, denoted E. Boundary conditions related to this equation will be discussed while walking through a derivation of the weak form.

While this formulation poses the question we wanted to ask, using the infinity norm of the strain creates a problem. The maximum strain over the domain as a function of location of material is

necessarily not everywhere differentiable. This can be intuitively seen by considering a case where one facet of the design is successively weakened until it is the "weakest part". This will necessarily instantaneously change how the maximum stress is behaving, and so the derivative will not be able to be defined.

The inherent non-differentiability makes prospects of optimization rather bleak. So instead, we find an approximate solution by optimizing for the strain energy, or compliance. This is a measure of the potential energy stored in an object due to its deformation, but also works as a measure of total stress over the structure. Using $\epsilon = \nabla^s \mathbf{u}$ the symmetric gradient of the displacement field, which is the strain, we have that the strain energy is given by

$$\int_E \frac{1}{2}\sigma : \epsilon \, d\Omega.$$

Re-writing the problem to minimize this strain energy gives

$$\begin{aligned} &\text{Minimize } \int_E \frac{1}{2}\sigma : \epsilon \, d\Omega \\ &\text{subject to } \|E\| \leq V_{\max}, \\ &\text{and } \nabla \cdot \sigma + \mathbf{f} = 0. \end{aligned} \tag{1.2}$$

Note that so far in this formulation, I am not stating what the decision variables are in this problem. The first decision to be made is the choice of using SAND or NAND, as discussed in Section 1.4, and this determines what the set of decision variables will be.

The objective function here, the compliance of the structure, also called the "strain energy", can be greatly simplified using the weak form of the elasticity equation constraint (derived in Section 2.4). By using the displacement itself as the test function, we arrive at

$$\int_\Omega \sigma : \epsilon = \int_\Omega \mathbf{f} \cdot \mathbf{u} + \int_{\partial\Omega} \mathbf{t} \cdot \mathbf{u}, \tag{1.3}$$

where $\mathbf{f}$ is the internal force, and $\mathbf{t}$ is the traction.

Because we are assuming that there are no internal forces, this simplifies to

$$\int_{\Omega} \sigma : \epsilon = \int_{\partial\Omega} \mathbf{t} \cdot \mathbf{u}, \tag{1.4}$$

giving a more condensed and usable problem description of

$$\begin{aligned} &\text{Minimize } \int_{\partial E} \mathbf{t} \cdot \mathbf{u} \\ &\text{subject to } \|E\| \leq V_{\max}, \\ &\text{and } \nabla \cdot \sigma + \mathbf{f} = 0. \end{aligned} \tag{1.5}$$

Many further issues arrive upon solving this problem. The value of the objective function is calculated using a finite element method, where the solution is the displacements. This is placed inside of a nonlinear solver loop that solves for a vector denoting placement of material. If we stick within real-world confines, and allow the material to either be present or not be present in a voxel, then this optimization problem becomes combinatorial, and very expensive to solve. Inequality constraints become necessary, resulting in the use of barriers which require modern strategies to effectively use. These and other considerations are discussed in Chapter 2.

1.4 Introduction to Simultaneous Analysis and Design

In my thesis, I explore the use of Simultaneous ANalysis and Design (SAND) in topology optimization, as well as the use of second-order solvers. In the SAND formulation, the PDE works as a constraint in the optimization problem, and so the PDE is "solved" at the same time the optimization step is taken. This gives one large linear system to solve for each optimization step. By contrast, Nested ANalysis and Design (NAND) formulations solve first only for the PDE constraint, and then perform a sensitivity analysis to gather information for the optimization step. This workflow is shown graphically in Figure 1.2. The larger system in SAND necessarily takes longer to solve than the two smaller systems that would be used in a second order NAND method combined. However, it increases the algorithm's ability to move throughout the decision space.

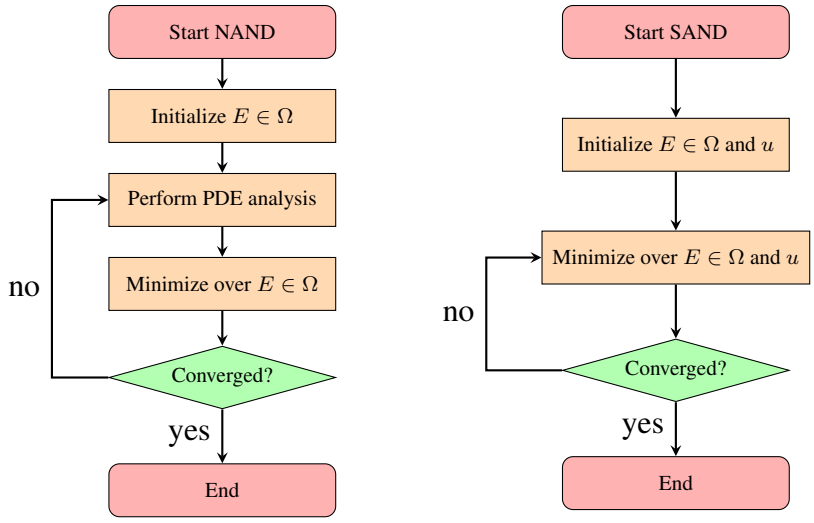
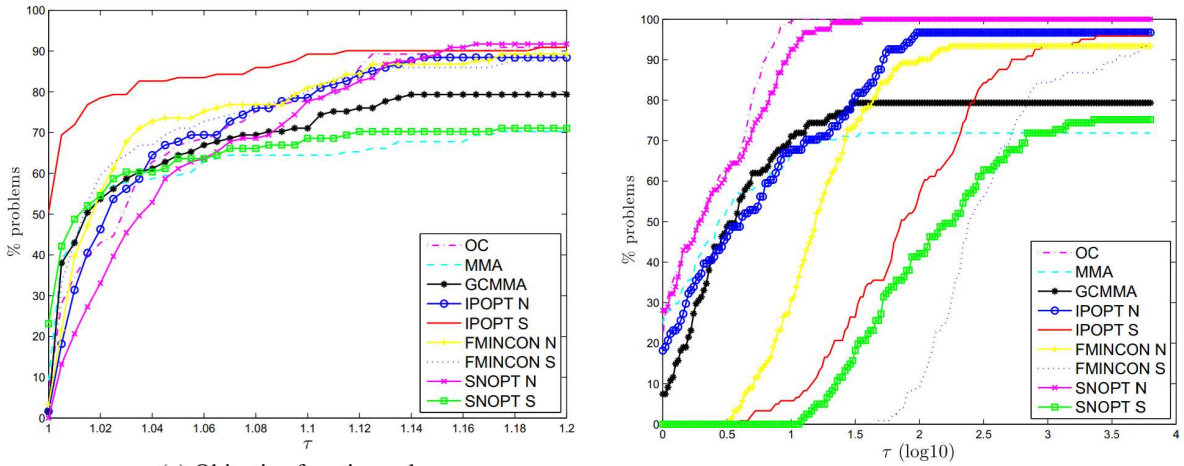


Figure 1.2: This flowchart describes the difference between the NAND technique and the SAND technique that is discussed in this thesis.

There is ample evidence in the literature going back to the mid 1990s to show that SAND formulations, especially combined with interior point solvers, generate designs with improved objective values compared to more commonly used solution methods [2, 13, 14]. The same research, however, also shows the great computational cost of using these second-order SAND methods, and also demonstrates issues solving even relatively simple problems within a reasonable amount of time. Results from [2] are shown in Figure 1.3.

The precise reason for these improved function values is unknown. However, a heuristic explanation exists that is straightforward – topology optimization problems have many local minima that are difficult to avoid, as demonstrated in Figure 1.4. SAND makes the solution space much larger – our solution can now move not only through the space of densities but also through the space of displacements. This allows for our trial solution at each step to move more freely through the solution space, and so to be more able to move past points that would be minima in the smaller solution space. This intuitively pairs very well with interior point optimization, which is discussed in more detail in Section 2.3. In interior point optimization, many local minima can be avoided while the barrier coefficient is large. The large barrier makes all but very large slopes unconsequential compared to the slope of the barrier, and so many local minima will be able to be moved



(a) Objective function value

(d) Computational time

Figure 1.3: A figure from Rojas-Labanda and Stolpe [2] showing interior point optimization with SAND method giving the best objective values most of the time, but taking up to 1,000 times longer to solve a problem when compared to other methods. Here methods marked "N" are NAND methods, and "S" denotes a SAND method.

over. As the barrier is decreased, the deepest minima will be able to be fallen into first, and so these formulations tend to hit the global optimum more often.

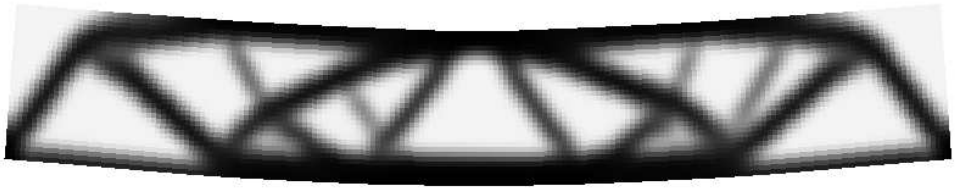


Figure 1.4: An asymmetric solution of a topology optimization problem. Each side of this design is a separate local minimum.

This thesis looks at ways to speed up the SAND so that it can be a feasible method for large problems. SAND, and especially SAND formulations that use interior point methods, are currently not often studied due to its lack of computational feasibility. In fact, NAND will always be the more computationally inexpensive option. As such, modern research into SAND has been looking into different formulations, as well as its use in smaller scale problems. These use pre-build optimization packages, and do not themselves work to improve the speed of the underlying algorithms. In this thesis, I apply modern optimization techniques to an interior point Newton-Raphson method to

give fast second-order convergence. I also develop a multi-layered preconditioner that allows for fast solves of the resulting linear problem. All of these processes are created in a way that allows for use of a distributed memory parallel system. I show that the SAND method with compliance minimization, which has been shown to typically give better objective values than traditional first order algorithms, is feasible in a modern computational landscape, even when solving problems in three space dimensions.

1.5 Test Problems

The most common application in topology optimization is the optimal design of a load-bearing elastic media. This is set up in one of two ways – either the maximum allowed compliance is given, and the total volume is minimized, or the amount of volume is given and the allowed compliance is minimized. In this thesis, I use the compliance minimization approach, although most of the techniques would be equally applicable to volume minimization.

Topology optimization of elastic media, and specifically compliance minimization, has found a home with usage by engineers of automobiles, planes, and spacecraft. In particular, the motorcycle Shown in Figure 1.5 was created by APWorks to show evidence of the usefulness of these techniques.

The algorithms explored in the first several chapters of this thesis are tested against a traditional topology optimization problem called the MBB Beam. This problem, originally considered in a collection of examples for an optimization system called CAOS [15], considers the optimal 2-d structure that can be built on a rectangle six units wide, and one unit tall. The bottom corners are fixed in place in the y direction using homogenous Dirichlet boundary conditions, and a downward force is applied in the center of the top of the beam by enforcing a Neumann boundary condition. The rest of the boundary is allowed to move, and has no external force applied, taking the form of a homogenous Neumann boundary condition. While the total volume of the domain is 6, 3 units of material are allowed for the structure. Because of the symmetry of the problem, it can be posed on a rectangle of width three and height one by cutting the original domain in half, and

Figure 1.5: A bicycle designed using topology optimization by APWorks and then 3D-printed – image from APWorks.

using homogenous Dirichlet boundary conditions in the x direction along the cut edge, as shown in Figure 1.6. However, the expected symmetry was an effective tool in debugging, and so I consider the full symmetric problem.

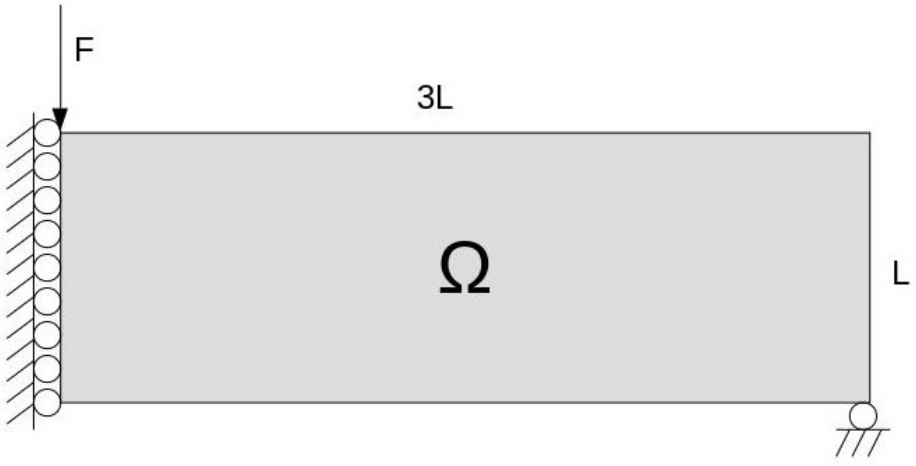


Figure 1.6: The MBB problem domain and boundary conditions.

In addition to this problem, I also explore a cantilever. The cantilever has height 1 and width 3, and is attached to a support at the top and bottom left corners. A downward force is applied to the bottom of the right side. Again, I allow half of the volume to be filled with material and minimize the compliance. This problem geometry is shown in Figure 1.7

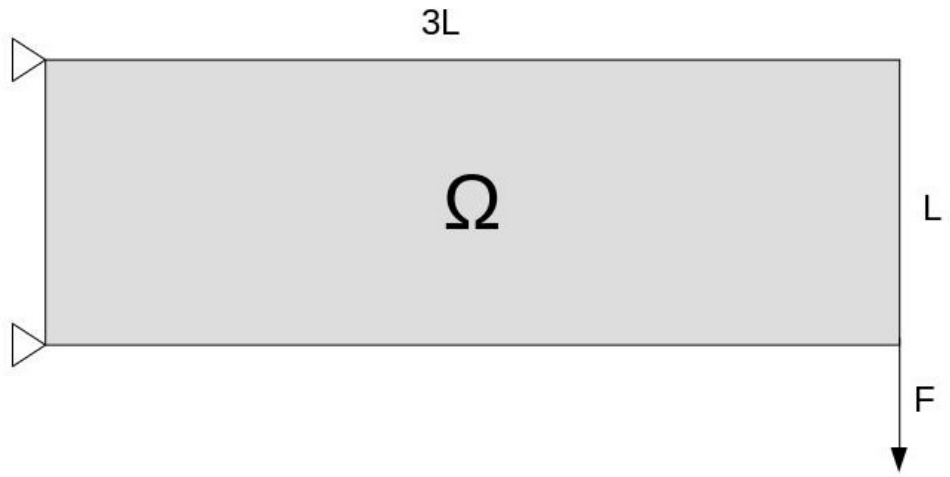


Figure 1.7: The Cantilever problem domain and boundary conditions.

With test problems established, I now continue this thesis with a more thorough theoretical description and analysis of the problem in Chapter 2. This includes building a formulation of the problem that is usable in the SAND context, and that will converge quickly. I discuss the formulation of the interior point optimization problem with a comparison of modern techniques used for these types of problems. For several aspects of the interior point formulation, I investigate the benefits of different methods before choosing one for my algorithm.

After introducing this interior point formulation, I look at numerical solution methods for my algorithm in Chapter 3. This includes modern globalization techniques, parallelization methods as well as linear solver selection, and the development of preconditioners for this problem.

The goal of this thesis is to take a combination of these algorithms and to combine them into one working algorithm that can be used to solve topology optimization problems of this type. While most of the techniques used have been well developed already in the literature, a combination of them for SAND topology optimization has not been discussed in literature to this point.

Chapter 2

Problem Formulation

Topology optimization of elastic media is an optimization problem that takes the form of

$$\begin{aligned} &\text{Minimize } \int_{\partial E} \mathbf{t} \cdot \mathbf{u} \\ &\text{subject to } \|E\| \leq V_{\max}, \\ &\text{and } \nabla \cdot \sigma + \mathbf{f} = 0, \end{aligned} \tag{2.1}$$

as shown in Section 1.3. However, this problem is combinatorially defined over an infinite dimensional space, and is not solvable as written.

Further, this thesis explores an "Interior Point Method" inside a "Simultaneous Analysis and Design" framework. While these are algorithms that have already been used and found to be good at generating good optimal designs, many details of these methods are not strictly defined. As such, there are many considerations I must take to move towards a computationally feasible interior point SAND topology optimization solver.

In this chapter, I explore modern theoretical techniques for solving topology optimization problems within these frameworks, as well as nonlinear, nonconvex optimization problems in general. In each section, I explore various methods for addressing some particular need, and select one for use in my implementation.

This chapter will work towards formulating a minimization problem that is computationally solvable. In each section, I discuss an issue with the formulation discussed up to that point, present possible solutions, and then select one of these solutions to use in my algorithm. By the end of this chapter, I will present an interior-point minimization problem that can be solved using a Newton-Raphson scheme.

2.1 Avoiding Combinatorial Optimization

The goal of topology optimization is to divide a domain into two parts – one that has material, and one that is void. However, even after discretization of the domain into voxels, there are simply too many disjoint possibilities to navigate finding an optimum.

Methods developed to avoid this challenge are based on allowing material to exist in a state with a "density" between 0 and 1. A density of 0 suggests the material is not there, and is therefore not a part of the structure, while a density of 1 suggests the material is present. Values between 0 and 1 do not necessarily reflect real-world phenomena (although some researchers have demonstrated that microstructures can be produced to mimic this density), but allow us to turn the combinatorial problem into a continuous one. I then look at density values ρ, with the constraint that $\rho_{\text{min}} \leq \rho \leq 1$.

In some NAND formulations of this problem – specifically those that do not use interior point methods – it is required that $0 < \rho_{\text{min}}$. Conversely, in formulations that use SAND or interior point methods (or both), we can use $\rho_{\text{min}} = 0$. For interior point methods, this occurs naturally because inequality constraints cannot be perfectly reached, and so the density cannot be made to be 0. In SAND, the elasticity equation is used as a constraint, and so the 0 density problem just leads to a constraint that is not full rank, which can be handled in a number of ways. When required, the minimum value ρ_{min} is typically chosen to be around 10^{-3} [1]. This value keeps the solver from the possibility of having an "infinite compliance" caused by a material with no stiffness, while still being small enough to provide accurate results.

The straightforward application of the effect of this "density" on the elasticity of the media would be to simply multiply the stiffness tensor C of the media by the given density, that is, the $C = \rho C_0$. However, this approach often gives optimal solutions where density values are far from both 0 and 1. As I want to find a real-world solution, where material either is present or it is not, a penalty is applied to these "in-between" values.

A simple and effective way to do this is to multiply the stiffness tensor by the density raised to some integer power penalty parameter p, so that $C = \rho^p C_0$. This makes density values farther away

from 0 or 1 less effective. Numerical experiments have shown that using $p = 3$ is a sufficiently high penalty parameter to create 'black-and-white' solutions [1]. This method is called the Solid Isotropic Material with Penalization (SIMP) interpolation, and was first discussed in [16], where the in-between values were shown to correspond to real-life micro-structures .

Using this density also allows me to re-frame the volume constraint on the optimization problem. Use of SIMP then turns the optimization problem into the following:

$$\begin{aligned} \text{Minimize } & \int_{\partial\Omega} \mathbf{t} \cdot \mathbf{u} \\ \text{Subject to } & \int_{\Omega} \rho \, d\Omega \leq V_{\max}, \\ & 0 \leq \rho \leq 1, \\ \text{and } & \nabla \cdot \sigma(\rho) + \mathbf{f} = 0. \end{aligned} \tag{2.2}$$

2.2 Density Filter Methods

The next issue I consider is that solutions to topology optimization problems are typically ill-posed. This can be easily seen by a lack of mesh-independence when numerically solved. As the mesh gains resolution, the optimal solution typically develops smaller and smaller structures. Optimal structures are often fractal, and so cannot be displayed with a finite number of voxels.

There are a few competing work-arounds to this issue, but the most popular for first order optimization is the sensitivity filter, while second order optimization methods tend to prefer use of a density filter. The density filter was first introduce in 2000 by Bourdin in [17] as an alternative to much more elaborate schemes used at the time, such as perimeter limiting [18], or by constraining the gradient of the density [19].

These techniques all have the additional purpose of solving another typical issue in topology optimization, checkerboarding. This issue is shown in Figure 2.1. In many cases, our algorithm gives far too low a compliance to a checkerboard pattern in a space where checkerboards are not a physically feasible solution. An explanation for the phenomenon is found in [20].

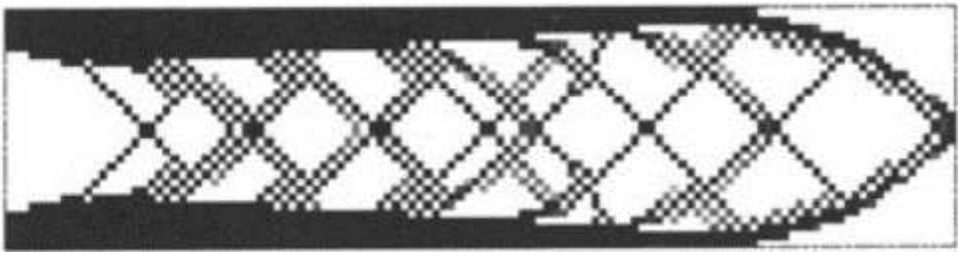


Figure 2.1: A solution to a cantilever problem showing checkerboarding

As the filters affect the gradient and Hessian of the compliance, the choice of filter has an effect on the solution of the problem. These filters are explained below.

2.2.1 Sensitivity Filter

The sensitivity filter is widely used in first-order methods for topology optimization. As such, it is not used in my algorithm. However, due to its common use in other algorithms today, I will cover it in some detail.

The sensitivity filter is a method of achieving mesh independence by applying a low-pass filter to the density derivatives of the compliance function. This cannot be directly expanded to work in second order methods, so it is exclusively used in first order methods. Given the density derivatives of the compliance, C, the sensitivity filter gives new derivatives by

$$\widetilde{\frac{\partial C}{\partial \rho_i}}(\mathbf{x}) = \int_\Omega \frac{\partial C}{\partial \rho_i}(\mathbf{y}) Z(\mathbf{x}-\mathbf{y}) d\mathbf{y}. \tag{2.3}$$

Here, and in future uses, the tilde symbol is used to express a "smoothed" version of the original function, so here $\widetilde{\frac{\partial C}{\partial \rho}}(\mathbf{x})$ is the smoothed partial Gateaux derivative of the compliance functional with respect to the density function.

My convolution function, denoted here by Z, is first determined by the non-normalized convolution H,

$$\text{where } H(\mathbf{r}) = \begin{cases} r_{\max} - \|\mathbf{r}\|, & \|\mathbf{r}\| < r_{\max} \\ 0, & \text{else} \end{cases} \tag{2.4}$$

for some selected filter radius $r_{\max}$.

The normalized convolution kernel Z ensures that any location is actually a weighted average of its neighbors, and is then defined as

$$Z(\mathbf{x}) = \frac{H(\mathbf{x})}{\int_\Omega H(\mathbf{x}-\mathbf{y})d\mathbf{y}}. \tag{2.5}$$

This convolution of the partial derivatives with regards to the cells prevents rapid fluctuations between neighboring cells. Heuristically, by forcing all steps in the optimization loop to be in some sense "smooth", the final solution should also be "smooth", preventing fractal structures and checkerboarding.

2.2.2 Density Filter

Another option that provides mesh independence is placing a filter on the densities. In this formulation, the objective function compliance is calculated as a function of filtered densities, $\tilde{\rho}$. Filtered densities are a convolution of the original design variable, ρ. This guarantees the density solution, $\tilde{\rho}$ will be filtered, and will not have high frequency variations. The convolution is similar to the sensitivity filter

$$\tilde{\rho}(\mathbf{x}) = \int_\Omega \rho(\mathbf{x}) Z(\mathbf{x}-\mathbf{y})d\mathbf{y}. \tag{2.6}$$

The convolution term Z is the same as given in Equation (2.5).

This convolution can be simply executed to make use of both a "filtered" and an "unfiltered" density variable. This becomes a constraint later, and is therefore possible to implement in second-order methods that require the Hessian, as opposed to derivative-effecting filters.

This density filter method is sometimes referred to as a 2-field density approach. By adding a third density that applies a heaviside function to the filtered density, this formulation becomes essentially equivalent to levelset methods, although filtering techniques look slightly different. All of these methods have been shown to give very similar results [21], and are outside the scope of my work here.

Other options do exist for these filters, such as using a modified Helmholtz equation as a filter [22]. This filtering method has the benefit of being much more sparse, and having known preconditioning methods. However, it is a much more heuristic method, and has less common use, and so I use the more well-known and tested approach of the density filter.

The filtered density is used in computing the compliance strain, in the elasticity equation, as well as in the volume constraint. The density constraint – ensuring that all densities are between some ρ_{min} and 1 – is applied to the unfiltered density. This ensures that no null-space will be added to our system comprised of wildly oscillating unfiltered densities which result in negligible filtered density values.

For ease of notation, I define the filter function

$$\tilde{\rho}(\mathbf{x}) = F(\rho) = \int_{\Omega} \rho(\mathbf{x}) Z(\mathbf{x} - \mathbf{y}) d\mathbf{y}. \tag{2.7}$$

This addition to the problem further expands our formulated optimization problem:

$$\begin{aligned} \text{Minimize } & \int_{\partial\Omega} \mathbf{t} \cdot \mathbf{u} \\ \text{Subject to } & \int_{\Omega} \tilde{\rho}\, d\Omega \leq V_{\max}, \\ & \tilde{\rho}(\mathbf{x}) = F(\rho), \\ & \rho_{\min} \leq \rho \leq 1, \\ & \text{and } \nabla \cdot \sigma(\tilde{\rho}) + \mathbf{f} = 0. \end{aligned} \tag{2.8}$$

2.3 Interior Point Methods

Our problem is further complicated by the inequality constraints bounding the density variable. An approach commonly used to handle these constraints that has been shown to find good objective values is interior point optimization. I begin by replacing the inequalities $\rho_{\min} \leq \rho$ and $\rho \leq 1$ with the slack variables and inequalities

$$\begin{aligned} s_1 &= \rho - \rho_{\text{min}}, \\ s_2 &= 1 - \rho, \\ s_1, s_2 &> 0. \end{aligned} \tag{2.9}$$

This does change the inequality to a strict one, but this does not create an issue as iterative methods will not give an exact solution anyway. This strict inequality keeps the solution strictly on the interior of the domain, and is necessary because of the modification I make to the objective function.

The crux of an interior point method is creating an infinite barrier to ensure we stay on the interior of the feasible domain. Using barriers created by logarithms, the objective function is modified to be

$$\text{Minimize } \int_{\partial\Omega} \mathbf{t} \cdot \mathbf{u} + \int_{\Omega} \alpha(\log(s_1) + \log(s_2)). \tag{2.10}$$

The added terms are referred to as the log barrier, and the positive value α is the barrier parameter. The objective is not defined when the slack variables are 0, showing again why the change to a strict inequality is necessary. As the barrier parameter approaches zero, the new objective function approaches the original. This new objective function ensures the inequalities on the slack variables are met, and give a somewhat smooth addition to the interior of the domain. However, derivatives of this function are now large near the barrier. This is especially problematic because the solution will be necessarily close to these barriers. Algorithms for decreasing this barrier parameter in a way that allows for fast convergence towards 0, while also maintaining well-conditioned subproblems, are discussed in Section 3.3. While the advantage of Newton's method is that it is a second order scheme, and in unconstrained contexts converges quadratically, log barrier methods achieve superlinear, but subquadratic convergence.

In addition to the logarithmic barrier method, inverse barrier methods have also been used, where an asymptote is created by adding a function of the form $\sum_i \frac{1}{c_i(x)}$ for the constraints $c_i(x) \geq 0$ to the objective function. This approach is generally considered worse for a couple of reasons. First, in practice, the second derivative of this function is sharper, and so makes the Newton step

less stable close to the boundary. Secondly, this method misses a big advantage of the log barrier method theoretically, which is its alignment with the Karuch-Khan-Tucker (KKT) conditions. The log barrier method has been used for linear optimization problems since the 1950s, but was first introduced for nonlinear optimization in [23].

Rephrasing our problem using a log barrier to have a strictly interior point problem gives the problem to solve as

$$\begin{aligned} \text{Minimize } \int_{\partial\Omega} \mathbf{t}\cdot\mathbf{u} - \alpha\int_{\Omega}\log(s_1) - \alpha\int_{\Omega}\log(s_2) \\ \text{Subject to } \int_{\Omega}\tilde{\rho}\,d\Omega \leq V_{\text{max}}, \\ \tilde{\rho}(\mathbf{x}) = F(\rho), \\ s_1 = \rho - \rho_{\text{min}}, \\ s_2 = 1-\rho, \\ \text{and } \nabla\cdot\sigma(\tilde{\rho}) + \mathbf{f} = 0. \end{aligned} \tag{2.11}$$

The final constraint, the elasticity equation, gives a method for relating the stress σ and strain ϵ to the filtered density $\tilde{\rho}$.

2.4 Elasticity Equation

The linear elasticity equation is typically formulated as

$$\nabla\cdot\sigma + \mathbf{f} = \rho_m\frac{\partial^2\mathbf{u}}{\partial t^2}. \tag{2.12}$$

Here, ρ_m is the mass density, and σ denotes the Cauchy Stress tensor, given by

$$\sigma = C : \varepsilon. \tag{2.13}$$

Here, C is the stiffness tensor, and ε is the strain, given by the symmetric gradient of the displacement, or

$$\varepsilon = \frac{1}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^T). \tag{2.14}$$

Therefore, in terms of displacement, $\mathbf{u}$, the elasticity equation becomes

$$\nabla \cdot (C : (\frac{1}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^T))) + \mathbf{f} = \rho_m \frac{\partial^2 \mathbf{u}}{\partial t^2}. \tag{2.15}$$

Compliance Minimization solves for optimal structures under a constant load, leaving a steady state displacement, and so the right hand side of this problem is necessarily 0. Replacing the stiffness tensor with the parameter described in the SIMP section gives

$$\nabla \cdot (\tilde{\rho}^p C_0 : \frac{1}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^T)) = -\mathbf{f}. \tag{2.16}$$

After multiplying by a test function $\mathbf{v}$, and integrating by parts, the weak form of this equation is given by

$$\int_\Omega (\nabla \mathbf{v}) : \left(\tilde{\rho}^p C_0 : \frac{1}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^T)\right) - \int_\Gamma \mathbf{v} \cdot \left(\tilde{\rho}^p C_0 : \frac{1}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^T)\right) \mathbf{n} = \int_\Omega \mathbf{f} \cdot \mathbf{v} \tag{2.17}$$

for all $\mathbf{v} \in H^1$. Defining the traction for Neumann boundary conditions as $\mathbf{t} = \left(\rho C_0 : \frac{1}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^T)\right) \cdot \mathbf{n}$ gives

$$\int_\Omega (\nabla \mathbf{v}) : \left(\tilde{\rho}^p C_0 : \frac{1}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^T)\right) = \int_\Omega \mathbf{f} \cdot \mathbf{v} + \int_\Gamma \mathbf{t} \cdot \mathbf{v} \ \ \forall \mathbf{v} \in H^1. \tag{2.18}$$

Further, given that C_0 must be a tensor that maps symmetric matrices to symmetric matrices, we can further simplify to the symmetric

$$\int_\Omega \frac{1}{2}(\nabla \mathbf{v} + (\nabla \mathbf{v})^T) : \left(\tilde{\rho}^p C_0 : \frac{1}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^T)\right) = \int_\Omega \mathbf{f} \cdot \mathbf{v} + \int_\Gamma \mathbf{t} \cdot \mathbf{v} \ \ \forall \mathbf{v} \in H^1. \tag{2.19}$$

Assuming an isotropic material, the stiffness tensor can be given using the Lamé parameters (in Einstein notation) as

$$(C_0)_{i,j,k,l} = \lambda\delta_{i,j}\delta_{k,l} + \mu(\delta_{i,k}\delta_{j,l} + \delta_{i,l}\delta_{j,k}). \tag{2.20}$$

This gives

$$\int_\Omega \partial_i \mathbf{v}_j(\tilde{\rho}^p\lambda\delta_{i,j}\delta_{k,l} + \tilde{\rho}^p\mu(\delta_{i,k}\delta_{j,l} + \delta_{i,l}\delta_{j,k}))(\partial_k \mathbf{u}_l + \partial_l \mathbf{u}_k) = \int_\Omega \mathbf{f}_i\mathbf{v}_i + \int_\Gamma \mathbf{t}_i\mathbf{v}_i \;\; \forall \mathbf{v} \in H^1 \tag{2.21}$$

$$\begin{aligned}\frac{\mu}{2}\int_\Omega \tilde{\rho}^p(\partial_i \mathbf{v}_j + \partial_j \mathbf{v}_i)(\partial_k \mathbf{u}_l + \partial_l \mathbf{u}_k)(\delta_{i,k}\delta_{j,l} + \delta_{i,l}\delta_{j,k}) + \lambda\int_\Omega \tilde{\rho}^p(\partial_i \mathbf{v}_i)(\partial_k \mathbf{u}_k) \\ = \int_\Omega \mathbf{f}_i\mathbf{v}_i + \int_\Gamma \mathbf{t}_i\mathbf{v}_i\end{aligned} \tag{2.22}$$

How this equation is used is the crux of the difference between SAND and NAND formulations. In NAND, the above equation is solved as is, using previously found densities. Sensitivity analysis is performed with the solution for the optimization loop. In SAND, the test function is actually a Lagrange multiplier controlling the equality constraint.

Additive Manufacturing, in whatever form used (especially Fused Deposition Modelling), often creates materials that have orthotropic elastic properties. Using these materials, the simplification of these equations is much more complex, and requires six parameters instead of the typically used two Lamé parameters. This would lead to a slightly more complex elasticity equation, which is outside the scope of this Thesis. The additional complications would take effect after Equation (2.19), and everything up to that point would still be valid for any material property. Once discretized, an orthotropic material would still form a sparse matrix that is symmetric and positive-definite, and therefore the results of this thesis starting in chapter 3 would still hold.

2.5 Final Primal Problem Formulation

With a theoretical framework applied, I can now present a fully formulated version of the topology optimization problem. There are five functions involved in the primal optimization problem, namely

1. filtered density, $\tilde{\rho}$
2. displacement, $\mathbf{u}$
3. unfiltered density, ρ
4. lower density slack variable, s_1
5. upper density slack variable, s_2

Using this notation, and the formulations presented so far in this chapter, I now have the final formulation of

$$\begin{aligned} \text{Minimize } & \int_{\partial\Omega} \mathbf{t}\cdot\mathbf{u} - \int_{\Omega} \alpha(\log(s_1)+\log(s_2)) \\ \text{subject to } & \int_{\Omega} \tilde{\rho}\, d\Omega \leq V_{\max}, \\ & \tilde{\rho} = F(\rho), \\ & \rho - \rho_{\min} = s_1, \\ & 1-\rho = s_2, \\ & \text{and } \nabla\cdot\sigma(\tilde{\rho}) + \mathbf{f} = 0. \end{aligned} \tag{2.23}$$

The formulation of this minimization problem consists of many separate known algorithms that all work well together and are well suited to this application.

There are many ways to take this problem and to then find the minimizing design. In large problems like this one, first order methods using a gradient descent technique would typically be used, as second order methods are often large and unwieldy, and are computationally expensive. However, interior point methods are often successfully used with second order methods [24], and we will be able to compute the second variational derivatives by hand, meaning the construction of the Hessian will be computationally straight-forward, and the Hessian itself will be sparse. With

these advantages at play, I decided to use a second order, primal-dual Newton-Raphson method for finding the minimum.

Chapter 3

Numerical Methods

The goal of this thesis is to demonstrate computational feasibility of using interior point methods to solve the simultaneous analysis and design formulation of topology optimization problems.

The minimization problem formulated over the previous chapter gives a starting place for solving my topology optimization problem. In this chapter, I discuss how I find a minimizing design. This process begins with a second order scheme based on a Newton-Raphson method. Once this is made, however, the problem is infinite-dimensional. Newton-Raphson schemes are notoriously non-global, and I have yet to discuss any method of working with the barrier parameter α, besides to say it should start with a moderate value and decrease to zero. In this chapter, I explore the computational and algorithmic formulation I use to solve this problem. This will include difficulties of implementing the nonlinear solver, working with not only these stated issues, but also with concerns about speed of solving the resulting linear problem.

Topology optimization, and structural optimization in general, is somewhat unique in the world of optimization, as it largely continues to rely on first-order methods. This is due to the necessarily large nature of the problem that easily reaches millions of decision variables, is not well-defined in its unmodified formulation, and is numerically unstable. Benefits of first order algorithms include a very fast solve of each optimization subproblem they look at. Using only first derivative information of the decision variables, these methods use relatively little memory and require a linear solve only when calculating the solution to the related partial differential equation (PDE) constraint. Second order methods – those that require a (at least approximated) Hessian – tend to solve a more robust range of optimization problems using fewer subproblems, and giving a better objective value [2]. However, the large size of these linear problems can create issues both with the computational power needed to solve them in a reasonable amount of time, and the memory needed to store the problem.

As mentioned in the introduction, having this many variables results in a large system, and this is typically minimized by use of first-order methods. However, in almost all other modern optimization problems, second order methods are preferred. In fact, as discussed in the introduction, second order methods have been shown to give better solutions to topology optimization algorithms [2]. As such, I will be using a Newton-Raphson algorithm for the solution of this problem.

3.1 Newton-Raphson Method

The Newton-Raphson method, or Newton's method, is an iterative scheme used to solve nonlinear equations. Given a nonlinear, vector-valued equation $F(\mathbf{x}) = 0$, I can take a previous guess of a solution, and use Newton's method to to improve the guess. A naive version of Newton's method would be to update an old guess $\mathbf{x}_{n-1}$ to a new one $\mathbf{x}_n$ using

$$\mathbf{x}_n = \mathbf{x}_{n-1} - \left(\nabla F(\mathbf{x}_{n-1})\right)^{-1} F(\mathbf{x}_{n-1}). \tag{3.1}$$

This will need to be modified to become a global algorithm using techniques discussed in Sections 3.4 and 3.5.

In the case of an optimization problem, the function we want to find a solution to is to make the gradient of the Lagrangian $\mathcal{L}$ equal to 0. This means the (still naive) Newton step becomes

$$\mathbf{x}_n = \mathbf{x}_{n-1} - \left(H(\mathbf{x}_{n-1})\right)^{-1} \nabla \mathcal{L}(\mathbf{x}_{n-1}), \tag{3.2}$$

where H is the Hessian of the Lagrangian. The first step, then, in formulating a Newton step is to formulate the Lagrangian.

For reference, the formulated optimization problem is the following:

$$\begin{aligned}
\text{Minimize } & \int_{\partial\Omega} \mathbf{t}\cdot\mathbf{u} - \int_{\Omega} \alpha(\log(s_1)+\log(s_2)) \\
\text{subject to } & \int_{\Omega} \tilde{\rho}\, d\Omega \leq V_{\text{max}}, \\
& \tilde{\rho} = F(\rho), \\
& \rho - \rho_{\text{min}} = s_1, \\
& 1-\rho = s_2, \\
& \text{and } \nabla\cdot\sigma(\tilde{\rho}) + \mathbf{t} = 0.
\end{aligned} \tag{3.3}$$

There are two equivalent ways of defining optimality conditions for this problem – I can use the KKT conditions or set the gradient of the Lagrangian equal to zero. This equivalency is part of the rationale for utilizing logarithmic barriers instead of other barrier functions as seen in Section 2.3. I will then use the Newton-Raphson method to find solutions to these conditions.

The formulation of a Lagrangian requires several Lagrange multipliers. I define z_1 and z_2 to be the Lagrange multipliers for my slack variables s_1, s_2, respectively. y_1 is the Lagrange multiplier for the elasticity constraint, applied as described in Section 2.4. The Lagrange multiplier for the density filter constraint is y_2, and y_3 corresponds to the total volume.

$$\begin{aligned}
\mathcal{L} = & \int_{\partial\Omega} \mathbf{u}\cdot\mathbf{t} - \alpha\int_{\Omega} \log(s_1)+\log(s_2) - \int_{\Omega} y_2(F(\rho)-\tilde{\rho}) \\
& - \int_{\Omega} \tilde{\rho}^p \left(2\mu\left(\varepsilon(\mathbf{y_1}) : \varepsilon(\mathbf{u})\right) + \lambda\left(\nabla\cdot\mathbf{u}\nabla\cdot\mathbf{y_1}\right)\right) + \int_{\Gamma} \mathbf{y_1}\cdot\mathbf{t} \\
& - \int_{\Omega} z_1(\rho - s_1) - \int_{\Omega} z_2(1-\rho-s_2) - y_3\left(V - \int_{\Omega}\tilde{\rho}\right).
\end{aligned} \tag{3.4}$$

In the following formulation, $d_{\{\cdot\}}$ is a test function that is naturally paired with the $\{\cdot\}$ function. In terms of the Lagrangian, this $d_{\{\cdot\}}$ is representative of the direction of the Gateaux derivative of the $\{\cdot\}$ function.

The KKT conditions can be broken up into several parts that must be achieved to reach a possible optimizer. I divide the necessary conditions found from the Gateaux derivatives of the Lagrangian into these categories below.

1. Stationarity – these equations ensure we are at a critical point of the objective function when constrained.

$$\int_\Omega d_{\tilde{\rho}} y_2 - p\tilde{\rho}^{p-1} d_{\tilde{\rho}} \left(2\mu\left(\varepsilon(\mathbf{y_1}) : \varepsilon(\mathbf{u})\right) + \lambda\left(\nabla \cdot \mathbf{u} \nabla \cdot \mathbf{y_1}\right)\right) + y_3 \int_\Omega d_{\tilde{\rho}} = 0, \tag{3.5}$$

$$\int_\Gamma \mathbf{d_u} \cdot \mathbf{t} - \int_\Omega \tilde{\rho}^p \left(2\mu\left(\varepsilon(\mathbf{y_1}) : \varepsilon(\mathbf{d_u})\right) + \lambda\left(\nabla \cdot \mathbf{d_u} \nabla \cdot \mathbf{y_1}\right)\right) = 0, \tag{3.6}$$

$$\int_\Omega -d_\rho z_1 + d_\rho z_2 - F(d_\rho) y_2 = 0. \tag{3.7}$$

2. Primal Feasibility – these equations ensure the equality constraints are met.

$$-\int_\Omega \tilde{\rho}^p \left(2\mu\left(\varepsilon(\mathbf{d_{y_1}}) : \varepsilon(\mathbf{u})\right) + \lambda\left(\nabla \cdot \mathbf{u} \nabla \cdot \mathbf{d_{y_1}}\right)\right) + \int_\Gamma \mathbf{t} \cdot \mathbf{d_{y_1}} = 0, \tag{3.8}$$

$$\int_\Omega -(\rho - s_1) d_{z_1} = 0, \tag{3.9}$$

$$\int_\Omega -(1 - \rho - s_2) d_{z_2} = 0, \tag{3.10}$$

$$\int_\Omega -(F(\rho) - \tilde{\rho}) d_{y_2} = 0, \tag{3.11}$$

$$\left(V - \int_\Omega \tilde{\rho}\right) d_{y_3} = 0. \tag{3.12}$$

3. Complementary Slackness – these equations essentially ensure the barrier is met – in the exact solution, we would require $s^T z = 0$.

$$\int_\Omega \left(-\frac{\alpha}{s_1} + z_1\right) d_{s_1} = 0, \tag{3.13}$$

$$\int_\Omega \left(-\frac{\alpha}{s_2} + z_2\right) d_{s_2} = 0. \tag{3.14}$$

4. Dual Feasibility – Multiplier on slacks and slack variables must be kept greater than 0.

$$s_1, s_2, z_1, z_2 \geq 0. \tag{3.15}$$

Newton's method is used to then find the appropriate zeros. Considering Newton's method in Equation (3.2), we can formulate an equation for the Newton step,

$$\Delta \mathbf{x} = -H(\mathbf{x}_{n-1})^{-1} \nabla \mathcal{L}(\mathbf{x}_{n-1}). \tag{3.16}$$

Below, $c_{\{\cdot\}}$ corresponds to the direction of a Gateaux derivative with respect to the $\{\cdot\}$ function. The combinations of the left-hand sides of each of the following equations then represents the Hessian. This system of equations can be solved for all $d_{\{\cdot\}}$ to find $c_{\{\cdot\}}$, which is then the Newton step for each function $\{\cdot\}$.

1. Stationarity

$$\begin{aligned}\int_\Omega d_{\tilde{\rho}} c_{y_2} - p(p-1)\tilde{\rho}^{p-2} d_{\tilde{\rho}} c_{\tilde{\rho}} \left(2\mu\left(\varepsilon(\mathbf{y_1}) : \varepsilon(\mathbf{u})\right) + \lambda\left(\nabla \cdot \mathbf{u} \nabla \cdot \mathbf{y_1}\right)\right) \\ - p\tilde{\rho}^{p-1} d_{\tilde{\rho}} \left(2\mu\left(\varepsilon(\mathbf{c_{y_1}}) : \varepsilon(\mathbf{u})\right) + \lambda\left(\nabla \cdot \mathbf{u} \nabla \cdot \mathbf{c_{y_1}}\right)\right) \\ - p\tilde{\rho}^{p-1} d_{\tilde{\rho}} \left(2\mu\left(\varepsilon(\mathbf{y_1}) : \varepsilon(\mathbf{d_u})\right) + \lambda\left(\nabla \cdot \mathbf{d_u} \nabla \cdot \mathbf{y_1}\right)\right) + \int_\Omega d_{\tilde{\rho}} \\ = \int_\Omega -d_{\tilde{\rho}} y_2 + p\tilde{\rho}^{p-1} d_{\tilde{\rho}} \left(2\mu\left(\varepsilon(\mathbf{y_1}) : \varepsilon(\mathbf{u})\right) + \lambda\left(\nabla \cdot \mathbf{u} \nabla \cdot \mathbf{y_1}\right)\right) - y_3 \int_\Omega d_{\tilde{\rho}}, \end{aligned} \tag{3.17}$$

$$\begin{aligned} - \int_\Omega p\tilde{\rho}^{p-1} c_{\tilde{\rho}} \left(2\mu\left(\varepsilon(\mathbf{y_1}) : \varepsilon(\mathbf{d_u})\right) + \lambda\left(\nabla \cdot \mathbf{d_u} \nabla \cdot \mathbf{y_1}\right)\right) \\ - \int_\Omega \tilde{\rho}^p \left(2\mu\left(\varepsilon(\mathbf{c_{y_1}}) : \varepsilon(\mathbf{d_u})\right) + \lambda\left(\nabla \cdot \mathbf{d_u} \nabla \cdot \mathbf{c_{y_1}}\right)\right) \\ = \int_\Gamma -\mathbf{d_u} \cdot \mathbf{t} + \int_\Omega \tilde{\rho}^p \left(2\mu\left(\varepsilon(\mathbf{y_1}) : \varepsilon(\mathbf{d_u})\right) + \lambda\left(\nabla \cdot \mathbf{d_u} \nabla \cdot \mathbf{y_1}\right)\right), \end{aligned} \tag{3.18}$$

$$\int_\Omega -d_\rho c_{z_1} \;+\; d_\rho c_{z_2} \;-\; F(d_\rho)c_{y_2} \quad = \quad -\int_\Omega -d_\rho z_1 \;+\; d_\rho z_2 \;-\; F(d_\rho)y_2. \quad (3.19)$$

2. Primal Feasibility

$$\begin{aligned} -\int_\Omega p\tilde{\rho}^{p-1}c_{\tilde{\rho}}\left(2\mu\left(\varepsilon(\mathbf{d_{y_1}}):\varepsilon(\mathbf{u})\right)+\lambda\left(\nabla\cdot\mathbf{u}\nabla\cdot\mathbf{d_{y_1}}\right)\right) \\ -\int_\Omega \tilde{\rho}^p\left(2\mu\left(\varepsilon(\mathbf{d_{y_1}}):\varepsilon(\mathbf{c_u})\right)+\lambda\left(\nabla\cdot\mathbf{c_u}\nabla\cdot\mathbf{d_{y_1}}\right)\right) \\ =\int_\Omega \tilde{\rho}^p\left(2\mu\left(\varepsilon(\mathbf{d_{y_1}}):\varepsilon(\mathbf{u})\right)+\lambda\left(\nabla\cdot\mathbf{u}\nabla\cdot\mathbf{d_{y_1}}\right)\right)-\int_\Gamma \mathbf{t}\cdot\mathbf{d_{y_1}} \quad (3.20) \end{aligned}$$

$$\int_\Omega(-c_\rho+c_{s_1})d_{z_1}=\int_\Omega(\rho-s_1)d_{z_1}, \quad (3.21)$$

$$\int_\Omega(c_\rho+c_{s_2})d_{z_2}=\int_\Omega(1-\rho-s_2)d_{z_2}, \quad (3.22)$$

$$\int_\Omega -(H(c_\rho)-\tilde{c_\rho})d_{y_2}=\int_\Omega(H(\rho)-\tilde{\rho})d_{y_2}, \quad (3.23)$$

$$\int_\Omega c_{\tilde{\rho}}d_{y_3}=-(V-\int_\Omega\tilde{\rho})d_{y_3}. \quad (3.24)$$

3. Complementary Slackness

$$\int_\Omega\left(\frac{\alpha}{s_1^2}c_{s_1}+c_{z_1}\right)d_{s_1}=-\int_\Omega\left(-\frac{\alpha}{s_1}+z_1\right)d_{s_1}, \quad (3.25)$$

$$\int_\Omega\left(\frac{\alpha}{s_2^2}c_{s_2}+c_{z_2}\right)d_{s_2}=-\int_\Omega\left(-\frac{\alpha}{s_2}+z_2\right)d_{s_2}. \quad (3.26)$$

4. Dual Feasibility

$$s_1,s_2,z_1,z_2\geq 0. \quad (3.27)$$

Solving this set of equations gives a Newton step for a given barrier size. This can, at best, allow for near-quadratic convergence, especially when in the asymptotic range – when the approximated solution is close to the actual solution. Challenges of using this technique go beyond solving this large set of problems. Newton methods are notoriously locally effective, but not globally, which

will be especially true in problems like this one that have many local minima. I also require a scheme to bring the barrier value to zero. Globalization of Newton's method can be forced by using either a merit function or a filter. There are many approaches for decreasing the barrier function value, but they can typically be split into monotone or adaptive methods. Newton's methods also struggle with constrained problems due to a phenomenon called the Maratos effect, discussed in Section 3.5. This can be partially mediated through either a second order correction scheme or a watchdog algorithm.

3.2 Discretization with Finite Elements

All nonlinear solvers require discretizing the problem into finite dimensions. For clarity and consistency throughout this section, I begin by describing how all functions I use are discretized.

There are in total 10 functions needed to discretize for any of these algorithms.

1. filtered density, $\tilde{\rho}$,
2. displacement, $\mathbf{u}$,
3. unfiltered density, ρ,
4. displacement multiplier, $\mathbf{y_1}$,
5. unfiltered density multiplier, y_2,
6. lower density slack, s_1,
7. upper density slack, s_2,
8. lower slack multiplier, z_1,
9. upper slack multiplier, z_2,
10. total volume multiplier, y_3.

As I want a solution that is an image, it makes sense to treat each cell as a voxel with regards to the density, and to use piece-wise constant functions to approximate the density. The density does not need to be continuous (and in fact should not be), so the discontinuous nature of these elements aids in finding a solution.

The functions related to density need to be approximated by the same scheme, and so the filtered density, slack variables, unfiltered density multiplier, and slack multipliers are all also discretized by piece-wise constant functions.

Displacement and its multiplier are both functions that are naturally continuous. I use Q_1 bilinear elements to approximate them on each cell.

Finally, a multiplier with regards to the total volume is needed, but as this is a single value, I regard it as a constant function over the entire domain.

The elasticity equation can be discretized into the linear system $A\mathbf{u} = \mathbf{t}$, where A is the matrix formed by the weak form of the elasticity equation using bilinear continuous finite elements. and $\mathbf{t}$ is the corresponding right-hand side.

The filter functions F, defined in Equation (2.7), can be re-written as a matrix with the knowledge that the density function will be discretized as a piece-wise constant function. Given that the center of a cell i can be written as $\mathbf{c_i}$, we have that

$$H_{i,j} = \begin{cases} r_{\text{max}} - \|\mathbf{c_i} - \mathbf{c_j}\|, & \|\mathbf{c_i} - \mathbf{c_j}\| < r_{\text{max}} \\ 0, & else. \end{cases} \tag{3.28}$$

And from this,

$$F_{i,j} = \frac{H_{i,j}}{\sum_k H_{i,k}}. \tag{3.29}$$

Each of the 10 functions must also be discretized in some manner. I here choose to discretize all functions related to density using piecewise constant finite elements, and discretize all functions related to the displacement using vector-valued linear continuous elements. That is, for the density related functions $\tilde{\rho}$, $\rho.s_1$, s_2, z_1, z_2, and y_2, I discretize as Q_0 elements, and the displacement-related functions $\mathbf{u}$ and $\mathbf{y}_1$ as Q_1 elements.

Using this discretization of the density filter, along with a discretization of the functions gives the following matrix equation:

$$\begin{pmatrix}
 & & D_m & & -D_m & & & & & \\
 & & & D_m & D_m & & & & & \\
D_m & & D_1 & & & & & & & \\
 & D_m & & D_2 & & & & & & \\
-D_m & D_m & & & & & F^T & & & \\
 & & & & & A & & C & & \\
 & & & & A & & & E & & \\
 & & & & F & & & -D_m & & \\
 & & & & & C^T & E^T & -D_m & B & \mathbf{m} \\
 & & & & & & & & \mathbf{m}^T &
\end{pmatrix}
\begin{pmatrix}
\Delta\mathbf{z}_1 \\ \Delta\mathbf{z}_2 \\ \Delta\mathbf{s}_1 \\ \Delta\mathbf{s}_2 \\ \Delta\sigma \\ \Delta\mathbf{u} \\ \Delta\mathbf{y}_1 \\ \Delta\mathbf{y}_2 \\ \Delta\rho \\ \Delta\mathbf{y}_3
\end{pmatrix}
=
\begin{pmatrix}
\mathbf{r}_{z_1} \\ \mathbf{r}_{z_2} \\ \mathbf{r}_{s_1} \\ \mathbf{r}_{s_2} \\ \mathbf{r}_{\sigma} \\ \mathbf{r}_{u} \\ \mathbf{r}_{y_1} \\ \mathbf{r}_{y_2} \\ \mathbf{r}_{\rho} \\ \mathbf{r}_{y_3}
\end{pmatrix}. \quad (3.30)$$

Each row of this block matrix corresponds to the discretization of one of the equations starting from (3.17). In particular, the $c_{\{\cdot\}}$ terms become the left-hand vector that is solved for as the Newton step.

The matrices shown in the above block matrix have the following properties:

- A is symmetric positive definite – this is the stiffness matrix corresponding to our linear elasticity problem.
- B is a mass matrix that turns out to be diagonal because we are using piece-wise constant discretization of the density-related functions.
- C and E are sparse, non-square matrices, corresponding to second derivatives of the Lagrangian with respect to either density and displacement, or the density and displacement Lagrange multiplier.
- F is the matrix corresponding to the density filter.
- $\mathbf{m}$ is a vector with each entry corresponding to the measure of a cell.

- All matrices marked D are diagonal – D_m has the vector $\mathbf{m}$ down the diagonal, D_1 has the value $\mathbf{m}z_1/s_1$, and $D_2 = \mathbf{m}z_2/s_2$.

The order of the discretizations used in the matrix will become evident in Section 3.7.4 when I discuss my construction of a Schur Complement Preconditioner. The discretization and manipulation of functions was performed using the finite element library deal.II [25].

3.3 Barrier Reduction Techniques

A common issue in interior point barrier methods lies in finding an appropriate scheme to reduce the barrier size. For this algorithm to achieve the quadratic convergence that makes Newton-Raphson schemes appealing, this value would also have to converge quadratically to 0. While this is not always possible, it is still a motivating factor to avoid merely linear convergence to 0. A smaller barrier value creates large high-order derivatives around the boundary of the feasible domain. These high-order derivatives prevent the second derivative information from the Newton step from giving a good approximation for a moderate radius. As the iterations approach the solution, it is possible to decrease this barrier value as the steps should become smaller anyways, giving a smaller search radius.

For simplicity, these strategies will all be discussed using the same simple interior point problem:

$$\begin{aligned} &\text{Minimize } f(\mathbf{x}) \\ &\text{such that } \mathbf{g}(\mathbf{x}) = \mathbf{c}, \\ &\text{such that } \mathbf{h}(\mathbf{x}) \geq \mathbf{d}, \end{aligned} \tag{3.31}$$

and where the Jacobian of $\mathbf{g}$ is A, and the Jacobian of $\mathbf{h}$ is B. The interior point method would give the new problem

$$\begin{aligned} &\text{Minimize } f(\mathbf{x}) - \alpha \sum_i \log(s_i) \\ &\text{such that } \mathbf{g}(\mathbf{x}) = \mathbf{c}, \\ &\text{such that } \mathbf{h}(\mathbf{x}) - \mathbf{d} = \mathbf{s}. \end{aligned} \tag{3.32}$$

The Lagrangian of such a system would be

$$\mathcal{L} = f(\mathbf{x}) - \alpha \sum_i \log(s_i) - \mathbf{y}^T(\mathbf{g}(\mathbf{x}) - \mathbf{c}) - \mathbf{z}^T(\mathbf{h}(\mathbf{x}) - \mathbf{d} - \mathbf{s}). \tag{3.33}$$

Requiring the gradient of Equation (3.33) to be zero leads to the following Newton step:

$$\begin{pmatrix} L_{xx} & -A & 0 & -B \\ A^T & 0 & 0 & 0 \\ B^T & 0 & -I & 0 \\ 0 & 0 & Z & S \end{pmatrix} \begin{pmatrix} \Delta\mathbf{x} \\ \Delta\mathbf{y} \\ \Delta\mathbf{s} \\ \Delta\mathbf{z} \end{pmatrix} = - \begin{pmatrix} \nabla f(\mathbf{x}) - \nabla\mathbf{g}(\mathbf{x})\mathbf{y} - \nabla\mathbf{h}(\mathbf{x})\mathbf{z} \\ \mathbf{g}(\mathbf{x}) - \mathbf{c} \\ \mathbf{h}(\mathbf{x}) - \mathbf{d} - \mathbf{s} \\ Z\mathbf{s} - \alpha \end{pmatrix}. \tag{3.34}$$

Where Z and S are diagonal matrices with **z** and **s** down the diagonal, respectively.

By adding the solution vector to our current iteration, i.e. $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}$, we hope to move closer to the minimum of the optimization subproblem in Equation (3.32). However, we also need to reduce α close to zero to find an accurate solution to Equation (3.31).

3.3.1 Monotone Barrier Method

The simplest method for barrier reduction is to reduce the barrier size whenever the KKT conditions within some subproblem are solved to some degree of accuracy (as determined by the l_∞ norm of the KKT conditions).

Using the example above, this corresponds to taking each found Newton step in a subproblem such as Equation (3.32) until the l_∞ norm of the right hand side vector of the problem is sufficiently small. The barrier parameter α would then be reduced by some method, and the new subproblem would be solved.

This approach is called the Fiacco-McCormick Method [26], and is widely used [27]. I use a method from the LOQO package [28] for determining barrier reduction after convergence each time. A sensible method is to choose the smaller parameter α achieved by either multiplying the barrier parameter α by some value between 0 and 1, or by raising it to some exponent between 1 and 2. I found a multiplier of .7 and an exponent of 1.3 to be effective values for this problem.

Monotone methods are noted for their simplicity and robustness. However, they often converge slower than necessary and remove the possibility of quadratic convergence that we would ideally achieve with Newton's Method. To this end, adaptive barrier methods have been developed that can maintain quadratic convergence.

3.3.2 Adaptive Barrier Methods

Adaptive barrier methods are typically much better than monotone reduction strategies [29]. These strategies typically update the barrier to be some value of the form

$$\alpha = \sigma \frac{\mathbf{s}^{\mathrm{T}}\mathbf{z}}{n}. \tag{3.35}$$

Here, $\frac{\mathbf{s}^{\mathrm{T}}\mathbf{z}}{n}$ gives the current average value for the complementarity values of the system. At a subproblem solution, the solution satisfies $s_i z_i = \alpha$. We expect this average value to give an idea of the current state of our barrier parameter. We then multiply the current state by some number σ, $0 < \sigma < 1$ to reduce the goal barrier parameter. Several algorithms exist to find a value σ that brings the barrier parameter α down quickly, so that the problem is solved quickly, but also that reduces α slowly enough that the Hessian does not become too ill-conditioned near the boundary of the feasible domain, resulting in a problem that is extremely difficult to solve.

The algorithms I considered are described below, along with a discussion of their usefulness in the context of my topology optimization problem.

- Mehrotra predictor-corrector algorithm

  The Mehrotra predictor-corrector algorithm [30] is an algorithm that is very efficient, and commonly used in small-scale optimization problems. It guesses what the new barrier value α should be, and then makes a step to find the minimum. The algorithm begins by taking a full feasible step assuming no barrier is present.

  This calculated step is called the affine step. The found slack variables and slack multipliers are used from this step to calculate a predicted barrier value that can be used for this step. This barrier value is then used in creating the actual step. A final corrector step is then

created that attempts to correct for the error created in the first two steps. While this method has been found to be very effective on interior point methods in linear programs where the Hessian is zero, it has been recently found to be quite ineffective in nonlinear programs [27]. The first step is referred to as a "predictor" step, where the barrier is taken to be zero. This hopefully gives us an idea of where the system would want to go on the boundary without yet looking at the barriers. Setting $\alpha = 0$, our Newton step from Equation (3.34) becomes

$$\begin{pmatrix} L_{xx} & -A & 0 & -B \\ A^T & 0 & 0 & 0 \\ B^T & 0 & -I & 0 \\ 0 & 0 & Z & S \end{pmatrix} \begin{pmatrix} \Delta\mathbf{x}^{\text{aff}} \\ \Delta\mathbf{y}^{\text{aff}} \\ \Delta\mathbf{s}^{\text{aff}} \\ \Delta\mathbf{z}^{\text{aff}} \end{pmatrix} = - \begin{pmatrix} f(\mathbf{x}) - \mathbf{y}^T\mathbf{g}(\mathbf{x}) - \mathbf{z}^T(\mathbf{h}(\mathbf{x}) - \mathbf{d}) \\ \mathbf{g}(\mathbf{x}) - \mathbf{c} \\ \mathbf{h}(\mathbf{x}) - \mathbf{d} - \mathbf{s} \\ Z\mathbf{s} \end{pmatrix}. \tag{3.36}$$

We denote the maximum primal step to the barrier to be $\Delta^{\text{aff}}_{\text{pri}}$, and the maximum dual step to be $\Delta^{\text{aff}}_{\text{dual}}$. The complementarity value after this step is $\alpha^{\text{aff}} = (\mathbf{s} + \Delta\mathbf{s}^{\text{aff}}_{\text{pri}})^T(\mathbf{z} + \Delta\mathbf{z}^{\text{aff}}_{\text{dual}})/n$. We now pick a new barrier value based off of the complementarity value by setting $\sigma = \left(\frac{\alpha^{\text{aff}}}{\alpha}\right)^3$, and solving the centering step

$$\begin{pmatrix} L_{xx} & -A & 0 & -B \\ A^T & 0 & 0 & 0 \\ B^T & 0 & -I & 0 \\ 0 & 0 & Z & S \end{pmatrix} \begin{pmatrix} \Delta\mathbf{x}^{\text{cen}} \\ \Delta\mathbf{y}^{\text{cen}} \\ \Delta\mathbf{s}^{\text{cen}} \\ \Delta\mathbf{z}^{\text{cen}} \end{pmatrix} = - \begin{pmatrix} f(\mathbf{x}) - \mathbf{y}^T\mathbf{g}(\mathbf{x}) - \mathbf{z}^T(\mathbf{h}(\mathbf{x}) - \mathbf{d}) \\ \mathbf{g}(\mathbf{x}) - \mathbf{c} \\ \mathbf{h}(\mathbf{x}) - \mathbf{d} - \mathbf{s} \\ Z\mathbf{s} - \sigma\alpha \end{pmatrix}. \tag{3.37}$$

This, of course, is still not quite right, and will never actually achieve an barrier parameter of zero, and so we perform a correction step to this by using

$$\begin{pmatrix} L_{xx} & -A & 0 & -B \\ A^T & 0 & 0 & 0 \\ B^T & 0 & -I & 0 \\ 0 & 0 & Z & S \end{pmatrix} \begin{pmatrix} \Delta\mathbf{x}^{\text{cor}} \\ \Delta\mathbf{y}^{\text{cor}} \\ \Delta\mathbf{s}^{\text{cor}} \\ \Delta\mathbf{z}^{\text{cor}} \end{pmatrix} = - \begin{pmatrix} 0 \\ 0 \\ 0 \\ Z^{\text{aff}}\mathbf{s}^{\text{aff}} \end{pmatrix}. \tag{3.38}$$

The step that is used is then $\Delta^{\text{cen}} + \Delta^{\text{cor}}$. When used in a linear context, this works very well, aside from the downside of needing to solve this equation 3 times. For small systems, the inverse of the matrix from Equation (3.34) could potentially be saved, turning the last two solves into matrix-vector products, but this becomes infeasible for large systems.

- Mehrotra probing

  Mehrotra probing is a modification of the MPC algorithm shown in [31], where the corrector step is not included. Instead, the "probe" of stepping with a zero barrier in the right hand side vector is used to calculate a new barrier parameter that is then used for the actual step. This has been shown to be far more robust and effective than the full MPC method in nonlinear programming problems [27]. However, concerns remain about needing to solve this system twice at each iteration, especially when dealing with large systems of equations.

- Quality functions

  This method was introduced by Nocedal in 2009 [27] to help solve very nonlinear problems. The idea is to select α by minimizing some quality function that gives information about the quality of the newly selected barrier parameter. Again, we say that $\alpha = \sigma\frac{\mathbf{s}^T\mathbf{z}}{n}$, with $0 < \sigma < 1$. Let $\beta^{\max}_{\text{pri}}(\sigma)$ and $\beta^{\max}_{\text{dual}}(\sigma)$ be the maximum steplengths of the primal and dual parts to the boundary for some given σ value, and $\Delta\mathbf{x}(\sigma)$ be the $\mathbf{x}$ part of the Newton step where σ is used. Then $\mathbf{x}(\sigma) = x_0 + \beta^{\max}_{\text{pri}}(\sigma)\Delta\mathbf{x}(\sigma)$. I want to find the σ that minimizes the KKT conditions of the nonlinear program by some norm. This has been shown to be

effective using an l_2 norm combined with a linear approximation of the KKT conditions, or

$$q_L(\sigma) = \|\nabla f(\mathbf{x}(\sigma)) - A(\mathbf{x}(\sigma))^T\mathbf{y} - \mathbf{z}(\sigma)\|_{l_2}^2 + \|\mathbf{c}(\mathbf{x})\|_{l_2}^2 + \|(X + \beta_{\text{pri}}^{\max}(\sigma)\Delta X(\sigma))(Z + \beta_{\text{dual}}^{\max}(\sigma)\Delta Z(\sigma))\mathbf{e}\|_{l_2}^2. \quad (3.39)$$

This was shown to perform better than a quadratic quality function in [27], although the reason is unknown. A golden section search is used to find the minimum, which requires solving the system several times. Once again, for small systems where an inverse can be saved, this is a very feasible method, but it becomes unwieldy for large systems.

- LOQO method

  The LOQO method is a heuristic method of updating the barrier parameter, first used in the LOQO optimization software package [28]. It is an adaptive method with a clear advantage over the Mehrotra and quality function technique in that it does not take an additional solve to determine the barrier parameter, and only uses already available information to update the barrier parameter.

  Like previous algorithms, LOQO determines the new barrier every step by determining a value σ to find a new barrier parameter through $\alpha = \sigma\frac{\mathbf{s}^T\mathbf{z}}{n}$. It uses the fact that systems where all values $s_i z_i$ are close to the average value tend to be well behaved [24]. It then uses this closeness to the average as a measure to determine how much to decrease the barrier parameter.

  The measure of "closeness to average" is given by

$$\xi = \frac{\min\{s_i z_i\}}{(\mathbf{s}^T\mathbf{z})/n}. \quad (3.40)$$

  From this, we get the multiplier

$$\sigma = 0.1\left(\min\left\{0.05\frac{1-\xi}{\xi}, 2\right\}\right)^3. \quad (3.41)$$

The major downside of using this method is that, when a subproblem is solved well enough that all values $s_i z_i$ are extremely close to the average value, the step it takes can be extremely aggressive. When this happens, a trust region is necessary in the form of a maximum allowed fraction travelled to the border. This is to prevent the step from reaching the (now ill-conditioned) area near the border.

I use the LOQO adaptive barrier method in my solver. Despite being rather heuristic, it does not need multiple large solves to determine a new barrier size, while still giving an aggressive, but reasonable, new barrier value. It speeds up the solve to use as few as 1/4 the Newton steps to find a solution to a given accuracy compared to a monotone method when tested against my test problems. In my problem, the LOQO algorithm can be used almost exactly as written, with the knowledge that I look over all entries of both s_1 and s_2.

3.3.3 Mixed Free and Monotone Barrier Method

The above adaptive methods work worse when far from the optimum (as evidenced by the need for a rather strict fraction-to-boundary trust region), so it is occasionally best to begin with a monotone barrier reduction, or to move to one when the above methods have a difficult time moving closer to the solution. In order to enforce this mixed method, [27] suggests the following scheme. In their paper, this was referred to as a "globalization scheme" but actually just globalizes the barrier search, not the problem as a whole.

My implementation of this mixed method begins optimistically with the LOQO adaptive scheme. If the barrier stops decreasing at a sufficient rate, I assume that the adaptive method has failed, and move to a monotone method. Once a full monotone step has been taken – that is, the subproblem with the given barrier size has been solved to sufficient accuracy – I restart the adaptive method in an attempt to regain the speed.

This barrier globalization scheme is simple to implement, and works well with the adaptive LOQO method mentioned above, and so I utilize both in my algorithm.

3.4 Globalization Techniques

Newton methods are infamously non-global, and so additional methods are needed to ensure that they lead towards a solution, especially in the early iterations of the process. The typical solution to this problem is to take a step, determine in some way if it is "good", and if it is not a good step, to attempt a smaller step in the same direction. That is, the equation for an iteration using Newton's method,

$$\mathbf{x}_n = \mathbf{x}_{n-1} - \left(H(\mathbf{x}_{n-1})\right)^{-1} \nabla \mathcal{L}(\mathbf{x}_{n-1}), \tag{3.42}$$

is modified to become

$$\mathbf{x}_n = \mathbf{x}_{n-1} - \beta \left(H(\mathbf{x}_{n-1})\right)^{-1} \nabla \mathcal{L}(\mathbf{x}_{n-1}), \tag{3.43}$$

where $0 < \beta < 1$ is modified to ensure convergence. In primal-dual systems, it turns out that two different values, β_{pri} and β_{dual} can be chosen, resulting in the update

$$\begin{aligned} \Delta\mathbf{x}_{n-1} &= \left(H(\mathbf{x}_{n-1})\right)^{-1} \nabla \mathcal{L}(\mathbf{x}_{n-1}), \\ (\mathbf{x}_n)_{\text{pri}} &= (\mathbf{x}_{n-1})_{\text{pri}} - \beta_{\text{pri}} (\Delta\mathbf{x}_{n-1})_{\text{pri}}, \\ (\mathbf{x}_n)_{\text{dual}} &= (\mathbf{x}_{n-1})_{\text{dual}} - \beta_{\text{dual}} (\Delta\mathbf{x}_{n-1})_{\text{dual}}. \end{aligned} \tag{3.44}$$

Typically, these βs will be chosen as 1, and then the step will be taken and the result examined. If the result is deemed "good", the β is kept. If not, the β is reduced, and the step is attempted again. This is repeated until a good step is found.

I explore two such ways of determining this β. While merit functions have historically been used as a way to determine whether or not to accept a Newton step, more modern and more simple methods called "filters" have begun showing themselves as more useful.

3.4.1 Merit Function

I consider an exact l_1 merit function for this problem. Exact merit functions are those where the minima of the merit function matches the minima of the problem. I choose to use an l_1 norm for its simplicity and wide usage in the field. This merit function is much easier to compute than,

for example, the augmented Lagrangian, with the only downside being its non-differentiability. As the methods I discuss never require a derivative of the merit function, this is not an issue. The exact l_1 merit function is, as stated, exact, and so will have a minimum in the same location as the objective function. This means that we should never accept a terribly bad step, or fall into a second minimum of this merit function – essentially, we can trust the values it gives [32]. This merit function has the form

$$\phi(\rho, \tilde{\rho}, \mathbf{u}) = \int_{\partial\Omega} \mathbf{u} \cdot \mathbf{t} + \int_{\Omega} \alpha(\log(s_1) + \log(s_2)) + \gamma \left(\int_{\Omega} \|A(\rho)\mathbf{u} - \mathbf{f}\|_{l_1} + \|\tilde{\rho} - F(\rho)\|_{l_1} + \|\mathbf{s}_1 - \alpha/\mathbf{z}_1\|_{l_1} + \|\mathbf{s}_2 - \alpha/\mathbf{z}_2\|_{l_1} \right). \quad (3.45)$$

An exact merit function is one where the minima of the merit function match the minima of the problem for any sufficiently large γ. In this scheme, the γ should be increased whenever a step is taken that forces the constraints to become less exactly met than before. One possible way to enforce this, from Nocedal and Wright [31], is

$$\gamma > \frac{.5\mathbf{x}^T L\mathbf{x} - \mathbf{x}^T \nabla\mathbf{f}}{\|\mathbf{c}\|_{l_\infty}}. \quad (3.46)$$

The penalty multiplier is updated to be at least this large whenever a new step is taken. In the context of my problem, $\mathbf{x}$ is the combined vector of the decision variables $\rho, \tilde{\rho},$ and $\mathbf{u}$, L is the submatrix of the Hessian of the Lagrangian relating these decision variables, and $\nabla\mathbf{f}$ is the corresponding right hand side. An entry c_i is then any other entry in the right hand side, as they correspond with constraints.

3.4.2 Filters

Filters are an alternative to merit functions that have been found to be especially useful over the past 20 years. Filters have many advantages over merit functions in that not only are they faster, but their effectiveness does not depend on the problem being solved, and they do not require any heuristic parameter selection. A filter is, in its essence, pairs of values representing previous

iterations. Both the objective function value and the constraint infeasibility are stored. A new step is accepted so long as there is no pair in the filter that is 'better' than the new point in terms of both its values. Immediate problems arise when considering barrier methods. The objective function is no longer truly what the algorithm is looking to minimize, but rather we are now looking at the barrier function. Changing the barrier parameter affects both the barrier function values and feasibility conditions, which can invalidate previous filter entries. Three possibilities for filter methods are discussed below:

- Fletcher-Leyffer Filter

  This method is the oldest, and rather simple. It uses the objective function and infeasibility value directly. This should not cause a problem so long as the barrier parameter is monotonically decreasing. The method is described in [33].

- Barrier Filter

  The Barrier filter decides to solve the first issue with filter methods by simply replacing the objective function value in the filter with the barrier function. By saving the objective value and barrier parts of the barrier function separately, whenever the barrier is updated, the barrier function values can be updated as well. The feasibility values can be similarly saved in a manner that allows us to update its value when a new barrier parameter is found. One implementation of this concept can be found in [34].

- Markov Filter

  The Markov filter is a great simplification over the other methods I have described. It only stores information from the previous step, and then accepts any step that is better in either the barrier function or feasibility conditions. While this is clearly the most optimistic filter, tests show that accepting more and larger steps actually causes it to converge much faster [35]. Not only that, but this filter does not have the problem some other filters have of getting "stuck" and having to temporarily revert to a merit function, which is a common problem in the cases of the Fletcher-Leyffer Filter and Barrier Filter.

In all of these filters, there is a concern that a very large step will be taken that only slightly improves the objective function. In this case, the rate of convergence can actually be hurt. To combat this, something similar to Armijo conditions are applied to how filters see the new objective function to help with convergence. In addition to this, merit functions are still necessary for barrier filters and Fletcher-Leyffer filters, but not Markov filters, which only look at the objective function and feasibility. The barrier issues can be largely resolved by updating the old objective function value to what it would be with the new barrier value.

All the filters discussed here besides the Markov filter still require a merit function, which comes with all of its mentioned problems as well. While work can be done to overcome these issues, the ease of use and demonstrable effectiveness of the Markov filter make it the obvious choice.

3.5 Avoiding the Maratos Effect

To ensure rapid global convergence for the subproblem given a barrier size, I implement a watchdog algorithm. While this algorithm can be slow in the process of reaching the asymptotic range, it is much faster once in the asymptotic range than second order corrections (which utilize multiple large solves on every step) or line search methods (which can on occasion force needlessly small steps and slow convergence) [31]. This algorithm essentially works by seeing if many Newton steps eventually find a "better" guess than the original, as determined by some merit function. If it does not, it takes a smaller, scaled step from the original location, and tries again.

```
1  Set barrier value, α
2  Set descent requirement, ν
3  Set initial guess x_0
4  While (Barrier above minimal value)
5  {
6    While (Convergence not reached)
7    {
8        For (i=0 to t̂ - typically 5 or 8)
```

```
{
    Compute step p_{k+i} with Newton's Method
    Compute x_{k+i+1} = x_{k+i} + p_{k+i}
    If (φ(x_{k+i+1}) < φ(x_k) + νD(φ(x_k), p_k) − φ is merit function or KKT norm)
      {
          Accept x_{k+i+1}
          k = k + i + 1
          Found Step = True
          Break for loop
      }
}
If( Found Step = False)
  {
  Compute step p_{k+t̂+1} with Newton's Method
  Find β_{k+t̂+1} so that φ(x_{k+t̂+2}) ≤ φ(k + t̂ + 1) + νβ_{k+t̂+1}D(φ(x_{k+t̂+1}); p_{k+t̂+1})
  x_{k+t̂+2} = x_{k+t̂+1} + β_{k+t̂+1}p_{k+t̂+1}
      If (φ(x_{k+t̂+1}) ≤ φ(x_k) or φ(x_{k+t̂+2}) ≤ φx_k + νD(φ(x_k); p_k))
      {
          Accept x_{k+t̂+2}
          k = k + t̂ + 2
      }
      Else
      {
          If(φ(x_{k+t̂+2}) > φ(x_k))
          {
              Find β_k such that φ(x_{k+t̂+3}) ≤ φ(x_k) + νβ_k D(φ(x_k); p_k)
              x_{k+t̂+3} = x_k + β_k p_k
              Accept x_{k+t̂+3}
              k = k + t̂ + 3
          }
          Else
          {
              Compute p_{k+t̂+2}
```

```
42                          Find β_{k+t̂+2} such that
                                φ(x_{k+3}) ≤ φ(x_{k+t̂+2}) + νβ_{k+t̂+2} D(φ(x_{k+t̂+2}); p_{k+t̂+2})
43                          x_{k+t̂+3} = x_{k+t̂+2} + β_{k+t̂+2} p_{k+t̂+2}
44                          Accept x_{k+t̂+3}
45                          k = k + t̂ + 3
46                      }
47                  }
48              }
49          }
50      Reduce Barrier Size
51  }
```

The use of the watchdog algorithm, combined with other methods given here, gives me a complete nonlinear solver that can find optimal solutions in few steps, is global, and avoids the Maratos Effect.

3.6 Numerical Continuation Approaches

Part of the difficulty of solving topology optimization problems is that the nonlinear problem becomes more difficult to solve when certain parameters are near desired values. For example, a larger density filter radius (as discussed in Section 2.2) gives a more well-conditioned problem, and a smaller penalty exponent on the density variable in Equation (3.4) removes many bad local minima. However, a large filter radius relative to the size of an individual voxel prevents the solution from taking full advantage of the resolution of the mesh. Similarly, a small penalty exponent prevents the final solution from being near "black-and-white". While not used in the final algorithm in this thesis, these numerical continuation methods are discussed here for completeness.

Numerical continuation methods are methods that assist in iteratively solving a nonlinear equation of the form

$$F(\mathbf{x}, \lambda) = 0. \tag{3.47}$$

These methods select initially some value λ_0 so that the equation $F(\mathbf{x}, \lambda_0) = 0$ is easy to solve, and as the iterative methods continue a sequence $\lambda_n \to \lambda$ is used to approach the actual value of λ. In practice, this leads to faster convergence as earlier iterations using the "nicer" values of λ lets $\mathbf{x}_n$ approach the actual solution $\mathbf{x}$ more quickly, and with less numerical instability.

In [36], the authors explore numerical continuation techniques to leverage using a smaller density penalty exponent when far from the solution, using the NAND formulation for benchmarking. They found this continuation of the penalty exponent, initially supported for its ease in solving, allows for finding better designs by avoiding local minima. This continuation approach has been used as early as in 1998 [37], although all current methods for its implementation are heuristic. A much more computationally feasible, although not necessarily as effective, method of implementing this continuation is shown in [36], in what they term an "automatic continuation". In [38], various continuations of the filter and density penalty exponent, although by using a 3-field density formulation, taking a heaviside filter in addition to the density filter.

All of these methods could largely be used as additional methods to improve the convergence speeds of topology optimization problems. However, the methods would seemingly have their own risks of giving less-optimal solutions, and are not yet shown to consistently improve solution time. As such, implementation of these methods is outside the scope of this Thesis.

3.7 Linear Problem Solving

In Section 3.1, I made a case for use of a Newton-Raphson interior point method for solving topology optimization problems, while acknowledging the difficulty of solving the necessary larger linear system. In this section, I look at methods that I have implemented in an attempt to make the linear solve more feasible for large-scale problems.

I begin by discussing the greatest advantage with regard to this problem – that the linear system is largely sparse! The block linear system sparsity pattern is shown in Figure 3.1, where we can see the block nature of the matrix. The one area of concern with regards to the sparsity is the convolution matrix representing the density filter, where some small percentage of entries are nonzero,

but this number increases with refinement. However, on large-scale problems, the filter radius can be expected to be a smaller portion of the domain diameter, and so this issue does not prevent a solution from being found. It is worth noting that, as mentioned in Section 2.2, some optimizers use slope constraints or a modified Helmholtz equation instead of a convolution filter to avoid this slight lack of sparseness.

Sparsity gives a maximum number of elements in a row of a matrix. This means that the number of operations needed for a matrix-vector multiplication with an $(n \times n)$ matrix with at most c elements in each row is $O(n)$, whereas for a dense matrix, the number of operations is $O(n^2)$. This means that for a matrix representing a problem with twice as many voxels, a matrix multiplication should only take twice as long. As our solving technique will be iterative, this gives us hope for solving large systems without extreme cost.

The sparsity of this problem, and its underlying structure, will be leveraged in creating preconditioners that allow us to invert this matrix quickly and efficiently.

3.7.1 Properties of Linear Problems and Appropriate Iterative Solvers

Looking at the problem overall, I have an indefinite matrix, meaning a General Minimal Residual (GMRES) [39] solver should be a good fit. GMRES converges in a number of iterations at most equal to the number of unique eigenvalues, so I need to look for a preconditioner that groups eigenvalues, or at least clusters them. Because my preconditioner is not linear, which we will soon see, Flexible General Minimal Residual (FGMRES) methods [40] can be used to maintain this convergence rate at the expense of some amount of memory.

3.7.2 How Accurately to Solve the Linear System

In today's linear algebra landscape, we have found that it is much more efficient to solve large linear systems using inexact iterative methods. This means that I have an inexact iterative linear solver inside of my inexact iterative nonlinear program. The inexactitudes here can actually be leveraged to our advantage to gain speed in our algorithm. When far away from the nonlinear solution, it is less necessary to get as precise a Newton step, as we expect it to be a worse approxima-

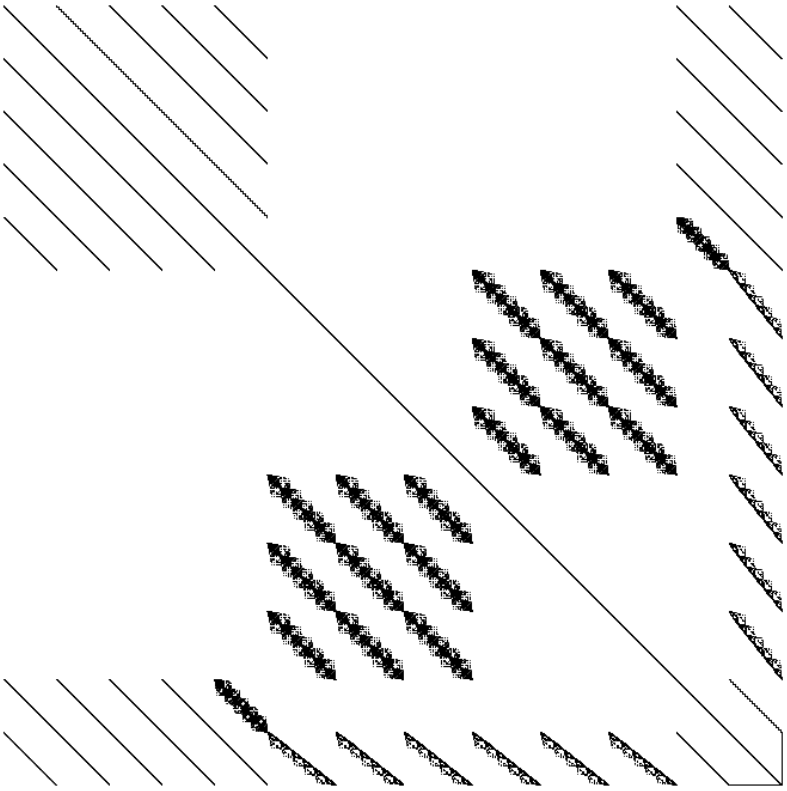


Figure 3.1: Sparsity pattern of block matrix used in finding a Newton step for a three-dimensional topology optimization problem – see Equation (3.58)

.

tion of the correct direction anyway. The use of this approximation is called the Eisenstat-Walker Method [41].

To find a step in my nonlinear algorithm, I solve

$$H\Delta\mathbf{x}_k = -\nabla\mathcal{L}, \tag{3.48}$$

where $\mathcal{L}(\mathbf{x}_k)$ is the Lagrangian formulated in Equation (3.4), given a state $(\mathbf{x}_k)$ and H is its Hessian. This is done to find the Newton-Raphson step $\Delta\mathbf{x}_k$. Eisenstat and Walker propose that, instead of solving this equation so that $H\Delta\mathbf{x}_k + \nabla\mathcal{L}(\mathbf{x}_k) = 0$, we solve this system only to the point where $\|H\Delta\mathbf{x}_k + \nabla\mathcal{L}(\mathbf{x}_k)\| \leq \nu_k\|\nabla\mathcal{L}(\mathbf{x}_k)\|$ for some $\nu_k < 1$.

For a given ν_k and t, a global inexact Newton's method will work to find a $\Delta \mathbf{x}_k$ such that

$$\|H\Delta \mathbf{x}_k + \nabla \mathcal{L}(\mathbf{x}_k)\| \leq \nu_k \|\nabla \mathcal{L}(\mathbf{x}_k)\| \tag{3.49}$$

and

$$\|\nabla \mathcal{L}(\mathbf{x}_k + \Delta \mathbf{x}_k)\| \leq (1 - t(1 - \nu_k))\|\nabla \mathcal{L}(\mathbf{x}_k)\|. \tag{3.50}$$

This forces first, a sufficiently good solve, and then, a sufficiently good step. The step is considered "sufficiently good" if the reduction in the norm of $\nabla \mathcal{L}(\mathbf{x}_k)$ is at least some multiple of the predicted reduction, that is

$$\|\nabla \mathcal{L}(\mathbf{x}_k)\| - \|\nabla \mathcal{L}(\mathbf{x}_k + \Delta \mathbf{x}_k)\| \geq t(\|\nabla \mathcal{L}(\mathbf{x}_k)\| - \|H\Delta x_k + \nabla \mathcal{L}(\mathbf{x}_k)\|) \tag{3.51}$$

$$\|\nabla \mathcal{L}(\mathbf{x}_k)\| - \|\nabla \mathcal{L}(\mathbf{x}_k + \Delta \mathbf{x}_k)\| \geq t(\|\nabla \mathcal{L}(\mathbf{x}_k)\| - \nu_k \|\nabla \mathcal{L}(\mathbf{x}_k)\|) \tag{3.52}$$

$$\|\nabla \mathcal{L}(\mathbf{x}_k + \Delta \mathbf{x}_k)\| \leq (1 - t(1 - \nu_k))\|\nabla \mathcal{L}(\mathbf{x}_k)\|. \tag{3.53}$$

In order to ensure q-quadratic convergence is maintained in this inexact context, I also have to follow the requirement on the accuracy on my linear solve that

$$\nu_k = O(\|\nabla \mathcal{L}(\mathbf{x}_k)\|). \tag{3.54}$$

The benefits of using such a method are evident when inexact solvers are used. Often, especially for complex block systems, systems are sufficiently ill-conditioned that inexact solution methods can continue to be slow to solve, even when somewhat close to the actual solution. Eisenstat-Walker gives a minimum condition to maintain quadratic convergence, without the need to wait for very high-accuracy solutions to these large systems.

3.7.3 Preconditioning Methods

The art of improving the solving speed of an iterative linear solver is known as preconditioning. In general, these problems are written

$$A\mathbf{x} = \mathbf{b}. \tag{3.55}$$

The number of iterations that an iterative solver takes to solve this system depends on characteristics of the matrix A. Using knowledge about the traits of the matrix A, as well as the characteristics of the linear solver algorithm, this system can be re-written

$$PA\mathbf{x} = P\mathbf{b}, \tag{3.56}$$

or alternatively as

$$AP(P^{-1}\mathbf{x}) = \mathbf{b}, \tag{3.57}$$

where the matrix P as used in Equation (3.56) is a "left preconditioner", and as used in Equation (3.57) is a "right preconditioner". Preconditioning changes to problem away from inverting A, and allows us to instead invert the preconditioned matrix PA or AP. Leveraging what we know about the matrix A, we can develop some invertible P that makes the preconditioned matrix easy and efficient to solve.

Many methods exist for developing a preconditioning matrix, and these depend very heavily on the matrix that is being solved. In the following sections, I explore several preconditioners that I have developed in an attempt to further speedup the solve of the problem. The preconditioners will eventually come in three parts: an outer preconditioner, that takes my large block system and simplifies it to a smaller (but more complex) matrix expression to invert, a mid-level preconditioner that leverages the structure of the matrix expression, and an inner preconditioner I use to very efficiently solve the block corresponding to the elasticity equation stiffness matrix.

3.7.4 Preconditioning for a GMRES or FGMRES algorithm

I solve the main block linear system with a FGMRES [40] solver. For these solvers, the number of iterations taken to solve the system is related to the number of distinct eigenvalues. This fact is easier to prove with the simpler GMRES algorithm, and is shown here.

The GMRES algorithm, or General Minimal Residual algorithm, works with the following steps when trying to solve the system $Ax = b$, with initial guess x_0.

The algorithm for GMRES slowly builds a krylov matrix Q, that is, the matrix $r, Ar, A^2r, ...A^mr$, where in this case r is the initial residual $r_0 = b - Ax_0$.

After some convenient scaling, the algorithm uses an Arnoldi iteration to compute an orthonormal basis to the Krylov subspace. As this matrix is built, we generate a low-rank approximation of the matrix using this basis, and solve for the solution to the approximation explicitly by finding a least-squares solution. This approximation gets better as the low-rank approximation becomes more and more accurate by the inclusion of more of the Krylov subspace in finding Arnoldi Vectors.

When the minimal polynomial of A has degree k, this Arnoldi iteration breaks down after K steps - that is, the full Krylov Subspace will already be spanned by the basis, and so the new basis vector found will be 0. At this point, the matrix has been fully written in terms of its krylov subspace, and GMRES will successfully converge.

As the size of the minimal polynomial is often related to the number of distinct eigenvalues, reducing the number of eigenvalues is a common way to precondition GMRES and FGMRES solvers. Further, the Arnoldi iteration approximates eigenvalues and eigenvectors using the krylov subspace. Then, the low-rank approximate matrices will be better approximations when the number of distinct eigenvalues is small, and the method will converge more quickly.

A more complete discussion of the convergence properties by Van Der Vorst and C.Vuik in [42].

Schur Complement Block Preconditioner

Schur Complement decomposition allows us to create a preconditioner that results in an upper triangular matrix with only "1" on the diagonal, meaning it only has the eigenvalue 1. Ideally, with

only one eigenvalue, FGMRES would converge in only one iteration. Because of slight variations in how the matrix is constructed for practical reasons, this preconditioner, if constructed exactly, can be experimentally shown to require 4 GMRES steps to solve. These extra eigenvalues stem from implementation of the Dirichlet Boundary conditions. In practice, around 10 steps will be needed, so the implementation of the boundary conditions are not a cause of concern.

The linear system found for the desirable Newton step is written in Equation (3.58). The matrix used in this system, which I will decompose, is

$$\begin{pmatrix}
 & & D_m & & -D_m & & & & \\
 & & & D_m & D_m & & & & \\
D_m & & D_1 & & & & & & \\
 & D_m & & D_2 & & & & & \\
-D_m & D_m & & & & & F^T & & \\
 & & & & & A & & C & \\
 & & & & A & & & E & \\
 & & & & F & & & -D_m & \\
 & & & & & C^T & E^T & -D_m & B & \mathbf{m} \\
 & & & & & & & & \mathbf{m}^T &
\end{pmatrix}. \tag{3.58}$$

I name the original matrix S_0 for reasons that will become clear later. The first Schur complement decomposition pivots around the top left 4-by-4 block matrix, given that

$$\begin{pmatrix}
 & & D_m & \\
 & & & D_m \\
D_m & & D_1 & \\
 & D_m & & D_2
\end{pmatrix}^{-1} = \begin{pmatrix}
D_3 & & D_m^{-1} & \\
 & D_4 & & D_m^{-1} \\
D_m^{-1} & & & \\
 & D_m^{-1} & &
\end{pmatrix}. \tag{3.59}$$

This inverse introduces the new diagonal matrices D_3, whose entries correspond to $-z_1/ms_1$, and D_4 with $-z_2/ms_2$. This decomposition also brings about matrices D_5 corresponding to the value of z_1/s_1 on a cell, D_6 with z_2/s_2, and $D_7 = D_1 + D_2$, corresponding to $m(z_1s_2 + z_2s_1)/s_1s_2$.

This decomposition allows the original matrix to be rewritten as $S_0 = T_1S_1T_1^T$, where T_1 is a triangular matrix with 1 on the diagonal, and S_1 is block diagonal. Here,

$$T_1 = \begin{pmatrix} I & & & & & & & & \\ & I & & & & & & & \\ & & I & & & & & & \\ & & & I & & & & & \\ D_5 & -D_6 & -I & I & I & & & & \\ & & & & & I & & & \\ & & & & & & I & & \\ & & & & & & & I & \\ & & & & & & & & I \end{pmatrix}, \tag{3.60}$$

and

$$S_1 = \begin{pmatrix} & D_m & & & & & & & \\ & & D_m & & & & & & \\ D_m & & D_1 & & & & & & \\ & D_m & & D_2 & & & & & \\ & & & D_7 & & & F^T & & \\ & & & & & A & & C & \\ & & & & A & & & E & \\ & & & F & & & & -D_m & \\ & & & & C^T & E^T & -D_m & B & \mathbf{m} \\ & & & & & & & \mathbf{m}^T & \end{pmatrix}. \tag{3.61}$$

S_1 can be further decomposed using the invertibility of D_7. Let us call the inverse of this diagonal matrix D_8. For simplicity, we name $G = FD_8F^T$.

Then, we have that $S_1 = T_2 S_2 T_2^T$ with

$$T_2 = \begin{pmatrix} I & & & & & & & & \\ & I & & & & & & & \\ & & I & & & & & & \\ & & & I & & & & & \\ & & & & I & & & & \\ & & & & & I & & & \\ & & & & & & I & & \\ & & & FD_8 & & & & I & \\ & & & & & & & & I \end{pmatrix}, \tag{3.62}$$

and

$$S_2 = \begin{pmatrix} & & D_m & & & & & & \\ & & & D_m & & & & & \\ D_m & & D_1 & & & & & & \\ & D_m & & D_2 & & & & & \\ & & & & D_7 & & & & \\ & & & & & A & & C & \\ & & & & A & & & E & \\ & & & & & & -G & -D_m & \\ & & & & C^T & E^T & -D_m & B & \mathbf{m} \\ & & & & & & & \mathbf{m}^T & \end{pmatrix}. \tag{3.63}$$

S_2 can be further decomposed using the diagonal block

$$\begin{pmatrix} & A \\ A & \end{pmatrix}^{-1} = \begin{pmatrix} & A^{-1} \\ A^{-1} & \end{pmatrix}. \tag{3.64}$$

I name $H = B - C^T A^{-1} E - E^T A^{-1} C$ This gives that $S_2 = T_3 S_3 T_3^T$ with

$$T_3 = \begin{pmatrix} I & & & & & & & & & \\ & I & & & & & & & & \\ & & I & & & & & & & \\ & & & I & & & & & & \\ & & & & I & & & & & \\ & & & & & I & & & & \\ & & & & & & I & & & \\ & & & & & & & I & & \\ & & & & E^T A^{-1} & C^T A^{-1} & & & I & \\ & & & & & & & & & I \end{pmatrix}, \tag{3.65}$$

and

$$S_3 = \begin{pmatrix} & & D_m & & & & & & & \\ & & & D_m & & & & & & \\ D_m & & D_1 & & & & & & & \\ & D_m & & D_2 & & & & & & \\ & & & & D_7 & & & & & \\ & & & & & A & & & & \\ & & & & & & A & & & \\ & & & & & & & -G & -D_m & \\ & & & & & & & -D_m & H & \mathbf{m} \\ & & & & & & & & \mathbf{m}^T & \end{pmatrix}. \tag{3.66}$$

Once again, we can decompose this S_3 matrix into $S_3 = T_4 S_4 T_4^T$ using

$$\begin{pmatrix} -G & -D_m \\ -D_m & H \end{pmatrix}^{-1} = \begin{pmatrix} \left(-HD_m^{-1}G - D_m\right)^{-1} HD_m^{-1} & \left(-HD_m^{-1}G - D_m\right)^{-1} \\ \left(-GD_m^{-1}H - D_m\right)^{-1} & -\left(-GD_m^{-1}H - D_m\right)^{-1} GD_m^{-1} \end{pmatrix}. \tag{3.67}$$

Naming $J = \left(-HD_m^{-1}G - D_m\right)^{-1}$, and $K = J^T = \left(-GD_m^{-1}H - D_m\right)^{-1}$, this gives

$$T_4 = \begin{pmatrix} I & & & & & & & & & \\ & I & & & & & & & & \\ & & I & & & & & & & \\ & & & I & & & & & & \\ & & & & I & & & & & \\ & & & & & I & & & & \\ & & & & & & I & & & \\ & & & & & & & I & & \\ & & & & & & & & I & \\ & & & & & & \mathbf{m}^T K & -\mathbf{m}^T K G D_m^{-1} & I \end{pmatrix}, \tag{3.68}$$

and

$$S_4 = \begin{pmatrix} & & D_m & & & & & & & & \\ & & & D_m & & & & & & & \\ D_m & & D_1 & & & & & & & & \\ & D_m & & D_2 & & & & & & & \\ & & & & D_7 & & & & & & \\ & & & & & A & & & & & \\ & & & & A & & & & & & \\ & & & & & & -G & -D_m & & & \\ & & & & & & -D_m & H & & & \\ & & & & & & & & \mathbf{m}^T(KGD_m^{-1})\mathbf{m} \end{pmatrix}. \quad (3.69)$$

Combining all of these decompositions gives

$$S_0 = T_1T_2T_3T_4S_4T_4^TT_3^TT_2^TT_1^T. \quad (3.70)$$

I note that $T_4^TT_3^TT_2^TT_1^T$ is upper triangular with 1 on the diagonal and has eigenvalue 1, and so a good preconditioner candidate is $P = (T_1T_2T_3T_4S_4)^{-1}$. $PS_0 = T_4^TT_3^TT_2^TT_1^T$ will only one eigenvalue, and be solved by a single GMRES iteration.

Because the matrix I have built has certain irregularities (such as constraints caused by boundary conditions), an exact run of the preconditioner still takes 4 GMRES iterations to solve exactly.

Most of the matrices I use in my preconditioner P are very inexpensive to compute. Even portions where A^{-1} need be applied are relatively inexpensive, as this stiffness matrix can be quickly inverted using a Multigrid method, discussed in Section 3.7.5 and a Conjugate Gradient (CG) solver. The most expensive portion of this decomposed inverse is finding the matrix K. To take a further look at the structure of K, I consider its formulation:

$$K = \left(-GD_m^{-1}H - D_m\right)^{-1} = \left(-FD_8F^TD_m^{-1}(B - C^TA^{-1}E - E^TA^{-1}C) - D_m\right)^{-1}. \quad (3.71)$$

Some immediate issues arise when looking at K^{-1}. Firstly, A^{-1} is a full matrix, and even if well-approximated by a very sparse matrix, it will be multiplied by a convolution matrix F, and its inverse, making K^{-1} still not sparse. Furthermore, K^{-1} is demonstrably nondefinite, and has a spectral radius greater than 1.

Currently, I solve this matrix K using GMRES iterations. I do not want to have to re-solve A^{-1} in each iteration, so I use a direct solver, and save a decomposition of A to avoid this problem. This is of course less than ideal, as the inverse of A is a dense matrix. In the future, for increased efficiency, more work needs to be done to efficiently solve this matrix K.

Rewriting as

$$K = \left(-FD_8F^TD_m^{-1}(B - C^TA^{-1}E - E^TA^{-1}C)D_m^{-1} - I\right)^{-1} D_m^{-1} \tag{3.72}$$

reveals more structure.

The solve of this inverse takes up the majority of the time on any given nonlinear subproblem. Its complex structure makes the solve very difficult, and I discuss attempts at preconditioning K^{-1} later in this chapter.

Preconditioner Approximation using Derivatives

A possibility for preconditioning the K matrix is to consider the origin of its constituent matrices. As they all come from second Gateaux derivatives of the Hessian of the Lagrangian, this information may be able to be leveraged to create an approximate matrix that would be easy to inverse and that would share many similar eigenvalues. Techniques similar to this are used to solve other block linear systems such as fluid flow in [43].

The A^{-1} matrix can be intuitively thought of as removing two derivatives of the Lagrangian - one with respect to $\mathbf{u}$, and the other with respect to $\mathbf{y}_1$. The matrices C and E each "add" these derivatives back. They also add one derivative with respect to ρ apiece. Using this heuristic framework, I attempted simplifying $B - C^TA^{-1}E - E^TA^{-1}C$ to act like the mass matrix B - which has two density derivatives of the Lagrangian.

The coefficient on the mass matrix B is $(p(p-1)\rho^{p-2})$, while the coefficients on the $C^T A^{-1} E$ 'mass matrix' would look like $(p^2\rho^{p-2})$. Combining these terms gives $(-p^2-p)\rho^{p-2}$. This mass matrix is diagonal because of the Q_1 element scheme used for the density-related elements, which makes it very easy to invert and use as a preconditioner. Constructing this mass matrix, and using its inverse as a preconditioner proved extremely ineffective. The eigenvalues of the actual matrix and its approximation are given in Figure 3.2. A clearer picture is given by the normalized eigenvalues in Figure 3.3. In this figure, we see that the structures of the eigenvalues are indeed very different, and not just their scaling. The number of iterations needed to solve when using this preconditioner actually increased, and so it was thrown out as a preconditioning possibility.

Polynomial Preconditioner

Below I show evidence that polynomial preconditioning of the K matrix would be inefficient, starting with a description of polynomial preconditioning. The polynomial preconditioning method laid out here is a simplified version of the preconditioner shown in [44].

It can be shown that a matrix $A = (I - B)$, where the magnitude of eigenvalues of B are strictly less than 1, has the inverse

$$A^{-1} = I + B + B^2 + B^3 + ... \tag{3.73}$$

A proof of this can be constructed using a diagonalization of the matrix B, where

$$B = P\Lambda P^{-1}, \tag{3.74}$$

and so

$$\lim_{n\to\infty}(I-B)(I+B+B^2+B^3+...+B^n) = \lim_{n\to\infty} I - B^{n+1} = \lim_{n\to\infty} I - P\Lambda^{n+1}P^{-1}. \tag{3.75}$$

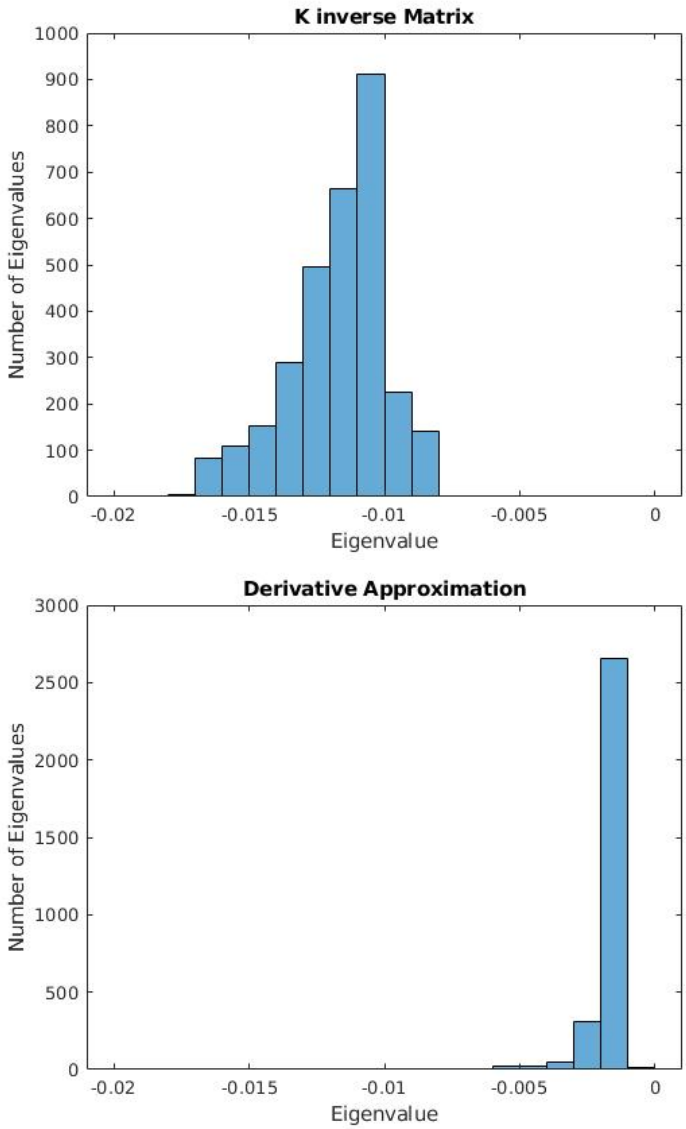


Figure 3.2: A plot containing the eigenvalues of K^{-1} and the approximating mass matrix. The eigenvalues being dissimilar provides an explanation for the lack of efficiency of this preconditioning attempt.

Because B has eigenvalues strictly less than 1, we have that

$$\lim_{n\to\infty} \Lambda^{n+1} = 0. \tag{3.76}$$

This inverse, however, must be truncated at some point, giving an approximate inverse

$$A_n^{-1} = \sum_{i=0}^{n} B^i. \tag{3.77}$$

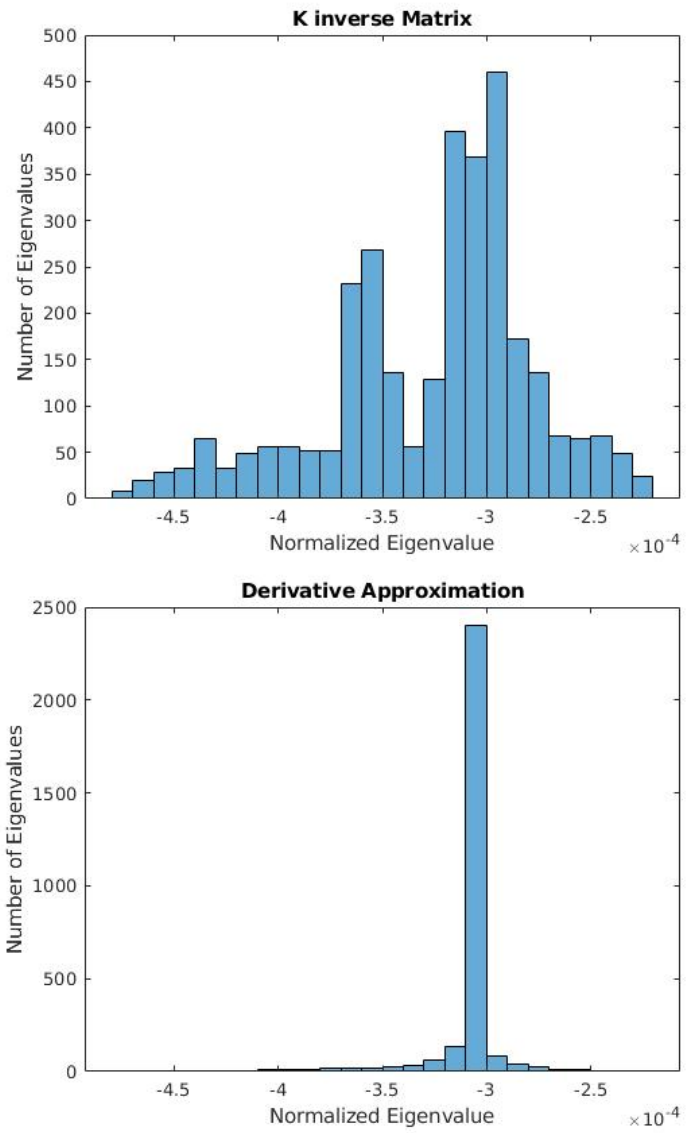


Figure 3.3: A plot containing the normalized eigenvalues of K^{-1} and the approximating mass matrix. The normalization allows for an easier comparison of eigenvalues than the plots of the actual eigenvalues given in Figure 3.2

This approximate inverse is often used with a GMRES solver, as the small eigenvalues are quickly clustered around 0, forcing a fast convergence.

I attempted to use this preconditioner on the matrix K discussed above, rewritten

$$K = (-GD_m^{-1}H - D_m)^{-1} = D_m^{-1}(I - (-GD_m^{-1}HD_m^{-1}))^{-1}. \tag{3.78}$$

The matrix $-GD_m^{-1}HD_m^{-1}$ has eigenvalues that are mostly less than 1, so it is reasonable to suspect that several of these eigenvalues would be successfully clustered by some order of preconditioning.

After constructing such a preconditioner, we can see the number of GMRES iterations still required to find the inverse of K using a given number of terms in the approximate inverse. The number of required terms for a variety of problems is given in Figure 3.4.

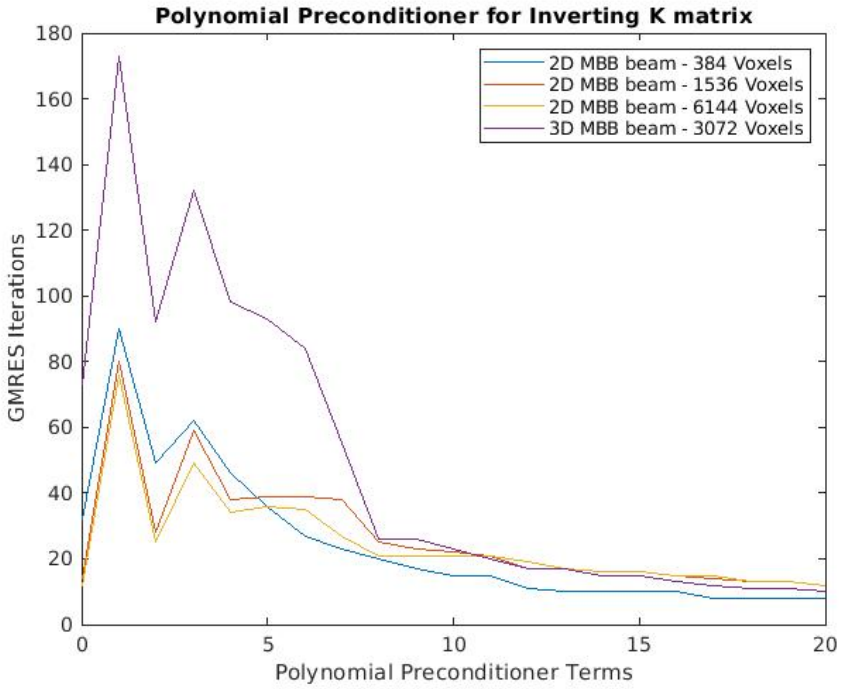


Figure 3.4: A graph of improvement in GMRES iterations needed to invert the K matrix compared to number of polynomial preconditioner terms.

This figure shows that the eigenvalues are in fact clustered by this preconditioner. However, for a preconditioner with n terms, each solution step takes an additional n matrix-vector multiplications. This preconditioner then does not reduce necessary computations, as the number of necessary multiplications actually increases.

After looking at approximate solves of larger systems as discussed in Section 3.9, we will see that I use as many as 1000 FGMRES iterations to solve for K here. However, the trend continues even in these large systems that the polynomial preconditioner does not reduce the computational

complexity of the problem. As such, I do not use a preconditioner for this intermediate step, and rely on GMRES by itself to approximately solve for the inverse of K.

3.7.5 Multigrid and Approximate Solves

A necessary step for efficiently calculating K^{-1} is to quickly approximate A^{-1}. There is ample evidence to suggest that multigrid methods are the method of choice for quickly finding a solution to the linear elasticity equation. This can be done using a small number of Conjugate Gradient steps with a Multigrid preconditioner.

Multigrid methods are a very old and rather intuitive method for preconditioning elliptic PDEs. They were first introduced in the 1960s in [45], and was thought of as an extension on pre-existing relaxation techniques. In the time since, multigrid preconditioning has earned enough attention to be a very well-studied subdiscipline in its own right. As computers became more powerful and more widely used in mathematical simulations, these methods became even more popular. I will discuss three versions of this method which could be used for this technique, and give results for the two I implemented in some detail here.

Geometric Multigrid

Multigrid methods have been widely used since the 1980s in numerical contexts such as ODEs. It has since gained great popularity as a preconditioner to parabolic PDEs. The first multigrid methods were all geometric, and matrix-based.

At a high level, multigrid methods pair well with iterative methods because it separates the reduction of high frequency errors with the reduction of low frequency errors. The low frequency errors can be reduced very quickly on a coarse mesh, and the remaining high frequency errors can be solved with only a few iterative steps on the finer mesh.

A brief description of the main steps of a basic 2-grid V-Cycle algorithm are given below:

1. Smoothing

   Krylov subspace methods, and iterative methods in general, reduce high-frequency error

much more efficiently than low-frequency error, and so have an effect of "smoothing" the residual.

2. Restriction

   Once the fine grid has been smoothed, the domain is remeshed to a coarser grid. This has the effect of making the low frequency part now appear as a high frequency error on this grid.

3. Smoothing

   Now that the low-frequency error appears to the iterative solver as being high frequency, applying a smoother once again can give an approximation of the error on the coarse grid

4. Interpolation and correction

   The coarse-grid error can now be interpolated onto the fine-grid solution to correct the low-frequency error on the fine grid.

5. Smoothing

   A final smoother is used to the corrected solution on the fine grid.

The algorithm presented above is a 2-grid V-cycle as it goes from the fine grid to the coarse, and then returns to the fine grid. This can be done with any number of coarser grids, finding the error on each subsequent level before returning back to the finest grid, by smoothing one level at a time. Other cycle types, such as the W-cycle, have been shown to be even more effective in practice. These cycle types spend more time on the coarser grids which are faster so wrk with, and do a better job reducing the low-frequency error quickly, and are therefore faster than the straightforward v-cycle. As such I use a W-cycle in all multigrid applications in my solution.

Algebraic Multigrid

Algebraic multigrid (AMG) methods are known for their ease-of-use in completing a linear solve. AMG uses algebraic methods to essentially infer a "mesh" from a given matrix. These methods were created because there was a realization of the effectiveness of geometric multigrid to solve ellipitic PDEs. However, they require much more information about the problem than just the matrix to be inverted, such as the grid, and some local operator. The grid the problem is defined on must also be somewhat regular and have a straightforward way to make a more coarse grid from

the original problem. The desire for such an effective solver on systems where less information could be known generated a desire for an algebraic (or more rarely abstract) multigrid solver.

The first algebraic multigrid was created in 1982 to solve a Laplace equation on an arbitrary mesh Brandt, McCormick and Ruge 1982a. All future AMG methods are extensions on this original that are largely based on heuristics.

Algebraic multigrid solvers are in general designed to solve the system $Ax = b$, where A is a symmetric, semi-positive definite matrix, using some smoothing operator R such as Gauss-Seidel or a Jacobi iteration. It is assumed that the smoother will quickly remove some domain of error. This easily removed error is considered to be the space of "Algebraicly high frequencies." At this point, the job of the algebraic multigrid method is to find a sequence of "algebraicly coarse subspaces" that would work well with the smoother. A set of coarse spaces is "good" if an arbitrary vector in the original space can be written as a linear combination of vectors that are algebraicly high frequency on at least one space, either the full space or one of the new coarse subspaces in the sequence. A well made description of these sorts of methods that delves deeply into the creation of two-level algebraic multigrid methods by Xu et al. can be found at [46].

Very often, many of these subspaces are generated using graph theory and by analyzing the locations of coefficients in the sparse matrix. The exact methods for generating these coarse spaces is not a necessary discussion for this thesis, but many advancements have been made in AMG methods in creating very efficient solvers.

Because so little extra information is needed for these systems, and because it can be easily applied to even irregular meshes, algebraic multigrid is widely used in many linear algebra packages. For my program, I used a pre-made AMG solver in the Trilinos scientific software package. This was implemented naively in a preconditioner for a Conjugate Gradient iterative solver.

Matrix-Free Geometric Multigrid

An improvement of the above strategy would be to use a matrix-free geometric multigrid (GMG) approach over the simpler to use algebraic multigrid (AMG). In recent years, the matrix-free approach has shown itself to be a more efficient option than either given above. In particular,

Zenodo showed in [47] that Matrix-free methods are faster at all levels of scaling for their test problem.

Geometric Multigrid in this context contains an added challenge of determining a "density" for the elements of each coarse mesh. In my implementation, I simply used an average density value. This complication does slow down my implementation of matrix-free GMG when compared to a problem that has no variable density. However, this matrix-free GMG still greatly outperforms AMG.

Using this technique allows me to find an approximate value for K by using GMRES to a fairly low level of accuracy. This allows my preconditioner to work quickly, even for large problems, while still effectively lowering the number of FGMRES iterations of the outermost linear solve. However, any decrease in the accuracy of interior preconditioner solves will result in an increase of required outer iterations. It is necessary to determine a strategy for how accurately the A^{-1} and K matrices should be determined for an efficient preconditioning.

My implementation works as follows: First, we recall that the elasticity equation, discussed in Section 2.4, can be written in its weak form as

$$\begin{aligned}\frac{\mu}{2}\int_\Omega \tilde{\rho}^p(\partial_i \mathbf{v}_j + \partial_j \mathbf{v}_i)(\partial_k \mathbf{u}_l + \partial_l \mathbf{u}_k)(\delta_{i,k}\delta_{j,l} + \delta_{i,l}\delta_{j,k}) + \lambda\int_\Omega \tilde{\rho}^p(\partial_i \mathbf{v}_i)(\partial_k \mathbf{u}_k) \\ = \int_\Omega \mathbf{f}_i \mathbf{v}_i + \int_\Gamma \mathbf{t}_i \mathbf{v}_i\end{aligned} \tag{3.79}$$

Once disctretized using

$$\mathbf{u} = \sum_n a_n \phi_n, \qquad\qquad \mathbf{v} = \sum_m b_m \phi_m,$$

the system can be re-written as $A\mathbf{a} = \mathbf{f}$ (where the vector of coefficients f does not appear as this equation must be true for all $\mathbf{v}$, and therefore for all possible $\mathbf{f}$). This matrix A can be written

$$\begin{aligned}A_{m,n} = \frac{\mu}{2}\int_\Omega \tilde{\rho}^p(\partial_i \phi_{mj} + \partial_j \phi_{ni})(\partial_k \phi_{ml} + \partial_l \phi_{nk})(\delta_{i,k}\delta_{j,l} + \delta_{i,l}\delta_{j,k}) \\ +\lambda\int_\Omega \tilde{\rho}^p(\partial_i \phi_{mi})(\partial_k \phi_{nk}).\end{aligned} \tag{3.80}$$

Subdividing my domain into individual voxels E such that $\Omega = \sum_r E_r$, and using the fact that my densities are discretized using a voxel-wise constant scheme - that is, on an element E_r, $\tilde{\rho} = \tilde{\rho}_r$ is constant - this gives

$$A_{m,n} = \sum_r \frac{\mu}{2} \int_{E_r} \tilde{\rho}_r^p (\partial_i \phi_{mj} + \partial_j \phi_{ni})(\partial_k \phi_{ml} + \partial_l \phi_{nk})(\delta_{i,k}\delta_{j,l} + \delta_{i,l}\delta_{j,k}) \\ + \lambda \int_{E_r} \tilde{\rho}_r^p (\partial_i \phi_{mi})(\partial_k \phi_{nk}). \tag{3.81}$$

Using a Q_1 finite element scheme, the vast majority of test functions are zero-valued in each cell. This means that on each cell, very few test functions need even be considered. Furthermore, the multiplication $A\mathbf{a}$ can be written

$$(A\mathbf{a})_m = \sum_r \sum_n \mathbf{a}_n \frac{\mu}{2} \int_{E_r} \tilde{\rho}_r^p (\partial_i \phi_{mj} + \partial_j \phi_{ni})(\partial_k \phi_{ml} + \partial_l \phi_{nk})(\delta_{i,k}\delta_{j,l} + \delta_{i,l}\delta_{j,k}) \\ + \lambda \int_{E_r} \tilde{\rho}_r^p (\partial_i \phi_{mi})(\partial_k \phi_{nk}) \tag{3.82}$$

This allows us to quickly find the multiplication $A\mathbf{a}$ without ever constructing the matrix A. This is especially powerful in multigrid, as it requires successive construction of matrices for successively coarser meshes.

To be able to use this setup in multigrid, there must also be an interpolation scheme. Moving from four voxels to one, the restriction used for Q_1 elements would be to examine only the values at each of the corners of the new voxel, and using those values as coefficients for the new test functions as seen in Figure 3.5. An example of the interpolation process is given in Figure 3.6. Similarly, the interpolation and restriction schemes for Q_1 elements are given in Figures 3.8 and 3.7, respectively. The density must also be interpolated, and to do this I simply average the values of the densities in the more refined voxels. A similar method is used by ASPECT in [48]. However, due to the nature of their fluid dynamics problem, a harmonic mean has been shown to be more accurate in their problem than the arithmetic mean I use here. Using this method, I can consistently

reduce error by over an order of magnitude through 2 V-cycles of multigrid, as shown in Figure 3.9.

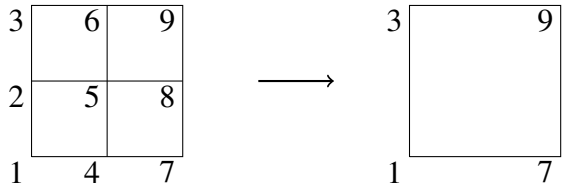


Figure 3.5: This diagram demonstrates how the restriction step of geometric multigrid is applied to a Q_1 element.

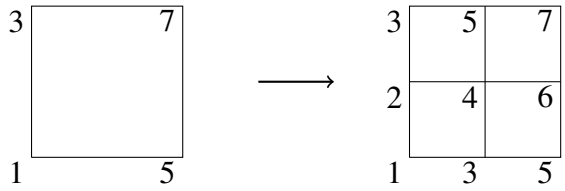


Figure 3.6: This diagram demonstrates how the interpolation step of geometric multigrid is applied to a Q_1 element.

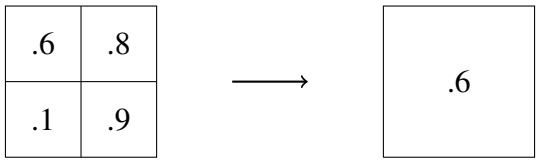


Figure 3.7: This diagram demonstrates how the restriction step of geometric multigrid is applied to a Q_0 element.

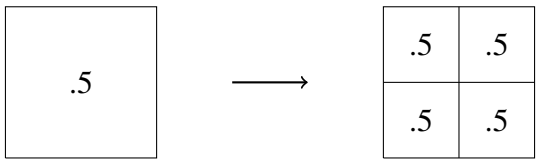


Figure 3.8: This diagram demonstrates how the interpolation step of geometric multigrid is applied to a Q_0 element. Note that this step is never actually performed in my multigrid implementation.

Further work could be done in this area to create a more precise homogenization of the coarsened grid. Based on the underlying grid pattern, the material in the coarsened grid may be anisotropic, and so the stress-strain tensor, C, could be modified to improve solve times. However, this would likely only increase speed by a small amount. Homogenization using this technique is beyond the scope of this thesis, but an outline of similar techniques can be seen in [49].

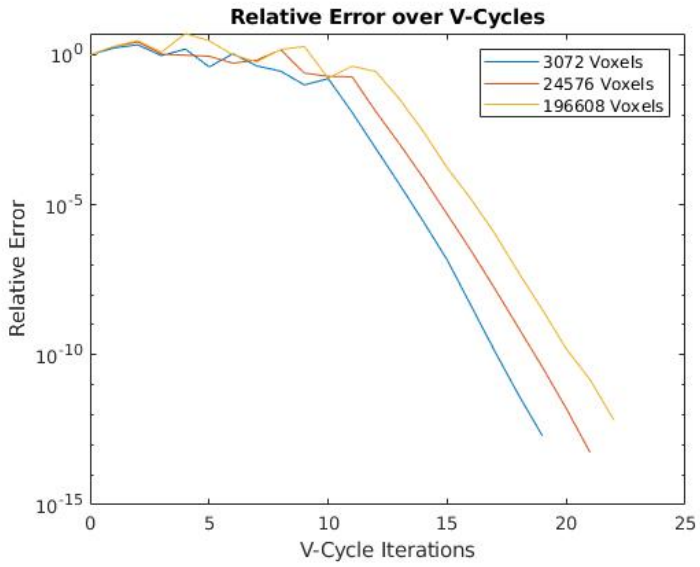


Figure 3.9: The residual error compared to the number of V-cycles in an elastic solve of the three-dimensional MBB problem using random densities on each voxel.

To fully understand the effect of these preconditioners, I conduct a short study to demonstrate the difference between matrix-free geometric multigrid, algebraic multigrid, as well as a solve without a preconditioner for comparison. I solve the elasticity equation on my MBB beam with uniformly randomly generated unfiltered densities between 0 and 1. This is solved using different numbers of refinements, and using different numbers of MPI processes.

A natural question to ask of this computationally expensive problem is whether the use of multiple processors can grant us any speedup, and if so, to what degree the speedup will occur. This speedup is measured in two ways - strong scaling and weak scaling.

- Strong scaling asks the question: "For a given problem, how much faster can I compute the solution as I add more processors?"
- Weak scaling asks: "As I make a problem more computationally expensive, how much do added processors help offset the added cost of the problem?"

Both of these are useful ways to analyze the efficiency of algorithms for use with multiprocessing. The PDE community tends to favor weak scaling as a method to measure parallel computing as opposed to strong scaling, as they are more concerned with solving a few large problems quickly, rather than solving many small problems arbitrarily quickly.

I begin by looking at my smallest scale three-dimensional example, where there are 3072 voxels, and 11907 degrees of freedom in the elasticity problem. I examine the solve time using a conjugate gradient solver with no preconditioner, with Trilinos' built-in algebraic multigrid preconditioner, and with my own, simply homogenized, matrix-free geometric multigrid preconditioner as seen in Figure 3.10. This graph shows time just for the solve, and does not include matrix assembly, initialization of the preconditioners, or any other necessary step. This makes sense for the purposes of this algorithm as the elasticity matrix will need to be inverted potentially several hundred times in the solve of the higher levels of preconditioners, but the setup will only need to be done once. The repeated inversion typically takes over 99% of the solve time, and so solving the system quickly is of utmost importance. Because of the number of times this needs to be solved, it becomes evident that strong scaling will be important for this linear solve. In other words, I need to minimize the amount of time an individual solve takes at a constant problem size. For this test, random densities are assigned to each voxel, which are filtered using the density filter given in Section 2.2.

The strength of the matrix-free geometric multigrid is demonstrated in this graph. A general rule for the number of processors to use for PDE computations and solves would be to use one processor for every 50,000 degrees of freedom. However, this problem, with only 11,907 degrees of freedom, speeds up through having 15 processors, and at all numbers of processors, the matrix-free geometric multigrid outperforms both of the competitors. The fastest solve time is more than 5 times as fast as the single-processor time.

The same test done on 4 refinements, (see Figure 3.20) giving 24576 voxels and 84099 degrees of freedom shows even more promise. A speedup of 20 times is given by using 20 times as many processors, and once again the matrix-free geometric multigrid is consistently the fastest. By using 20 processors, the system solve can be solved in a comparable time to the 3072-voxel system (.537 seconds as compared to .474 seconds). Using 10 processors, this takes less than twice the time of the smaller system, even though it has 8 times as many degrees of freedom.

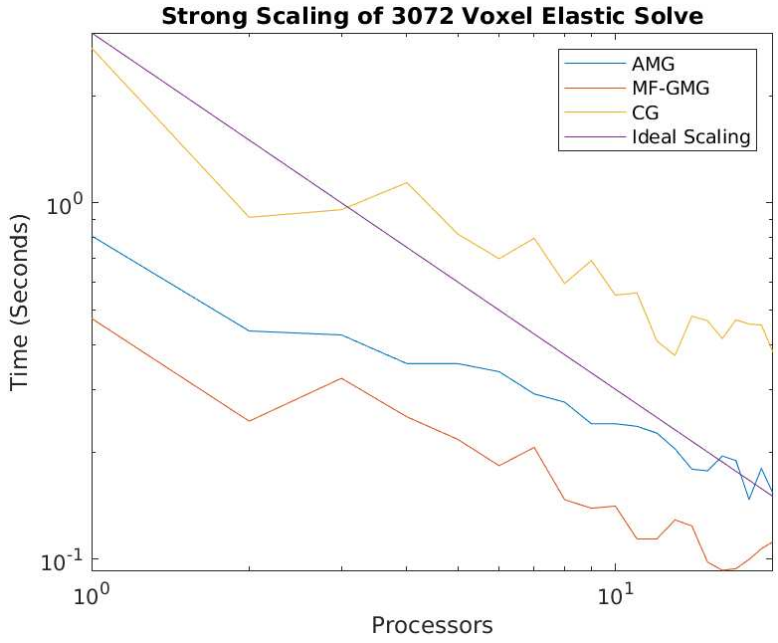


Figure 3.10: Strong scaling of preconditioners of the elasticity equation applied to the MBB Beam domain using 3072 voxels.

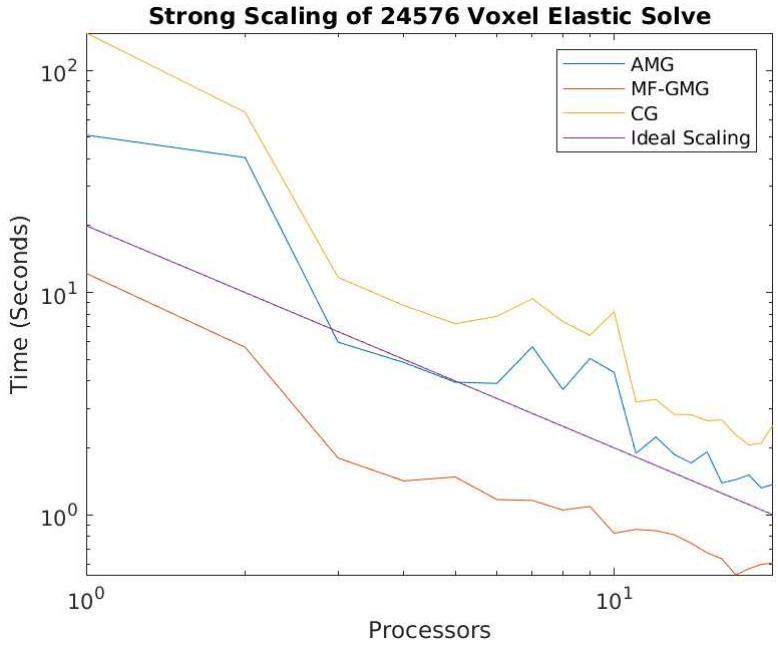


Figure 3.11: Strong scaling of preconditioners of the elasticity equation applied to the MBB Beam domain using 24576 voxels.

Finally, I also performed this study with 5 refinements and 196608 voxels as seen in Figure 3.12. Here, we see more of the same, where the problem strongly scales very well, and as the problem gets larger, there is even better strong scaling.

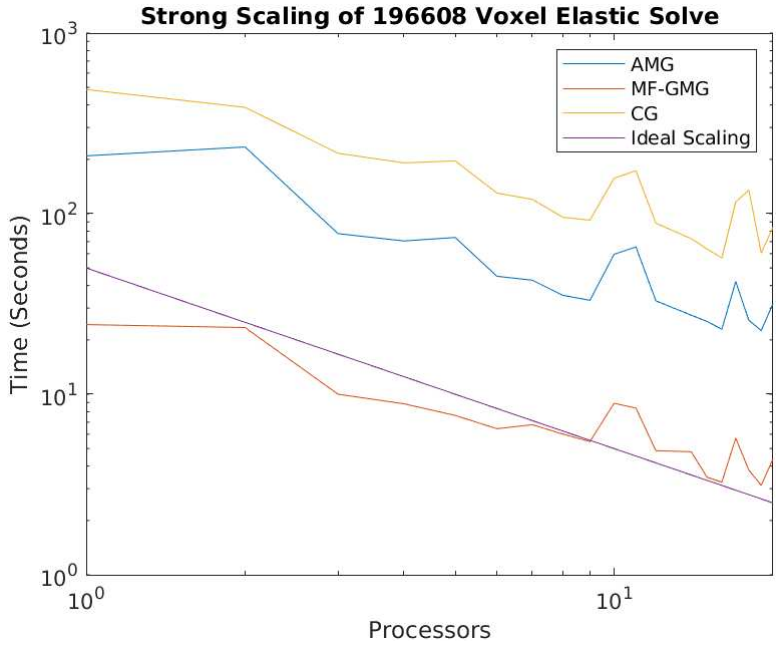


Figure 3.12: Strong scaling of preconditioners of the elasticity equation applied to the MBB Beam domain using 196608 voxels.

Combining all of these results, we can consider the weak scaling of the problem. In this case, the weak scaling results of the problem are very good – using sufficiently many processors, problems that are 8 times as big can be solved in less than 8 times the amount of time. This works especially well in the MF-GMG case, as seen in Figure 3.13.

This section demonstrated the well-behavedness of the elasticity equation, even when using random "densities" on each voxel. In the next section, I examine the K^{-1} matrix, given in Equation (3.78), and examine our ability to strongly and weakly scale the solve.

3.8 Strong and Weak Scaling of the K^{-1} Matrix

The K^{-1} matrix is the most complicated portion of my preconditioning setup. As seen in Section 3.7.4, this takes the from

$$K^{-1} = -FD_8F^TD_m^{-1}(B - C^TA^{-1}E - E^TA^{-1}C) - D_m. \tag{3.83}$$

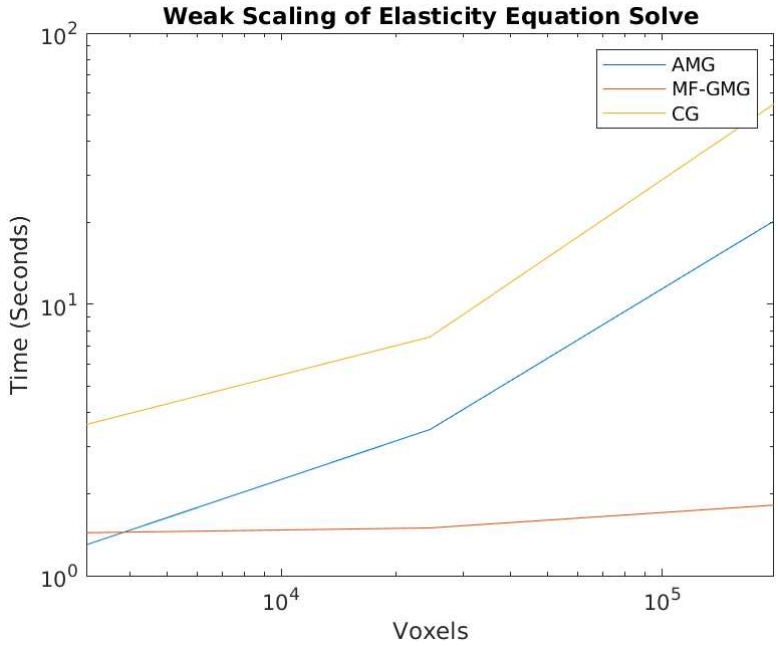


Figure 3.13: Weak scaling of preconditioners of the elasticity equation applied to the MBB Beam domain at various resolutions. Performed using 1 processor for the 3072 Voxel problem, 8 processors for the 24576 voxel problem, and 64 processors for the 196608 voxel problem. Ideal weak scaling would result in constant time for all problem sizes.

I have attempted several preconditioners in conjunction with inverting this expression, but they have been largely ineffective. Despite this, in this section, I discuss the weak and strong scaling properties of performing this inverse using a straightforward FGMRES iterative solve.

In Section 3.9 below, I show that solving the entire matrix is more effective when using a fixed number of iterations of my MF-GMG than when inverting the elasticity matrix to a certain accuracy. The K^{-1} expression has multiple A^{-1} stiffness matrices contained inside of it. To that end, I consider how the number of MF-GMG iterations used in the approximation of A^{-1} affects the K^{-1} solve. Figure 3.14 demonstrates that more A^{-1} iterations typically lead to fewer FGMRES iterations being needed to invert K^{-1} to some desired accuracy. However, this is a case of diminishing returns. In the 3072 voxel case, using 20 or 100 MF-GMG iterations results in the same number of FGMRES iterations to invert K^{-1}. However, as the MF-GMG solve is the most expensive part of a multiplication by K^{-1}, the 100 iteration case takes nearly 5 times as long per FGMRES iteration.

Importantly, when increasing the refinement of my mesh, the number of K^{-1} iterations required to achieve the same level of convergence increases dramatically, from ~ 60 to ~ 1400, an increase by over a factor of 20. Compounded with the inevitable increased complexity, this means that the weak scaling of this problem will inevitably be poor.

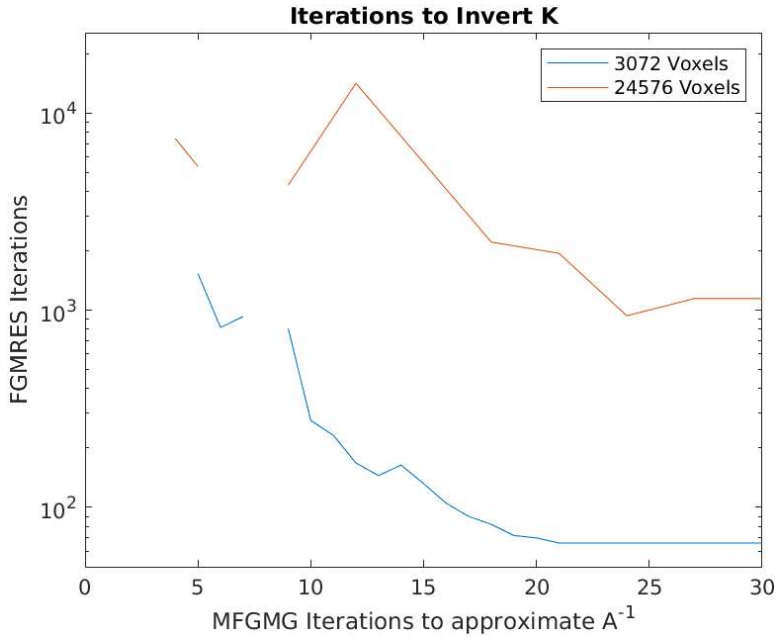


Figure 3.14: Number of FGMRES iterations needed to invert K^{-1} for various numbers of MF-GMG iterations to approximate the inner A^{-1} for various numbers of voxels. Some parts of this plot are missing due to failed solves when A^{-1} is solved to low accuracy.

Starting the discussion of strong scaling of K^{-1} with Figure 3.15, it is abundantly clear that this inverse scales very strongly. In fact, the difference between using 1 and 15 processors in the 3072 voxel version is a speedup of more than 13 times. In the 24576 voxel problem, similar results are observed.

The issue comes when we begin looking towards weak scaling. The number of iterations of MF-GMG needed to solve my system to a particular accuracy stayed almost constant regardless of the problem size. The effective weak scaling of this part of the solve is possible because of this near-constant iteration number. Unfortunately, this consistency does not remain for the K^{-1} matrix. Moving from the 3-refinement, 3072 voxel problem to the 4-refinement, 24576 voxel

problem, the minimum number of iterations I found to successfully invert K^{-1} to an accuracy of 10^{-12} went up from 69 iterations to 1100 iterations – this is an increase by a factor of around 16. Interestingly, the complexity of the solve does not increase nearly as much between the 24576 and 196608 voxel version, only requiring 23% more FGMRES iterations in this case.

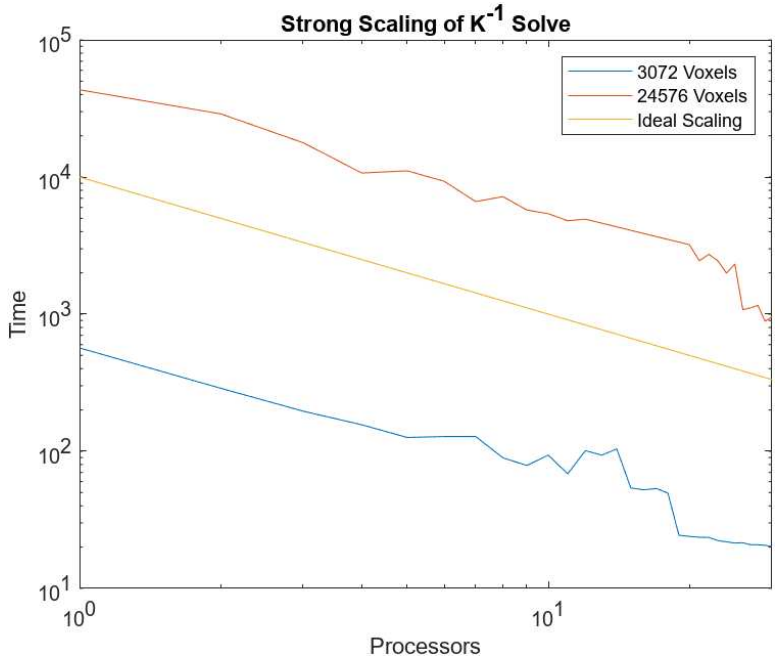


Figure 3.15: Strong scaling of the solve of the K^{-1} matrix for the MBB Beam domain using various numbers of voxels.

Table 3.1: Weak scaling of the K^{-1} matrix for the MBB Beam domain. Note the very poor weak scaling mostly occurs between 3072 and 24576 voxels, and essentially all of the poor scaling is a result of the vastly increased number of FGMRES iterations needed to complete the solve.

Refinements	Voxels	Processors	Solve Time	FGMRES Iterations	Time Per Iteration
3	3024	1	567	69	8.217
4	24576	8	7200	1100	6.545
5	196608	64	10400	1356	7.669

3.9 Tiered Preconditioning with Matrix Free Geometric Multigrid

The goal of the preconditioning schemes presented thus far is to solve my Newton-Raphson method matrix to some relative tolerance a_1 governed by the Eisenstat-Walker Method, discussed in Section 3.7.2. In doing so, I must apply the linear transformation given by the matrix K given above to some relative tolerance a_2. In doing this, I must apply A^{-1} to some relative tolerance a_3. In this section, I perform a study to determine these accuracies a_2 and a_3.

For this study, I consider two methods of determining termination of the solves of A and K, and also consider the interplay between solver types for K and my overall block system. First, I will use a relative tolerance cutoff for each solve, where the solver will take as many iterations as necessary to reach the given tolerance. Secondly, I will use a set number of iterations for each solve. Both of these methods will be tested when using both GMRES and FGMRES for the K solver. After finding that using GMRES for the K matrix was competitive with using FGMRES (see study below), I also tested the possibility of switching the system solve from FGMRES to GMRES, although this was wholly not useful. For completeness, the test solve was performed both to a relative accuracy of 10^{-6} and 10^{-12}. This allows for a consideration of changing these parameters as the necessary accuracy from the Eienstadt-Walker algorithm (see Section 3.7.2) changes. My test case is a relatively coarse version of the three-dimensional MBB Beam discussed in Section 1.5, whose design domain contains 3072 voxels. The results in Figure 3.16 are validated using a more refined version of the same problem in Figure 3.20.

This study shows that solving K using FGMRES is a better option than solving using GMRES, although these options are comparable. It also provides evidence that a rather inaccurate solve of the elasticity equation gives a faster system solve, to a point. Eventually, the approximation is so poor that the preconditioner is unusable, and the linear system is not solved.

Unfortunately, in both of these cases, the parameters that give the fastest solves are immediately adjacent to parameters that fail to solve the system at all – see 33 GMRES iterations compared to

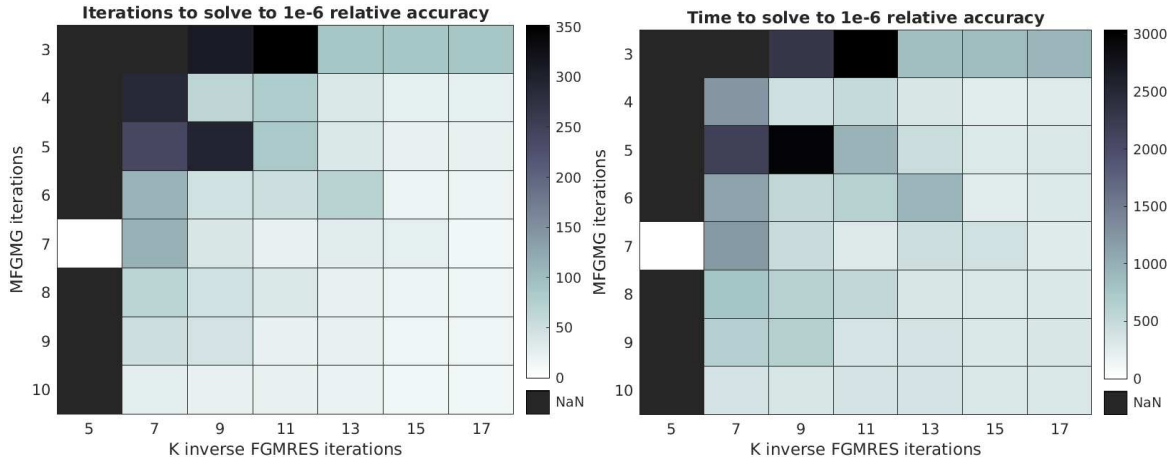


Figure 3.16: Parameter tuning results for iterations of MF-GMG, FGMRES to use to approximately invert A, K in 3072 voxel MBB Problem, when the full matrix is inverted using FGMRES to a relative accuracy of 10^{-6}. The left figure displays the number of outer FGMRES iterations needed to solve the system matrix to the required accuracy, while the right side displays the time. The number of outer iterations strictly decreases as the number of FGMRES and MF-GMG used increases, while the time initially decreases, but then increases again as the internal iterations used becomes more than needed.

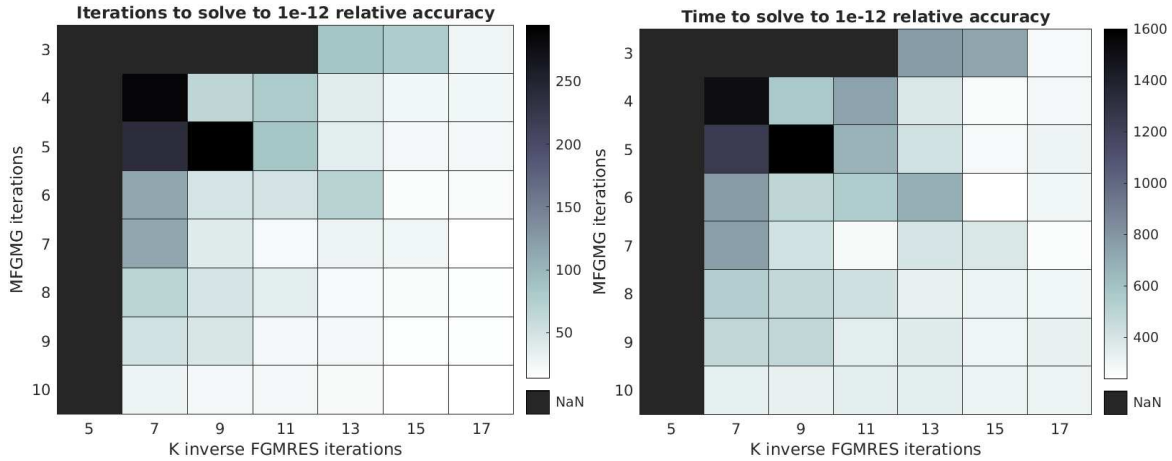


Figure 3.17: Parameter tuning results for iterations of MF-GMG, FGMRES to use to approximately invert A, K in 3072 voxel MBB Problem, when the full matrix is inverted using FGMRES to a relative accuracy of 10^{-12}. The left figure displays the number of outer FGMRES iterations needed to solve the system matrix to the required accuracy, while the right side displays the time. The number of outer iterations strictly decreases as the number of FGMRES and MF-GMG used increases, while the time initially decreases, but then increases again as the internal iterations used becomes more than needed.

11 or 10 MF-GMG iterations in Figure 3.20. This causes much uncertainty in wanting to use this algorithm of scaled up problems. The 'sweet spot' for optimizing parameters for this problem may shift, even slightly, and this could result in a failed solve. This problem becomes even more evident when the 24576 voxel problem is considered. I solve the 24576 voxels model to a relative accuracy

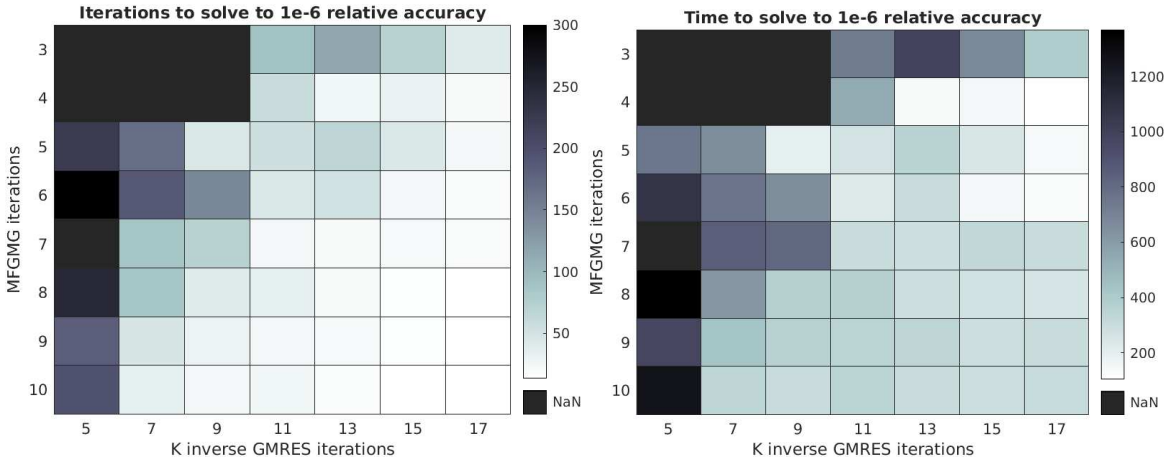


Figure 3.18: Parameter tuning results for iterations of MF-GMG, GMRES to use to approximately invert A, K in 3072 voxel MBB Problem, when the full matrix is inverted using FGMRES to a relative accuracy of 10^{-6}. The left figure displays the number of outer FGMRES iterations needed to solve the system matrix to the required accuracy, while the right side displays the time. The number of outer iterations strictly decreases as the number of GMRES and MF-GMG used increases, while the time initially decreases, but then increases again as the internal iterations used becomes more than needed.

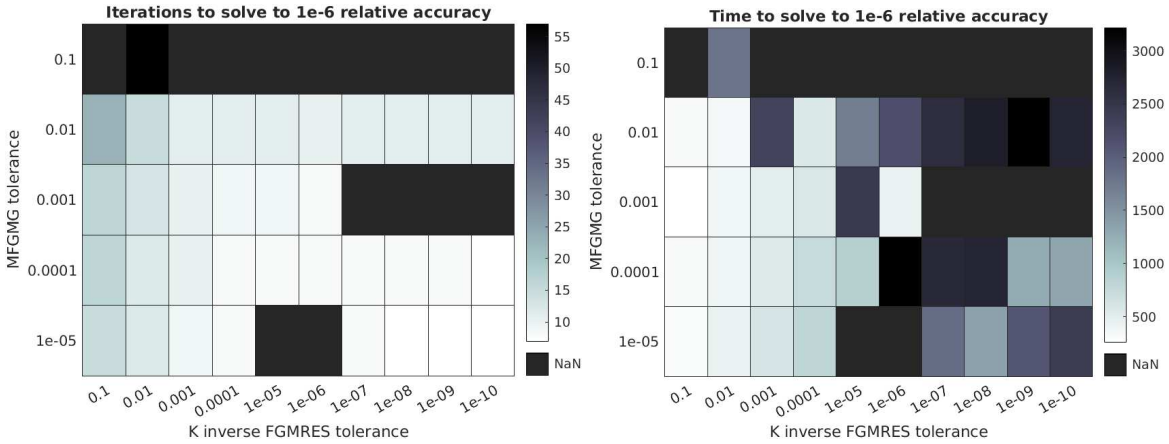


Figure 3.19: Parameter tuning results for relative tolerance of MF-GMG, FGMRES to use to approximately invert A, K in 3072 voxel MBB Problem, when the full matrix is inverted using FGMRES to a relative accuracy of 10^{-6}.The left figure displays the number of outer FGMRES iterations needed to solve the system matrix to the required accuracy, while the right side displays the time. The number of outer iterations strictly decreases as the relative tolerance of GMRES and MF-GMG becomes more strict, while the time is always quite large when K is solved very accurately.

of 10^{-12}, giving a similar pattern to the 3072 voxel case, only with more iterations necessary for the solve. The results of this experiment are given in Figure 3.20.

While the consistency does seem to translate to the larger 24576-voxel version of this problem, the number of FGMRES steps greatly increases. A useful way to consider the timing of the solve

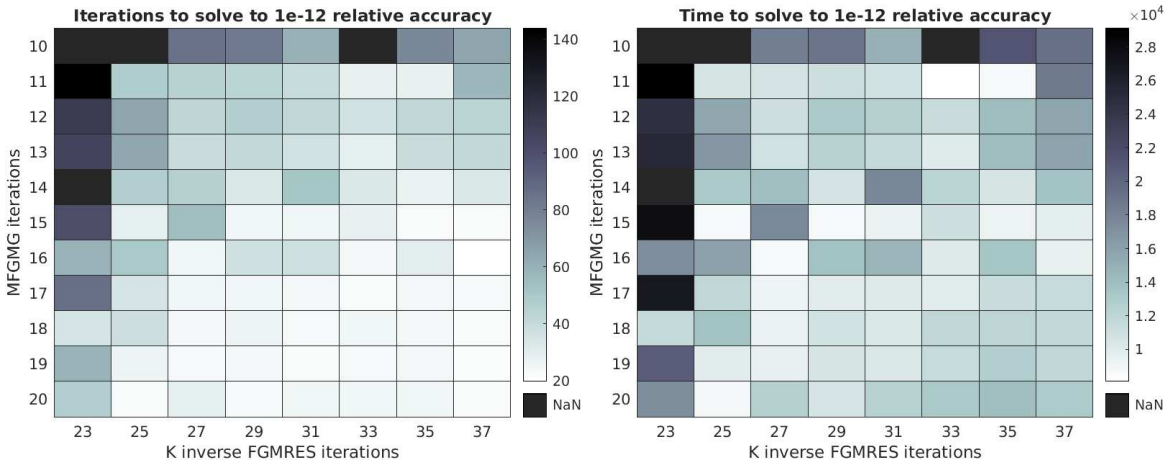


Figure 3.20: Parameter tuning results for iterations of MF-GMG, FGMRES to use to approximately invert A, K in 24576 voxel MBB Problem, when full matrix inverted using FGMRES to a relative accuracy of 10^{-12}. The left figure displays the number of outer FGMRES iterations needed to solve the system matrix to the required accuracy, while the right side displays the time. The number of outer iterations strictly decreases as the relative tolerance of GMRES and MF-GMG becomes more strict, while the time is always quite large when K is solved very accurately.

is to examine the number of multigrid iterations on A that must be completed over the course of the whole solve. This is useful because the vast majority of the time for the solve is spent within the multigrid solve. In the 3072-voxel version, the minimal number of MF-GMG iterations is $7 \times 24 \times 22 = 3,696$. This does not bode well for weak scaling of this problem, and further encourages the development of a preconditioner for this step.

The second result of this study is that using a fixed number of iterations of the GMRES for the K system produces faster system solves than solving K to some relative accuracy. This can be interpreted with the same intuition as given for the speed of using GMRES. A more consistent underlying preconditioner can allow the outer FGMRES to more efficiently find the optimal solution.

3.10 Parallelization Considerations

An advantage of iterative solvers and sparse matrices is that I can use well-developed libraries that already implement their parallelization. In fact, most aspects of this algorithm are already well-made for parallel implementation. A notable exception to this is the density filter matrix.

The use of a "filter radius" means that information may be required from cells that are not just neighboring "ghost cells," and in fact information may be needed from a processor that does not own any neighboring cells. To circumvent this issue, I currently store information about all cell centers in vectors, and use these fully replicated vectors to determine filter information. While this is not a fast process, it only needs to occur once at the beginning of the solve (and potentially again after a mesh refinement), and so it is a rare enough event that it does not become an impediment to solving.

3.11 Problem Solution

The results of the combination of these algorithms to give a MBB beam is much as expected, and they align with the results achieved from other methods. In Figure 3.21, I show the results of both the filtered an unfiltered densities of the two-dimensional MBB Beam problem. There is a clear issue here – which of these is the solution we should use? After all the work to ensure a solution that has only values 0 and 1, our choices are now either a (clearly not viable) 0-1 solution, or a solution that is clearly not 0-1. The solution here must use the filtered version of the density, even though it is clearly not purely black and white. In actual applications, a differentiable approximation of the heaviside function can be applied to the filtered density, giving an almost 0-1 solution that is likely to be viable and optimal. As mentioned in Section 2.2, this pseudo-heaviside function can be used in the problem formulation, giving a 3-field density setup. An issue with these approaches is that functions that are closer to a true heaviside function are less well approximated with first and second derivatives, which makes the problem more difficult to solve using gradient-based approaches. In addition, the solution will still not give a true 0-1 solution, and so post-processing is still required. Whether the benefits then outweigh these costs is an open question. Several algorithms exist, and with growing popularity, that avoid this issue by forcing the solution to be 0-1. Two of these, the ESO, and BESO, are described in the appendix.

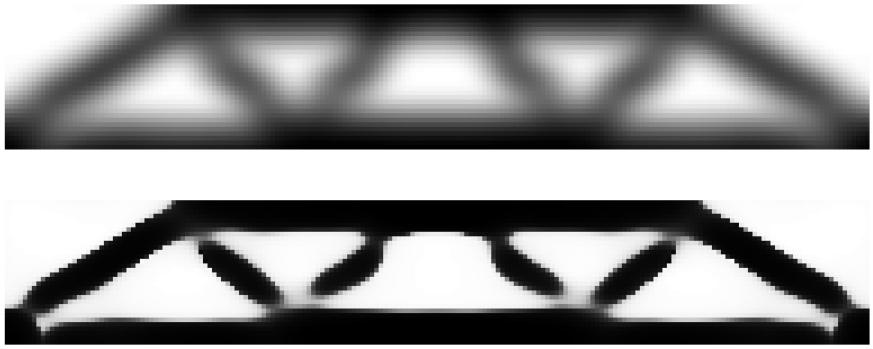


Figure 3.21: The filtered and unfiltered densities for a solved MBB problem.

3.12 Summary of Chapter

In this chapter, I have transformed the problem given in Equation (3.4) into a computationally solvable, discretized problem, and have created and tested various methods to attempt to speed up the solving of this problem.

Using an optimize-then-discretize approach, I first used Gateaux derivatives to find first and second derivatives of the Lagrangian. This gave me a system of PDEs to solve that would give a Newton step, as shown in Section 3.1.

Then, I discussed various methods for reducing the barrier given from the interior point scheme. Reducing this barrier at the correct rate allows us to maintain quadratic convergence near the solution. While I discuss many such schemes in Section 3.3, I select an appropriate adaptive scheme that is globalized using a mixed monotone method.

I then discuss necessary steps for Newton's method to be made global, and also to avoid a common pitfall in Newton methods called the Maratos effect. I select a watchdog method in Section 3.5, along with a Markov filter in Section 3.4, the most aggressive methods available for avoiding these pitfalls, in order to give the best chance at a speedy solve.

With these techniques researched and implemented, I have a novel nonlinear algorithm for solving a SAND interior point topology optimization problem. While all of these methods have been used on their own for some time, and are all well researched, the implementation and combination

of them, especially in this context, constitutes a new algorithm that provides new possibilities for solving this type of problem.

Once I established this nonlinear optimizer, I turned my attention to what computational steps were necessary to take such a Newton step, namely discretizing using finite elements and solving the resulting large, block linear system. I broke my preconditioning technique down into three parts, a Schur complement decomposition, an approximate solve of the resulting matrix equation, and a matrix-free, simply homogenized geometric multigrid.

The Schur complement decomposition step is actually composed of four separate decompositions, given in Section 3.7.4. The structure of the matrix and orderings of the variables in the system were selected to aid in this step. Using invertible blocks along the diagonal, I reduce the system size by more than 10 times. However, this results in a large linear expression to solve.

I attempted to construct several preconditioners to speed up the solve of the large linear expression, including a polynomial preconditioner and a derivative-approximation approach in Section 3.7.4. Neither of these methods proved fruitful, and so I turned instead to tuning the approximate solve of this matrix. My approach for tuning this solve is given in Section 3.9.

Finally, I used a matrix-free geometric multigrid approach to efficiently solve the underlying linear elasticity equation. This method is shown in Section 3.7.5 .This preconditioner includes a homogenization technique for dealing with the different densities on adjacent voxels when moving to coarser grids.

Throughout this chapter, I have given consideration to strong and weak scaling of this problem. While all of these steps scale very strongly, the middle step, solving the linear expression, does a poor job at scaling weakly. Future work should focus on this step, and an effective preconditioner here would allow for much larger, faster solves with this method.

3.13 Results

At the end of this section, it makes sense to compare my solution time with that of other solvers. However, the comparison is far from simple, so I split it into two parts.

First, I want to compare how long it takes my system to solve with a straightforward FGMRES solve of the full matrix. I tested this on the 3072-vovel three-dimensional problem. The first non-preconditioned iterative solve took 932,964 iterations of FGMRES, totalling almost 100 minutes on 5 processors. This was just for the first Newton step of the nonlinear solve, which is typically much faster to solve than the following steps (as the initial displacements are zero, resulting in a large simplification of the block matrix). The second step never completed in this method, even after running for 10,000,000 iterations of FGMRES.

Secondly, there is a partially iterative approach, where the elasticity equation is inverted directly, and this inverse is saved. This allows for very fast applications of the elasticity inverse, and is quite useful in evaluating a solution. I used this to great effect until I reached the 196608 voxel problem, when the direct solver I was using deemed the problem too large to solve, at 630531 degrees of freedom. In the asymptotic case, for an arbitrarily large problem, the repeated use of multigrid can be expected to be faster than any direct solve, and so my efforts were focused on this approach.

This thesis presented algorithms to successfully precondition the outer block matrix so that only a smaller (albeit more complicated) matrix K need be inverted. I also present an effective preconditioner for the elasticity equation that sits inside this K. Unfortunately, we can see that without an effective preconditioner on this K, scalability of this algorithm is still extremely limited. In this thesis, I explored the use of a derivative approximation for this K matrix, as well as a polynomial preconditioner, but both of these methods have been shown to be ineffective. Possible future attempts at preconditioning this matrix may involve methods such as deflation, or Krylov subspace recycling, both of which may be the missing link to make this algorithm scalable. However, as is, the current algorithm still is limited in problem scale.

Chapter 4

Large Scale Results

While the majority of topology optimization researchers work exclusively on two-dimensional problems, the utility of these algorithms is most shown in larger scale three-dimensional applications. To demonstrate the applicability of the SAND algorithms, I have implemented this algorithm with a three-dimensional version of the MBB-beam problem. In this problem, I simulate a beam with unit height and depth, and a width of 6. The four bottom corners of the domain are held constant in the directions of height and depth. A downward force is applied halfway along the beam across the top. The solution is presented in Figure 4.1. There are currently no published papers using SAND algorithms with interior point optimization in a three-dimensional problem.

The solution, given in Figure 4.1, closely resembles a stripped-down I-beam. This is reassuring, as an I-beam is a structure engineers often choose to use for similar load cases. This sort of rediscovery of known structures is common in topology optimization – the 2-d MBB problem, for example, generates a truss.

4.1 Multiprocessor Scaling

For this large solve to be completed, I again look to the scaling of this system. The strong and weak scaling of the inner preconditioners of this solve are given in Sections 3.7.5 and 3.8. The scaling of the full nonlinear step is presented here. Below, the strong and weak scaling of the full algorithm, as well as of select intermediate steps in the algorithm, will be presented.

4.1.1 Strong Scaling

As stated above, strong scaling is typically not as big a concern in the finite element community, as it is generally understood and accepted that solving small problems with extremely fast wall time is not feasible in general. However, for this problem, there are interior solves that must be completed several times. For these problems, strong scaling is important. The strong scaling of

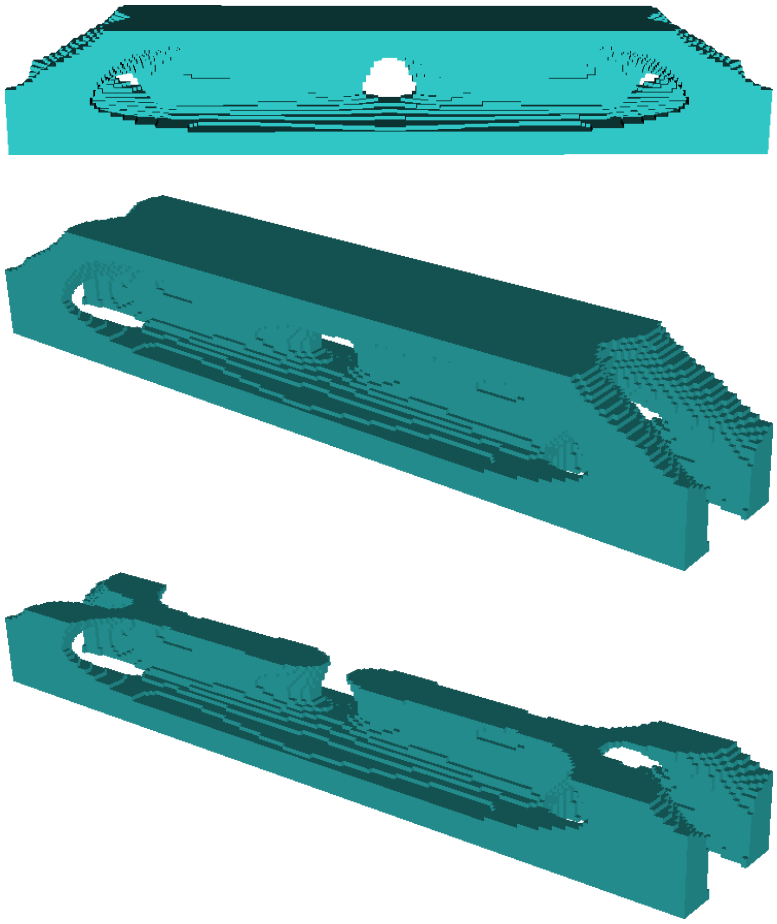


Figure 4.1: Several views of the three-dimensional MBB Beam – Front view, isometric view, and an isometric view with the "top" of the structure missing.

the elasticity equation is given in Section 3.7.5, and the strong scaling of taking a full step is given in Figure 4.2.

As we can see in Figure 4.2, even with the small three-dimensional problem, strong scaling works well with taking a full step in the nonlinear optimization problem. There is a large amount of scaling up until around 30 processors, at which point the advantage seems to go away. For larger problem sizes, we continue to have good strong scaling with even more processors.

4.1.2 Weak Scaling

Weak scaling of this problem still has many issues, largely because of the missing K preconditioner. The results of weakly scaling the inverse of the elasticity operator are given in Section 3.7.5, while some weak scaling results for the full problem are given in Table 4.1. The problem

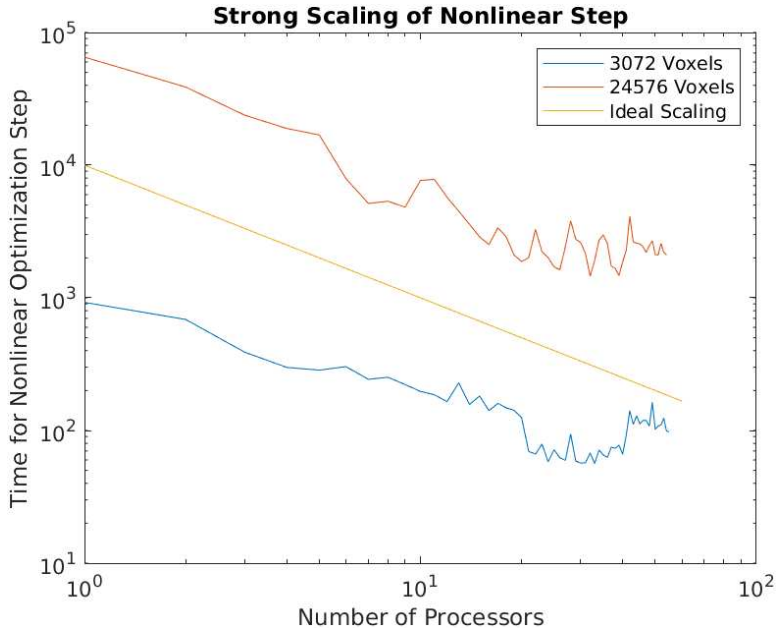


Figure 4.2: The strong scaling of taking a single nonlinear optimization step. The speedup comes directly from strongly scaling the inverse of the elasticity operator, which takes the vast majority of time on all of these runs.

Table 4.1: Weak scaling of a full step for the MBB Beam problem. Note that, unlike in the weak scaling of the K^{-1} solve, very poor weak scaling occurs not just between 3072 and 24576 voxels, but also between 24576 and 196608. This is due to the increased number of inner FGMRES iterations used to approximate K^{-1} that is necessary for convergence.

Refinements	Voxels	Processors	Solve Time
3	3024	1	924
4	24576	8	5350
5	196608	64	15200

can actually be seen by looking into the study I performed on tuning the K and A inverse solves in Section 3.9. To solve the K inverse, upwards of 10 times as many iterations were needed on the GMRES solve to have the outer system converge. The good news here is that if a preconditioner for this K matrix is found, so that a similar number of iterations would be needed independent of the problem size, then the weak scaling would be improved to levels where much larger scale problems would be feasible.

Chapter 5

Conclusion

In this thesis, I have demonstrated that the simultaneous analysis and design in interior point topology optimization framework is ripe for improvement, and I have shown that large problems can in fact be solved in a relatively reasonable timeframe, even in three dimensions. The code I developed that demonstrates these advancements can be found at [50]. I implemented several nonlinear optimization techniques that are appropriate to this problem to reduce the number of linear solves necessary to complete the nonlinear optimization, and developed preconditioners to complete these linear solves more quickly.

5.1 Summary of Results

As demonstrated in Chapter 2, interior point algorithms have a vast variety of methods that can be used on many problems, and selecting which methods to use is extremely problem specific. I worked through and analyzed these methods with a goal of solving the problem in as few iterations as possible, while still maintaining the property of being a global algorithm. This nonlinear solve is further complicated by a desire to never save an inverse of a matrix, especially of the full system. This desire comes from the lack of scalability of these directly computed inverses, which typically are full matrices and as such require memory on the order of $O(n^2)$. These self-imposed restrictions resulted in a novel algorithm for completing the solve of this Newton step.

Once a method for completing a nonlinear step was selected, the difficulty became completing the step as quickly as possible. This solve was sped up by using a multi-tiered preconditioning technique. There are three layers to this preconditioning – one for the block system, one for the resulting expression of linear transformations, and one for the complicated elasticity equation. Two of these three necessary preconditioners were shown here in detail, in Sections 3.7.4 and 3.7.5. The block system is simplified effectively with a Schur complement preconditioner, and the innermost

elasticity operator is preconditioned with a matrix-free geometric multigrid preconditioner that is homogenized to work with the variable densities inherent in SIMP problems.

5.2 Feasibility of an Interior Point Method used with Simultaneous Analysis and Design in Topology Optimization

In this thesis, and specifically as discussed in Section 5.1, I have made various improvements to the computational feasibility of interior-point SAND algorithms. These led to my ability to solve a relatively high-resolution problem in three dimensions.

Given this updated and refined algorithm that can solve three-dimensional problems in much less time than previously required, the question of possible practical usability remains. There are, however, many problems still lurking inside this algorithm that make it less than ideal.

5.2.1 Multiple Constraints

A benefit of the interior point optimization setup based on the Karush-Kahn-Tucker conditions is that a number of constraints can be simply written in the problem, and that solving an adjoint equation is unnecessary to utilize the constraint. Many common constraints in topology optimization such as constraints on the mass, stress, and compliance would be simple enough to write in this framework. The problem arises when several of these constraints need to be met at the same time. For each of these constraints, two new block rows and columns needs to be added to the system matrix, and another Schur complement decomposition would need to be added to the preconditioner.

The popular GE Bracket [**?**] shown in Figure 5.1 is a real world topology optimization case studied by a wide number of groups that has no less than 4 load cases, which would all need to have unique associated constraints. These additional constraints are not unique to this problem, and a typical real-world problem could be expected to have several constraints, which would make the solution much more difficult to find.

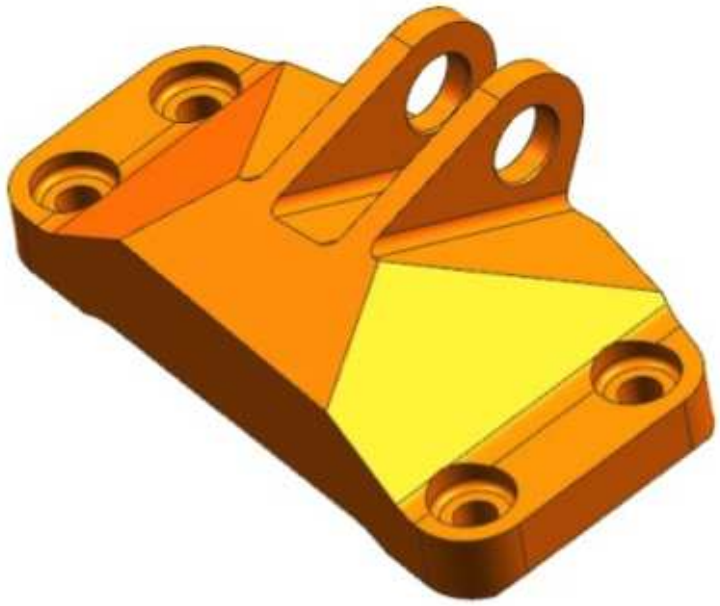


Figure 5.1: The GE Bracket design domain, as given by GE and Grabcad.com.

5.2.2 Post-Processing

Another reality of computer-generated design is the necessity of post-processing after completion of topology optimization. At a minimum, an isosurface of the density field must be made to define the part being made. The calculated density field is not, and was never intended to be, the final design. As such, having a density field that gives a lower compliance in the optimization problem does not necessarily correspond to a lower compliance in the final problem. The isosurface may even have protrusions from the shape that may form most of a truss-like structure, but because of the isosurface value, may not be complete, as shown in Figure 5.2.

Some topology optimization techniques such as BESO, as discussed in Appendix A.3 can overcome at least this isosurfacing by considering only black-and-white solutions. However, even these parts need to be smoothed after voxels are identified, and this causes a change in all responses of note, including the mass and compliance.

Other post-processing steps such as thickening minimum part size, re-meshing the part, smoothing, and removal of unnecessary partial trusses further remove the topology optimization result and the generated part design.

These post-processing steps remove much of the reliable usability of the results given in [2], which is largely the motivation for this thesis. However, short of recreating the results in the benchmark study with some common post-processing steps being added, the true benefit of these results for real-world application is rather questionable. It can be assumed that creating a more optimal result from a topology optimization problem will lead to a better design after some common post-processing steps, but I have not found any studies looking into this assumption to see how large of an effect stays in the long run.

Figure 5.2: A demonstration of a shortcoming of topology optimization algorithms that use density – if the isosurface is created at a level where part of a structure is not included, clearly non-optimal designs can be formed. This figure was generated by using a higher density cutoff for the isosurface of the result.

5.2.3 Philosophy of Generative Design

In Generative Design, any tool that aids in design generation is viewed exactly as a tool that can be used by an engineer. Practitioners of Generative Design have a goal of having a number of good, feasible designs, that can then be selected from and tweaked by the engineer using this tool. Viewed in this context, an individual generative design algorithm's speed becomes more relatively valuable when compared with its measured "best fit," as no one of these algorithmic results will necessarily even be used. Again using the GE-Bracket problem from [**?**] as an example, the real benefit of the contest was to give the team at GE a number of possible designs to consider, rather

than to give a definite way to always get the best design. Several examples of these designs can be seen in Figure 5.3.

Many companies that utilize topology optimization techniques use it as part of a larger workflow of generative design. After topology optimization is used, the model will still need to be generated with an isosurface, and then the remaining design needs to be post-processed and edited. Simultaneous analysis and design in the context of interior point optimization as a part of this larger workflow shows its weakness as the same computational power it takes to generate this design could be used to generate many designs with other methods, and this design is by no means guaranteed to be the best after post-processing.

5.2.4 Summary of Applicability of Simultaneous Analysis and Design in Interior Point Topology Optimization

This simultaneous analysis and design in interior point topology optimization algorithm, and second order algorithms in general, are typically viewed as too computationally inefficient to be useful. This thesis took steps towards making the algorithm more computationally feasible by working both on the nonlinear optimization scheme, as well as the linear solver. These steps have resulted in a sufficiently fast solve so that solutions of relatively large, 3-dimensional problems can be found, but more work is needed before this algorithm could see widespread use in generative design contexts.

5.3 Possible Future Work

The interior point SAND method is still noteworthy for its benefits of finding better minima when compared to other solvers. My thesis has demonstrated that the algorithm can be made to solve much larger problems than previously considered in a reasonable timeframe. However, before it can be successfully implemented in part of a generative design framework, more work is needed to continue the speedup of this algorithm. This work should be focused on inverting the linear expression I call K in Section 3.7.4. There are several techniques outside the scope

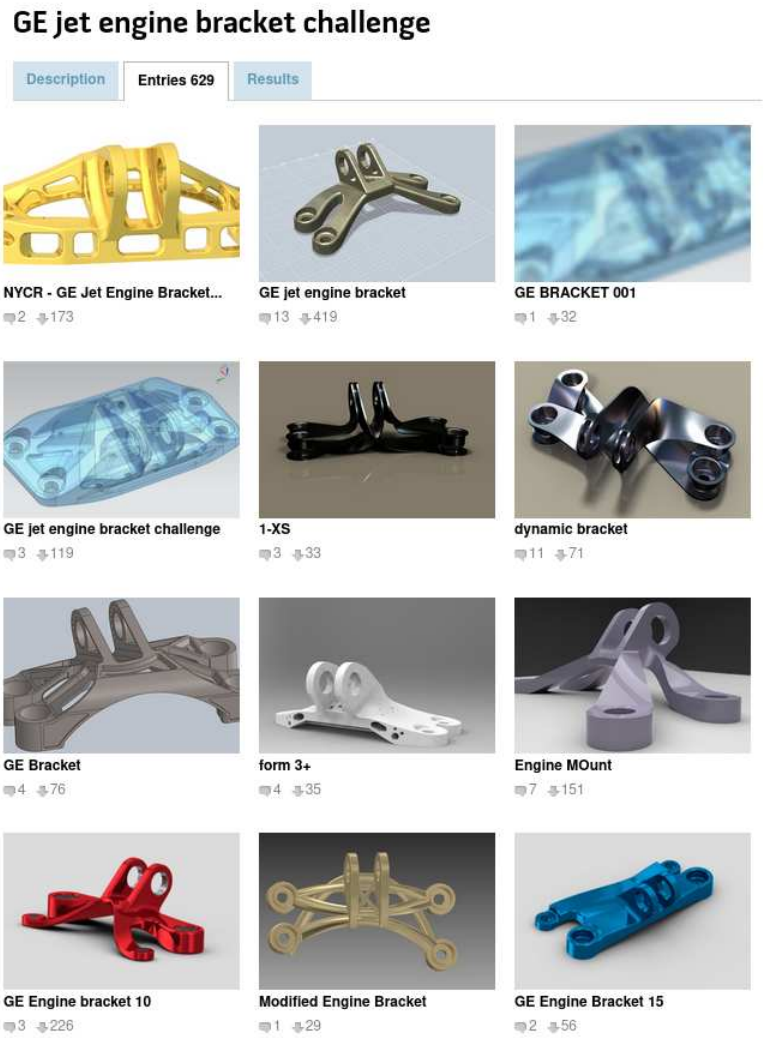


Figure 5.3: Many possible designs from the grabCAD GE bracket contest. Photograph taken from grabCAD website.

of this thesis that could be a starting place for preconditioning this expression, including Krylov subspace recycling, or deflation. Information about these techniques can be found in [51] and [52], respectively.

Bibliography

[1] M. P. Bendsøe and O. Sigmund. *Topology Optimization*. Springer Berlin Heidelberg, 2004.

[2] S. Rojas-Labanda and M. Stolpe. Benchmarking optimization solvers for structural topology optimization. 52(3):527–547, May 2015.

[3] G. N. Vanderplaats. Structural design optimization status and direction. *Journal of Aircraft*, 36(1):11–20, January 1999.

[4] S. Jung, J. Ro, and H. Jung. A hybrid algorithm using shape and topology optimization for the design of electric machines. *IEEE Transactions on Magnetics*, 54(3):1–4, 2018.

[5] W. Prager and G. I. N. Rozvany. Optimal layout of grillages. *Journal of Structural Mechanics*, 5(1):1–18, January 1977.

[6] G.I.N. Rozvany, T.G. Ong, W.T. Szeto, R. Sandler, N. Olhoff, and M.P. Bendsøe. Least-weight design of perforated elastic plates— I. *International Journal of Solids and Structures*, 23(4):521–536, 1987.

[7] M. P. Bendsøe and N. Kikuchi. Generating optimal topologies in structural design using a homogenization method. *Computer Methods in Applied Mechanics and Engineering*, 71(2):197–224, November 1988.

[8] O. Sigmund. On the design of compliant mechanisms using topology optimization. *Mechanics of Structures and Machines*, 25(4):493–524, January 1997.

[9] T. Borrvall and J. Petersson. Topology optimization of fluids in Stokes flow. *International Journal for Numerical Methods in Fluids*, 41(1):77–107, 2002.

[10] S. Das and A. Sutradhar. Multi-physics topology optimization of functionally graded controllable porous structures: Application to heat dissipating problems. 193:108775, August 2020.

[11] J. Zhu, H. Zhou, C. Wang, L. Zhou, S. Yuan, and W. Zhang. A review of topology optimization for additive manufacturing: Status and challenges. 34(1):91–110, January 2021.

[12] Y. Kuo, C. Cheng, Y. Lin, and C. San. Support structure design in additive manufacturing based on topology optimization. *Structural and Multidisciplinary Optimization*, 57(1):183–195, July 2017.

[13] R. T. Haftka and M. P. Kamat. Simultaneous nonlinear structural analysis and design. 4(6):409–416, 1989.

[14] S. Sankaranarayanan, R. T. Haftka, and R. K. Kapania. Truss topology optimization with simultaneous analysis and design. *AIAA Journal*, 32(2):420–424, February 1994.

[15] J. Rasmussen. The structural optimization system CAOS. *Structural Optimization*, 2(2):109–115, June 1990.

[16] M. P. Bendsøe. Optimal shape design as a material distribution problem. *Structural Optimization*, 1(4):193–202, December 1989.

[17] B. Bourdin. Filters in topology optimization. *International Journal for Numerical Methods in Engineering*, 50(9):2143–2158, 2001.

[18] R. B. Haber, C. S. Jog, and M. P. Bendsøe. A new approach to variable-topology shape design using a constraint on perimeter. *Structural Optimization*, 11(1-2):1–12, February 1996.

[19] J. Petersson and O. Sigmund. Slope constrained topology optimization. *International Journal for Numerical Methods in Engineering*, 41(8):1417–1434, April 1998.

[20] A. Díaz and O. Sigmund. Checkerboard patterns in layout optimization. *Structural Optimization*, 10(1):40–45, August 1995.

[21] O. Sigmund and K. Maute. Topology optimization approaches. *Structural and Multidisciplinary Optimization*, 48(6):1031–1055, August 2013.

[22] B. S. Lazarov and O. Sigmund. Filters in topology optimization based on Helmholtz-type differential equations. *International Journal for Numerical Methods in Engineering*, 86(6):765–781, December 2010.

[23] F. A. Lootsma. Logarithmic programing–a method of solving nonlinear-programming problems. *Philips Research Reports*, 22(3):329, 1967.

[24] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, April 2005.

[25] D. Arndt, W. Bangerth, B. Blais, M. Fehling, R. Gassmöller, T. Heister, L. Heltai, U. Köcher, M. Kronbichler, M. Maier, P. Munch, J. Pelteret, S. Proell, K. Simon, B. Turcksin, D. Wells, and J. Zhang. The deal.II library, version 9.3. *Journal of Numerical Mathematics*, 29(3):171–186, September 2021.

[26] A. V. Fiacco and G. P. McCormick. *Nonlinear Programming–Sequential Unconstrained Minimization Techniques*. Oxford University Press (OUP), August 1969.

[27] J. Nocedal, A. Wächter, and R. A. Waltz. Adaptive barrier update strategies for nonlinear interior methods. *SIAM Journal on Optimization*, 19(4):1674–1693, January 2009.

[28] R. J. Vanderbei. LOQO user's manual — version 3.10. *Optimization Methods and Software*, 11(1-4):485–514, January 1999.

[29] P. Armand, J. Benoist, and D. Orban. Dynamic updates of the barrier parameter in primal-dual methods for nonlinear programming. *Computational Optimization and Applications*, 41(1):1–25, October 2007.

[30] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2(4):575–601, November 1992.

[31] J. Nocedal and S. Wright. *Numerical Optimization*. Springer New York, 2006.

[32] R. J. Vanderbei and D. F. Shanno. An interior-point algorithm for nonconvex nonlinear programming. *Computational Optimization and Applications*, 13(1/3):231–252, 1999.

[33] R. Fletcher and S. Leyffer. Nonlinear programming without a penalty function. *Mathematical Programming*, 91(2):239–269, January 2002.

[34] M. Ulbrich, S. Ulbrich, and L. N. Vicente. A globally convergent primal-dual interior-point filter method for nonlinear programming. *Mathematical Programming*, 100(2):379–410, June 2004.

[35] H. Y. Benson, R. J. Vanderbei, and D. F. Shanno. Interior-point methods for nonconvex nonlinear programs: Filter methods and merit functions. *Computational Optimization and Applications*, 23(2):257–272, 2002.

[36] S. Rojas-Labanda and M. Stolpe. Automatic penalty continuation in structural topology optimization. *Structural and Multidisciplinary Optimization*, 52(6):1205–1221, July 2015.

[37] O. Sigmund and J. Petersson. Numerical instabilities in topology optimization: A survey on procedures dealing with checkerboards, mesh-dependencies and local minima. *Structural Optimization*, 16(1):68–75, August 1998.

[38] L. Li and K. Khandelwal. Volume preserving projection filters and continuation methods in topology optimization. *Engineering Structures*, 85:144–161, February 2015.

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

[40] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM Journal on Scientific Computing*, 14(2):461–469, March 1993.

[41] S. C. Eisenstat and H. F. Walker. Globally convergent inexact Newton methods. *SIAM Journal on Optimization*, 4(2):393–422, May 1994.

[42] H.A. Van der Vorst and C. Vuik. The superlinear convergence behaviour of GMRES. *Journal of Computational and Applied Mathematics*, 48:327–341, 1993.

[43] D. Silvester and A. Wathen. Fast iterative solution of stabilised Stokes systems part II: Using general block preconditioners. *SIAM Journal on Numerical Analysis*, 31(5):1352–1367, October 1994.

[44] P. F. Dubois, A. Greenbaum, and G. H. Rodrigue. Approximating the inverse of a matrix for use in iterative algorithms on vector processors. *Computing*, 22(3):257–268, September 1979.

[45] R.P. Fedorenko. A relaxation method for solving elliptic difference equations. *USSR Computational Mathematics and Mathematical Physics*, 1(4):1092–1096, January 1962.

[46] J. Xu and L. T. Zikatanov. Algebraic multigrid methods, 2016.

[47] T. C. Clevenger and T. Heister. The deal.II tutorial step-50: Geometric multigrid on adaptive meshes distributed in parallel. https://zenodo.org/record/4004166, 2020.

[48] M. R. T. Fraters, W. Bangerth, C. Thieulot, A. C. Glerum, and W. Spakman. Efficient and practical Newton solvers for non-linear Stokes systems in geodynamic problems. *Geophysical Journal International*, 218(2):873–894, April 2019.

[49] S. Knapek. Matrix-dependent multigrid homogenization for diffusion problems. *SIAM Journal on Scientific Computing*, 20(2):515–533, January 1998.

[50] J. O'Connor. Simultaneous analysis and design with interior point topology optimization. https://zenodo.org/record/7689822, 2023.

[51] K. M. Soodhalter, E. de Sturler, and M. Kilmer. A survey of subspace recycling iterative methods, 2020.

[52] J. Frank and C. Vuik. On the construction of deflation-based preconditioners. *SIAM Journal on Scientific Computing*, 23(2):442–462, January 2001.

[53] K. Svanberg. The method of moving asymptotes—a new method for structural optimization. 24(2):359–373, February 1987.

[54] C. Zillober. A globally convergent version of the method of moving asymptotes. *Structural Optimization*, 6(3):166–174, September 1993.

[55] B. Hassani and E. Hinton. A review of homogenization and topology optimization III—topology optimization using optimality criteria. *Computers & Structures*, 69(6):739–756, December 1998.

[56] N. Aage, E. Andreassen, and B. S. Lazarov. Topology optimization using PETSc: An easy-to-use, fully parallel, open source topology optimization framework. *Structural and Multidisciplinary Optimization*, 51(3):565–572, August 2014.

[57] X. Huang and Y. M. Xie. A new look at ESO and BESO optimization methods. *Structural and Multidisciplinary Optimization*, 35(1):89–92, May 2007.

Appendix A

Currently Used Methods

Researchers in the field have also historically used a Nested ANalysis and Design (NAND) problem formulation. In the NAND formulation, the PDE constraint is solved outside of the optimization loop at each step, and then information given by the solution to the PDE is passed into the optimization problem. This is different from how traditional constraints are treated in that it breaks each subproblem into two parts – the PDE and the optimization step. Two different linear solves occur in this method, but they are both of a moderate size. Pre-existing preconditioning and solving methods can be easily applied to the PDE, allowing for fast linear solves.

The major downside of this simplification is a less robust search. NAND reduces the design domain to not include the physical descriptors of the build, but only the shape of the structure itself. Designs found using NAND tend to be less optimal, if convergence is able to occur at all.

The topology optimization community historically has used a variety of optimization techniques that have been specifically created in the community. For example, the Method of Moving Asymptotes (MMA) [53], the Globalized Method of Moving Asymptotes (GMMA) [54], and the Optimality Criteria method (OC) [55] are optimization techniques developed between 1987 and 1998 for this particular problem. All of these algorithms, by necessity of the computing power of the time, are first-order algorithms that work only with first derivative information of the objective function using sensitivity analysis. They all also use the NAND formulation. I will next discuss often-used algorithms for comparison.

A.1 Method of Moving Asymptotes

The Method of Moving Asymptotes (MMA) is a nonlinear optimization technique that is widely used today for structural optimization problems [12, 56]. It is discussed here to be con-

trasted with an interior-point second order method. MMA [53] takes the form

$$\text{minimize } f_0(\rho) \quad \rho \in \mathbb{R}^n \tag{A.1}$$

$$\text{subject to } f_j(\rho) \leq \hat{f}_j, \quad j \in \mathbb{R}, \tag{A.2}$$

$$\text{and } \underline{\rho}_i \leq \rho_i \leq \bar{\rho}_i, \quad i = 1, 2, ..., n \tag{A.3}$$

The crux of MMA is creating first order approximations of the objective function and inequality constraints in the form of asymptotic functions. This gives a subproblem that can be easily solved using a dual method. The algorithm is given below:

1. The first step is to choose an appropriate initial guess $\rho^{(0)}$. Because we currently have no information about how the density of material should be arranged, beginning with an even distribution of material is the typical approach for all nonlinear solvers. Denoting the volume of the domain as V_{Total} and the maximum volume of a structure as V_{Max} gives

$$\rho_i^{(0)} = \frac{V_{\text{Max}}}{V_{\text{Total}}}. \tag{A.4}$$

2. Asymptotes are now selected for speedy convergence. Each density variable is optimized according to a function – generated in the next step – that is constructed to have an asymptote both above and below the current value. These asymptotes are heuristically modified as the problem continues to prevent oscillation of density values and to help with speedy convergence.
3. Next, in iteration k, we generate approximate functions to use in the MMA subproblem to be solved for the step in the iteration. The Method of Moving asymptotes approximates both the objective function and constraint functions as

$$f_j^{(k)}(\rho) = r_j^{(k)} + \sum_i \left(\frac{p_{i,j}^{(k)}}{U_i^{(k)} - \rho_i} + \frac{q_{i,j}^{(k)}}{\rho_i - L_i^{(k)}} \right). \tag{A.5}$$

The placement of the upper and lower asymptotes for each element of ρ are kept the same for all functions being approximated for the subproblem. Using the following parameters gives the correct function and gradient values at $\rho^{(k)}$, while also limiting the function to one asymptote being used in each dimension.

$$p_{i,j}^{(k)} = \begin{cases} (U_i^{(k)} - \rho_i^{(k)})^2 \frac{\partial f_j}{\partial \rho_i}, & \frac{\partial f_j}{\partial \rho_i} > 0 \\ 0, & \frac{\partial f_j}{\partial \rho_i} \leq 0 \end{cases}, \tag{A.6}$$

$$q_{i,j}^{(k)} = \begin{cases} 0, & \frac{\partial f_j}{\partial \rho_i} \geq 0 \\ -(\rho_i^{(k)} - L_i^{(k)})^2 \frac{\partial f_j}{\partial \rho_i}, & \frac{\partial f_j}{\partial \rho_i} < 0 \end{cases}, \tag{A.7}$$

$$r_j^{(k)} = f_j^{(k)}(\rho) - \sum_i \left(\frac{p_{i,j}^{(k)}}{U_i^{(k)} - \rho_i^{(k)}} + \frac{q_{i,j}^{(k)}}{\rho_i^{(k)} - L_i^{(k)}}\right). \tag{A.8}$$

4. We then generate a subproblem using these approximations. This gives a separable approximate Lagrangian, which can be written for any element of the density as

$$l_i(\rho_i, y) = q_{i,0}^{(k)} \left(\frac{1}{\rho_i - L_i^{(k)}} - \frac{1}{\rho_i^{(k)} - L_i^{(k)}}\right) + y p_{i,1} \left(\frac{1}{U_i^{(k)} - \rho_i} - \frac{1}{U_i^{(k)} - \rho_i^{(k)}}\right). \tag{A.9}$$

$l_i'' > 0$ everywhere, so $\rho_i(y)$, that is, the ρ_i that minimizes l_i given a Lagrange multiplier y, can be found using the following:

$$\rho_i(y) = \begin{cases} \max(\underline{\rho}_i, \alpha_i^{(k)}), & l_i'(\max(\underline{\rho}_i, \alpha_i^{(k)}), y) \geq 0, \\ \min(\bar{\rho}_i, \beta_i^{(k)}), & l_i'(\min(\bar{\rho}_i, \beta_i^{(k)}), y) \leq 0, \\ \frac{\sqrt{q_{i,0}} U_i + \sqrt{y p_{i,1}} L_i}{\sqrt{q_{i,0}} + \sqrt{y p_{i,1}}}, & \text{else.} \end{cases} \tag{A.10}$$

Here the final formula is found by setting $l_i' = 0$.

5. Using the separable approximation to our Lagrangian gives a dual problem over a single variable. The dual objective value, W, is given by

$$W(y) = r_0 - y(V_{\text{max}} - V^{(k)}) + \sum_i \frac{y p_{i,1}^{(k)}}{U_i - \rho_i(y)} + \frac{q_{i,0}^{(k)}}{\rho_i(y) - L_i}. \tag{A.11}$$

The total volume Lagrange multiplier is y, which can then be solved using a golden search method. The golden search method is usable because of the guaranteed convexity of the asymptotic approximation.

Steps 2-5 are then repeated until convergence of the compliance is reached.

This method is clearly quite detailed, and also rather difficult to implement. A competing strategy which is much simpler is to instead use a fixed point method based upon some criteria of an optimal solution.

A.2 Optimality Criteria Method

A simple method for updating the density can be created by using the premise that, at an optimal solution, the voxels with the most sensitivity should all be at a maximum density value. This method, developed and implemented by Hassani in 1998 [55], computes sensitivities for all cells, and guesses a total volume multiplier Λ, and updates as

$$B_i = -\Lambda \frac{\partial \tilde{C}}{\partial \rho_i}, \tag{A.12}$$

$$\rho_i^{(k+1)} = \begin{cases} \max(\rho_{\text{min}}, \rho_i^{(k)} - \zeta), & \max(\rho_{\text{min}}, \rho_i^{(k)} - \zeta) > \rho_i^{(k)} B_i, \\ \min(1, \rho_i^{(k)} + \zeta), & \min(1, \rho_i^{(k)} + \zeta) < \rho_i^{(k)} B_i, \\ \rho_i^{(k)} B_i, & \text{else.} \end{cases} \tag{A.13}$$

In each iteration, Λ is changed until the total volume reached is equal to the total allowed volume. This method increases the cell density in cells whose sensitivity magnitude is greater than $1/\Lambda$, and decreases it in cells whose sensitivity is less. This simple method is very competitive with the MMA [2], and is just as widely used.

A.3 Evolutionary Structural Optimization and Bidirectional Structural Optimization

Evolutionary Structural Optimization is an approach that rejects almost all of what was discussed in this paper, beginning with the SIMP method. By using an evolutionary search, it attempts to iterate over the original combinatorial space in a way that finds local minima fairly quickly. The original ESO algorithm begins with a design where the full volume is filled. Performing a structural analysis, the elements that have the lowest Von Mises stress are removed (or are replaced with a much weaker "void material"), creating a new structure. This process is repeated until the mass is at the desired value, or until removing more material would cause a constraint to be broken.

Bidirectional ESO works much in the same way, except it allows both material to be removed and also material to be added back. In this case, after a structural analysis is performed, a portion of the "void material" that experienced the largest Von Mises stress is placed back into the design, and uses the original material. These methods are both explained in detail in [57].

These methods have gained popularity for being simple to implement, fast, and finding local minima that are visually different than those from algorithms using SIMP.