TOPOLOGY OPTIMIZATION FOR ADDITIVE MANUFACTURING USING SIMP METHOD

Seth Grinde
Montana Tech

TOPOLOGY OPTIMIZATION FOR ADDITIVE MANUFACTURING USING SIMP METHOD

by

Seth Grinde

A thesis submitted in partial fulfillment of the requirements for the degree of

General Engineering
Mechanical Option

Montana Tech
2018

Abstract

The increasing effectiveness of additive manufacturing has contributed to increased fabrication of complex parts with less material waste. With this process, complex shapes that can reduce the weight of the component can be explored. Topology optimization of a component uses computer software to remove and add material in locations throughout the design volume. The optimized design output results in a reduced weight component that meets the performance requirements of the original design. There are many optimization methods, one of which is the solid isotropic material with penalization (SIMP) method. An objective function is defined to give the optimization method an objective for the algorithm to iterate against while a design variable is altered after each iteration to achieve the objective. Different constraints are applied to keep the optimization method within a set of bounds defined by the user and the components original geometry. A penalization factor is applied to the optimization method algorithm to refine the final solution to solid and void regions so that a three-dimensional printer can manufacture the component.

Various optimization programs were explored for the topology optimization of a beam designed for three point loading. A solid beam that has not been optimized is used as the initial design for optimization as well as a baseline for comparison of the different optimization software packages. Five different methods for optimization were used which include: MATLAB with penalization; MATLAB without penalization and variable thicknesses; ParetoCloud optimization; and two simple methods previously used for component lightening. The components were printed with a fused filament fabrication process that extrudes material building the component layer by layer. The printed beams were then tested in a three point bending test until failure. Comparisons of the different optimized beams were performed using calculations from the resulting load-deflection curves.

Keywords: Topology Optimization, SIMP, MATLAB Optimization, ParetoCloud Optimization, Variable Thickness

Dedication

I would like to dedicate this thesis to God and my parents, Craig and Susan Grinde, whose love and support has made me into a person that I can be proud of. I would not be where I am today without their endless encouragement.

Acknowledgements

This research was financed and developed from the generosity of the Army Research Laboratories (ARL). I would like to thank ARL for providing me with the opportunity to work on an exciting research project.

I would like to thank my advisor, Dr. Bruce Madigan, for focusing my attention to topology optimization and offering valuable suggestions for a path forward. I would also like to thank my committee members, Dr. Peter Lucon, Dr. Jerry Downey, and Scott Coguill for providing me with beneficial information and taking time from their busy schedules to read and review my writing.

I am also thankful for the assistance of Ronda Coguill for assisting me in the manufacturing and testing of my beams.

I am grateful for the assistance provided by my officemate Dale Brush in helping me generate the necessary MATLAB code.

Table of Contents

List of Tables

List of Figures

List of Equations

Glossary of Terms

Term	Definition
$\boldsymbol{\sigma_b}$	Buckling stress
ρ	Material density
ρ_e	Material density at an element in the design domain
A	Cross sectional area
b	Beam thickness
c	Compliance
C_{ijk}^{o}	Original stiffness tensor
C_{ijk}	Penalized stiffness tensor
E_B	Flexural modulus
E_o	Actual Young's modulus
E_e	Penalized Young's modulus at an element in the design domain
F	Force vector
F_b	Max force before buckling
F_f	Force of load applied during three point loading
h	Beam height
I_b	Second moment of inertia of smallest cross section
K	Structure stiffness matrix
L	Beam span
L_b	Length of slot support material
m	Slope of tangent line to the linear region of the load-deflection curve
p	Penalization factor
u	Displacement vector
vf	Volume fraction
V_o	Original volume
x	Design variable
x_e	Design variable at an element in the design domain

1. Introduction

1.1. Background

Modern technology has enabled the ability to construct complex parts that were previously considered difficult to manufacture. Classically, production of complex parts involved the removal of material to obtain the final product. With developing 3D printing technologies, additive manufacturing has become a more feasible method for production of complex geometries. Additive manufacturing creates components by building up material where it is needed instead of removing material (where it is not needed) from the component. This process allows for a more efficient use of material, with less waste. The additive technologies being developed within the past 32 years, (Pena, Micali, & Lal, 2014), are used to produce complex shapes and parts that were previously considered too difficult to manufacture. Advancements in additive manufacturing have provided the ability for quick and efficient implementation of manufacturing components with less overall effort in production. However, for some circumstances, materials used in the additive manufacturing process may not be readily available or too expensive making it even more important to implement methods that efficiently utilize material (Weller, Kleer, & Piller, 2015). One such method appropriate for additive manufacturing to more efficiently utilize material is to first design a component using topology optimization.

Topology optimization is the mathematical process of retaining and removing material of a design within the design domain. Topology optimization has become more advanced and prevalent in use since its introduction to the homogenization method of topology optimization in 1988 (Bendsoe & Kikuchi, 1988). By removing unnecessary material from the design, the resultant component will be lighter with theoretically equal strength. Typically, a new or improved design is just an incremental change in an existing design. Therefore, the implementation of topology optimization in component design improvement changes from an incremental improvement to a substantial improvement in function when applied to the original component design. When efficient use of additive manufacturing material and simplicity of component improvement is of importance, topology optimization is an effective method in the design process.

1.2. Types of Optimization

Two other subsets of structural optimization are related to topology optimization. Size and shape optimization are the more common methods practiced for optimizing a component. Size optimization can be considered as the location of links and joints within the components design domain. In trusses in structural beam analysis, the links and joints are designed to carry the load while providing lightening effects by only occupying a fraction of the beams total design domain. Size optimization also includes the variance of material thickness. If a truss with links and joints under loading were to have a fixed height and width, the thickness of the links could be optimized to reduce weight and maintain the required strength. Shape optimization modifies the design geometry to meet the required objective(s). If a beam with circular holes were to undergo loading, shape optimization would be able to change the geometry of the holes to withstand the loading as best as possible. By themselves, size and shape optimization can only optimize a single aspect of the design. The combination of the two types of optimization allows for simultaneous optimization of the size and shape of the geometry Topology optimization is a combination of shape and size optimization. By relocating material and altering the shape of the structure, topology optimization can provide a more accurate representation of an optimized design. Shape and size optimization add certain design elements to the topology optimization process. Each method is trying to maximize the optimized structures stiffness, this maximization of the structures stiffness can be considered the objective function of each method. Size optimization method (Figure 1a) adds a design element by positioning the links and joints in an optimal position and changing the size of the cross-sectional area of each link. Therefore, size optimization will rely on these design variables for optimization. Shape optimization (Figure 1b) maximizes the optimized structures stiffness by altering the shape of the voids. However, the voids in the structure must keep the same position and area within the structure. For this example, the area and location of the voids are considered design constraints (Broxterman, 2017). Topology optimization (Figure 1c) uses both the design variables and constraints of size and shape optimization to satisfy the objective function. Because of the combination of design variables and constraints from size and shape optimization, the optimization becomes more complex, but presents a more accurate optimization result to the objective function.

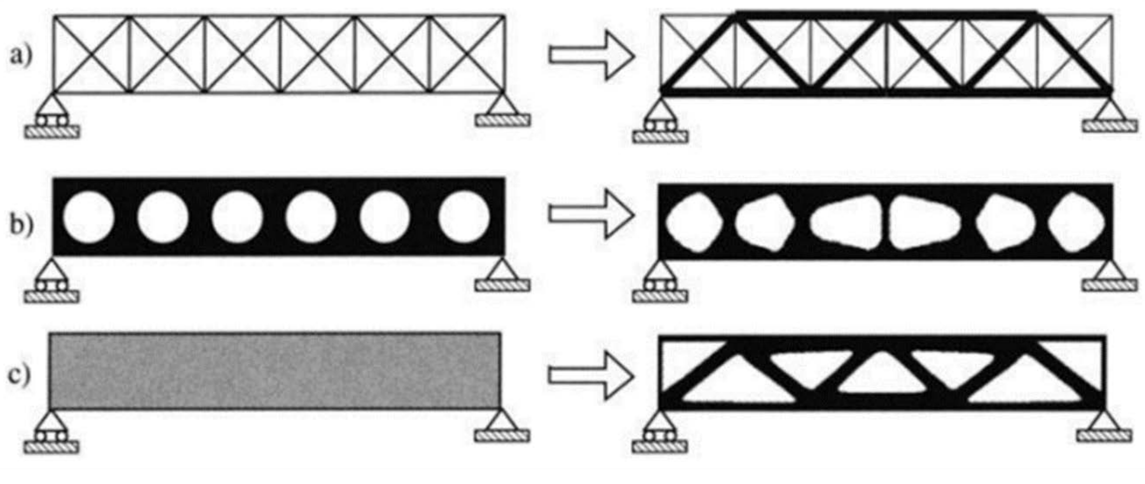


Figure 1: Three categories of structural optimization. a) Sizing optimization of a truss structure, b) shape optimization, and c) topology optimization. The initial problems are shown at the left hand side and the optimal solutions are shown at the right. (Bendsoe & Sigmund, 2003)

Anything that is affected by the size and shape of material within the design domain can be set as an objective function. Heat transfer, vibrations, and structural stiffness are the more common objective functions because of their wide applications. The design variables and constraints vary between each problem, so it is up to the designer to analyze the problem for the required specification.

The calculations involved in topology optimization will be discussed to provide an understanding of the functionality of optimization methods. An understanding of how the optimization methods calculate optimal designs is necessary to discover possible improvements with optimization and to decide which constraints and variables to alter to rectify certain issues. The objective is to then use this foundational understanding to modify an optimization method to produce an improved component which will then be compared to other optimization methods.

2. Topology Optimization

The main goal of a topology optimization program is to decide which elements in a components design domain will be solid material or a void with respect to the objective function. The optimization process relies on different constraints and alters the design variables that are specific to each problem to generate a unique solution. For a structural optimization problem, a maximum stiffness of the structure is usually desired. In a simply supported beam, a large stiffness will reduce the deflection from a load. Since the compliance of a structure is inversely proportional to the stiffness, a minimum compliance is another consideration to the objective function.

2.1. Optimization Calculation Introduction

To begin understanding the analysis done by topology optimization, consider a solid simply supported beam with a single center load (Figure 2) where the goal is to reduce the beams weight.

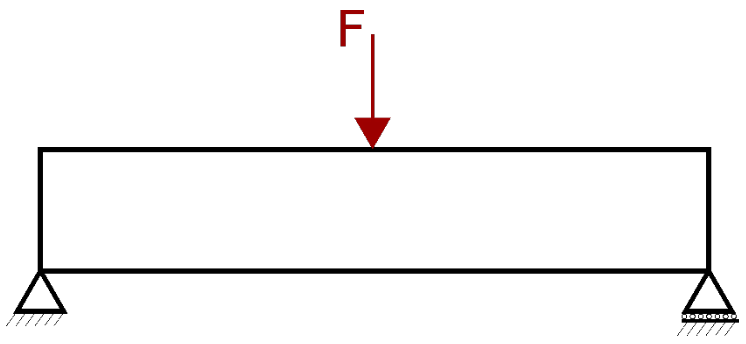


Figure 2: Simply supported beam with single center load

To define the problem, the beam design needs to remain within the build domain, must deform as little as possible, and is made of a single isotropic material. The problem definition establishes beam weight and build domain as constraints, the deformation (compliance) as the objective function, and the material as a design variable. For structural optimization, there are three forms of constraints: behavioral, design, and equilibrium (Christensen & Klarbring, 2010). For a linear discretized problem, the equilibrium constraint involves the design variable. Behavioral and design constraints are similar where the behavioral constraint involves the state variable and the design constraint involves the design variables. The state variable represents a given structure as a function of the structures response to loading, displacement is typically used. The design variable is a function that can be changed during the optimization process, this design variable will be defined as the material density. The objective of the solid simply supported beam was to minimize the displacement of the beam. This objective can also be reverted to the maximizing of stiffness or minimizing the compliance. The equilibrium constraint can then be defined as

$$K(x)\,u = F(x) \tag{1}$$

where *K* is the structures stiffness matrix, *u* is the displacement vector, *F* is the force vector, and *x* is the design variable. From this equation, the stiffness matrix is a function of the design variable and the force vector may also depend on the design variable. The displacement vector is considered to be the state variable where the displacement shows where the beam is and where it will be located in a linear displacement. The objective function as stated before is to minimize the compliance. Because the compliance matrix is the inverse of the stiffness matrix, the objective function can be to maximize the stiffness or minimize the compliance which is written as

$$\begin{gathered} \min_{u,x_e} c = F^T\,u \\ s.t \quad K(x_e) = F \end{gathered} \tag{2}$$

where, the compliance (*c*) of the beam is minimized subject to (*s.t*) the stiffness matrix. The stiffness matrix is a function of the design variable x_e on each element in the design domain (Bendsoe & Sigmund, 2003). For topology optimization that focuses on the placement of material in the design domain, the design variable will be assigned as a material property. The most commonly used properties are the materials elastic modulus and density. For a void in the beam, the material property would ideally become a zero so that there is no force resistance from the stiffness matrix. For material located within the design domain, the material property would become fully available for force resistance. To begin getting the desired solid or void, an artificial density function will need to be applied to a Young's modulus. For this example, ρ_e will represent the design variable x_e as the density of each element in the design domain. A density of 1 will represent a solid while a density of 0 will represent a void. This density will then be applied to a Young's modulus to find the strength of material for that element.

$$E_e = \rho_e * E_o \tag{3}$$

where E_e is the penalized Young's modulus at an element within the design domain and E_o is the actual Young's modulus of the isotropic material. When the optimizer varies the design variable to minimize the compliance, the Young's modulus of each element will change based on the design variable at that element which, in turn, will change the material properties of the elements.

When running the optimization with just the minimization of the compliance in mind, the result will be a solid beam that fills the whole design domain. The stiffest structure that will be the easiest to compute is a solid beam. The lack of material removal means that a volume constraint will need to be applied to guide the optimizer to an optimized beam with solid and voids. The volume constraint then takes the form

$$\int \rho_e \leq vf * V_o \tag{4}$$

where vf is the chosen volume fraction and V_o is the original volume. This constraint now forces the optimizer to select material to be removed until the new design volume is less than or equal to the desired volume (Bendsoe & Sigmund, 2003).

2.2. SIMP Method

With the optimizer's objective function working on minimizing the compliance and the volume constraint forcing a reduced volume solution, the result is closer to an end solution. However, the density function will vary between 1 and 0 ($0 < \rho < 1$). This variation in density will create a variable density gradient that will exist in the new build domain. Without the ability of manufacturing components with a variable density gradient, the gradient will need to be corrected. The solid isotropic material with penalization (SIMP) method is a common solution to the density gradient problem. The intermediate density material gets a stiffness tensor that is represented by $C_{ijkl}(\rho) = C_{ijkl}^0 * \rho^p$ where the original stiffness tensor of solid material C_{ijkl}^0 is penalized by a penalized density ρ^p. The penalization factor p helps force a solution of either a solid or a void. The penalization of density results in a solid or void solution by lowering the local stiffness of the element causing the final design to not favor the lower density gradients because its cost is greater than the benefit. In other words, the stiffness that the element would provide is too small for the amount of material that the element would provide making it "uneconomical" for the final solution (Sigmund & Petersson, 1998). A penalization factor greater than one is chosen for solutions that aim to transform the intermediate densities into solid and void regions, while a penalization factor equal to one will result in a density gradient.

Without a penalization factor, the resulting beam will have a density gradient while a penalization factor will create a solid and void region. From Figure 3a, the gray area is a density gradient while the black and white is solid and void respectively. This gradient is caused by a penalization factor of one being applied so that the density is not forced to change to become a solid or void.

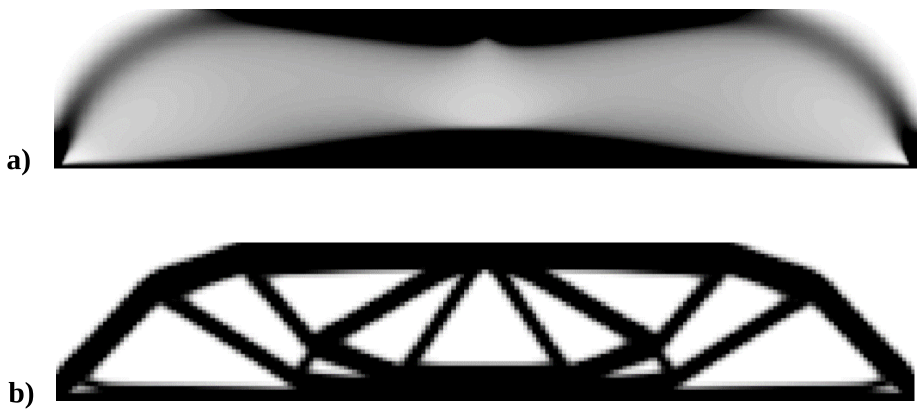


Figure 3: Beams optimized using the 99 line MATLAB code (Sigmund, 2001) a) Optimized beam with a penalization factor p=1, b) Optimized beam with a penalization factor p=3

In Figure 3b, the density gradient has been penalized to form black and white, solid and void regions. The solid and void regions can exist because the penalization factor force the density gradients to become a solid or a void. This penalization can then be applied to the material properties and volume constraints of each element within the design domain.

$$E_e = \rho_e^p * E_o$$

$$\sum_{e=1}^{n} \rho_e^p * V \leq vf * V^o \tag{5}$$

Because the stiffness matrix depends on the stiffness of *E_e* in each element and the global element stiffness matrix *K_e* is a function of *E_e* (Bendsoe & Sigmund, 2003), the new stiffness matrix can be written as

$$K = \sum_{e=1}^{N} K_e(E_e) \tag{6}$$

By combining Equations (5) and (6), the final problem statement can be written as

$$\begin{gathered} \min_{\rho_e} \; c = F^T \, u \\ s.t. \sum_{e=1}^{N} (K_e * \rho_e^p) u = F \\ \sum_{e=1}^{n} \rho_e^p * V \leq vf * V^o \\ 0 \leq \rho_e \leq 1 \\ e = 0 \ldots n \end{gathered} \tag{7}$$

The objective function is to minimize the compliance of the structure by altering the density at each element, e. The stiffness contributed to the structure by each element is penalized by the density. To force the optimizer to find a solution that reduces the volume of the overall structure, the objective function is subject to the volume constraint where the volume fraction is a user defined fraction of the original design. The objective function must also consider the density of each element to remain in the complete void to solid range.

Typically, the larger the penalization factor, the better the intermediate density removal. However, with the increase in penalization factor, the required time for the software to converge on an answer increases. Therefore, a balance between accuracy and required computational time must be found.

2.3. Sensitivity Analysis

During the optimization process, it is necessary to run a sensitivity analysis. The sensitivity analysis will determine which parameters and design variables will affect the result the most due to the significance of the variable. If a variable is perturbed slightly and greatly changes the result, that variable is going to have a higher significance. If the variable is perturbed and the result has minimal change, the variable is going to have a smaller significance. The significance of each variable is important to the optimization process because the significance may be used to determine how uncertain the result is based on the varying parameters. A large significance will cause the solution to be easily alterable depending on the input variables.

For most problems, the sensitivity of the structure is assumed to be differentiable with respect to the design (Choi & Kim, 2005). A general outlook on the sensitivity analysis indicates that the sensitivity of the structure depends on the design. A cross sectional area of a simply supported beam would change the weight of the structure. The volume of the beam can be expressed as

$$V(a) = b * h * L \tag{8}$$

where V is the beams volume, b is the thickness, h is the height, and L is the length of the beam. If the thickness of the beam is a design variable, the design sensitivity of the beams volume with respect to b is

$$\frac{dV}{db} = h * L \tag{9}$$

Because this function can be explicitly formed with respect to the specific design, the function is explicitly dependent (Choi & Kim, 2005). In most cases however, the sensitivity of the structure is not explicitly dependent on the design. The stiffness of a structure is dependent not only on the shape of the structure, but also the elastic modulus, which is part of the finite element solution. From the equilibrium Equation (1), the derivative of the equilibrium equation and the objective function *c* can be calculated as

$$K(\rho)\frac{du}{d\rho} = \frac{df}{d\rho} - \frac{dK}{d\rho}u \tag{10}$$

$$\frac{dc}{d\rho} = f^T\frac{du}{d\rho} \tag{11}$$

If the displacement vector *u* is known from the finite element analysis, *df/dρ* and *dK/dρ* can be calculated using input values of *K(ρ)* and *F(ρ),* then *du/dρ* can be found from Equation(10). The value *du/dρ* can then be substituted into Equation (11) to compute *dc/dρ* which is the sensitivity of the compliance problem.

2.4. Problems

The formation of an optimized component using topology optimization is not quite perfect and requires special care for the optimization process. Sigmund and Petersson (1998) describe three main problems that topology optimization faces including the formation of checkerboard patterns, mesh dependencies, and local minima.

2.4.1. Checkerboard

The checkerboard patterns refer to the formation of solids and voids that form a checkerboard pattern where the corners of the solids and voids meet at a point. It has been found that the checkerboard problem is most common in minimal compliance problems. Figure 4 is a representation of the checkerboard problem where black areas are solid material and white areas are voids.

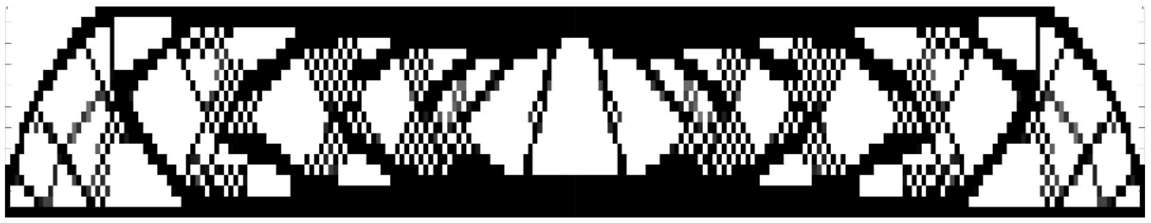


Figure 4: A simply supported beam with a checkerboard solution

The checkerboard problem has been proven to be the result of the checkerboard pattern being calculated as the stronger optimized result because of the difficulty in numerical modeling of the checkerboard pattern (Diaz & Sigmund, 1995). When a component is optimized and the checkerboard pattern emerges as the result, the optimizer assigned a large artificial stiffness to the checkerboard pattern. When the maximum stiffness of the structure (inverse of minimum compliance) is the objective function, the optimizer views the checkerboard as a viable option. Sigmund and Petersson (1998) summarized various solutions to the checkerboard pattern problem.

2.4.2. Higher-Order Finite Elements

Elements with a higher number of nodes have a higher degree of freedom with respect to the displacement as compared to those with a lower node count. Because a lower degree of freedom allows for quicker calculations, a lower node count is more common in optimizations (Diaz & Sigmund, 1995). However, it has been found that finite elements with eight to nine nodes have eliminated most checkerboard patterns (Jog & Haber, 1996). For the SIMP method, the checkerboard pattern was only eliminated when the penalization factor was small (Diaz & Sigmund, 1995).

2.4.3. Filter

A filter proposed by Diaz and Sigmund (1995) is another possible action to prevent checkerboards in the final solution. The optimized structure after each iteration can be viewed as a matrix of elements that have a color value assigned to them where white is a void and black is a solid element. A noise filtering technique is applied to where the color (density) of each element is averaged with respect to its neighboring elements. This filter will fill the checkerboard pattern area with weight averaged material so that the checkerboard pattern will be removed. By modifying the sensitivity of the optimization analysis with this filter, the checkerboard problem can be corrected.

2.4.4. Mesh Dependency

Mesh dependency refers to varying results with different mesh sizes applied to the same problem. A component with a larger mesh size will have a different result than that of one with a smaller mesh size even though the components are bound the same constraints and variables. With different mesh sizes, the two components should represent the same solution with different detail, however, with different mesh sizes, the two components will have different optimized material locations. For most solutions, the density of the mesh was found to have the largest impact on the final solution. With a larger mesh density, the final solution will have thinner members than a smaller density mesh because the resolution of the high mesh density can provide the finer detail needed for thin members (Figure 5).

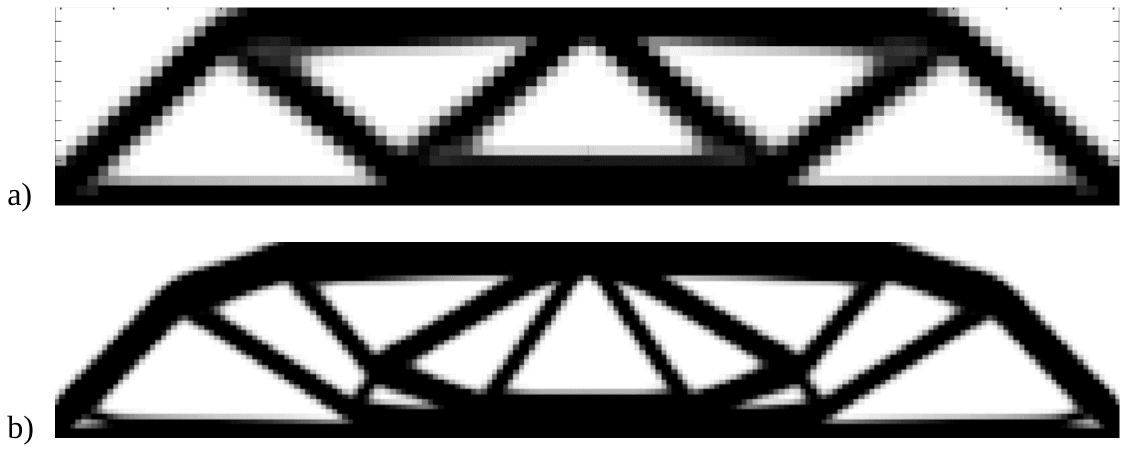


Figure 5: Simply supported beam with penalization factor of 3 and half beam dimensions of a) 50x20 elements and b) 100x40 elements

If manufacturability is a limit to the design process, a smaller mesh density can be used to simplify the solution with close to optimal results. A larger mesh density can be used for a solution with more accurate optimization results if there are no manufacturing limitations. The mesh density must be considered in the design of the component optimization. The result of each is an iterative process that will lead to an optimized solution that will fit the manufacturing capabilities. Other sensitivity analysis modifications created by Ambrosio and Buttazzo (1993), Sigmund and Petersson (1998), and Diaz and Sigmund (1995) are summarized in a report by Zhou, Shyy, and Thomas (2001).

2.4.5. Local Minima

Local minima refers to the variance in optimization solutions between components of the same problem with different parameters. For many problems that need optimization, changing one parameter can greatly affect the outcome. It is difficult to determine the optimum parameters to use to generate the best optimized solution without running the optimizer repeatedly with the different parameters. This trial and error method would take a lot of setup and computing time. The different minima are mostly due to the difference in local minima of the function. A non-convex function can have different minima depending on the parameters used, however, a convex function is more desired because the function will only have one minimum that fits the whole solution. The convex function will force the optimizer to converge onto a single solution of the global minimum instead of a local minimum that may or may not be the global minimum of that function.

3. Methods

After a basic understanding of topology optimization and its methods for optimizing structures was acquired, various software and methods for optimizing a simply supported beam were explored. This section will introduce the methods used for optimizing three dimensionally printed beams. The beams are designed to fit into a three-point loading machine for testing with a volume reduction of 50%. Five different beams are optimized using different software and theory. The optimized beams were tested and compared to a solid beam without the volume constraint applied to it.

3.1. MATLAB Optimized Beam

MATLAB is a programing language that is designed for numerical computing. Because of its computing capabilities, it has been useful for solving simple topology optimization problems. A 99 line MATLAB topology optimization code was created by O. Sigmund (2001) to optimize simply supported beams in two dimensional space. This method and a variation to the code was used to generate beams for testing. Sigmund's code is provided in Appendix A.

3.1.1. MATLAB Penalization

The first beam optimized in MATLAB is the penalized beam. This beam uses the unaltered MATLAB topology optimization code created by O. Sigmund (2001) (Appendix A).

3.1.1.1. Penalization Method

Sigmund's MATLAB code uses the SIMP method for penalizing intermediate densities. This method assumes that the material density is a design variable and that the material properties are constant within each element in the design domain (Sigmund, 2001). The material properties of the elements are penalized by multiplying the material properties of solid material to the density raised to a certain power, or penalization factor. This method is used if the penalization factor is greater than or equal to three. The SIMP method must also include a perimeter constraint, gradient constraint, or filtering methods for the optimizer to find solutions (Sigmund, 2001). Without the gradient constraint and filtering, the resulting beam will have intermediate densities and a checkerboard pattern. The MATLAB optimization code has allowed for the gradient constraints to be met and explored for producing optimized beams.

For the optimization program to run with less computing time, the code simplifies the problem. The problem can only take place in two dimensions, so the final solution can print an image in the x and y plane. A rectangle is also assumed to be the design space such that the dimensions of the rectangle x and y are defined by the user in the problem statement. By simplifying the problem to a rectangle, the number of elements is defined by a matrix of length x and y which was previously defined. Therefore, the larger the defined beam, the larger the number of elements, the finer the mesh size, then the more accurate the solution. To save more time, the beam is assumed to be symmetrical across its x axis center. The assumed symmetry of the beam allows the solver to use half of a beam in its calculations. A free body diagram of the half beam used in the MATLAB calculation is displayed in Figure 6 with elements 1 through n in the y and x direction.

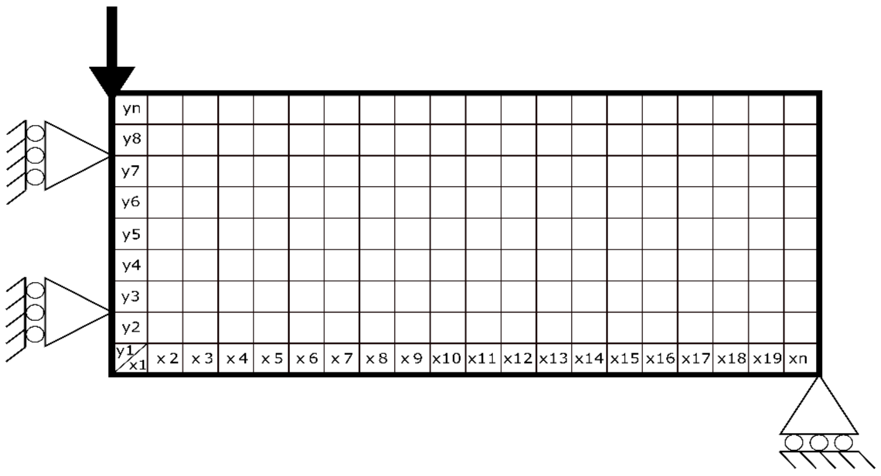


Figure 6: Half beam mesh from xy matrix

The x and y matrix lengths are for the half beam and not the full beam. When the optimizer uses a half-length beam, it assigns a roller support to the mid span boundary condition. The roller support prevents rotational and lateral movement in the x direction so there is only deflection in the y direction. A roller support at the right end of the beam is used to prevent the beam from deflecting at the support while still being able to move in the x direction. When the optimizer is run, the output will be an image consisting of a pixel at each of the matrix values for the material placement in the beam after each iteration. The inputs for the code are the element size in the x and y direction that define the size of the half beam. The penalization factor is chosen to be three or higher (typically lower than five) and the volume fraction is defined based off user desire. A filter size is also an input into the code that must be determined. The filter size allows the user to change the sensitivity of the elements to confirm the existence of solutions.

The first section of the code is used to identify the global displacement vector and the element displacement vector. The global and element displacement vector are then analyzed in the objective function to form a solution. To obtain the global displacement vector, material is initially distributed through the entire design domain. Once each element has an assigned material, the design is run through a finite element analysis to determine the global displacement vector. Once the global displacement vector is found, the elemental displacement vector is extrapolated from the global displacement vector. A sensitivity analysis is run for each element that is then updated by the applied filter. The resulting density distribution for the iteration is printed and recorded before the loop starts over. The 99 line code will continue to run under a while loop until the design variable criteria of change in the design variables becomes less than 1% change.

3.1.1.2. Penalized Beam Design

To start designing the MATLAB penalized beam, the objective of the design and computing requirements needs to be considered. The weight reduction of the beam for the testing is decided to be 50% of the original solid beams volume. The 50% weight reduction will set the volume fraction (volfrac) in the MATLAB function input to be 0.5. For this example, the beam dimensions were chosen to be 4 inches long by 0.75 inch tall by 0.25 inch thick. To get a modeled beam the same size as the actual beam dimensions, the size of the beam elements should represent the desired beam dimensions for more accurate results. Because the length of the beams span is 2.667 times larger than the height of the beam, the number of elements in the x direction needs to be close to 2.667 times larger than the elements in the y direction. A large number of elements in either direction will increase the number of elements in the design domain. However, a larger number of elements resulted in increased computing time. With limited computing capability, the element sizes cannot be too large or else convergence will take too long. With the computing time in mind, the element sizes in the x and y direction are chosen to be 100 in the x direction and 37 in the y direction. These dimensions will keep the final result more defined while keeping the computing time relatively low.

A penalization factor is applied to the density to remove the intermediate densities in the material. A penalization factor greater than one will start to remove the density gradient. A lower penalization factor will remove some of the intermediate densities, however, some areas of intermediate density will still remain. In Figure 7, three beams are shown with different intermediate densities caused by different penalization factors.

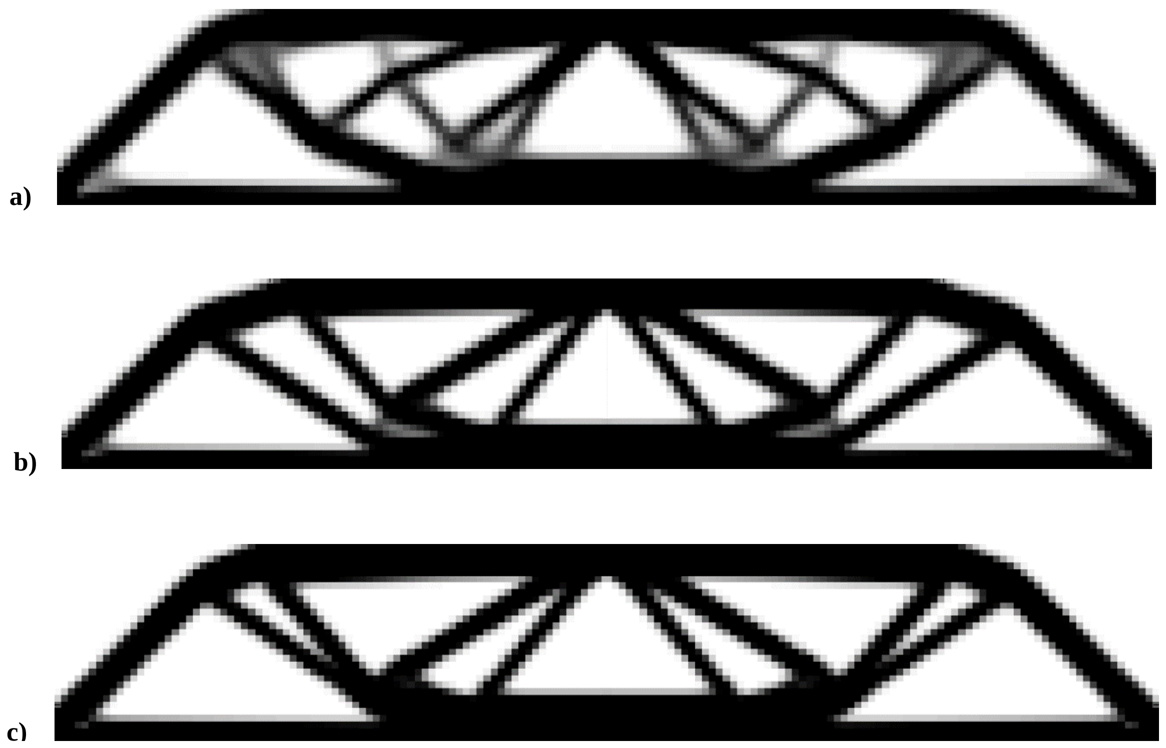


Figure 7: MATLAB optimized beam with penalization factor of a) p=2, b) p=3, c) p=4

Because the penalization factors are different in Figure 7, the intermediate densities will be penalized differently resulting in different material placement in the final solution. As the penalization factor increases, the amount of intermediate density should decrease. The amount of gray area in Figure 7a decreases to a clearer black and white image in Figure 7c when a higher penalization factor is applied to the same problem. Dadalau, Hafla, and Verl (2009) demonstrated that a penalization factor of one has a majority of intermediate densities while a penalization factor of two removed most intermediate density with some still remaining. A penalization factor of three resulted in a solution that had clear solid and void regions. By increasing the penalization factor, the optimizer was able to remove all intermediate densities. The removal of intermediate densities did come at a cost, which was that as the penalization factor increased the compliance also increased (Dadalau, Hafla, & Verl, 2009). For a minimal compliance problem, the additional cost is not desired. From Dadalau, Hafla, and Verl's study, a penalization factor large enough to remove intermediate densities but small enough to keep the compliance low is required. For this penalization factor range, a penalization factor of three was chosen for the optimization of the beam.

The final input to Sigmund's MATLAB code is the minimum filter radius. The filter will give a better chance for existence of solutions by restricting the resulting design (Sigmund, 2001). If the distance between the centers of neighboring elements falls within the filter radius, then the weighted average of those elements is taken and applied to the elements.

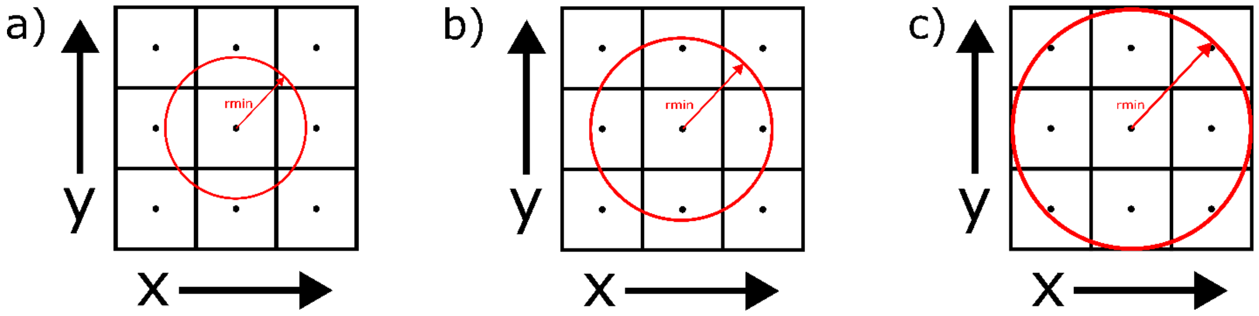


Figure 8: Filter radius sizes a) small filter radius with no intersections b) medium filter radius with some intersections c) large filter radius with complete intersections

A larger filter radius will encompass more elements making a larger area of elements uniform in value (Figure 8c). The larger number of included elements can create larger zones of intermediate density along a solid void boundary because the solid elements and void elements are grouped together making the weighted average between solid and void. For a more accurate boundary between the solid and void, a smaller filter radius can be used (Figure 8b). However, if the filter radius is too small, neighboring elements will not be grouped together and each element will have their assigned value (Figure 8). A single element in the filter radius leads to the checkerboard problem where one element is a solid and perpendicular elements are voids. A filter radius of 1.5, which fits between being too large and too small, was used for the beams.

When the final iteration outputs the material distribution figure, the final step before the optimized beam can be printed is to create a three-dimensional model from the two-dimensional image. Because the optimized beam was penalized, the result will be a solid and void solution. This solution creates an image that is black and white where black is solid material and white is a void.

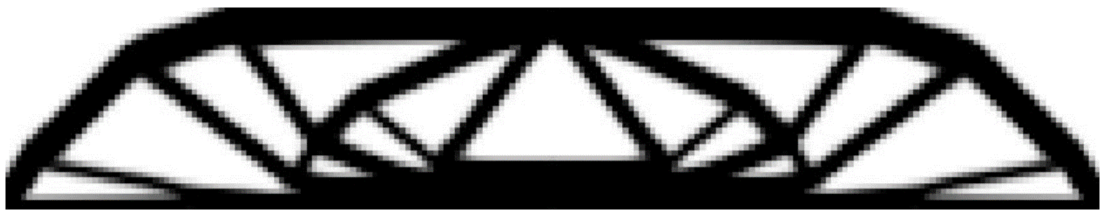


Figure 9: Resulting optimized beam from MATLAB with penalization factor of 3.0, size of 100x35, and filter radius of 1.5

The image is imported into SolidWorks and the black portions are traced creating a border between the white and black regions. The area is then extruded and mirrored so that the result is a full three-dimensional beam. Because the bending of the beam causes the supported ends to move towards each other, additional nonstructural supports must be added to the beam would not fall through before reaching its failure point.

3.1.2. MATLAB Non-Penalization

The second beam optimized in MATLAB is the MATLAB non-penalized beam. This beam was optimized using the 99 line MATLAB topology optimization code created by O. Sigmund (2001) and additional code for intermediate density processing.

3.1.2.1. MATLAB Non-Penalization Method

Topology optimization is used to generate an improved component that fits an objective function. The process of improving a component is typically done by penalizing intermediate densities so that the end component is a solid and void structure where the solid is 100% of the materials density and the void is 0% of the materials density. These two densities make it easier for the additive manufacturing process because, currently, there are no easy ways to print a density gradient. However, during the penalization process, the stiffness contribution of the density gradient is skewed. When the gradient is penalized, the density is nudged to either a solid or a void which then makes the elements contribution to the components stiffness a value different than its true value. This section proposes an alternate method to this slight loss in stiffness due to penalization of the density gradient.

A possible way to relate the variable density to a solid or void without altering the true value of stiffness contribution is to relate the density at each element in the gradient to a thickness of 100% density. For example, if an element in the density gradient is calculated to be 50% of the density of material, then the thickness of the solid material may be 50% of the thickness. If the objective of the optimization process is to reduce the weight of the component, by varying the thickness of the component where there is a density gradient, the weight will ideally be reduced while keeping the true stiffness contribution of each element intact without penalization.

To go about this process, a simply supported beam was optimized using Sigmund's MATLAB code. Instead of using a penalization factor of three like before, the penalization factor was set to one. This new penalization factor made it so there was no penalization of intermediate densities creating a density gradient (Figure 3a). From here, two different methods for relating the densities of each element to a thickness are used.

The first method was a user interpreted thickness and boundary location. The output image of the code was changed so that instead of a grayscale image, the image would be colored. The colored image helped distinguish areas of similar densities. In the density gradient, there are slight boundaries between colors that created an area of similar density that could be related to a thickness. These boundaries are then traced in SolidWorks and extruded to a thickness based off the color of the layer. From the color scale, blue and red are both extremes where blue was 100% thickness and red was 0% thickness. Any color in the density gradient that was closer to red (orange and yellow) was a lower density region that related to a thinner section. Colors that are closer to blue on the scale (teal and green) are higher density regions that related to a thicker section. In general, the boundary of the beam had a higher density than the mid-section of the beam. The colors of each area created by the color boundary are then compared to the color scale of the image. The areas are extruded from mid-plane so that the beam was a mirror image. By keeping each side symmetrical, the structure of the beam would remain in line with the applied force to avoid eccentric loading. A volume calculation was run to check to see if the modeled beam was 50% of the initial volume. The thicknesses of the beam are scaled until the volume of the beam met the desired volume. Because the distance between the ends of the beam shorten due to bending, additional non-structural supports are added to the ends of the beam to prevent fall through. While in SolidWorks, the final model was loaded in a finite element analysis to find yield initiation points. By locating the yield initiation points, the results of the finite element analysis could be compared to the failure locations in the printed beam. This method, while simple, loses accuracy of results. The goal of the non-penalized beam was to find a more accurate representation of an optimized beam without varying the results caused by penalization. Therefore, a second method was developed to avoid the “guess and check” work of finding the thicknesses and find the true thicknesses of the density gradient.

The second method is similar to the first, however, it uses more MATLAB code to increase the accuracy of the related thicknesses and density boundary locations. Once the non-penalized beam image is produced from Sigmund’s MATLAB code, the image is saved to the MATLAB directory so that is can be used for image processing. The first step is to identify the boundaries of the similar density regions. Each pixel in the image is an element in the mesh of the beam. The pixels each have an intensity value assigned to them that determine the color that they are assigned. When pixels of similar intensity are called, they are assigned a value of 1 while all other uncalled pixels are assigned a value of 0. This process will run for different intensity values each time displaying the pixels with a 1 value as white and those with a 0 value as black. The black and white images produced are then considered the different layers of the density gradient. The next step is to identify the thickness of the previously identified layers. The intensity of each pixel is used to compare to the color scale of the original image. The color scale has 64 different colors in it that can be related to a thickness. If the intensity of the pixel relates to red (a void in the beam and color number 1 in the color scale) then the pixel is assigned a value of 0. For a pixel that is related to blue (a 100% density value in the beam and color number 64 in the color scale) the assigned value is 100. The assigned values relate to the percent of the material density so that the thickness can be applied. This value assignment is done for each layer so that an accurate thickness is applied to each layer.

3.1.2.2. Non-Penalized Beam Design

Like the MATLAB penalized beam, the objective and constraints of the problem need to be defined before optimizing. To keep the test the same, the objective and constraints for each beam are required to remain the same. The objective of the optimization is to minimize the compliance while reducing the weight of the beam by 50% of its original volume. The beam needs to remain the same size as well to provide comparable results.

In Sigmund's code, the inputs are similar to the penalized beam. The dimensions of the elements are 200 elements in the x direction and 75 elements in the y direction. The scale of x elements to y elements remains to be 2.667 times larger to provide for the 2 inch half beam length to the 0.75 inch height. The weight reduction is still desired to be 50% of the beams original volume so the volfrac is chosen to be 0.5. To remain consistent in the filtering technique, the filter radius is also kept at 1.5. The difference between the penalized beam and the non-penalized beam inputs is the penalization factor. Because the goal of the non-penalized beam is to not penalize intermediate densities, the penalization factor is chosen to be one. This penalization factor will make the optimizer return the true density of each element instead of a penalized value because raising the design variable to the first power will result in the same value. In order to process the image with a clearer view of the density boundaries, the optimization code was changed to output the image as a colored image with a 64 color scale instead of a gray scale image. The output of the optimizer is then a colored image of half of a beam with intermediate densities. This image is then processed to produce the desired thicknesses for the various densities.

For the second method of the MATLAB non-penalized beam, a MATLAB code was written to accurately determine the boundary locations and thicknesses of the gradient (Appendix B). The first step is to import the colored beam image from the optimization code. The image is then separated into its three main colors of red, green, and blue. On the color scale, a 100% red color signified a zero density so a new image was created by removing red and combining green and blue. The result is a blue, green, and black image with black being a zero density area (Figure 10). The black color was desired because the pixel intensity value assigned to it is zero which makes the thickness calculation easier to manage.

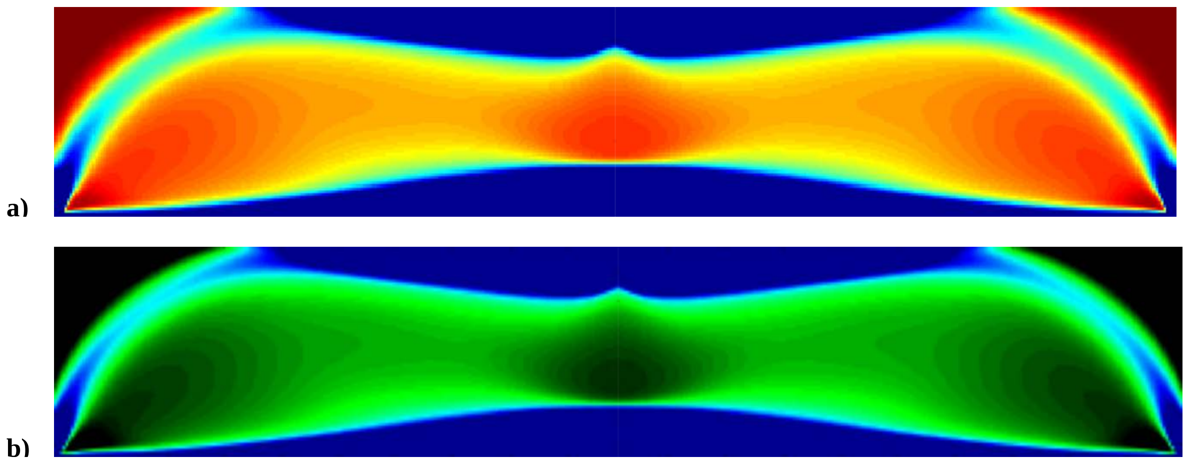


Figure 10: Non-penalized beams with layer locations from color variations a) beam with red, green, and blue as density gradient layout b) beam with red removed for simplifying calculations

The green and blue image displays a clearer representation as to where the layer boundaries are located. In the green region, pixels of similar color are grouped together so that they form an area. These colored pixel areas share the same intensity value which makes the layer location and thickness calculations easier to calculate.

Once the new image is created, layer locations are calculated. For each pixel in the image, a region of interest calculator is run to determine which pixel values lie within a certain range. The first iteration will look at the intensity of each pixel and group like pixels together. All pixels that fall within the starting range will be given a value of one while all pixels that fall outside of the range are given a value of zero. The next iteration will adjust the range so that the beginning of the second iteration range is at the end of the first iteration range. This change in range creates new values for the pixels that fall within the new range. This process continues until each pixel is grouped together with pixels of the same values. Each iteration will output an image that is black and white (Figure 11).

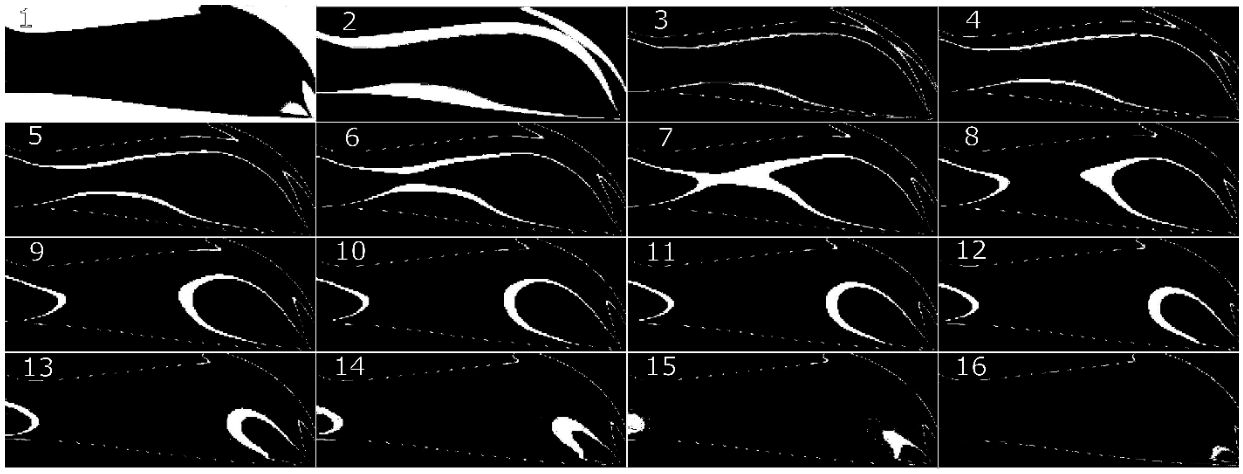


Figure 11: MATLAB generated layer locations for a half beam

The white region are the pixels that fell within the certain range that are assigned a value of one while the black region are the pixels that are outside of the range and are assigned a value of zero. For each iteration the region of interest is adjusted so that the next layer calculated is next to the previous one, decreasing in thickness. In Figure 11, panel 1, both the blue and black pixels from Figure 10b are grouped into one layer. This grouping by itself can be misleading because it appears that the area of full thickness and no thickness are in the same layer when the colored image shows that they are each on opposite extremes of the thickness. The grouping of blue and black pixels will be corrected in the thickness calculations.

After the layer locations are found, the thickness of each layer needs to be determined. The pixel intensity is once again used. Because the beam image used for thickness calculations consists of green, blue, and black as the main colors, the image is broken down to three different sections. A higher blue intensity value will resemble a higher density while a small value of each color will result in a black region of a lower density value. The green value is designated for the density gradient between the full and empty density. A series of filters is run to decide the intensity of the pixel so that it can be related to a thickness. If a pixel has no other color but blue, then the thickness will be 100% of the original thickness. If a pixel has no color in it (black area) then the thickness is not going to exist. For areas that only have a green value, the value is related to the color scale to determine what the thickness percentage will be. Because the color scale is linear, a linear equation is used to relate the green intensity value to the color scale value. This equation is represented by

$$\%t_n = i_p(0.0976) + 14.16 \tag{12}$$

where $\%t_n$ is the new percent thickness of the layer and i_p is the pixel intensity. The slope and y-intercept from Equation (12) are derived from known values at the extremes in the linear equation range. This thickness percentage is then related to a thickness of the beam when multiplied by the original beam thickness. Because the remainder range is a mixture of blue and green pixel values, the previously used linear equation cannot be used. This range is made up of two similar ranges that use a manual filter to determine the thickness of the layer. In one segment of the range, the blue values of the pixels remain mostly unchanged at its peak value while the green values vary. The other segment of the manual filter range has the green value close to its peak value while the blue value changes. In this range, a manual filter is applied so that any pixel that has a green and blue value will give an appropriate thickness. The color scale used for determining the element thickness is displayed in Figure 12 which includes the ranges for thickness calculation methods.

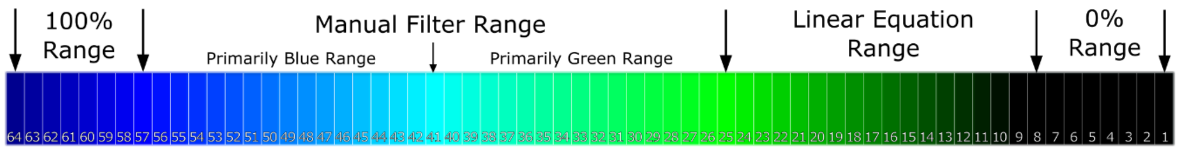


Figure 12: 64 color scale used to define thicknesses in color regions

As the color in the non-penalized beam image moves from right to left in Figure 12, the thickness will increase. A higher number (the highest being 64) will result in a thicker layer in the beam while a lower number will result in a zero thickness layer. While the manual filter range takes up the largest section in the color scale, the manual filter range has a smaller area of effect in the actual non-penalized beam. This area is only in effect on the boundary between the completely green and completely blue areas. When the thickness is calculated, it is calculated for the whole thickness of the design domain. Because the beam must keep the material at its center to prevent eccentric loads, the thickness calculations will be based off of the central plane. The layers will then have to extrude in two opposite directions from the central plane so the thickness calculation must be halved.

To get a three dimensional model for printing, the layer locations and thickness calculations are used. The layer location images are imported into SolidWorks starting with the thickest section (Figure 11 panel 1) and working down to the smallest (Figure 12 panel 16). After each panel is imported, the image is located in the same position each time so that the images are in the exact location as the one before it. The white segments depicting the layer is then traced and extruded to the thickness calculated for that layer. This process is repeated for the next layer until all images are traced. The half beam is then mirrored to create a full model of a beam. To prevent the beam from falling through the supports, non-structural supports are added to the sides at the base.

An automatic method for importing the thickness values at layer locations was also explored. When the thickness calculator calculates the thickness at each element, it creates a matrix of thickness values. The values can then be converted into xyz coordinates. Each element in the matrix is a certain element over in the x and y direction. The thickness value for that element then becomes the z value. A table can then be made to output the new xyz coordinates where x and y locate the element in the beam and z is the thickness of material. Running a mesh using these numbers gives a representation to how the final design will look (Figure 13).

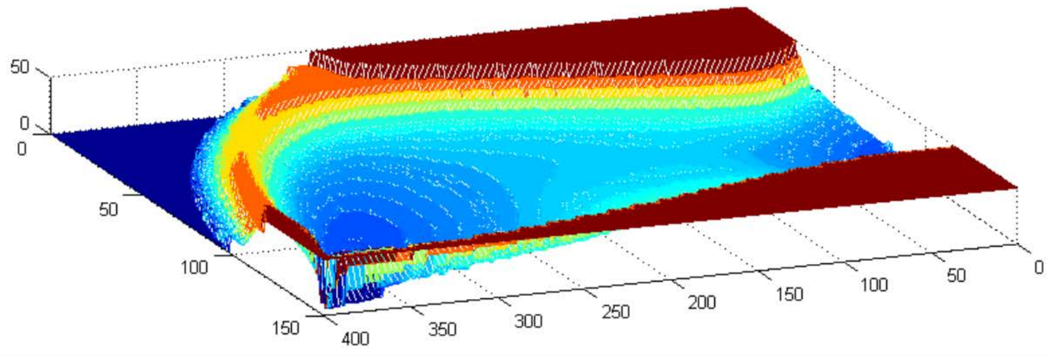


Figure 13: Three dimensional representation of thickness calculations at element locations

The dark blue areas in Figure 13 are the zero thickness sections and the dark red areas are the full thickness sections. The scale is based off of percentage of the original beam thickness which is 0.25 inch. The scale for the thickness only goes to 50 because the plot represents a quarter section of the beam so the thickness is only 50% of the total beam thickness. The plot would be mirrored across the x, and y plane to get half of the beam and then mirrored across the y and z plane to get the other half. The new xyz coordinates can be saved as a xyz file to which SolidWorks can read and import the values. These values can then be used to turn the data points into a solid body. However, the large number of data points produced requires more computing power than available. A MATLAB code written by Sven Holcombe (2011) (Appendix D) uses the xyz coordinates to produce a STL file. This STL file can then be more easily opened in SolidWorks as a graphics body. This graphics body is only a surface of the beam section (Figure 14) because the xyz coordinates only produce locations of the surface and no points within the solid areas of the beam.

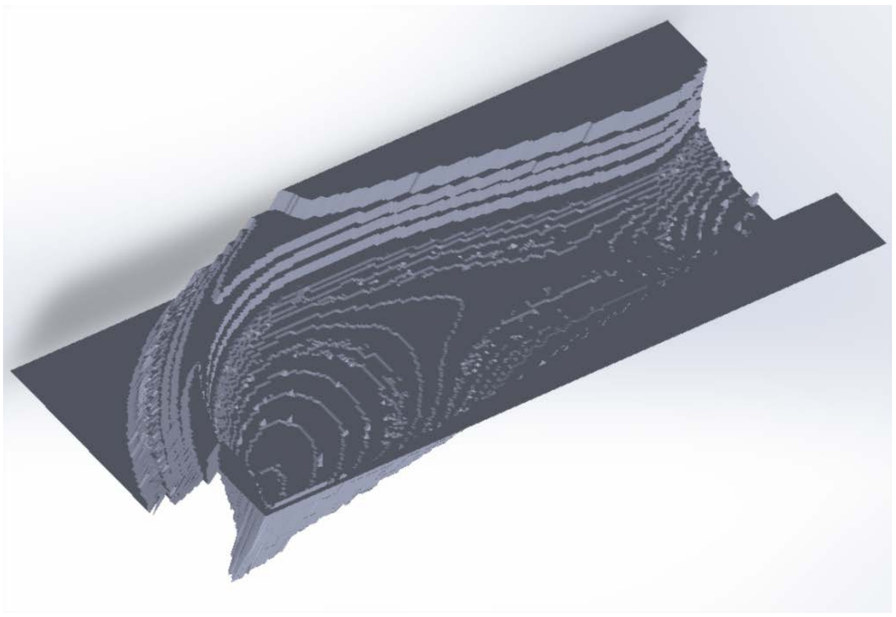


Figure 14: Graphics body of beam section created in MATLAB and imported into SolidWorks

Post importing extrusions could possibly be done to give the surface depth, however, the amount of surfaces and points is too large for SolidWorks to import as a solid body. A possible solution is a method to simplify the surface. If an element has the same thickness value as an element next to it, the points can combine to produce a single surface shared by many points. A method of simplifying the amount of surfaces and points could increase the speed and ability to extrude the STL surface.

3.2. ParetoCloud

ParetoCloud is a browser based software used for designing components with topology optimization (SCIART, 2017). The software was created by SCIART, LLC and is used in this research for a comparison to the other optimized beams.

3.2.1. ParetoCloud Method

Like the MATLAB penalized method, ParetoCloud uses a penalization method for solving the topology optimization problem. The results of the ParetoCloud optimization method has solid and void regions due to intermediate density penalization. A STL file is imported into the browser to be used as a design domain. Different boundary conditions can be added to the part such as loads and fixed areas. The boundary conditions help to define the problem for optimization. Because the part is imported, the problem can also be more complex than a simply supported beam with different loading and fixed areas. However, for comparison, a simply supported beam will be optimized. The user also has more control on the final design of the component. If symmetry in any of the three dimensions are expected, the user can select which axis the symmetry is on. The symmetry option will provide a simpler solution for the optimization. Like most topology optimization software, a finite element analysis is used with the boundary conditions defining the problem. The deflection and stresses within the component are displayed once the finite element analysis is complete. The data from the finite element analysis is also used for further optimization processes. This software also focuses on the ability to print the final solution. When the topology optimization part of the program is ready to be run, different print parameters are considered for the design. The draw direction can be selected to be any of the axis so that when optimized, the result will limit the amount of support material needed making it easier to print the component. The user must also define the final volume fraction of the component much like the other optimization software. When optimized, the result will be a solid and void component that is exported as an STL file.

3.2.2. ParetoCloud Beam Design

The Paretocloud software, like the MATLAB program, uses a pre-defined design domain to optimize. Instead of a two dimensional design domain, ParetoCloud makes use of the full three dimensional design space. To get a three dimensional model, a full beam is created in SolidWorks and saved as a STL file for ParetoCloud to import. Small areas that will define the location of the force and supports also need to be located on the beam. Because the load and supports are close to a point load, the area is made to be small. The load and support areas had dimensions of 0.25 inch by 0.1 inch. The area where the load is applied is located on the top of the beam centered at the beams midpoint while the area for the supports are located on the bottom of the beam on either end. Once completed, the STL is imported into ParetoCloud for boundary conditions to be applied. A fixed xyz boundary condition was applied to one of the support areas and a fixed yz boundary condition was applied to the other support. These boundary conditions keep the xyz fixed support from displacing in any of the three directions. The yz fixed support is restrained from displacing downwards and to the side. For this support, the beam is still allowed to move lengthwise as the beam bends. A normal force is applied to the designated area on the top of the beam. This force is made to be 20lbf. The material of the beam must also be defined. The material properties of the printed plastic is entered into the optimizer so that when it runs the finite element analysis and optimization, it will have accurate material properties for calculations. No other boundary conditions are needed to define this problem so the next step is to run the finite element analysis. A medium mesh was selected to improve the accuracy of the results as compared to the coarse mesh but save computing time as compared to the fine or very fine mesh. Before running the finite element analysis, component symmetry can be defined. For a simply supported beam, z axis symmetry can be applied perpendicular to the front face of the beam. If no symmetry is applied, one side of the beam can experience more material excavation than the other which will create a beam with eccentricity. The last step before optimizing the beam is to select the volume fraction. To compare this optimized beam with the rest, it must have the same volume fraction as the others so a volume fraction of 0.5 is selected to reduce the volume by 50%. With all the boundary conditions and constraints in place, the optimizer can be run with the result displayed in Figure 15.

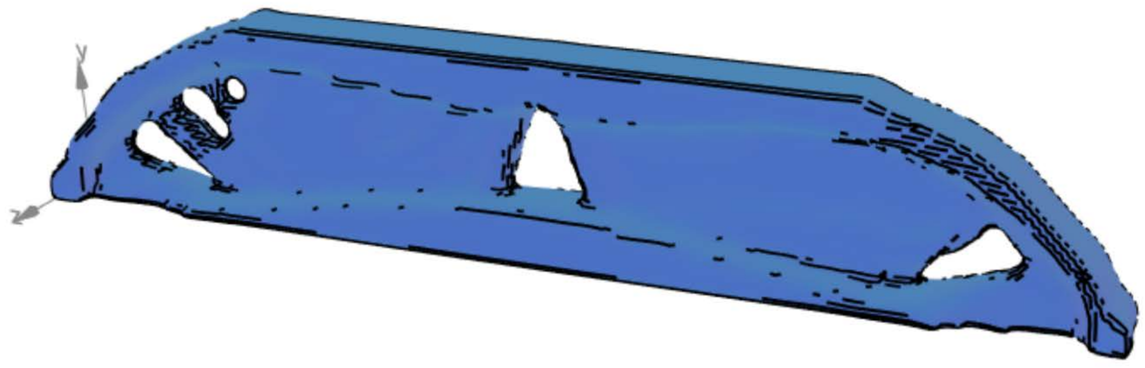


Figure 15: ParetoCloud optimized beam result

The output of the optimizer is a STL file of the optimized beam with a 50% reduction in volume. The same as the other optimized beams, the span of the beam is the exact length as the supports for testing. The exported STL file is imported into SolidWorks as a solid body to add the additional non-structural supports so the beam does not fall through when the load is applied.

3.3. Applied Variables and Constraints

For the three beams modeled using computer software: MATLAB penalized beam, MATLAB non-penalized beam, and ParetoCloud beam, the design constraints and boundary conditions are given in Table I.

Table I: Optimized beams design variables and constraints

Sample	Penalization Factor	Volume Fraction	Mesh Size	Filter Radius
MATLAB Penalized Beam	3.0	0.5	100x37	1.5
MATLAB Non-Penalized Beam	1.0	0.5	200x75	1.5
ParetoCloud Beam	N/A	0.5	Medium (50000 elements)	N/A

The ParetoCloud beam had some additional inputs that the MATLAB beams did not require. A 100 lb load was applied to the top center of the beam over an area of 0.25 inch by 0.1 inch. A xyz fixed constraint was placed on one bottom edge over the same area as the load. A slider condition was applied to the other end of the bottom of the beam over the same area as the load. A draw direction in the z-axis was also applied to resemble the printing direction.

3.4. Simple Machining

Before additive manufacturing became a feasible solution to manufacturing components, simpler solutions are implemented to reduce the weight of components. The most common are slots and holes cut from the material. These solutions are used because of the simplicity in the weight reducing methods. A hole and slot can easily be cut into a component with common tools.

For a simply supported beam, the upper half of the beam undergoes compression while the lower half experiences tension. The further out from the center of the beam (neutral axis) the higher the bending stress. The bending stress can be modeled by

$$\sigma_{b,max} = \frac{Mc}{I_c} \tag{13}$$

Where $\sigma_{b,max}$ is the bending stress, M is the bending moment, c is the distance from the neutral axis to the extreme fiber at the top and bottom of the beam, and I is the moment of inertia of the beams cross section. From Equation (13), the bending stress increases the further out from the neutral axis meaning that at the neutral axis there will be no bending stress and locations close to the neutral axis will have very small bending stress. If material is to be removed from the center, the beams stiffness would theoretically have a very small change than if material was removed elsewhere.

An added benefit of the slot and holes method is the lack of stress concentration points. The nature of drilled holes and slots allows for a curved hole without any sharp cornered holes. For components that will experience loading, a stress concentration point can create a spot for crack initiation.

3.4.1. Slots

3.4.1.1. Slots Method

The purpose of a slot is to remove material from the center of the beam where there is less bending stress to reduce the weight of the beam. Timothy Demers (2009) modeled a cantilever beam with a slot running through the center length of the beam while leaving material on either end to connect the top and bottom sections of the beam. This method was then applied to the simply supported beams. For the simply supported beam, two slots are modeled so that the slots run through the thickness of the beam and along the neutral axis. Two slots are chosen because the load applied at mid-span would buckle the upper section of the slot. The middle section of the beam under the load would remain intact so that no buckling from the applied load would occur.

The largest concern with this slot design is the buckling in the upper section of the slot. The material above the neutral axis experiences compression under a three-point loading test so the small sections of material above the slot are more likely to experience buckling. Therefore, the maximum force that can be applied may be limited by the size of the remaining material. The equation for buckling force is given by

$$F_b = \frac{\pi^2 E I_b}{(K L_b)^2} \tag{14}$$

where F_b is the maximum applied force before buckling, E is the modulus of elasticity, I_b is the moment of inertia of the cross sectional area of the upper section of the beam, K is the effective length factor, and L_b is the length of the beams upper section. When the cross-sectional area of the upper beam material is small, the moment of inertial lowers which then causes the maximum force that can be applied to decrease. A wider slot cut from the beam will result in a smaller cross-sectional area of the upper material causing the maximum force to lower. Also, from Equation (14), the longer the length of the upper beam material, the smaller the amount of force can be applied before buckling meaning a longer slot will make it so the amount of force the upper beam material can resist is decreased. With these considerations, two slim slots are considered so that the chances of failure due to buckling can be reduced.

3.4.1.2. Slots Design

For the slotted beam, there is no software used to optimize the size of the slot. The objective of the beam is to still remove 50% of the original beams volume, however, with the slot, it becomes increasingly difficult to keep strength and reduce the weight to the desired amount. The length of the slot must increase and the cross sectional area of the connecting material must decrease. This concept greatly decreases the stiffness of the beam. In order to combat this, the design of the slot hole will be smaller so that the resulting beam volume is greater than 50% of the beams original volume. Even being larger than the 50% volume, the slotted beam is not expected to perform as well as the other optimized beams.

The slot height is designed to be a third of the beams total height which will leave a third of the beams height above and below the slot as connecting material. Because the beam height is 0.75 inch, the slot height and connecting material will be 0.25 inch. With the length of the beam being 4 inches, the slot length must be shorter than half of the beam length to allow for two slots to be designed. If the slots are too short, the removed material will not be close enough to the 50% removed material. However, if the slots are too long, there will not be enough material on the ends to support any load. With support material in mind, the total length of the slot is chosen to be 1.75 inches. This slot length allows 0.25 inch between the two slots in the middle and 0.125 inch by the supports on the ends. The dimensions and layout of the slotted beam are drawn in Figure 16.

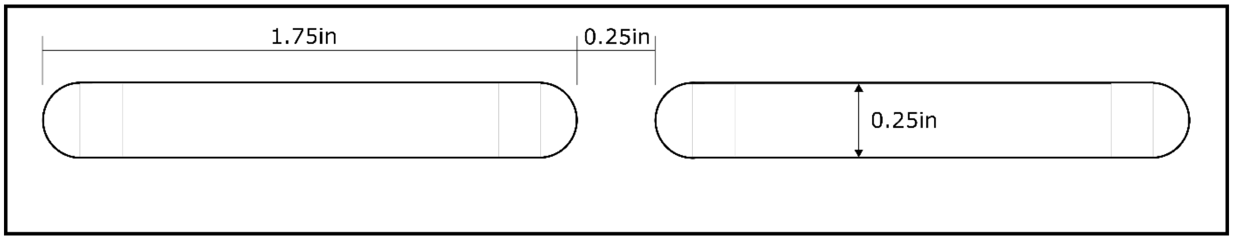


Figure 16: Slotted beam dimensions and layout

To begin modeling the slotted beam for printing, a solid beam with the dimensions of 4 inches by 0.75 inch by 0.25 inch was modeled in SolidWorks. The slots are then cut from the beam along the neutral axis. As with the previous beam, additional non-structural supports are added to the ends of the beam to prevent the beam falling between the supports when loaded.

3.4.2. Holes

3.4.2.1. Holes Method

Similar to the slot method, the hole method removes material from the center of the beam along the neutral axis. This process will remove material that has a low stiffness contribution from the component making it lighter. This method is commonly used because of the easy manufacturing of the lightening holes. A simple drill can cut the holes in strategic locations.

To avoid the buckling concern in the slot method, the hole method utilizes connecting material between the holes to give strength. The length of the beams upper material is now the diameter of the hole which is shorter than the slot. Using Equation (14), the maximum force acting perpendicular to the cross sectional area is larger than that of the slot. However, because there are more instances of supporting material between the holes, there is less material removed from the beam. If the objective of the optimization is to remove a certain amount of volume from the beam, then the size of the holes will need to increase which will then greatly decrease the strength of the beam. For small material removal objectives, the hole method is a viable solution, however, when the need for more material removal is required, the hole method may not be able to remove enough material while maintaining strength.

3.4.2.2. Hole Design

As with the slot method, there is no optimization software that is used for material removal using the hole method. The hole method is also not going to reach the 50% volume removal that is required by the objective function. In order to remove 50% of the material using the hole method, the holes would have to be large and closely spaced. At that point, the strength of the beam would have greatly decreased.

The holes at the ends of the beam are made smaller than the rest of the holes. Described in a report by Demers (2009), the ends of the beam experience more stress than the middle so more material can be removed closer to the center of the beam while the holes close to the edge are kept smaller to resist deformation. Eight holes in total are designed into the beam with constant dimensions and locations (Figure 1Figure 17).

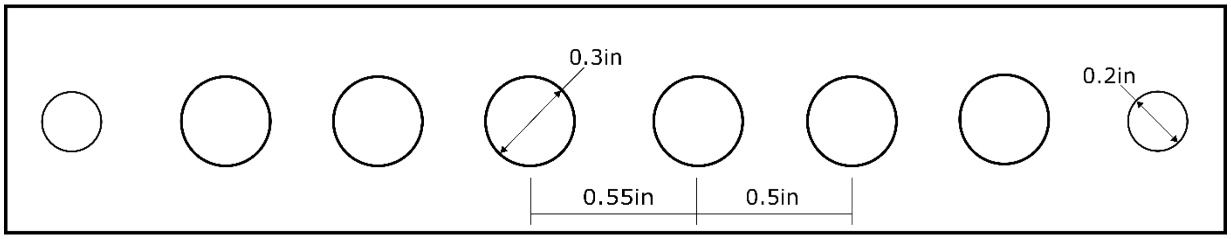


Figure 17: Dimensions and layout of the beam with lightening holes

The two outer holes are 0.2 inch in diameter and the other six are 0.3 inch in diameter. The distance between the centers of the holes is 0.5 inch. However, the distance between the two holes close to the center of the beam where the load will be applied is 0.55 inch. The increase in hole spacing provides more support material to resist the load. To model the beam with holes, another solid beam is extruded in SolidWorks and the holes are cut from the beam along the neutral axis.

4. Printing

Once the beams are designed and modeled, they are sent to a three dimensional printer. A Stratasys uPrint SE 3D printer was used to print the beams. The uPrint SE printer uses a fused filament fabrication process to build up layers of material. An ABS P430 filament was fed through a head that moves in the x, y, and z directions. The head heats up the filament to a temperature, which the extruded material will bond to the material it is extruded upon. The printing head prints out cross sectional areas slices. Each slice builds up the part in the z direction. Layers are added to the part until the full dimensions of the part are reached. For sections that have empty space below a solid section, support material is used. The support material is used as a base for the solid material printed above an empty space, so that the printed material will not deform under its own weight upon extrusion. After the 3D printed part is finished, the support material is then removed.

4.1. Printed Material Tests

Before printing of the test beams began, tests were performed on the printed filament to determine the printed filament material properties. Three different standard tests were completed on the printed filament that was to be used on the test beams. The test data were used determine if the filament may be assumed isotropic and provide empirical models on how the material reacts under compression, shear, and tensile forces. The empirical models were used to estimate the compression modulus, shear modulus, modulus of elasticity, and poisons ratio. Two different print orientations were considered for these test. After printing the test specimens flat (0 degrees), they were rotated 90 degrees from the 0 degree orientation to the long edge to test the effect of build direction (Figure 18).

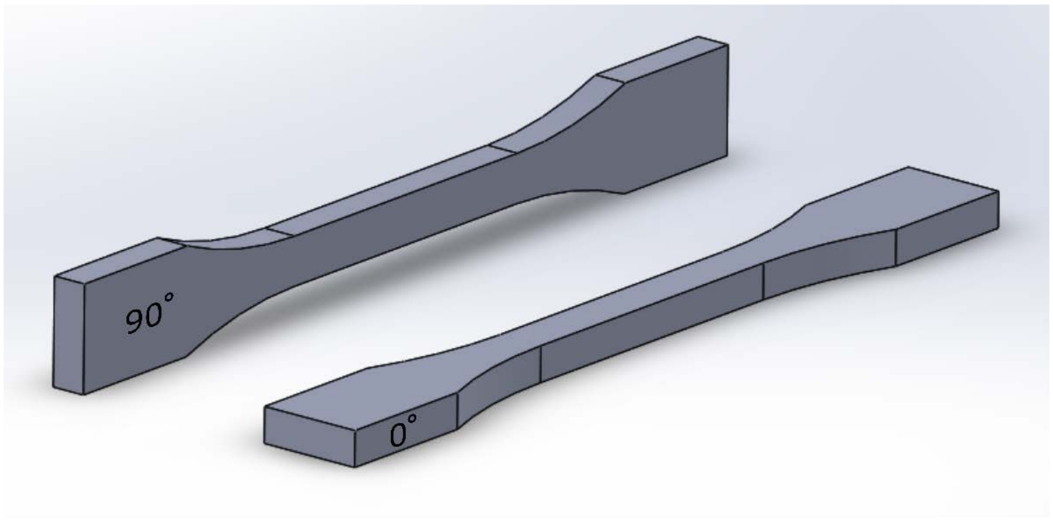


Figure 18: Print orientations of tensile testing specimens

Different fill levels of color pigment was also explored, because certain colors require more pigments in the material to create the desired color effect. The hypothesis is that the higher the filler material, the lower yield point that the material will produce because the pigments added may not be as strong as the plastic Various tests were performed to generate data that were used to quantify the change of material properties with varying fill loading. Strain gages were attached to each specimen to get an accurate strain measurement. The stress was calculated by Equation (15).

$$\sigma = \frac{F}{A} \tag{15}$$

where σ is the stress, F is the force applied, and A is the cross sectional area of the test specimen perpendicular to the applied force.

4.1.1. Compression Testing

A compression test was used to determine the compression modulus of the samples. The compression modulus was used as a material property input for computer modeling for a more accurate static loading analysis. The test specimens were cylinders that were 0.75 inch in diameter and 1 inch tall. The speed of compression was set to 0.05 in/min, making the test a quasi-static loading test. A general purpose CEA-13-240UZ-120 strain gage was a used on each compression test along the axis of loading. Two green and two white cylinders were tested with their orientation the same direction according to ASTM D695. Because the compression modulus was the desired material property for this test, the test proceeded until the test specimen experienced plastic deformation. After yielding, the test specimen passed its elastic range so the compression modulus could be calculated by taking the slope of the linear plastic range. The compression modulus was calculated for each color and presented in Table II.

Table II: Compression testing modulus results

Sample	Compression Modulus (ksi)
Green Compression 1	320
Green Compression 2	312
White Compression 1	317
White Compression 2	297

The compression modulus was the calculation at the linear region and therefore, the closest to the actual compression modulus. The green samples had a larger average compression modulus. The variation of compression modulus between the two green samples was 8 ksi while the variation of compression modulus between the white samples was 20 ksi. Because only two samples were tested, the green compression results can be hypothesized to have a more reliable compression modulus.

4.1.2. Shear Testing

A shear test was used to determine the shear modulus, which was used in computer modeling for a more accurate static loading analysis. Shear was induced in the test specimen by the V-notch beam method. A beam that had a length, width, and height of 3 inches by 0.75 inch by 0.25 inch respectively had two notches taken out of the side. The v-notches were 0.15 inch deep and had an internal angle of 90 degrees (Figure 19).

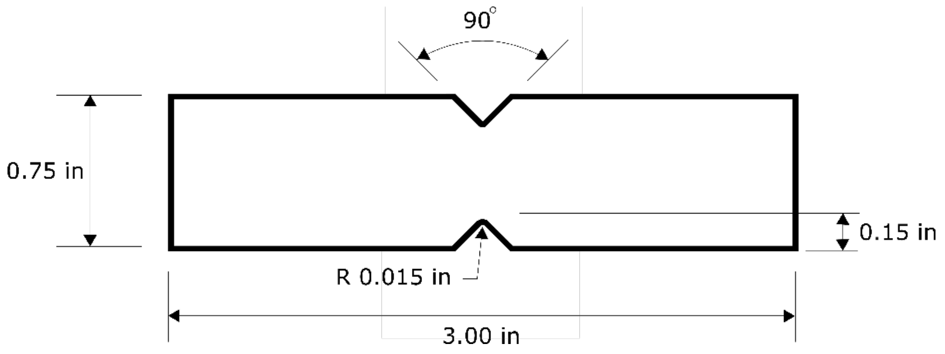


Figure 19: V-notch specimen dimensions

These notches were placed in the middle of the top and bottom of the beam. Two green and two white beams were tested using general purpose EA-06-062TY-350 strain gages as the data acquisition devices along the axis of loading using ASTM D5379. The first green and white beams were printed flat on the build plate (0 degree build orientation) while the second green and white beams were printed on edge with the v-notch on the top and bottom (90 degree build orientation). The speed of the test was set to 0.05 in/min as in the compression test. The stress strain curve provided a linear area where the shear modulus was calculated. The calculated shear modulus for each color and build direction is displayed in Table III.

Table III: Shear testing modulus results

Sample	Build Angle	Shear Modulus (ksi)
Green Shear 1	**0°**	111
White Shear 1		121
Green Shear 2	**90°**	112
White Shear 2		96

The shear modulus from the white test specimen at the 0 degree build angle had the highest shear modulus while the white test specimen at the 90 degree build angle had the lowest shear modulus. Tests performed on the green test beams produced the most consistent results with a variance of 1ksi between the two build angles. The white test beams had a relatively large variance in shear modulus of 25ksi. Because only two samples were tested, there were not enough results to quantify the data. However, the green samples were hypothesized to give more consistent results.

4.1.3. Tension Testing

The tension test was used to determine the tensile modulus of the printed material. The tensile modulus is used in computer modeling of the optimized beams for a more accurate static loading analysis. For this tension test, a flat bar with a slim midsection had opposing forces act axially to create the tensile forces. The midsection that experiences the deformation had a length, width, and height of 2.25 inches by 0.5 inch by 0.25 inch respectively. The white material tension bar had a general purpose CEA-06-125UT-350 bi-axial strain gage while the green material had a WK-06-125TM-350 bi-axial strain gage. Like the shear test, the tensile test bars were printed in two different orientations with white and green filaments. The first orientation was flat on the build plate at 0 degrees and the second was on its side at 90 degrees. The tensile test was performed in accordance with ASTM D638. The modulus of elasticity was calculated from the slope of the linear region on the stress strain curve. These values can be seen in Table IV.

Table IV: Tension testing modulus results

Sample	Build Angle	Elastic Modulus (ksi)
Green Tension 1	**0°**	422
White Tension 1		304
Green Tension 2	**90°**	385
White Tension 2		352

The green colored bars had a larger modulus of elasticity than the white colored specimens. The variation between the two build directions was also smaller for the green than the white. Between the 0 degrees and the 90 degrees, the green bar had a difference of 37ksi while the white had a difference of 48ksi. For the green specimen, the 0 degree orientation had a larger modulus than the 90 degree specimen. The opposite is true for the white specimens, the 90 degree orientation had a larger modulus than the 0 degree orientation. Because only two samples were tested, the results were not enough to quantify the data so the green material can only be hypothesized to have a consistently larger elastic modulus.

4.1.4. Material Testing Conclusion

From these tests, the green colored filament was hypothesized to perform better than the white colored filament. Except for the sheer modulus, the green filament test samples had larger values for the compressive modulus and Young’s modulus. The higher modulus values were desirable for the three point bending tests that will be conducted for the optimized beams. The green filament was also hypothesized to have a more consistent result when compared to build angle. As the angle changed, the various values for the green filament changed less than the values for the white filament. For the purpose of testing, the smaller variation in values based on the print direction is more desirable because the filament can be more closely related to an isotropic material.

4.2. Printed Beams

After the green material was selected for the testing, each beam is printed in the same orientation. The 4 inch by 0.75 inch face is flat on the build table and it is extruded upwards the 0.25 inch. This direction was initially chosen because most beams will not require support material when built up from this side. The beams were also printed at 100% fill, which does not necessarily mean that the beams will be clear of voids, but, the beams will be printed at the printer's maximum density capabilities. Between each cross-sectional layer and rows on the cross section, there will be small gaps because the extruded material has a cross sectional area that resembles a circle. Circular cross-sectional areas cannot form a perfect fit with neighboring material. As each layer prints, the orientation alters direction by 90 degrees between layers. The angle between the bottom of the beam and the first extrusion direction is positive 45 degrees. The next layer has an angle between the bottom of the beam and the extrusion direction of negative 45 degrees.

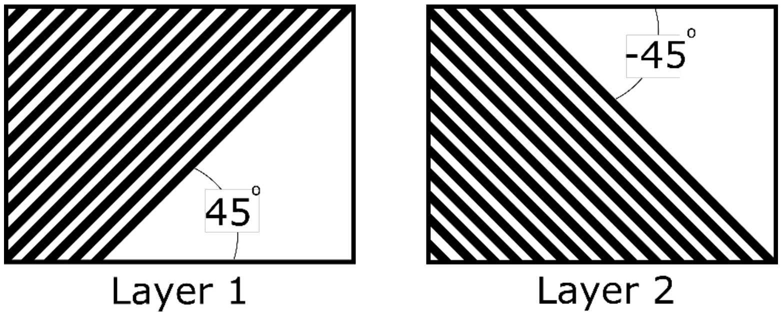


Figure 20: Extrusion line angles

The angles will alternate between this positive 45 degrees and negative 45 degrees each layer. The alternating angles helps bond the layers and prevents large gaps between the layers from running through the full thickness of the beam.

A total of 24 beams were printed including four beams by each different method for optimization. Figure 21 depicts the beam configurations where a) is the solid beam that will be used as a standard for comparison, b) is the beam with lightening holes, c) is the beam with slotted holes d) is the MATLAB penalized beam, e) is the MATLAB non-penalized beam, and f) is the ParetoCloud beam.

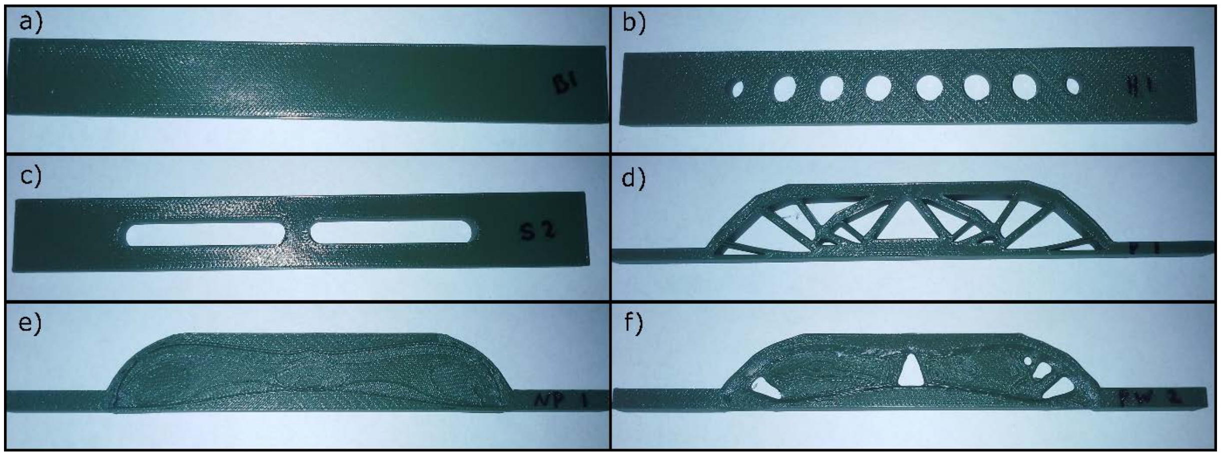


Figure 21: a) printed solid beam b) printed beam with holes c)printed beam with slots d) printed MATLAB penalized beam e) printed MATLAB non-penalized beam f) printed ParetoCloud beam

Each beam was labeled on the support section to keep track of the beam. B1-B4 represents the four solid beams, H1-H4 represents the four beams with holes, S1-S4 represents the four beams with slots, P1-P4 represents the four beams optimized using MATLAB with penalization, NP1-NP4 represent the four beams optimized using MATLAB without penalization, and PW1-PW4 represent the four beams optimized using ParetoCloud.

4.3. Problems with Printing

Additive manufacturing is still evolving. The technology and methods for three dimensional printing have not been perfected and still requires some external work from the user.

For topology optimization, the optimized beams are assumed to have a full density with complete bonding between extrusion lines and layers. A full print density is not always the case for certain additive manufacturing methods. Each method has strengths and weaknesses that can have a large effect on the final product.

4.3.1. Print Density

For topology optimization, it is important for the final product to obtain solid sections. After the optimization process, the result is a component with distinct solid and void regions. The solid regions were created to be 100% of the material density. Any variation in this density will cause errors in the printed component. For the fused filament fabrication method, voids in the solid regions were observed because of the way that the layers are created. To view the voids in the beam, a MATLAB penalized beam print was interrupted to show the cross sectional area (Figure 22). The first arrow in Figure 22 shows a section where there was improper bonding between extrusion lines. When the beam was printed, the extrusion head traces the outline of the beam where there is a boundary between solid and void. While this method gives a smooth finish on the outer surfaces, it can create problems within the component. In some thin members the extruded lines of the boundaries are too far apart to properly bond with each other, but too close together for filler material. As each layer builds, a long crack will be built within the full thickness of the section. With this crack between the extruded lines, the lines are allowed to slip past each other or buckle under loading due to the decreased cross sectional area.

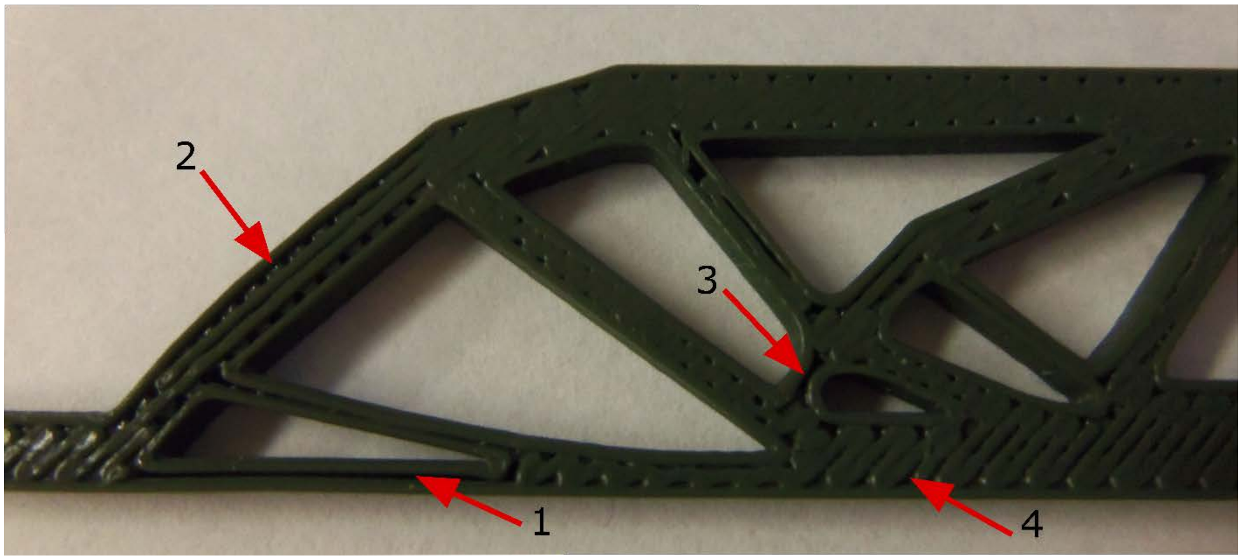


Figure 22: Cross section of the MATLAB penalized beam with marked voids

The second arrow in Figure 22 provides an example of how the layer direction effects the porosity of the component. Because each layer is built by alternating the extrusion lines by 90 degrees between 45 degrees and -45 degrees with the horizontal, not every space in the component can be completely filled. If a section of the component were to have an angle equal to 45 degrees, then the extrusion line would be able to run parallel to the boundary at each pass. By running parallel to the boundary, the extrusion lines will be able to better fill the subsequent area in the layer until converging with the next boundary. However, the MATLAB optimized beam did not have support members that matched these angles. At the second arrow, the support does not match the filler line angle and there is a slight ark which causes the extrusion line to diverge from the boundary. The extrusion fills up the support area with as much filament as possible within the constraints of the 45 degree angle. The inability to follow the boundary leaves voids on either side of the extrusion lines for this layer. The third arrow shows a void that is a combination of the first two issues. The extruded boundary lines create a small gap because they are not close enough to bond with each other and the gap cannot be filled because its angle is not a close enough match to the line angle. The fourth arrow shows smaller void regions that exist between the extruded lines. When the extruder finishes one line, it curves into the next line which leaves a small gap at the curve. Instead of curving into the next line, it may be beneficial to move alongside the boundary until it has cleared the previous line, then continue on to make the next line.

4.3.2. Printing Errors

Imperfections are often created during printing. These imperfections can have an effect on the performance of the beams because they can change the geometry of certain areas or create stress concentration points.

The ParetoCloud beam (PW1) in Figure 23 was printed with extrusion defects on the top of the beam.

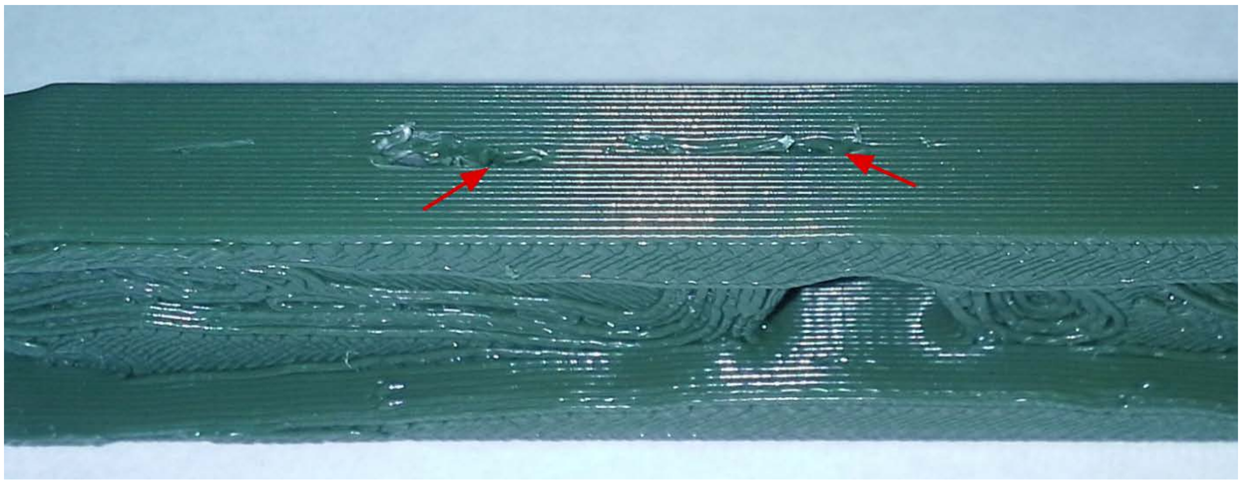


Figure 23: ParetoCloud optimized beam with extrusion defects

The extrusion defect was created when the filament material did not properly bond to previous layers. The filament moved to a position where it interfered with other layers. The intersection of layers created gaps and deposits of material in the top of the beam. Once the layers built up enough to move over the hole, the layering resumed as normal. The hole was close to the center of the beam where maximum moment and shear exist for a simply supported beam. A stress concentration point and removal of material in this location can cause a lower resistance to loading.

A similar printing error occurred while printing the third beam with holes (H3). Instead of creating a hole in the surface of the beam, there was a small deposit of tangled filament. The tangle was imbedded on the side of one of the center holes, displayed in Figure 24.

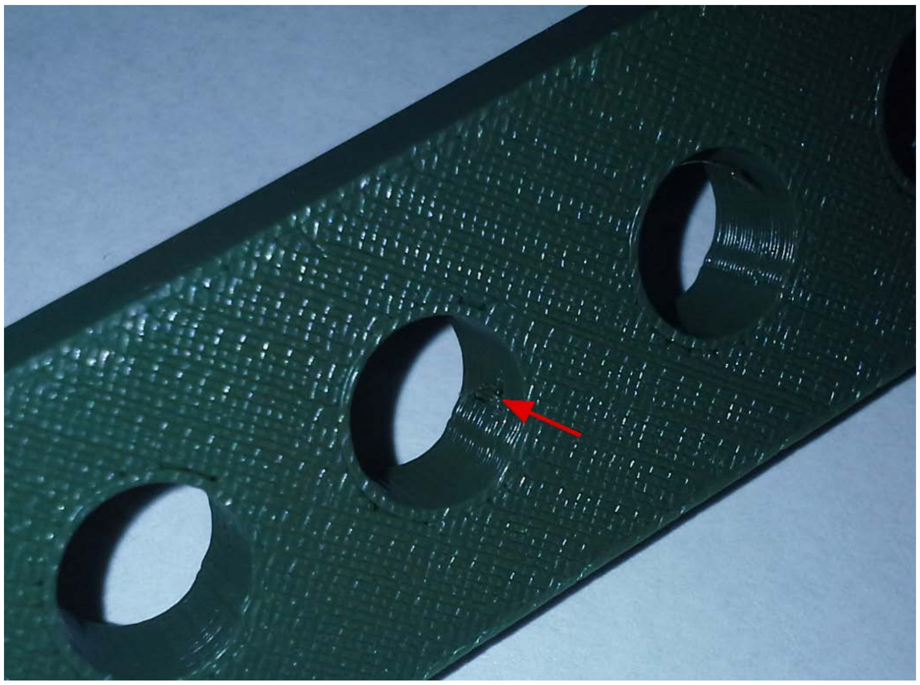


Figure 24: Beam with lightening holes with extrusion defects

The material that formed the inclusion either came from excess material from the extruding head or from material already within the beam. If the extruding head deposited excess material on the surface of the hole, it had no structural significance to the deposit. If the deposit was imbedded into the surface layers, then there will still not be a large issue because the tangled filament became part of the structure but, not in the same orientation as the other surface layers. However, if the tangled filament came from within the beam, there may be a hole below the surface where the clump originated. A hole below the clump is a larger problem because the hole is a stress concentration point where a fracture can form.

The last significant printing error occurred in the third MATLAB non-penalized beam (NP3). Unlike the other printing errors, this error did not induce a stress concentration point, but changed the geometry of the beam. A patch of material was improperly extruded that caused a divot of mixed melted material, displayed in Figure 25.

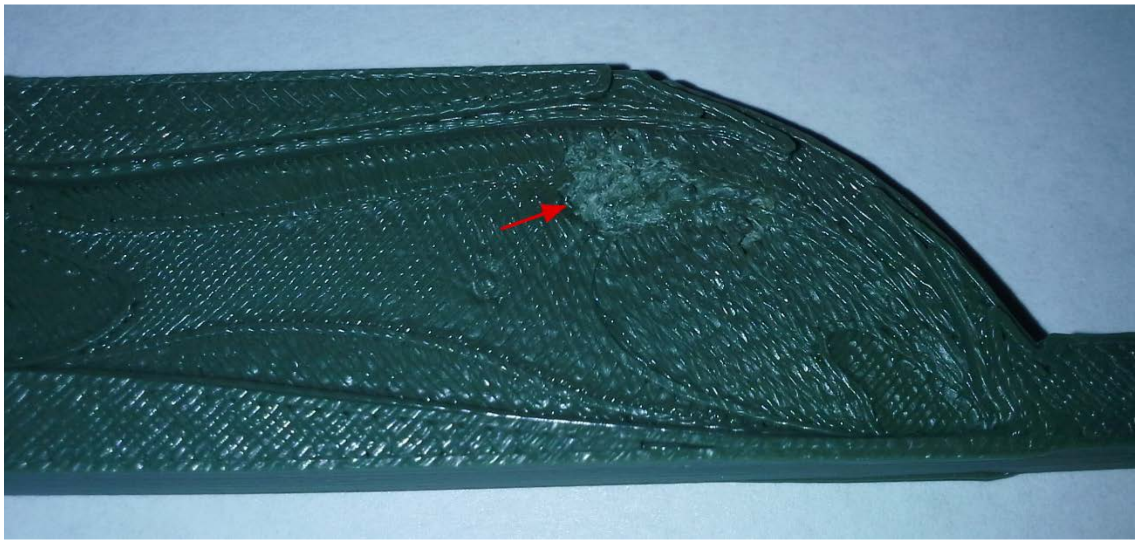


Figure 25: Printing defect in non-penalized MATLAB beam, NP3

This depression of material has a depth of four layers in a circular area with a diameter of about 0.23 inch. With a diameter close to a third of the height of the beam, the divot may have an impact on the beams performance. The accuracy of the thicknesses of the beam are important to the non-penalized beams geometry and strength so an area that lowers the thickness may decrease the strength of the overall beam.

4.3.3. Printing Layer Tolerance

The design of a component can vary from the actual build of the component when the means of manufacturing cannot match the tolerance of the design. For additive manufacturing using the fused filament fabrication process, the tolerance was not enough to accurately create the non-penalized MATLAB beam. The layer thicknesses of the beam changed by five thousandths of an inch between each layer at its smallest. The printer used to make the beam was only able to print to a tolerance of ten thousandths of an inch, twice as large as necessary to accurately print the layers. The larger tolerance meant that areas that were separated by two different thicknesses were then combined into one (Figure 26).

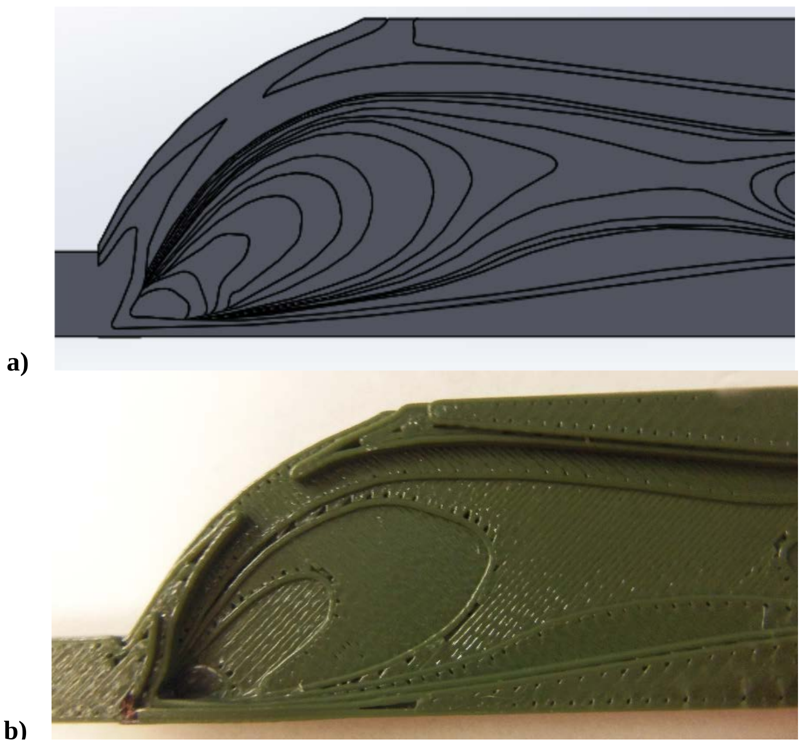


Figure 26: Non-penalized beam layers with a) designed layer locations and b) printed layer locations

In Figure 26a, there are 14 layers that have five thousandths of an inch thickness difference between neighboring layers. If the tolerance of the printer was small enough to print these thickness differences, then the printed beam would also have 14 layers in the same locations. However, the printed beam (Figure 26b) has five layers total in the same locations as the design. These layers are a simplification of the 14 layers of the designed beam. The simplification of the layer thicknesses causes some layers to decrease in thickness and others to increase in thickness to average out and combine into one layer. Because the layers are averaged, the volume of the beam should still be accurate to the 50% reduction. However, the structural stiffness may be skewed because the layers are no longer true to the density values used to calculate the thicknesses.

5. Break Results

After the beams were printed, a three-point bending test was performed using ASTM D790 to develop load-deflection curves for further calculations. The load-deflection curve was used to determine the flexural modulus, flexural resilience, flexural toughness, and maximum average load at the yield initiation point. These results are then compared to determine individual beam strengths and weaknesses. Fracture locations are also analyzed to determine the areas that experience the highest stresses and how the design could be changed to accommodate the high stress area. Each beam is compared to the original solid beam to check the variation in strength with half of the optimized beams volume removed. The length of each beam is four inches, so the support length of the three-point bending test was set to four inches with the applied load in the center of the beams span.

The locations of the supports and applied load were marked on the beam so that each beam could be arranged in the same location. Because the total beam length beam with the extra supports was six inches long and the desired beam dimensions for testing was four inches long, the support marks were placed one inch from each side. The location of the load was placed in the center of the beam three inches in from the side. The load application was applied at a rate of 0.1 in/sec until the beam failed. During the test, the load applied to the center of the beam and the extension of the load was recorded at time intervals of 0.005 second. The load and deflection data were collected from the three point bending test and plotted against each other to produce a load deflection-curve. The crosshead position was used to determine the deflection of the beams. The data from the load-deflection curve were then used to calculate flexural beam properties for comparison by use of a MATLAB code (Appendix C). The flexural modulus, flexural resilience, flexural toughness, and load at yielding were calculated and used to compare the results of the various beams.

5.1. Beam Breaking

5.1.1. Solid Beam

Four solid beams were subjected to the three-point bending test. Beams B1, B2, B3, and B4 were each printed with the same dimensions, in the same orientation, and had no visible defects. Because the beams were so similar, the results of the tests were expected to have minimal deviation from each other.

The first beam to be tested was B1. The setup of the beam in the three-point loading can be seen in Figure 27 with the load initially being applied. The same setup was done for each solid beam.

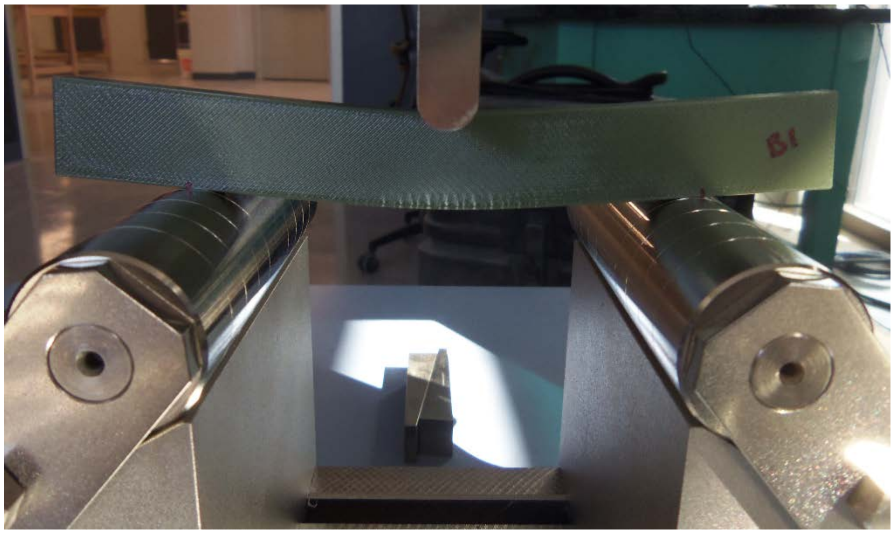


Figure 27: Three-point bending test layout of solid beam, B1

The white striations in the lower section of the beam indicated plastic deformation of the beam. The plastic deformation occurs where expected because the beam experiences the largest bending moment at the center of the beam. The lower section also experienced yielding because there is a tensile force experienced at this location. Typically, materials require less force to yield under tension than under compression. Yielding striations were also evident parallel to the applied force because of the printing pattern. As discussed in section 4.3.1, the edge of the beam was not fully filled in because of rounding in the extruded lines near the edge. The rounding causes small voids for stress concentration to occur, which will then create yielding. Each striation can be contributed to a void from the rounded printed lines and is spaced out accordingly.

After the beam reached its failure point, the solid beam broke in two pieces with the fracture in the middle of the beam at the point of highest deflection. The fracture ran the entire height of the beam alternating directions 90 degrees between the layer angles (Figure 28).

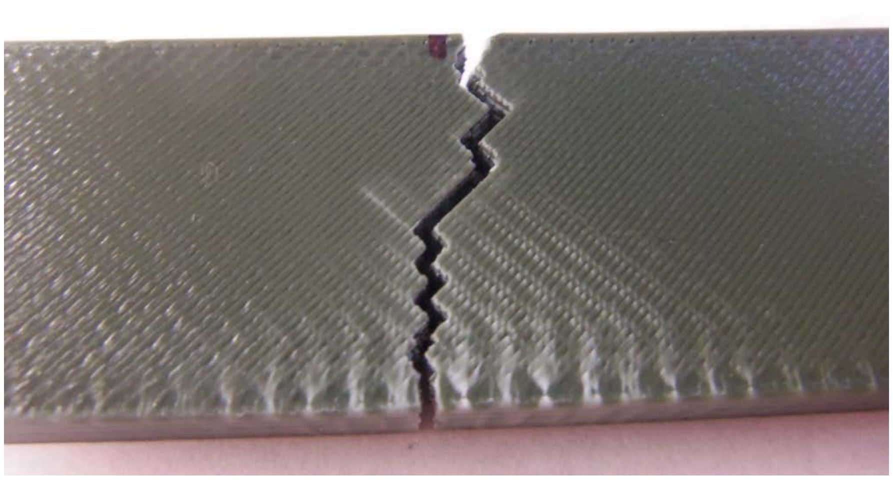


Figure 28: Final fracture geometry of solid beam, B3

The initial fracture was perpendicular to the bottom of the beam where the vertical yielding existed because of the rounded extrusion lines. After the initial crack formed, the fracture followed an oscillating pattern with sharp directional changes. The angle of the fracture can be attributed to the angle of the extrusion lines where they alter between 45 degrees and -45 degrees per layer. The yielding between extrusion layers at the -45 degree angle can be seen on the surface of the beam in Figure 28. Each angled yield striation has spacing similar to the vertical yield striations because the angled yield striations initiate from the end of the vertical yield striations. The spacing of the yield striations are then translated into the fracture. Even though the fracture grew at 45 degree angles, the fracture still traveled upwards with little horizontal deviation. Because the largest bending stress occurs at the center of the beam, the fracture will change directions between the two angles to stay within range of the higher bending stress.

Once each solid beam was broken, the data were collected and imported into MATLAB for load-deflection calculations at each time interval. The load-deflection curve of all four solid beams were plotted on a single plot (Figure 29) to see the variance in results. An average load-deflection curve was used to develop an average flexural modulus. The slope of the line tangent line of the linear region of the average load-deflection curve was used to calculate the average flexural modulus.

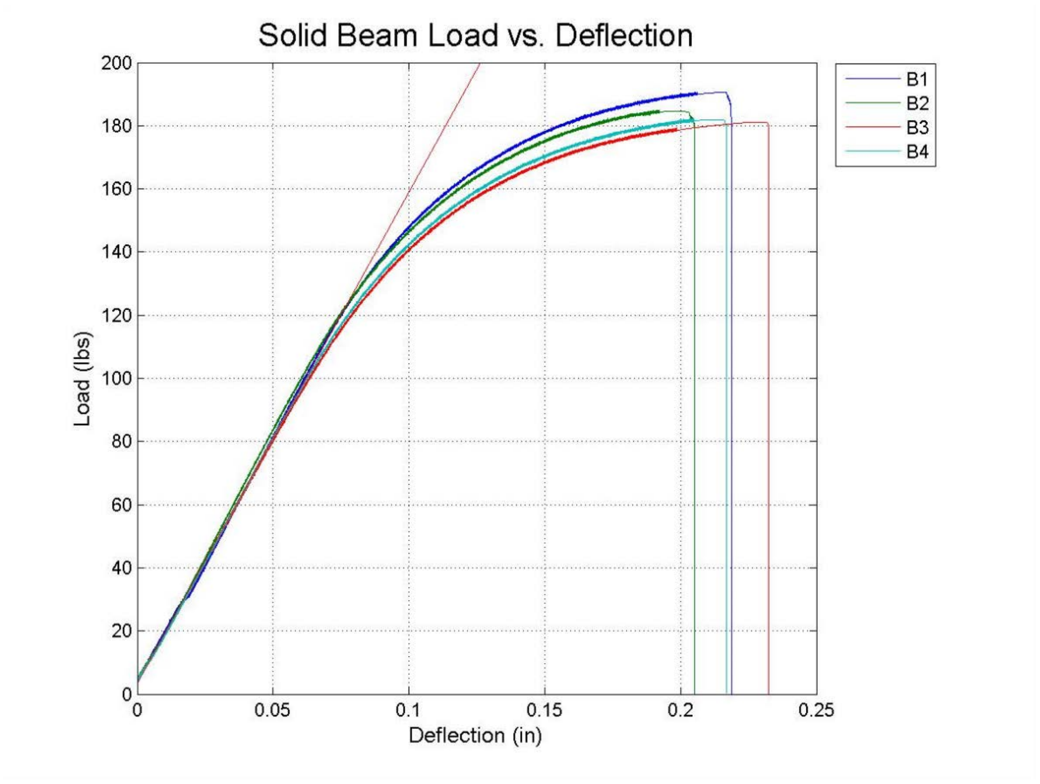


Figure 29: Load-deflection curves for the solid beams with average linear tangent line

The result of the load-deflection curve shows that there is not a large amount of variance in the data. Beam B1 had a maximum force of 190.6 lbs before breaking while beam B3 (the beam with the lowest resistance to force) had a maximum force of 181.1 lbs before breaking. The difference of 9.5lbs shows that the results are in an acceptable range. In the load-deflection curves, an expected linear region reveals the beam’s elastic deformation. The linear range for each beam is nearly identical to one another with slight divergence at a load of about 115 lbs, and this close relationship gives a more accurate measurement for flexural modulus. As the load increases past the linear range, the deflection begins to increase at a faster rate. The increase in deflection shows the initiation of plastic deformation. This deformation occurs at a load of about 115 lbs. After the beams plastically deform, the load continued to rise until failure at an average load of about 185 lbs.

5.1.2. Beam with Holes

The next beams that were tested were the beams with the lightening holes H1, H2, H3, and H4. All four beams were printed in the same orientation with small printing errors in beam H3. With this printing error, possible strength variations were expected.

Each beam was set in the three-point loading test the same as the previous beams to keep accuracy in testing. The setup and initial deformation of beam H3 is given in Figure 30.

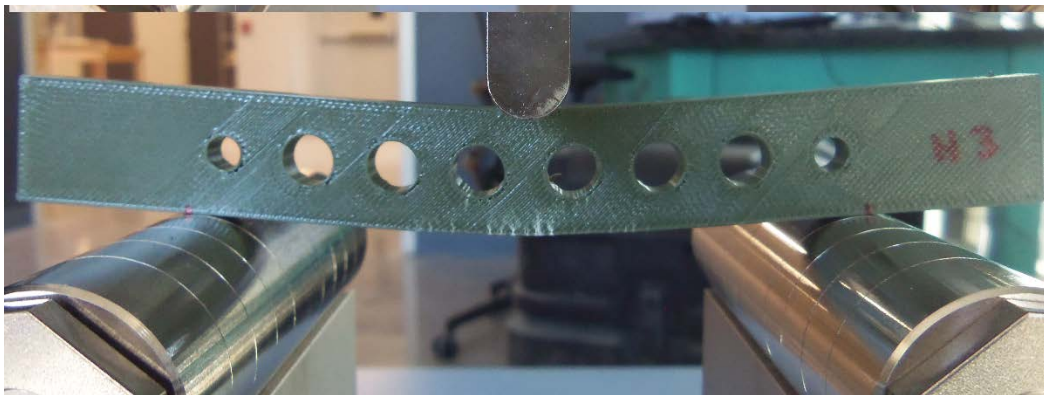


Figure 30: Beam with lightening holes in the three-point loading test

Like the solid beam, the initial deformation shows the yield striations perpendicular to the bottom of the beam. However, with the beams with lightening holes, yield striations begin to form at the base of the center holes. Yielding started to develop at the bottom of the beam and at the bottom of the center holes.

Resembling the solid beam, after enough loading was applied, the beams with holes fractured in two pieces. The fracturing in each beam occurred near the middle of the beam running through one of the center holes (Figure 31).

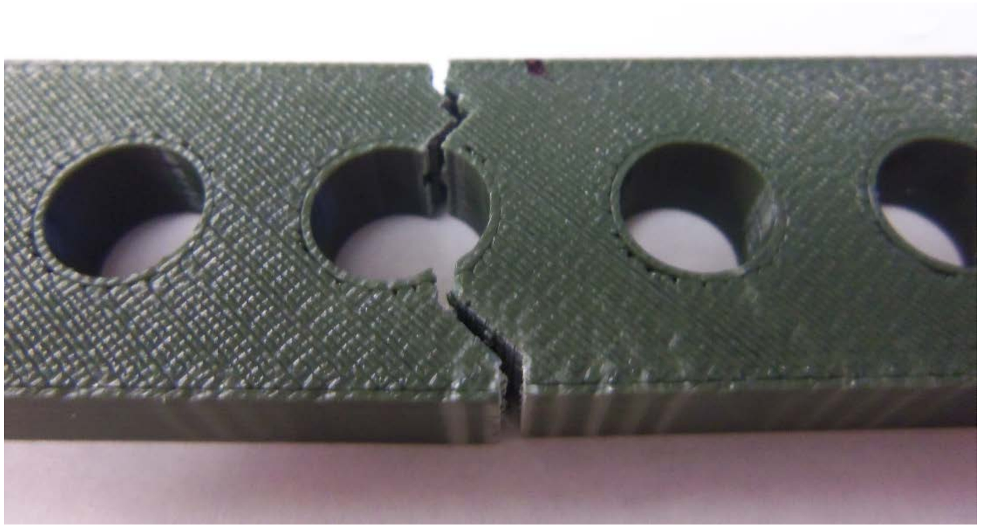


Figure 31: Fracture of beam H1

The initial fracture started at the bottom of the beam close to the center where the higher bending stress is located. The fracture grows perpendicularly to the bottom of the beam where the yield striations exist. Like the solid beam, there were rounded extrusion lines by the edge of the beam which caused small voids for high stresses to form. After the fracture traveled past the vertical yielding, the fracture traveled along the extrusion line boundary towards the hole. Instead of altering directions like the solid beam, the fracture in beam H1 moves directly towards the hole. The smaller cross-sectional area allowed for a higher stress to from under the hole rather than the full cross section in the middle of the beam. Once through the hole, the fracture traveled through the upper section of the beam opposite to how the fracture traveled through the lower section. The fracture moved between the extrusion layers again at the 45 degree angle. Once close enough to the top of the beam, the fracture traveled vertically slightly above the hole. The cross section above and below the hole provided high stress areas for the fracture to travel through.

Each beam was broken with three-point loading to obtain the necessary flexural data for the load-deflection curve. The loads and load extension were imported into MATLAB to generate the load-deflection curves for each beam. The load-deflection curve for the four solid beams is given in Figure 32.

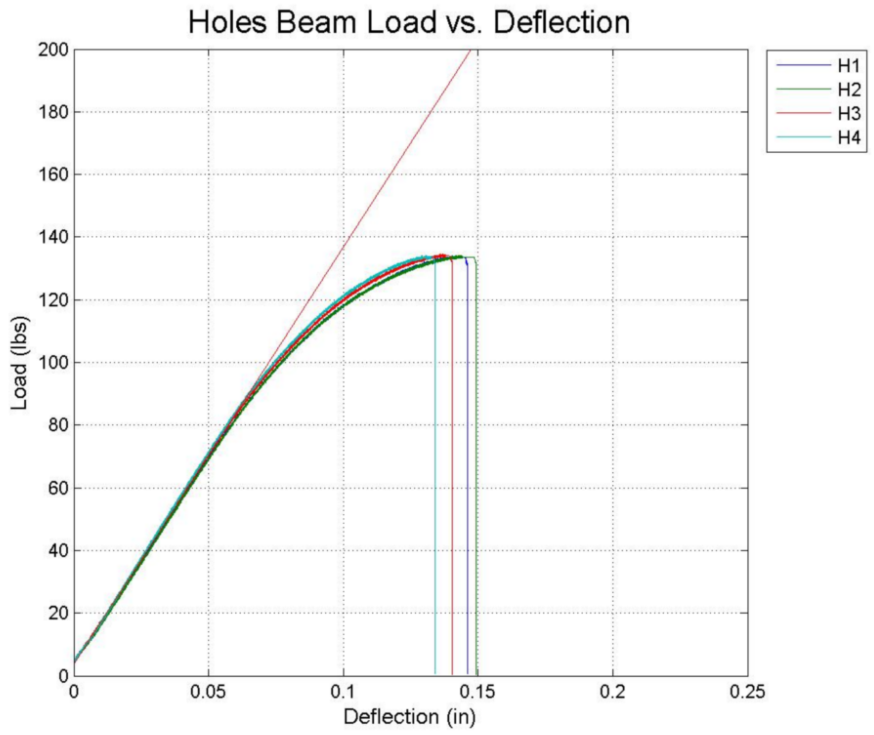


Figure 32: Load-deflection curves for beams H1, H2, H3, and H4 with average linear tangent line

The load-deflection curves for each beam in Figure 32 show a close relationship with each other. The maximum load before breaking was 134.6 lbs on beam H3, while the maximum force experienced by beam H4 was the smallest at 134lbs. The difference of 0.6 lb is small showing that the printing error in beam H3 did not affect the results. The linear range in each load-deflection curves closely match each other resulting in an accurate reading of the linear tangent line for the flexural modulus calculation. After the elastic region, the four curves begin to diverge slightly from each other. The plastic range still increases as expected with a decreasing value for the slope. Near the end of the beams plastic range, the slope seems to decrease close to zero then suddenly fractures resembling a slightly brittle fracture.

5.1.3. Beam with Slots

The four beams with slots were then tested in three-point loading. These beams were marked S1, S2, S3, and S4 for easy classifications. The slotted beams were printed in the same orientation as the previous beams for comparison.

The marks on the beams indicating support locations were lined up on the supports and the center mark indicating the location of the load was used to check for center. This setup is displayed in Figure 33.

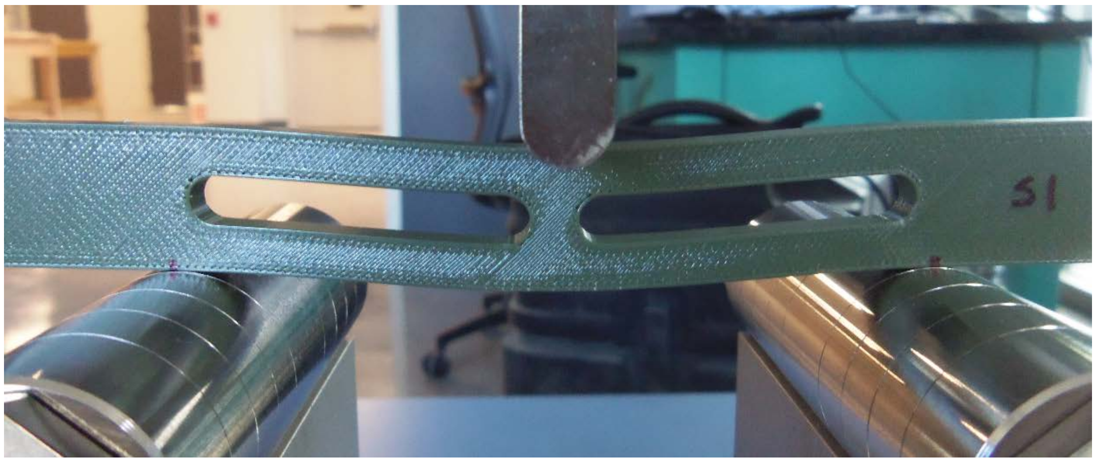


Figure 33: Slotted beam, S1, in three-point loading test

The slotted beams were able to deflect more without showing signs of plastic deformation. The yield striations in Figure 33 were just starting to appear after the beam deflected more than the solid beam. Because the support sections above and below the slot were smaller, they were able to bend like a cantilever beam. When bending like a cantilever beam, the higher stresses occur at the fixed point of the beam. The fixed point in the slotted beams are the support locations so failure at the end of the slot by the support was expected.

Unlike the other beams, the slotted beams did not fracture into two pieces. Once enough load was applied, the beam yielded enough to where there was not enough resistance of force to continue. As expected, the yielding of the slotted beam was at the end of the slot by the support

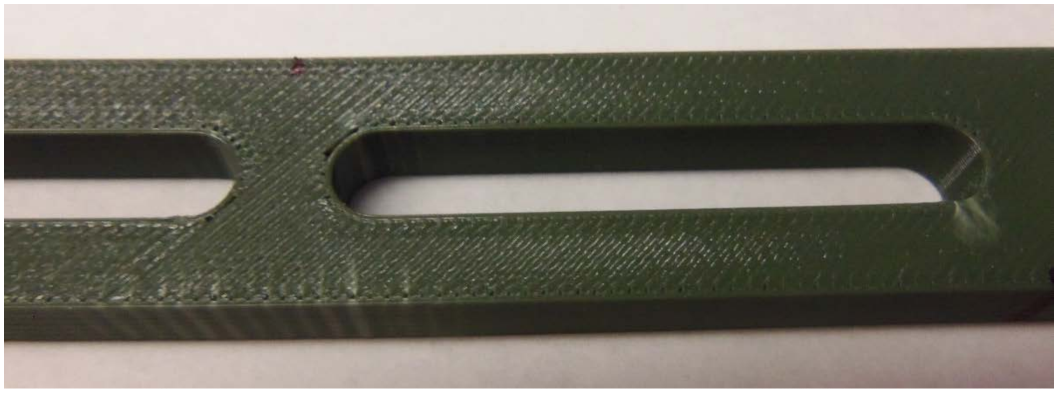


Figure 34: Yielding locations in slotted beam, S1

At the base of the beam, in the center, there are the expected yield striations. With the slotted beam, there was no crack propagation initiating at these points. Instead, the majority of yielding occurred at the end of the slot. The yielding is directed at the location of the support at an angle that represents the angles of the extrusion lines. Because the lower support section of the slot resembles a cantilever beam near the support location, the tensile forces were located at the top of the cantilever beam (bottom of the slot). The next point of higher yielding is the opposite side of the slot in the upper support section near the location of applied load. The support sections of the slot only act as a cantilever beam near the supports, so the tensile forces acted at the bottom of the support in the middle of the beam. The yielding followed the tensile forces acting in the beam at the bottom section of the slot near the support to the upper section of the slot at the point of loading.

The load and extension data from the test were collected for each beam so the load-deflection curves could be generated with a linear tangent line. Each beam, S1, S2, S3, and S4 had their respective data plotted in Figure 35.

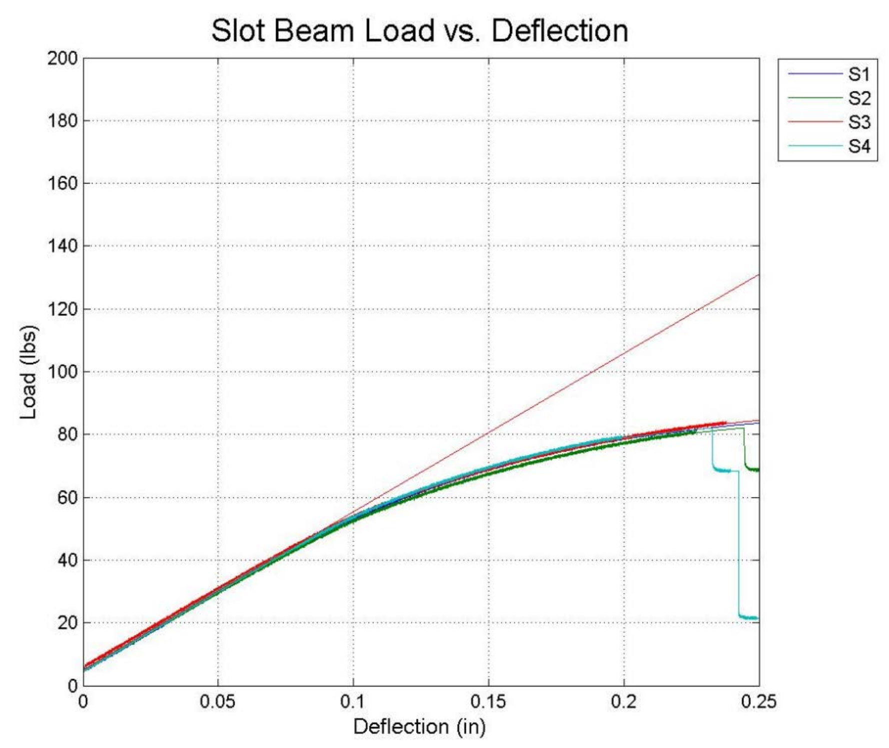


Figure 35: Load-deflection curves for beams S1, S2, S3, and S4 with average linear tangent line

The curves for the four beams are highly consistent. Beam S3 was able to resist the largest force before fracturing at 84.5 lbs and the smallest maximum force experienced by the four beams was 82 lbs by beams S2 and S4. The difference of 2.5lbs shows that the printing of each beam was accurate enough for testing. The elastic segments of the load-deflection curves for the four beams are practically identical. The matching elastic region provides an accurate average slope result for calculation of the flexural modulus. After the elastic region, the curves slightly diverge from each other. The initial failure of the beams occurred suddenly without a drop of stress indicating a slight brittle failure. After the initial failure, the beams were able to carry more load for a small amount of deflection before failing again.

5.1.4. MATLAB Penalized Beam

The MATLAB penalized beams were the first topology optimized beams to be tested in three-point loading. Beams P1, P2, P3, and P4 were printed in the same orientation without any major printing errors. The beams were expected to perform similarly to each other.

The marks near the ends of the penalized beam indicate the support locations for the three-point loading test. These marks helped to line up the beam so that the load would be applied to the top center of the beam as designed. The setup for beam P1 is displayed in Figure 36.

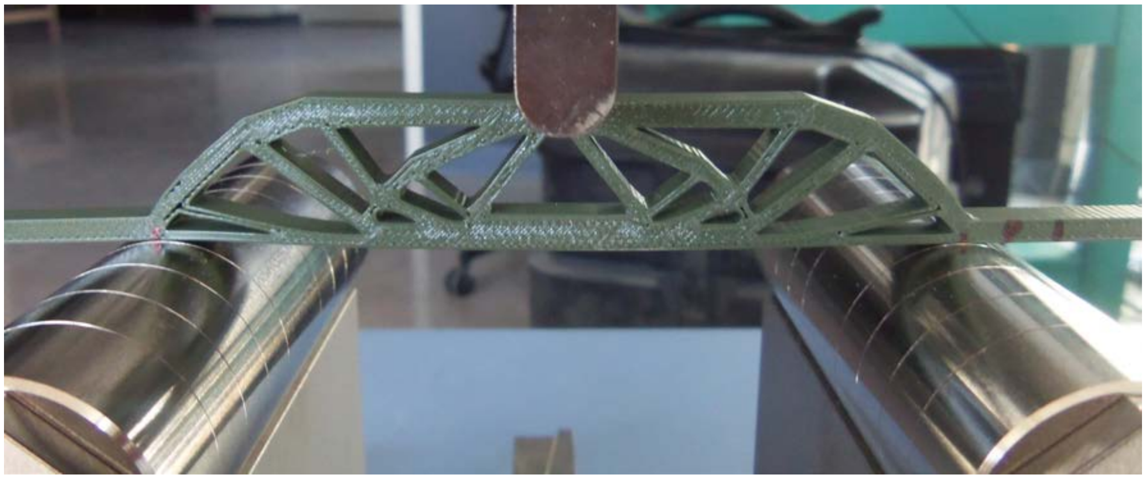


Figure 36: MATLAB penalized beam, P1 in three-point loading

The initial bending of beam P1 showed fewer yield striations in in the bottom of the beam. Instead, separation of the extrusion lines began to occur in the left side of the beam in Figure 36. When the beam bent enough, some extrusion lines failed where the initial separation occurred. The failure of the extrusion lines took place in areas where the extrusion lines could not completely bond with neighboring layers due to printing angle. As the beam continued to bend, vertical yielding at the bottom of the beam began to develop.

The beam failed when bent to its ultimate point, beyond the plastic yield point. The fracture occurred near the middle of the beam where the yield striations occurred. The fracture was expected at the point of extrusion line separation on the left side of the beam, instead, there was a momentary loss of load resistance until the beam was able to support the load again. The fracture in Figure 37 is initiated off center of the beam at the yield striations.

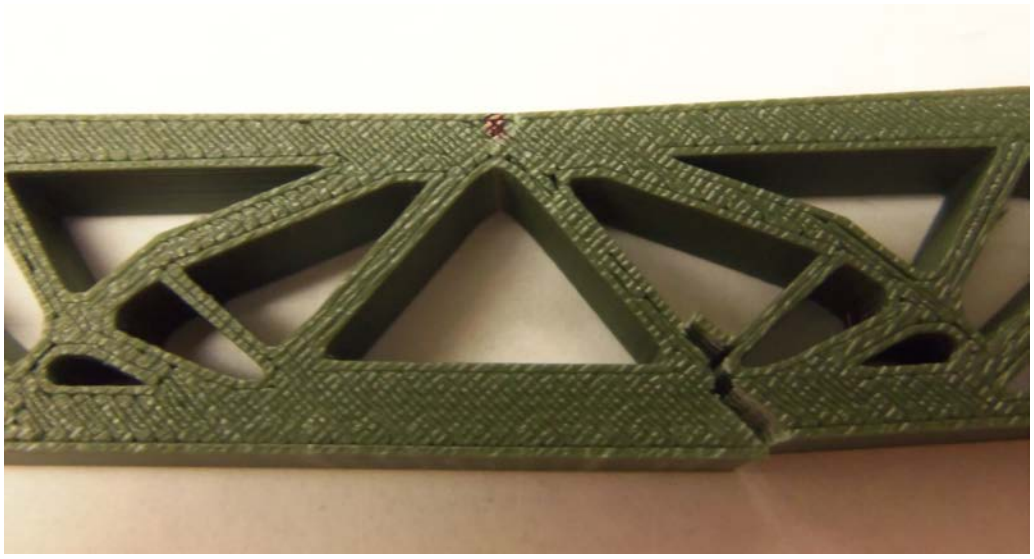


Figure 37: MATLAB penalized beam, P1 fracture location

Initially, the fracture started from the bottom of the beam offset from the center where the highest bending stress occurs in a solid beam. The fracture propagated along the extrusion line until reaching an intersection of void spaces. The thin members between the void spaces created an area within the beam where improper bonding of extrusion lines developed. After reaching this point, the fracture traveled through the imperfections to the boundary of a void space. The boundary layer was fractured, leaving only the top of the beam to support the load. Separation of extrusion lines developed at the top of the beam at which point the test ended because of the beams inability to resist loading.

The load and deflection data were collected from the test to develop the load-deflection curve for the MATLAB penalized beams. The data for each penalized beam were plotted in Figure 38.

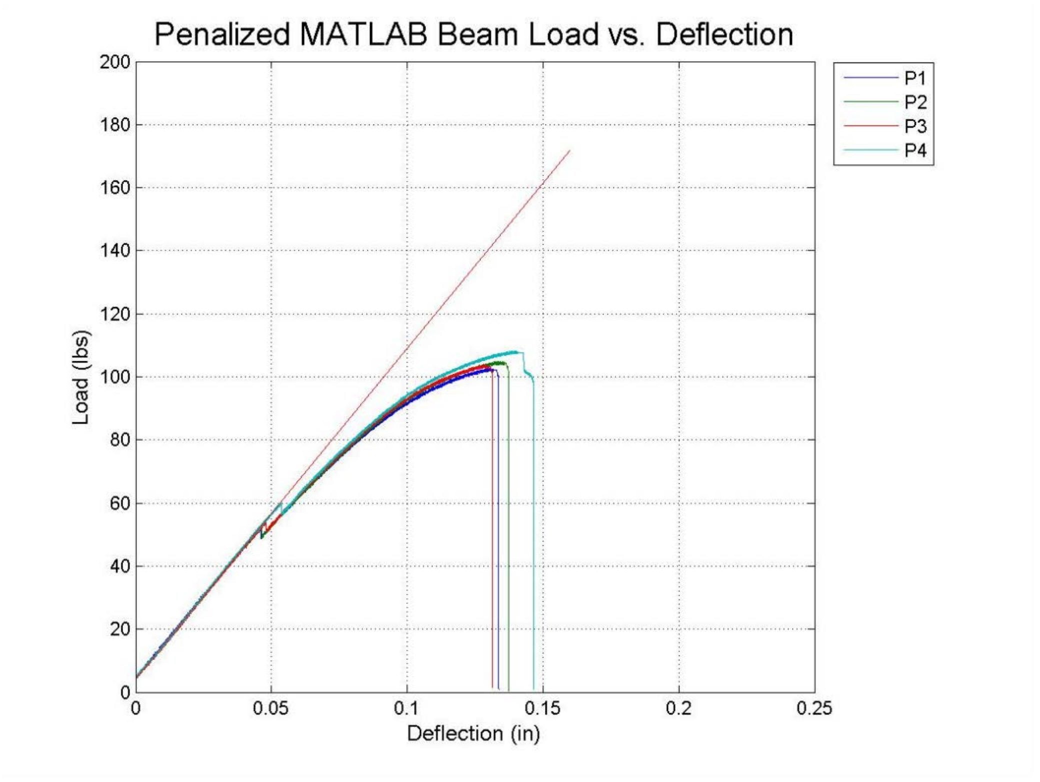


Figure 38: Load-deflection curves for MALAB penalized beams with average linear tangent line

The elastic range of each beam, from a deflection of 0 to 0.05 inch in Figure 38, were nearly identical giving an accurate representation for the linear tangent line to calculate the flexural modulus. In the plastic region, the maximum load applied was to beam P4 at 108.3 lbs and the smallest maximum load applied was to beam P1 at 102.7 lbs. The difference of 5.6 lbs is closely related considering the internal voids created in the extrusion process. As the load was increased in the beam, the separation of extrusion lines on the side of the beam created a load drop at an average deflection of 0.045 in. After the load drop, the load and deflection continued to rise linearly in the elastic range before plastically deforming at 65 lbs. In the plastic region, the four beam curves began to slightly diverge from each other. At the end of the plastic range, before failure, there is a slight area of slope reduction. The slope reduction indicates more yielding is occurring with less load resistance. Each beam then failed suddenly with indicating a partially brittle failure.

5.1.5. MATLAB Non-Penalized Beam

The next optimized beam to be tested was the MATLAB non-penalized beam. Beams, NP1, NP2, NP3, and NP4 were printed in the same orientation as the previously printed beams. Because the variable thickness material was centered in the beam, support material was added during the printing process to prevent printed material from falling through the middle. The support material produced a variation in surface finish between the two faces of the beam that may have produced a variation in results.

The setup for three-point loading of beam NP1 is displayed in Figure 39. The support marks were used to align the beam to the center of the load.

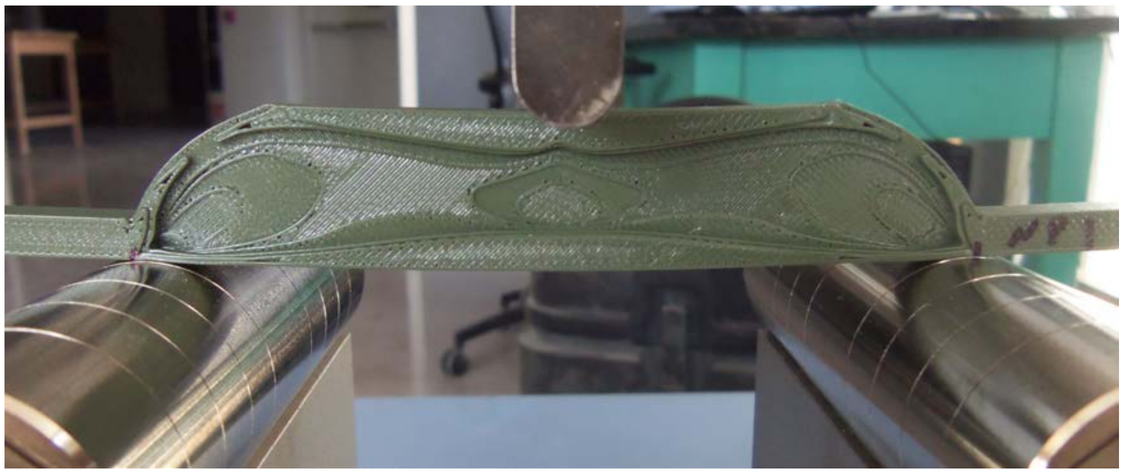


Figure 39: MATLAB non-penalized beam, NP1 setup in three point loading

The initial bending of the beam created yielding striations along the bottom center of the beam. Lesser amounts of yielding then started to develop near the ends of the beam by the supports where the thickness of the beam was the smallest. After more load extension, yielding started to develop in the center of the beam in the thinnest center section. With the majority of yielding at the center of the beam, the failure was expected to happen at the point of loading in the center.

For the non-penalized MATLAB beams NP1, NP2, and NP4, the failure of the beam developed in the middle under the applied load (Figure 40). The failure started at a yield striation at the bottom of the beam where the voids in the printing existed.

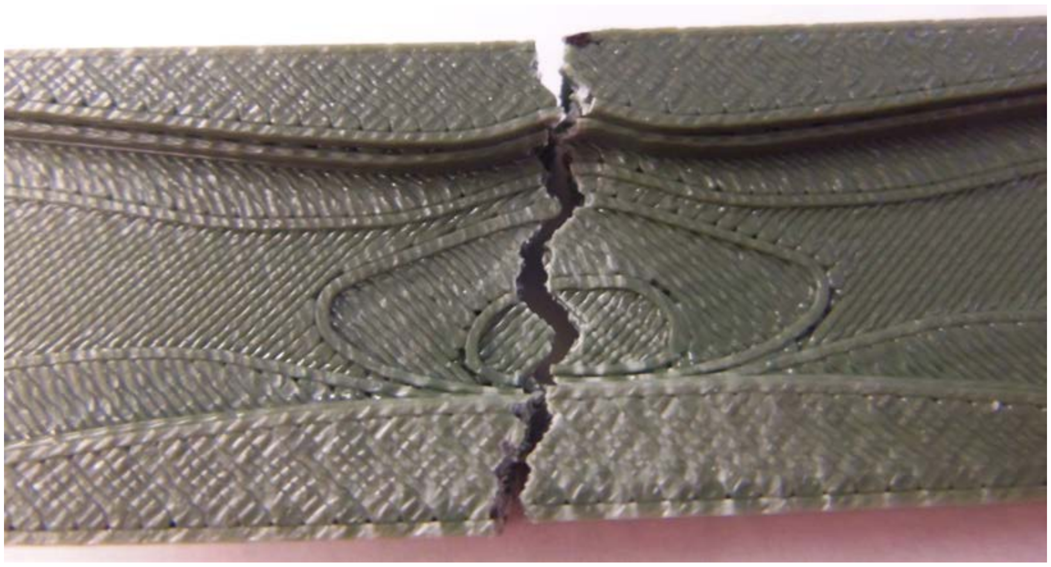


Figure 40: MATLAB non-penalized beam, NP1 fracture location

The fracture grew towards the top of the beam through the thickest part of the beams lower section. Through the lower section, the fracture did not follow the angle of the extrusion lines. The yielding striations provided a more vertical path for fracture propagation through the extrusion lines. Once into the thinner section, the fracture began to follow the angle of the extrusion lines. The fracture altered directions by 90 degrees while continuing up the height of the beam. In some areas where a change in thickness occurs. The fracture traveled vertically towards the top of the beam instead of following the extrusion line angles. This fracture propagation behavior shows proper bonding between extrusion lines and layers. When the fracture grew to the thickest part of the upper section of the beam, the fracture propagated directly upwards, centered under the load.

The third non-penalized MATLAB beam (NP3) experienced failure in a different location than the other three beams. The failure of NP3 developed close to the side of the beam where the printing error occurred (Figure 41).

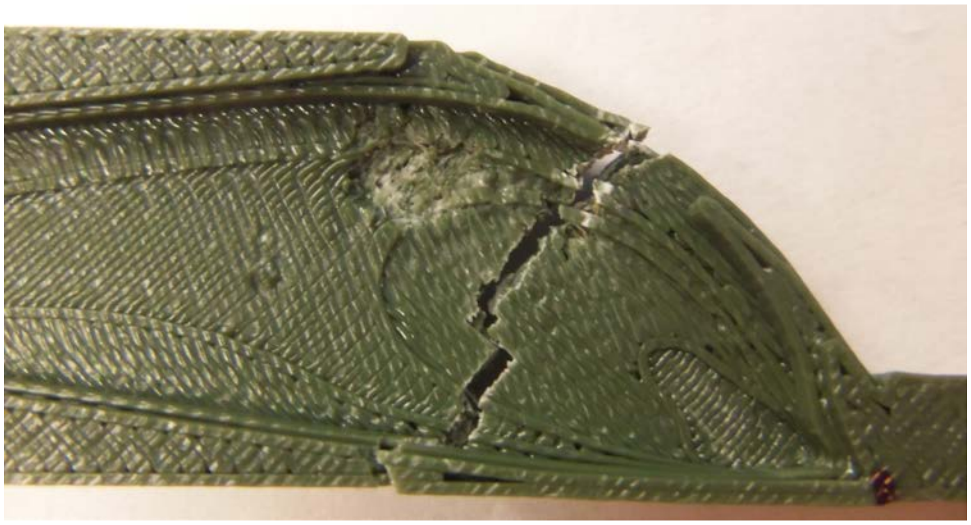


Figure 41: MATLAB non-penalized beam, NP3 fracture location

Instead of a central fracture initiation point, the fracture in NP3 started closer to the support. The fracture grew along the angles of the extrusion lines with minimal alterations in angle directions. After growing through the thin layers of the beam, the fracture developed along the printing error boundary.

Once all four beams non-penalized MATLAB beams were broken, the load-deflection curves were generated from the load and extrusion data. The load-deflection plots for each beam were then plotted in Figure 42.

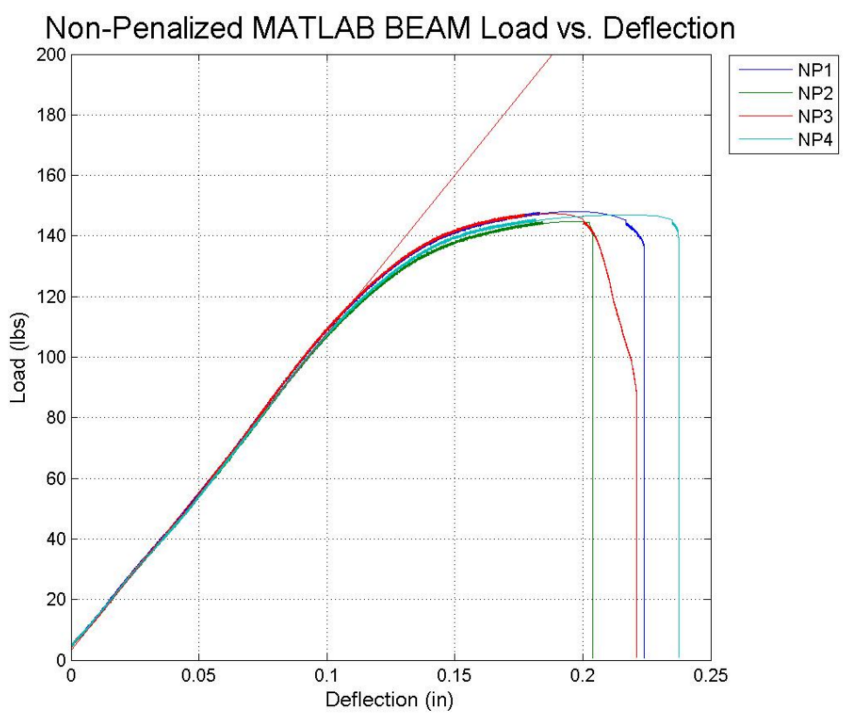


Figure 42: Load-deflection curve for the MATLAB non-penalized beams with average linear tangent line

The elastic region of the non-penalized MATLAB beams all matched each other in slope giving an accurate slope measurement for flexural modulus calculation. The maximum load experienced by a non-penalized beam was 148.1 lbs by beam NP1, while the smallest maximum force experienced by a beam was 144.8 lbs by beam NP2. The difference in maximum force was 3.3 lbs which shows a close relationship in data. The printing error in NP3 did not make a large impression in the data because it fell within the range of the two beam extremes. After the load passes the elastic range, the curves for each beam begin to diverge. The slope in the plastic range levels out, then gradually falls before the final fracture. The decreasing slope indicates a more ductile failure with the beams increased extension with decreased load resistance.

5.1.6. ParetoCloud Beam

ParetoCloud optimized beams PW1, PW2, PW3, and PW4 were the final optimized beams tested. The four beams were printed in the same orientation with the only major printing error existing on the top of the first beam, PW1 (Figure 23Figure 25). Because of the printing inclusions, PW1 was expected to have a variance in result compared to the other ParetoCloud beam.

Consistent with the previous three point loading test procedures, the support marks on the ParetoCloud beams were aligned with the supports and the load applicator was centered in the middle of the beam. The load was initially applied with yielding establishing in the lower section (Figure 43).

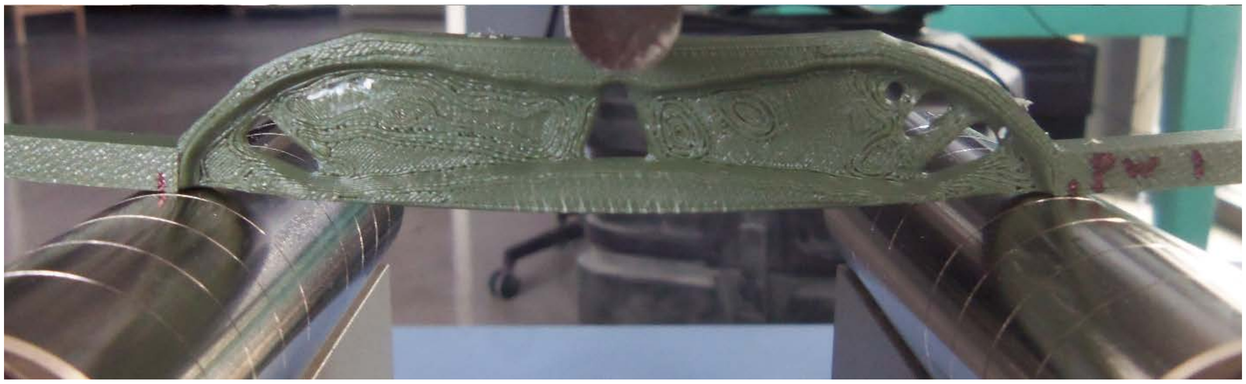


Figure 43: ParetoCloud beam, PW1 in three point loading test

As the beam was bent, the yielding began to form and grow from the bottom of the beam in the center, upwards parallel to the applied load. The yielding striations were evenly spaced because of the voids left from the curved extrusion lines by the beam's boundary.

From the yielding locations, a fracture formed that caused the ultimate failure. The fracture location and propagation path are displayed in Figure 44.

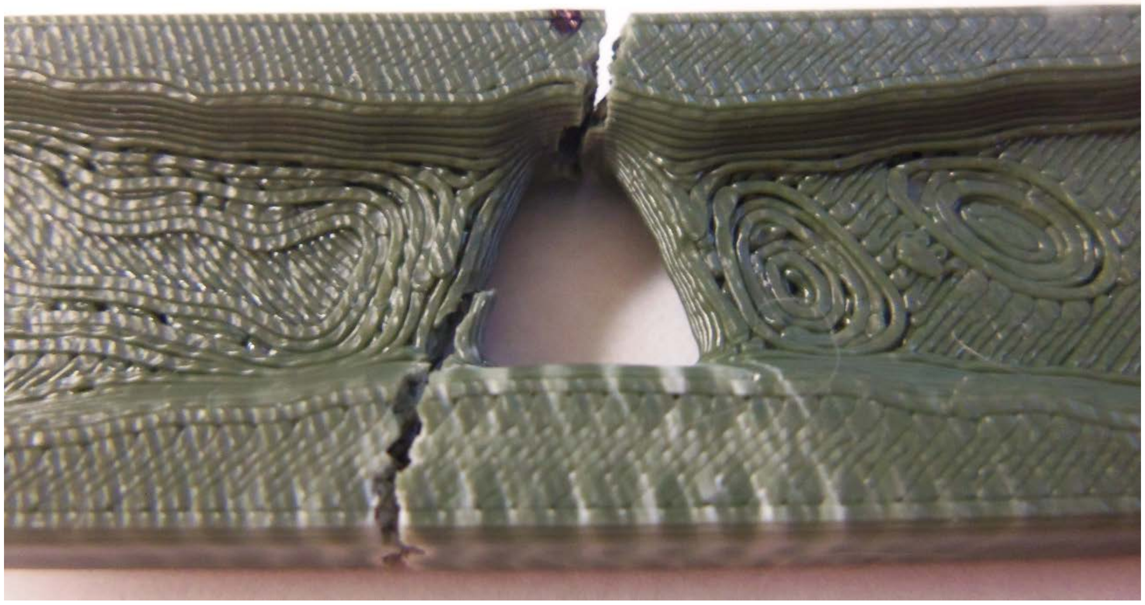


Figure 44: ParetoCloud beam, fracture location and propagation path

The yield striations in the bottom section of the beam extend upwards towards the top of the beam and downwards towards the bottom of the beam. The middle section of the beam has a void which causes the thicker bottom section to have a boundary parallel to the bottom and thickness of the beam. This boundary section produces more voids from the curving of extrusion lines at the boundary. Because the extrusion lines are not vertical, the void inclusions by the boundaries are not vertically adjacent to each other. The fracture will then propagate from the farthermost tension fiber at the bottom of the beam upwards at a slight angle to the next void inclusion. The fracture does not follow the extrusion line angles because the voids by the boundaries are close enough together to provide a lower stress fracture path. The fracture continues to propagate towards the edge of the designed void where the cross-sectional area is smallest. After breaking through the boundary layer, the fracture propagates through the top of the beam in the middle at the highest stress location.

The load and extension data were collected to generate the load-deflection curves for each ParetoCloud beam with the linear tangent line. The load-deflection curve for the four beams were plotted on the same graph to facilitate comparison (Figure 45).

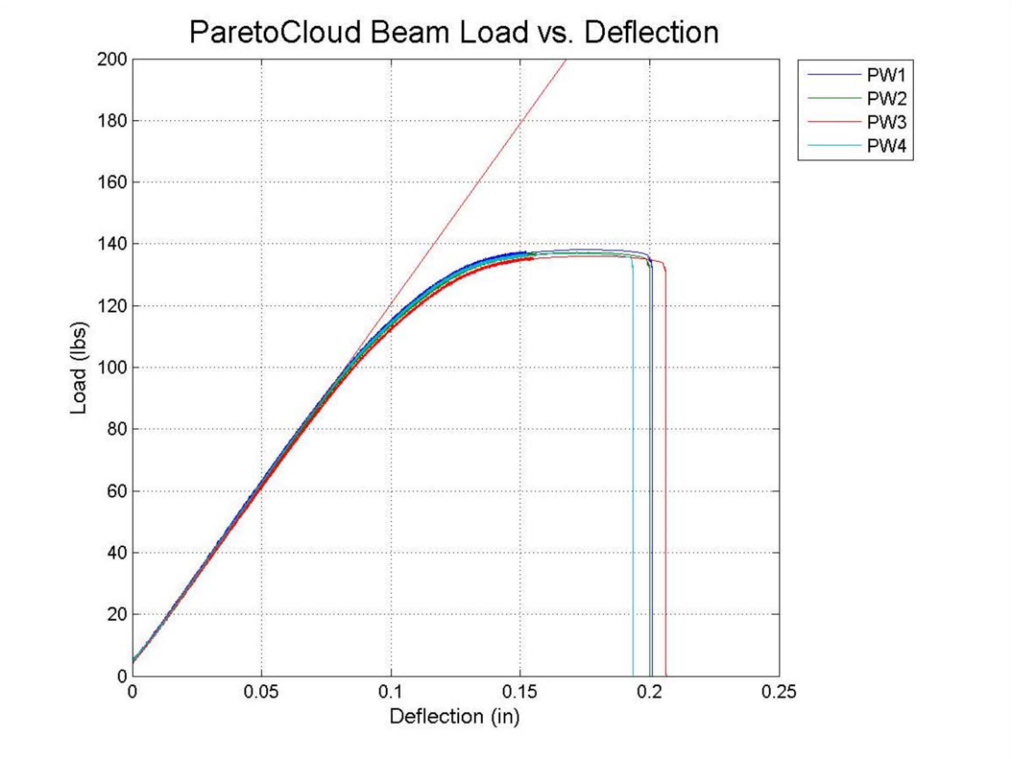


Figure 45: Four ParetoCloud beam load-deflection curves with linear tangent line

In the linear range of the ParetoCloud load-deflection plot, each beam's curve was practically identical, exhibiting comparable results. The consistency of the four curves in the elastic region gave a closer linear line for the calculation of the flexural modulus. After the linear range, the four curves slightly diverge in the plastic deformation range. In the plastic deformation range, the slope of each beam's curve lessens to zero, then becomes negative before failure. The decreasing slope shows greater ductile deformation before failure

5.2. Comparison

To compare properties of the assorted optimized styles of beams, numerical data were calculated from the load-deflection curves for each beam. Values for flexural modulus, flexural resilience, flexural toughness, and maximum average load before yielding were determined.

One of the most important properties for the optimized beams is the flexural modulus. The objective function of the optimization process was to minimize compliance or maximize the stiffness of the component. The flexural modulus provides a measure of stiffness. A beam that has a higher flexural modulus also has higher stiffness because the beam will be able to withstand more load with less deflection before plastically yielding. The flexural modulus for each optimization method was calculated using the slope of the averaged linear tangent line to the linear region of each beam's load-deflection curve. In MATLAB, a linear line was fit to the linear region of the average load-deflection curves for determining the slope. The flexural modulus was then calculated using Equation (16).

$$E_B = \frac{L^3 m}{4bd^3} \tag{16}$$

where L is the span, m is the slope of the tangent line to the linear region of the load-deflection curve, b is the beam thickness, and d is the beam height.

The flexural resilience of the beams quantifies the beams ability to absorb energy while experiencing elastic deformation then release the energy after returning to the beams original state. A higher flexural resilience will result in a beam that can absorb more energy before plastically deforming. The flexural resilience is an important calculation because it will determine whether the beam will be able to withstand the necessary load in the design without plastic deformation. The flexural resilience of each optimization method was calculated by taking the integral of the load-deflection curve in the linear slope region. This integration was done using trapezoidal numerical integration (trapz) in MATLAB from the start of the load-deflection curve to the end of the linear region.

The flexural toughness is the beams ability to absorb energy without fracturing. A beam with a higher flexural toughness will exhibit a larger absorption of energy before failing. Flexural toughness of each optimization method was calculated by integrating the load-deflection curve from start to failure of the beam. The integration was done in MATLAB using trapz.

The loading at the point of yielding was found to determine the maximum average load that the beams could resist before yielding. A higher load shows a beams greater ability to resist plastic deformation. The load was determined from the load-deflection curve where the linear region ended.

The results of each calculation are given in Table V. The solid beam is used to compare the optimized beams to the original design before optimization.

Table V: Calculations from average load-deflection plots for the six tested beams

	Volume (in^3)	**Flexural Modulus (psi)**	**Flexural Resilience (in*lb)**	**Flexural Toughness (in*lb)**	**Load at Yielding (lb)**
Solid Beam	0.750	236214	4.33	25.89	115
Beam with Holes	0.632	201835	2.77	11.08	85
Beam with Slots	0.540	76432	2.58	13.66	50
MATLAB Penalized Beam	0.365	158722	2.30	8.36	65
MATLAB Non-Penalized Beam	0.376	158519	5.77	19.69	110
ParetoCloud Beam	0.401	177370	4.43	18.49	100

With the volume constraint applied to the optimization process, the volume of the optimized beams should have been half of the solid beam’s volume. With the solid beam’s volume being 0.75 in^3, the half volume target for the optimized beams was 0.375 in^3. The beam that came the closest to half of the solid beams volume was the non-penalized MATLAB beam. The highest flexural modulus calculated for the beams was from the beam with lightening holes. The MATLAB non-penalized beam then had the highest flexural resilience, flexural toughness, and load at yielding.

The high flexural modulus in the beam with holes may be attributed to its larger volume. With a volume close to 1.7 times larger than the 50% volume constraint, the beam with holes would not be a viable option for a 50% volume removal requirement. The ParetoCloud beam had the highest flexural modulus when compared to the optimized beams with a volume reduction closer to the desired 50%. The two versions of the MATLAB beams had a calculated flexural modulus that was approximately 18.6 ksi lower than the ParetoCloud beam. Though the ParetoCloud had a higher flexural modulus which resulted in a higher stiffness, the two MATLAB beams were close in comparison.

The average load-deflection plots for each beam were employed in the calculations that produced the data in Table V. A comparison of plots and visual of how the calculations relate to the plots and respective beams is given in Figure 46.

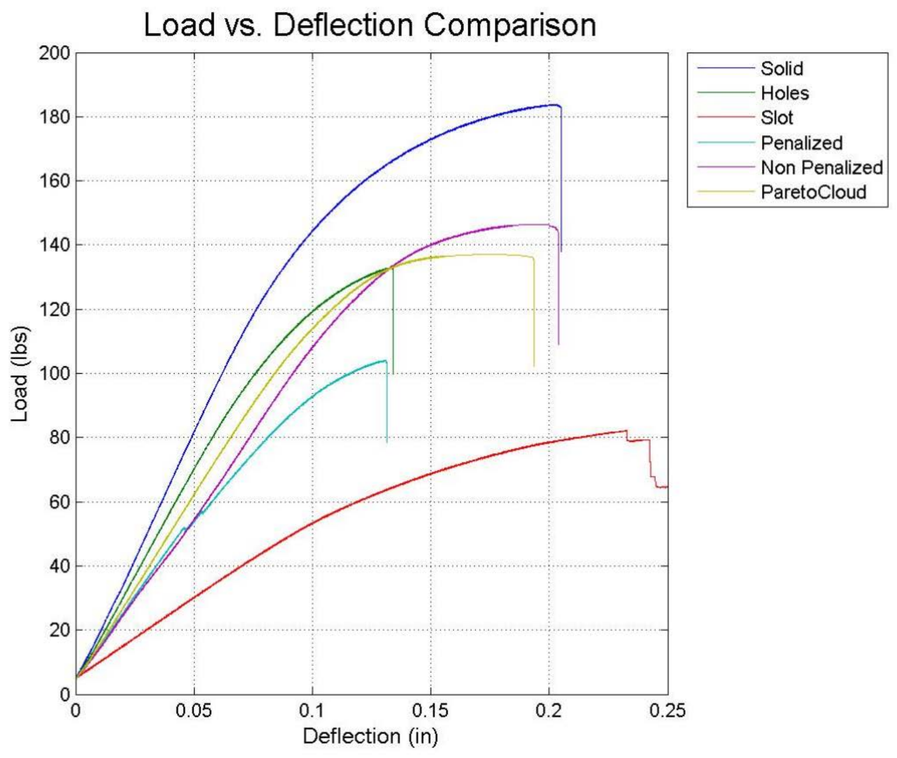


Figure 46: Average load-deflection plot comparison for the six tested beams

The curve for the beam with holes had a steeper slope than the other non-solid beams, resulting in the highest flexural modulus. The ParetoCloud beam had the next steepest slope resulting in the second highest flexural modulus with the two MATLAB beams having the next highest flexural modulus. The flexural resilience of the non-penalized MATLAB beam was the largest because of the length and slope of the elastic range. Even with the non-penalized beam yielding at close to the same as the solid beam, the flexural resilience of the non-penalized beam was larger than that of the solid beam because of the non-penalized beams lower stiffness. The non-penalized beam also resisted the largest load with the same deflection at failure as the solid beam.

5.3. Discussion of results

The two optimized beams that gave the better results were the non-penalized MATLAB beam and the ParetoCloud beam. The ParetoCloud beam had a volume of 0.401 in^3, which was 0.026 in^3 larger than the desired 0.375 in^3 volume. The non-penalized beam was 0.001 in^3 larger than the desired volume. The non-penalized beam had a more accurate volume reduction than the ParetoCloud beam. The flexural modulus calculated for the ParetoCloud beam was larger than the non-penalized beam which results in a higher stiffness. This larger stiffness may result from the extra material in the ParetoCloud beam. The non-penalized beam followed the constraints more accurately than the other beams. The non-penalized beams and the ParetoCloud beams also had very similar designs which may have contributed to their similar performances. The middle sections along the neutral axis were designed to have the thinnest cross-sectional area while the top and bottom of the beams were designed with thicker sections. The cross-sectional areas resembled an I-beam (Figure 47).

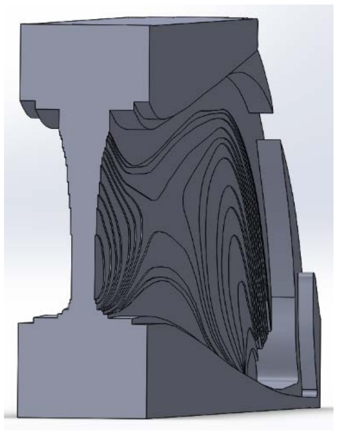
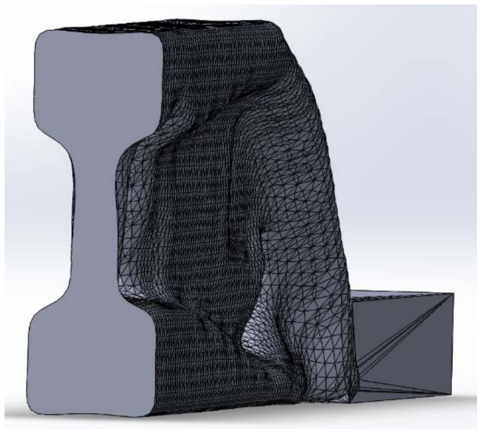
a)  b) 

Figure 47: Cross sections of a) MATLAB non-penalized beam and b) ParetoCloud beam

The thinner section along the neutral axis is a relation to the maximum bending stress of Equation (13). Because the further from the neutral axis material is located, the higher a stress will be experienced. The beams with holes and slots had the right design with minimal material along the neutral axis, however, there was still a need for material to resist loading. By thinning the material near the center of the beam and keeping the full thickness at the top and bottom of the beam (where the greater stresses are located), the non-penalized and ParetoCloud beams were able to resist the loading in a more efficient way. The non-penalized beam was able to resist a higher load because there were no designed holes in its cross section that would decrease the allowable load. A smaller cross section from a designed hole will produce higher stresses because in Equation (13), a smaller cross section creates a smaller second moment of inertia, which then produces a higher stress in the part. The stresses then can reach the failure point of the material faster than if the cross section was larger. In locations that experience higher stresses, there should be an increase in material placement to increase the cross-sectional area. The top, bottom, and middle of the beam experience the highest stresses in three-point loading, so more material should be added in these areas. The non-penalized beams and ParetoCloud beams had a decreasing thickness in this area, and for this decrease, the beams failed in the middle sections.

The penalized MATLAB beam experienced failure at a lower force than the other optimized beams because of the small sections designed from the optimization. A smaller element size would create a beam that would have larger supporting sections that may have been better able to resist the loading. With a smaller element size, the accuracy of the optimization would have lowered, as discussed in section 2.4.4. The lower accuracy in calculations may have then lowered the beams stiffness, which may have then given the same results as the higher element size that was tested.

The beams with holes and slots, while a quick and easy method of reducing weight, did not perform as well as the optimized beams. The beam with holes had the lowest compliance, but also had the most material kept in the final design. If the reduction of material is essential to the final design of a component, then the beam with holes cannot be efficiently used. The slotted beam had the largest deflection, which in the minimal compliance problem, is not ideal.

Of the five reduced weight beams, the beam with holes, ParetoCloud, and the non-penalized MATLAB beam performed the best. The beam with holes could only supply a higher stiffness if weight reduction is not essential to the component design. More mass removed with larger holes would reduce the effectiveness of the beam because the reduced cross-sectional area would increase the stresses. The ParetoCloud beam produced a slightly larger flexural modulus than the non-penalized beam, but was slightly outperformed in flexural resilience, plastic energy, flexural toughness, and fracture energy by the non-penalized beam. These results show that the MATLAB non-penalization method is a comparable method to the higher performing optimized beams. If material reduction is important, the non-penalization method or the ParetoCloud method are the better choices.

Various printers possess different printing characteristics that have an effect on the final printed component. The fused filament fabrication process used for the simply supported beams created small voids within the beams that caused stress concentration points. Because of the printing limitations, the method for printing must be considered in the design. For the MATLAB penalized beam, the thinly designed links showed improper bonding between the two boundary extrusion lines. To avoid this, the design of the beam may constrain the maximum distance between small links to twice the filament diameter. The maximum distance would only be applied if the filler filament could not extrude between the two boundary extrusion lines. By keeping the small links within a certain range, proper bonding of the two boundary lines would be assured. To achieve the desired link size, the mesh size can be altered. Smaller element sizes will create a design with smaller links, while larger element sizes will create larger link sizes. Because there were areas in the MATLAB penalized beams that had voids caused by difference between the extrusion line angles and the link angles, a design constraint might be applied. To avoid the voids from the extrusion line angles, the link angles and the extrusion line angles should match. If the printing parameters that make the extrusion line angles cannot be changed, then the design of the component must be. A constraint that forces the designed links to match the angle of the extrusion lines would help to eliminate the voids within the component. However, with forcing the design to match certain angles, the compliance of the beams may not minimize to the same value as before the angle design constraint was applied. The additive manufacturing tolerance can also be considered when applying additional design constraints. The MATLAB non-penalized beam, for example, had a thickness step size of 0.005 inch at its smallest. For the uPrint used to print the beams, the tolerance was 0.01 inch, double that of the thickness step size. Because of the printer’s inability to print to the required accuracy of the non-penalized beam, thickness layers were combined so that the printer could print the beams. When the beam is being optimized, the accuracy of the printer must be considered as a design constraint so that the designed component can closely match the printed component. Though the optimized design may be ideal for computational modeling, the manufacturing of the component creates flaws that should be designed for.

6. Conclusions

The focus of this thesis research was to determine how topology optimization using the SIMP method functioned and what changes could be made to improve the optimization process. The main optimization method explored was the 99 line MATLAB code written by O. Sigmund (2001). With Sigmund’s code, there were multiple variables that have an impact on the outcome of the optimization. The penalization factor used to penalize the intermediate densities can change the amount of intermediate density in the final design. A large penalization factor creates clear solid and void boundaries, but also increases the compliance. A low penalization factor will decrease the compliance, but there will be more instances of intermediate density, which a printer cannot print. Because a lower penalization factor was discovered to decrease the compliance the most, a method for using a penalization factor of one was developed. The variable thickness method uses the density values in the intermediate density to determine an equal strength thickness value at full density. The variable thickness method would theoretically produce a component with a higher stiffness than the penalized beams. Another optimization method that combines the design of penalization and non-penalization was explored. The ParetoCloud beam results in a beam that had solid and void regions along with a variable thickness. To compare to classic means of volume reduction, beams with holes and slots were explored.

Not only do the beams need to be designed for the objective function, but also for the additive manufacturing process. The design of the beams were ideal in perfect printing conditions, however, there were some limitations in the printing process that had an effect on the results. The printing angles created voids in the MATLAB penalized beam while certain links in the beam were not sized properly for boundary layer bonding or filling between the boundary lines. The tolerance of the printer also created the printed beam to vary from the designed beam with the combination of layers in the MATLAB non-penalized beam. To design a beam that will be printed, the limitations of the printing process should be considered. A variation in mesh size could alter the link sizes so that the boundary layers can be properly sized for proper bonding and filling. A design constraint could be applied to the optimizer to match the link angles to that of the printer's extrusion line angles. Another design constraint could be applied to make the minimal thickness variation the same as the printer's tolerance. The optimization of the beams is ideal for computer modeling, however, design constraints that relate to the limitations of the printer should be applied for printing.

Three-point bending tests were performed to determine each beam's flexural modulus, flexural resilience, plastic energy, flexural toughness, and failure energy. Each beam was compared to a solid beam that fills the entire design domain of the beams. The beam with the lightening holes had the closest flexural modulus to the solid beam, but the method of design was unable to remove the required 50% of volume. The main constraint for the optimization of the beams was to remove 50% of the beams initial material. The beam with lightening holes was unable to fulfil the constraint which did not making it a viable option for optimization. The ParetoCloud and MATLAB non-penalized beams were able to produce the highest stress resistance with the closest to the 50% volume reduction. The optimization process for the ParetoCloud beam was not fully able to get to the 50% volume reduction at 0.401 in^3 while the non-penalized beam was the closest at 0.376 in^3. With the volume reduction in mind, the MATLAB beams both met the standards that were required of the optimization while the others were too large. Even with less material than the others, the non-penalized beam was able to perform at a comparable level as the other beams. This high performance proves that the variable thickness non-penalized beam is a viable option for topology optimization.

7. References Cited

Ambrosio, L., & Buttazzo, G. (1993). An optimal design problem with perimeter penalization. *Calculus of Variations and Partial Differential Equations, 1*(1), 55-69.

Bendsoe, M. P. (1998). Generating optimal topologies in structural design using a homogenization method. *Computer Methods in Applied Mechanics and Engineering, 71*(2), 197-224.

Bendsoe, M. P., & Kikuchi, N. (1988). Generating optimal topologies in structural design using a homogenization method. *Computer Methods in Applied Mechanics and Engineering, 71*(2), 197-224.

Bendsoe, M. P., & Sigmund, O. (2003). *Topology Optimization, theory, methods and applications.* Berlin: Springer.

Broxterman, S. (2017). *Using Topology Optimization for Actuator Placement within Motioin Systems.* Master's Thesis, Delft University of Technology, Mechanical, Maritime and Materials Engineering, Delft.

Choi, K. K., & Kim, N. H. (2005). *Structural Sensitivity Analysis and Optimization 1: Linear Systems.* New York, NY: Springer Science+Business Media, Inc.

Christensen, P. W., & Klarbring, A. (2010). *An introduction to structural optimization.* Berlin: Springer Netherland.

Dadalau, A., Hafla, A., & Verl, A. (2009). A new adaptive penalization scheme for topology optimization. *Production Engineering, 3*(4-5), 427-434.

Demers, T. M. (2009). *A designer's approach for optimizing an end-loaded cantilever beam while achieving structural requirements.* Mechanical Engineering. Hartford: Rensselaer Polytechnic Institute.

Diaz, A., & Sigmund, O. (1995). Checkerboard patterns in layout optimization. *Structural Optimization, 10*(1), 40-45.

Holcombe, S. (2011, November 24th). stlwrite. Retrieved from https://www.mathworks.com/matlabcentral/fileexchange/20922-stlwrite-filename--varargin-

Jog, C. S., & Haber, R. B. (1996). Stability of finite element models for distributed-parameter optimization and topology design. *Computer Methods in Applied Mechanics and Engineering, 130*(3-4), 203-226.

Pena, V., Micali, M. K., & Lal, B. (2014). U.S. Federal Investment in the Origin and Evolution of Additive Manufacturing: A Case Study of the National Science Foundation. *3D Printing and Additive Manufacturing, 1*(4), 185-193.

SCIART. (2017). CloudTopopt. Madison, Wisconsin. Retrieved from http://www.cloudtopopt.com/

Sigmund, O. (2001). A 99 line topology optimization code written in Matlab. *Structural and Multidisciplinary Optimization, 21*(2), 120-127.

Sigmund, O., & Petersson, J. (1998). Numerical Instabilities in Topology Optimization: A Survey on procedures Dealing With Checkerboards, Mesh-Dependencies and Local Minima. *Structural Optimization, 16*(1), 68-75.

Weller, C., Kleer, R., & Piller, F. T. (2015). Economic implications of 3D printing: Market structure models in light of additive manufacturing revisited. *International Journal of Production Economics, 164*, 43-56.

Zhou, M., Shyy, Y., & Thomas, H. (2001). Checkerboard and minimum member size control in topology optimization. *Structural and Multidisciplinary Optimization, 21*(2), 152-158.

Appendix A: 99 Line MATLAB Code

```
%%%% A 99 LINE TOPOLOGY OPTIMIZATION CODE BY OLE SIGMUND, JANUARY 2000 %%%
%%%% CODE MODIFIED FOR INCREASED SPEED, September 2002, BY OLE SIGMUND %%%
function top(nelx,nely,volfrac,penal,rmin);
% INITIALIZE
x(1:nely,1:nelx) = volfrac;
loop = 0;
change = 1.;
% START ITERATION
while change > 0.01
    loop = loop + 1;
    xold = x;
    % FE-ANALYSIS
    [U]=FE(nelx,nely,x,penal);
    % OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
    [KE] = lk;
    c = 0.;
    for ely = 1:nely
        for elx = 1:nelx
            n1 = (nely+1)*(elx-1)+ely;
            n2 = (nely+1)* elx   +ely;
            Ue = U([2*n1-1;2*n1; 2*n2-1;2*n2; 2*n2+1;2*n2+2;
2*n1+1;2*n1+2],1);
            c = c + x(ely,elx)^penal*Ue'*KE*Ue;
            dc(ely,elx) = -penal*x(ely,elx)^(penal-1)*Ue'*KE*Ue;
        end
    end
    % FILTERING OF SENSITIVITIES
    [dc]   = check(nelx,nely,rmin,x,dc);
    % DESIGN UPDATE BY THE OPTIMALITY CRITERIA METHOD
    [x]    = OC(nelx,nely,x,volfrac,dc);
    % PRINT RESULTS
    change = max(max(abs(x-xold)));
    disp([' It.: ' sprintf('%4i',loop) ' Obj.: ' sprintf('%10.4f',c) ...
        ' Vol.: ' sprintf('%6.3f',sum(sum(x))/(nelx*nely)) ...
        ' ch.: ' sprintf('%6.3f',change )])
    % PLOT DENSITIES
    colormap(gray);imagesc(-x); axis equal; axis tight; axis off;pause(1e-6);
    img_name = sprintf('img%d.jpg',loop);
    file_name = [File_Path',img_name];
    %imwrite(-x,file_name);
    I = imagesc(-x);
    saveas(I,file_name);
end

%%%%%%%%%% OPTIMALITY CRITERIA UPDATE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [xnew]=OC(nelx,nely,x,volfrac,dc)
l1 = 0; l2 = 100000; move = 0.2;
while (l2-l1 > 1e-4)
    lmid = 0.5*(l2+l1);
    xnew = max(0.001,max(x-move,min(1.,min(x+move,x.*sqrt(-dc./lmid)))));
    if sum(sum(xnew)) - volfrac*nelx*nely > 0;
        l1 = lmid;
    else
```

```
        l2 = lmid;
    end
end
%%%%%%%%%% MESH-INDEPENDENCY FILTER %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [dcn]=check(nelx,nely,rmin,x,dc)
dcn=zeros(nely,nelx);
for i = 1:nelx
    for j = 1:nely
        sum=0.0;
        for k = max(i-floor(rmin),1):min(i+floor(rmin),nelx)
            for l = max(j-floor(rmin),1):min(j+floor(rmin),nely)
                fac = rmin-sqrt((i-k)^2+(j-l)^2);
                sum = sum+max(0,fac);
                dcn(j,i) = dcn(j,i) + max(0,fac)*x(l,k)*dc(l,k);
            end
        end
        dcn(j,i) = dcn(j,i)/(x(j,i)*sum);
    end
end
%%%%%%%%%% FE-ANALYSIS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [U]=FE(nelx,nely,x,penal)
[KE] = lk;
K = sparse(2*(nelx+1)*(nely+1), 2*(nelx+1)*(nely+1));
F = sparse(2*(nely+1)*(nelx+1),1); U = zeros(2*(nely+1)*(nelx+1),1);
for elx = 1:nelx
    for ely = 1:nely
        n1 = (nely+1)*(elx-1)+ely;
        n2 = (nely+1)* elx   +ely;
        edof = [2*n1-1; 2*n1; 2*n2-1; 2*n2; 2*n2+1; 2*n2+2; 2*n1+1; 2*n1+2];
        K(edof,edof) = K(edof,edof) + x(ely,elx)^penal*KE;
    end
end
% DEFINE LOADS AND SUPPORTS (HALF MBB-BEAM)
F(2,1) = -1;
fixeddofs   = union([1:2:2*(nely+1)],[2*(nelx+1)*(nely+1)]);
alldofs     = [1:2*(nely+1)*(nelx+1)];
freedofs    = setdiff(alldofs,fixeddofs);
% SOLVING
U(freedofs,:) = K(freedofs,freedofs) \ F(freedofs,:);
U(fixeddofs,:)= 0;
%%%%%%%%%% ELEMENT STIFFNESS MATRIX %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [KE]=lk
E = 1.;
nu = 0.3;
k=[ 1/2-nu/6   1/8+nu/8 -1/4-nu/12 -1/8+3*nu/8 ...
    -1/4+nu/12 -1/8-nu/8  nu/6       1/8-3*nu/8];
KE = E/(1-nu^2)*[ k(1) k(2) k(3) k(4) k(5) k(6) k(7) k(8)
    k(2) k(1) k(8) k(7) k(6) k(5) k(4) k(3)
    k(3) k(8) k(1) k(6) k(7) k(4) k(5) k(2)
    k(4) k(7) k(6) k(1) k(8) k(3) k(2) k(5)
    k(5) k(6) k(7) k(8) k(1) k(2) k(3) k(4)
    k(6) k(5) k(4) k(3) k(2) k(1) k(8) k(7)
    k(7) k(4) k(5) k(2) k(3) k(8) k(1) k(6)
    k(8) k(3) k(2) k(5) k(4) k(7) k(6) k(1)];
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This Matlab code was written by Ole Sigmund, Department of Solid          %
% Mechanics, Technical University of Denmark, DK-2800 Lyngby, Denmark.      %
% Please sent your comments to the author: sigmund@fam.dtu.dk               %
%                                                                           %
% The code is intended for educational purposes and theoretical details     %
% are discussed in the paper                                                %
% "A 99 line topology optimization code written in Matlab"                  %
% by Ole Sigmund (2001), Structural and Multidisciplinary Optimization,     %
% Vol 21, pp. 120--127.                                                     %
%                                                                           %
% The code as well as a postscript version of the paper can be              %
% downloaded from the web-site: http://www.topopt.dtu.dk                    %
%                                                                           %
% Disclaimer:                                                               %
% The author reserves all rights but does not guaranty that the code is     %
% free from errors. Furthermore, he shall not be liable in any event        %
% caused by the use of the program.                                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Appendix B: Non-Penalized Beam Thickness Calculation MATLAB Code

```
Seth Grinde 02/09/2018


% Calls colored image
A=imread('density gradient beam images as jet colormap');
A=imresize(A,[150,400]);
Figure
Imshow(A)

% Separates image into RGB
A1=A(:,:,1);
A2=A(:,:,2);
A3=A(:,:,3);

a = zeros(size(A,1), size(A,2));
a2 = zeros(size(A2,1), size(A2,2));

R = cat(3,A1,a,a);
G = cat(3,a2,A2,a2);
B = cat(3,a2,a2,A3);

% Creates an image of GB
Y=G+B;

Y(:,1:53,:)=[];
Y(1:20,:,:)=[];
Y(114:130,:,:)=[];
Y(:,311:347,:)=[];

Y=imresize(Y,[150,400]);
figure
imshow(Y)

%Creates individual layer image
for x=0:15:240
    x;
    x2=x+15;
    BWA = roicolor(A2,x,x2);
    BWA(:,1:53,:)=[];
    BWA(1:20,:,:)=[];
    BWA(114:130,:,:)=[];
    BWA(:,311:347,:)=[];
    figure
    imshow(BW)

    img_name2 = sprintf('Thickness%d.jpg',x);
    filename = ['File location',img_name2];
    imwrite(BWA,filename)
end
```

```
%Displays A2 and A

A2(:,1:53,:)=[];
A2(1:20,:,:)=[];
A2(114:130,:,:)=[];
A2(:,311:347,:)=[];
figure
imshow(A2)

A(:,1:53,:)=[];
A(1:20,:,:)=[];
A(114:130,:,:)=[];
A(:,311:347,:)=[];
figure
imshow(A)


%% Thickness Calculations

[m,n,o]=size(Y);
z=zeros(m,n);
i=0;
for y=1:1:m
    for x=1:1:n
        if Y(y,x,3)>80 && Y(y,x,3)<210 && Y(y,x,2)<100
            p=100;
        elseif Y(y,x,2)>15 && Y(y,x,3)<=5
            p=(Y(y,x,2)*.09765625)+14.16014525;
        elseif Y(y,x,2)>=230 && Y(y,x,3)>5
            if Y(y,x,3)>5 && Y(y,x,3)<85
                p=39.0625;
            elseif Y(y,x,3)>=85 && Y(y,x,3)<170
                p=51.5625;
            elseif Y(y,x,3)>=170 && Y(y,x,3)<=255
                p=64.0625;
            end
        elseif Y(y,x,3)>=210 && Y(y,x,2)<=230
            p=76.5625;
        elseif Y(y,x,3)>=5 && Y(y,x,3)<=25 && Y(y,x,2)<230 && Y(y,x,2)>40
            p=29.6875;
        elseif Y(y,x,3)<30 && Y(y,x,2)<=40
            p=0;
        end
        z(y,x)=p/2;
        i=i+1;
        s(i,:)=[x,y,z(y,x)];
    end
end

x=s(:,1);
y=s(:,2);
z2=s(:,3);
k=[x y z2];
figure(4)
```

```
axis image
mesh(z)
```

Appendix C: Beam Load-Deflection Plot Maker/Flexural Properties Calculator MATLAB Code

```
Seth Grinde 04/29/2018

%%%%%%%%%%%%%%%%%%%%%%%%%%%% Data Reader %%%%%%%%%%%%%%%%%%%%%%%%%%%%
filename = 'Data File Location';
delimiter = ',';
startRow = 2;

fileID = fopen(filename,'r');
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, 'EmptyValue'
,NaN,'HeaderLines' ,startRow-1, 'ReturnOnError', false);

fclose(fileID);

B1Force = dataArray{:, 1};
B1Stroke = dataArray{:, 2};
B2Force = dataArray{:, 3};
B2Stroke = dataArray{:, 4};
B3Force = dataArray{:, 5};
B3Stroke = dataArray{:, 6};
B4Force = dataArray{:, 7};
B4Stroke = dataArray{:, 8};
H1Force = dataArray{:, 9};
H1Stroke = dataArray{:, 10};
H2Force = dataArray{:, 11};
H2Stroke = dataArray{:, 12};
H3Force = dataArray{:, 13};
H3Stroke = dataArray{:, 14};
H4Force = dataArray{:, 15};
H4Stroke = dataArray{:, 16};
S1Force = dataArray{:, 17};
S1Stroke = dataArray{:, 18};
S2Force = dataArray{:, 19};
S2Stroke = dataArray{:, 20};
S3Force = dataArray{:, 21};
S3Stroke = dataArray{:, 22};
S4Force = dataArray{:, 23};
S4Stroke = dataArray{:, 24};
P1Force = dataArray{:, 25};
P1Stroke = dataArray{:, 26};
P2Force = dataArray{:, 27};
P2Stroke = dataArray{:, 28};
P3Force = dataArray{:, 29};
P3Stroke = dataArray{:, 30};
P4Force = dataArray{:, 31};
P4Stroke = dataArray{:, 32};
NP1Force = dataArray{:, 33};
NP1Stroke = dataArray{:, 34};
NP2Force = dataArray{:, 35};
NP2Stroke = dataArray{:, 36};
NP3Force = dataArray{:, 37};
NP3Stroke = dataArray{:, 38};
NP4Force = dataArray{:, 39};
```

```
NP4Stroke = dataArray{:, 40};
PW1Force = dataArray{:, 41};
PW1Stroke = dataArray{:, 42};
PW2Force = dataArray{:, 43};
PW2Stroke = dataArray{:, 44};
PW3Force = dataArray{:, 45};
PW3Stroke = dataArray{:, 46};
PW4Force = dataArray{:, 47};
PW4Stroke = dataArray{:, 48};

clearvars filename delimiter startRow formatSpec fileID dataArray ans;

%%%%%%%%%%%%%%%%%%%%%%%%%%%% Volume Calculations %%%%%%%%%%%%%%%%%%%%%%%%%%%
density=15.547;
% Solid beam volume calculation
BV=((17.47+17.52+17.5+17.47)/(4*density))-(2*.75*.25*1);
% Beam with holes volume calculation
HV=((15.68+15.63+15.68+15.64)/(4*density))-(2*.75*.25*1);
% Beam with slots volume calculation
SV=((14.25+14.24+14.18+14.21)/(4*density))-(2*.75*.25*1);
% Penalized MATLAB beam volume calculation
PV=((6.64+6.65+6.66+6.63)/(4*density))-(2*.125*.25*1);
% Non-penalized MATLAB beam volume calculation
NPV=((7.4+7.41+7.41+7.4)/(4*density))-(2*.2*.25*1);
% ParetoCloud beam volume calculation
PWV=((7.78+7.78+7.79+7.78)/(4*density))-(2*.2*.25*1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Beam Dimensions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

b=.25;          %beam thickness
d=.75;          %beam height
L=4;            %beam length

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Load Calculation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Average Load
Bf=(B1Force+B2Force+B3Force+B4Force)/4;
Hf=(H1Force+H2Force+H3Force+H4Force)/4;
Sf=(S1Force+S2Force+S3Force+S4Force)/4;
Pf=(P1Force+P2Force+P3Force+P4Force)/4;
NPf=(NP1Force+NP2Force+NP3Force+NP4Force)/4;
PWf=(PW1Force+PW2Force+PW3Force+PW4Force)/4;

%%%%%%%%%%%%%%%%%%%%%%%%%%% Deflection Calculations %%%%%%%%%%%%%%%%%%%%%%%%%

%Average Deflection
Bd=(B1Stroke+B2Stroke+B3Stroke+B4Stroke)/4;
Hd=(H1Stroke+H2Stroke+H3Stroke+H4Stroke)/4;
Sd=(S1Stroke+S2Stroke+S3Stroke+S4Stroke)/4;
Pd=(P1Stroke+P2Stroke+P3Stroke+P4Stroke)/4;
NPd=(NP1Stroke+NP2Stroke+NP3Stroke+NP4Stroke)/4;
PWd=(PW1Stroke+PW2Stroke+PW3Stroke+PW4Stroke)/4;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Solid Beam %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
figure(1)
set(figure(1),'Position',[200 250 1000 500])
plot(B1Stroke,B1Force,B2Stroke,B2Force,B3Stroke,B3Force,B4Stroke,B4Force)
grid on
xlabel('Deflection (in)','FontSize',12)
ylabel('Load (lbs)','FontSize',12)
title('Solid Beam Load vs. Deflection','FontSize',18)
legend('B1','B2','B3','B4','Location','northeastoutside')
% Flexural Resilience Calculation
[strain,num]=min(abs(Bf-115));
Solid_Resilience = trapz(Bd(1:num),Bf(1:num));
% Flexural Modulus Calculation
[val,loc]=min(abs(Bd-.05));
g=polyfit(Bd(1:loc),Bf(1:loc),1);
BFM=(g(1)*L^3)/(4*b*d^3);
h=refline(g(1),g(2));
set(h,'Color','r')
xlim([0,0.25])
ylim([0,200])
% Flexural Toughness Calculation
Bf2=Bf;
Bf2(isnan(Bf2))=[];
Bd2=Bd;
Bd2(isnan(Bd2))=[];
Solid_Toughness = trapz(Bd2,Bf2);
% Energy Calculation
SolidPW=Solid_Resilience*BV;
SolidFW=Solid_Toughness*BV;
% Present Data
a1=[num2str(BV), ' in^3'];
a2=[num2str(BFM), ' psi'];
a3=[num2str(Solid_Resilience), ' in*lb/in^3'];
a4=[num2str(SolidPW), ' in*lb'];
a5=[num2str(Solid_Toughness), ' in*lb/in^3'];
a6=[num2str(SolidFW), ' in*lb'];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Holes %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(2)
set(figure(2),'Position',[200 250 1000 500])
plot(H1Stroke,H1Force,H2Stroke,H2Force,H3Stroke,H3Force,H4Stroke,H4Force)
grid on
xlabel('Deflection (in)','FontSize',12)
ylabel('Load (lbs)','FontSize',12)
title('Holes Beam Load vs. Deflection','FontSize',18)
legend('H1','H2','H3','H4','Location','northeastoutside')
% Flexural Resilience Calculation
[strain,num]=min(abs(Hf-85));
Holes_Resilience = trapz(Hd(1:num),Hf(1:num));
% Flexural Modulus Calculation
[val,loc]=min(abs(Hd-.05));
i=polyfit(Hd(1:loc),Hf(1:loc),1);
HFM=(i(1)*L^3)/(4*b*d^3);
r=refline(i(1),i(2));
set(r,'Color','r')
```

```
xlim([0,0.25])
ylim([0,200])
% Flexural Toughness Calculation
Hf2=Hf;
Hf2(isnan(Hf2))=[];
Hd2=Hd;
Hd2(isnan(Hd2))=[];
Holes_Toughness = trapz(Hd2,Hf2);
% Energy Calculation
HolesPW=Holes_Resilience*HV;
HolesFW=Holes_Toughness*HV;
% Present Data
c1=[num2str(HV), ' in^3'];
c2=[num2str(HFM), ' psi'];
c3=[num2str(Holes_Resilience), ' in*lb/in^3'];
c4=[num2str(HolesPW), ' in*lb'];
c5=[num2str(Holes_Toughness), ' in*lb/in^3'];
c6=[num2str(HolesFW), ' in*lb'];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Slot %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(3)
set(figure(3),'Position',[200 250 1000 500])
plot(S1Stroke,S1Force,S2Stroke,S2Force,S3Stroke,S3Force,S4Stroke,S4Force)
grid on
xlabel('Deflection (in)','FontSize',12)
ylabel('Load (lbs)','FontSize',12)
title('Slot Beam Load vs. Deflection','FontSize',18)
legend('S1','S2','S3','S4','Location','northeastoutside')
% Flexural Resilience Calculation
[strain,num]=min(abs(Sf-50));
Slot_Resilience = trapz(Sd(1:num),Sf(1:num));
% Flexural Modulus Calculation
[val,loc]=min(abs(Sd-.05));
j=polyfit(Sd(1:loc),Sf(1:loc),1);
SFM=(j(1)*L^3)/(4*b*d^3);
r=refline(j(1),j(2));
set(r,'Color','r')
xlim([0,0.25])
ylim([0,200])
% Flexural Toughness Calculation
Sf2=Sf;
Sf2(isnan(Sf2))=[];
Sd2=Sd;
Sd2(isnan(Sd2))=[];
Slot_Toughness = trapz(Sd2,Sf2);
% Energy Calculation
SlotPW=Slot_Resilience*SV;
SlotFW=Slot_Toughness*SV;
% Present Data
b1=[num2str(SV), ' in^3'];
b2=[num2str(SFM), ' psi'];
b3=[num2str(Slot_Resilience), ' in*lb/in^3'];
b4=[num2str(SlotPW), ' in*lb'];
b5=[num2str(Slot_Toughness), ' in*lb/in^3'];
b6=[num2str(SlotFW), ' in*lb'];
```

```
%%%%%%%%%%%%%%%%%%%%%%%%% Penalized MATLAB Beam %%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(4)
set(figure(4),'Position',[200 250 1000 500])
plot(P1Stroke,P1Force,P2Stroke,P2Force,P3Stroke,P3Force,P4Stroke,P4Force)
grid on
xlabel('Deflection (in)','FontSize',12)
ylabel('Load (lbs)','FontSize',12)
title('Penalized MATLAB Beam Load vs. Deflection','FontSize',18)
legend('P1','P2','P3','P4','Location','northeastoutside')
% Flexural Resilience Calculation
[strain,num]=min(abs(Pf-65));
Penalized_Resilience = trapz(Pd(1:num),Pf(1:num));
% Flexural Modulus Calculation
[val,loc]=min(abs(Pd-.04));
k=polyfit(Pd(1:loc),Pf(1:loc),1);
PFM=(k(1)*L^3)/(4*b*d^3);
r=refline(k(1),k(2));
set(r,'Color','r')
xlim([0,0.25])
ylim([0,200])
% Flexural Toughness Calculation
Pf2=Pf;
Pf2(isnan(Pf2))=[];
Pd2=Pd;
Pd2(isnan(Pd2))=[];
Penalized_Toughness = trapz(Pd2,Pf2);
% Energy Calculation
PenalizedPW=Penalized_Resilience*PV;
PenalizedFW=Penalized_Toughness*PV;
% Present Data
d1=[num2str(PV), ' in^3'];
d2=[num2str(PFM), ' psi'];
d3=[num2str(Penalized_Resilience), ' in*lb/in^3'];
d4=[num2str(PenalizedPW), ' in*lb'];
d5=[num2str(Penalized_Toughness), ' in*lb/in^3'];
d6=[num2str(PenalizedFW), ' in*lb'];

%%%%%%%%%%%%%%%%%%%%%%%%% Non-Penalized MATLAB BEAM %%%%%%%%%%%%%%%%%%%%%%%%%

figure(5)
set(figure(5),'Position',[200 250 1000 500])
plot(NP1Stroke,NP1Force,NP2Stroke,NP2Force,NP3Stroke,NP3Force,NP4Stroke,NP4Fo
rce)
grid on
xlabel('Deflection (in)','FontSize',12)
ylabel('Load (lbs)','FontSize',12)
title('Non-Penalized MATLAB BEAM Load vs. Deflection','FontSize',18)
legend('NP1','NP2','NP3','NP4','Location','northeastoutside')
% Flexural Resilience Calculation
[strain,num]=min(abs(NPf-110));
nonpen_Resilience = trapz(NPd(1:num),NPf(1:num));
% Flexural Modulus Calculation
[val,loc]=min(abs(NPd-.1));
l=polyfit(NPd(1:loc),NPf(1:loc),1);
```

```
NPFM=(l(1)*L^3)/(4*b*d^3);
r=refline(l(1),l(2));
set(r,'Color','r')
xlim([0,0.25])
ylim([0,200])
% Flexural Toughness Calculation
NPf2=NPf;
NPf2(isnan(NPf2))=[];
NPd2=NPd;
NPd2(isnan(NPd2))=[];
nonpen_Toughness = trapz(NPd2,NPf2);
% Energy Calculation
nonpenPW=nonpen_Resilience*NPV;
nonpenFW=nonpen_Toughness*NPV;
% Present Data
e1=[num2str(NPV), ' in^3'];
e2=[num2str(NPFM), ' psi'];
e3=[num2str(nonpen_Resilience), ' in*lb/in^3'];
e4=[num2str(nonpenPW), ' in*lb'];
e5=[num2str(nonpen_Toughness), ' in*lb/in^3'];
e6=[num2str(nonpenFW), ' in*lb'];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Pareto Beam %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(6)
set(figure(6),'Position',[200 250 1000 500])
plot(PW1Stroke,PW1Force,PW2Stroke,PW2Force,PW3Stroke,PW3Force,PW4Stroke,PW4Fo
rce)
grid on
xlabel('Deflection (in)','FontSize',12)
ylabel('Load (lbs)','FontSize',12)
title('ParetoCloud Beam Load vs. Deflection','FontSize',18)
legend('PW1','PW2','PW3','PW4','Location','northeastoutside')
% Flexural Resilience Calculation
[strain,num]=min(abs(PWf-100));
Pareto_Resilience = trapz(PWd(1:num),PWf(1:num));
% Flexural Modulus Calculation
[val,loc]=min(abs(PWd-.06));
m=polyfit(PWd(1:loc),PWf(1:loc),1);
PWFM=(m(1)*L^3)/(4*b*d^3);
r=refline(m(1),m(2));
set(r,'Color','r')
xlim([0,0.25])
ylim([0,200])
% Flexural Toughness Calculation
PWf2=PWf;
PWf2(isnan(PWf2))=[];
PWd2=PWd;
PWd2(isnan(PWd2))=[];
Pareto_Toughness = trapz(PWd2,PWf2);
% Energy Calculation
ParetoPW=Pareto_Resilience*PWV;
ParetoFW=Pareto_Toughness*PWV;
% Present Data
f1=[num2str(PWV), ' in^3'];
f2=[num2str(PWFM), ' psi'];
```

```
f3=[num2str(Pareto_Resilience), ' in*lb/in^3'];
f4=[num2str(ParetoPW), ' in*lb'];
f5=[num2str(Pareto_Toughness), ' in*lb/in^3'];
f6=[num2str(ParetoFW), ' in*lb'];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Combined Plots %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(7)
set(figure(7),'Position',[200 250 1000 500])
plot(Bd,Bf,Hd,Hf,Sd,Sf,Pd,Pf,NPd,NPf,PWd,PWf)
grid on
xlabel('Deflection (in)','FontSize',12)
ylabel('Load (lbs)','FontSize',12)
title('Load vs. Deflection Comparison','FontSize',18)
legend('Solid','Holes','Slot','Penalized','Non
Penalized','ParetoCloud','Location','northeastoutside')
xlim([0,0.25])
ylim([0,200])

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Table Formation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C={'Solid' a1 a2 a3 a4 a5 a6;'Slot' b1 b2 b3 b4 b5 b6;'Holes' c1 c2 c3 c4 c5
c6;'Penalized' d1 d2 d3 d4 d5 d6;'Non Penalized' e1 e2 e3 e4 e5
e6;'ParetoCloud' f1 f2 f3 f4 f5 f6};
T=cell2table(C,'VariableNames',{'Beam' 'Volume' 'Flexural_Modulus'
'Flexural_Resilience' 'Plastic_Energy' 'Flexural_Toughness'
'Fracture_Energy'})

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Save Plots %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
numplot=7;
for z=1:numplot
    saveas(figure(z),fullfile('File Location',['figure' num2str(z) '.jpg']))
end
```

Appendix D: STL Write MATLAB Code

```
function stlwrite(filename, varargin)
%STLWRITE   Write STL file from patch or surface data.
%
%   STLWRITE(FILE, FV) writes a stereolithography (STL) file to FILE for a
%   triangulated patch defined by FV (a structure with fields 'vertices'
%   and 'faces').
%
%   STLWRITE(FILE, FACES, VERTICES) takes faces and vertices separately,
%   rather than in an FV struct
%
%   STLWRITE(FILE, X, Y, Z) creates an STL file from surface data in X, Y,
%   and Z. STLWRITE triangulates this gridded data into a triangulated
%   surface using triangulation options specified below. X, Y and Z can be
%   two-dimensional arrays with the same size. If X and Y are vectors with
%   length equal to SIZE(Z,2) and SIZE(Z,1), respectively, they are passed
%   through MESHGRID to create gridded data. If X or Y are scalar values,
%   they are used to specify the X and Y spacing between grid points.
%
%   STLWRITE(...,'PropertyName',VALUE,'PropertyName',VALUE,...) writes an
%   STL file using the following property values:
%
%   MODE          - File is written using 'binary' (default) or 'ascii'.
%
%   TITLE         - Header text (max 80 chars) written to the STL file.
%
%   TRIANGULATION - When used with gridded data, TRIANGULATION is either:
%                        'delaunay'  - (default) Delaunay triangulation of X,
Y
%                        'f'         - Forward slash division of grid quads
%                        'b'         - Back slash division of quadrilaterals
%                        'x'         - Cross division of quadrilaterals
%                    Note that 'f', 'b', or 't' triangulations now use an
%                    inbuilt version of FEX entry 28327, "mesh2tri".
%
%   FACECOLOR     - Single colour (1-by-3) or one-colour-per-face (N-by-3)
%                    vector of RGB colours, for face/vertex input. RGB range
%                    is 5 bits (0:31), stored in VisCAM/SolidView format
%
(http://en.wikipedia.org/wiki/STL_(file_format)#Color_in_binary_STL)
%
%   Example 1:
%     % Write binary STL from face/vertex data
%     tmpvol = false(20,20,20);      % Empty voxel volume
%     tmpvol(8:12,8:12,5:15) = 1;    % Turn some voxels on
%     fv = isosurface(~tmpvol, 0.5); % Make patch w. faces "out"
%     stlwrite('test.stl',fv)        % Save to binary .stl
%
%   Example 2:
%     % Write ascii STL from gridded data
%     [X,Y] = deal(1:40);             % Create grid reference
%     Z = peaks(40);                  % Create grid height
%     stlwrite('test.stl',X,Y,Z,'mode','ascii')
```

```
%
%   Example 3:
%     % Write binary STL with coloured faces
%     cVals = fv.vertices(fv.faces(:,1),3); % Colour by Z height.
%     cLims = [min(cVals) max(cVals)];      % Transform height values
%     nCols = 255;  cMap = jet(nCols);      % onto an 8-bit colour map
%     fColsDbl = interp1(linspace(cLims(1),cLims(2),nCols),cMap,cVals);
%     fCols8bit = fColsDbl*255; % Pass cols in 8bit (0-255) RGB triplets
%     stlwrite('testCol.stl',fv,'FaceColor',fCols8bit)

%   Original idea adapted from surf2stl by Bill McDonald. Huge speed
%   improvements implemented by Oliver Woodford. Non-Delaunay triangulation
%   of quadrilateral surface courtesy of Kevin Moerman. FaceColor
%   implementation by Grant Lohsen.
%
%   Author: Sven Holcombe, 11-24-11



% Check valid filename path
path = fileparts(filename);
if ~isempty(path) && ~exist(path,'dir')
    error('Directory "%s" does not exist.',path);
end

% Get faces, vertices, and user-defined options for writing
[faces, vertices, options] = parseInputs(varargin{:});
asciiMode = strcmp( options.mode ,'ascii');

% Create the facets
facets = single(vertices');
facets = reshape(facets(:,faces'), 3, 3, []);

% Compute their normals
V1 = squeeze(facets(:,2,:) - facets(:,1,:));
V2 = squeeze(facets(:,3,:) - facets(:,1,:));
normals = V1([2 3 1],:) .* V2([3 1 2],:) - V2([2 3 1],:) .* V1([3 1 2],:);
clear V1 V2
normals = bsxfun(@times, normals, 1 ./ sqrt(sum(normals .* normals, 1)));
facets = cat(2, reshape(normals, 3, 1, []), facets);
clear normals

% Open the file for writing
permissions = {'w','wb+'};
fid = fopen(filename, permissions{asciiMode+1});
if (fid == -1)
    error('stlwrite:cannotWriteFile', 'Unable to write to %s', filename);
end

% Write the file contents
if asciiMode
    % Write HEADER
    fprintf(fid,'solid %s\r\n',options.title);
    % Write DATA
    fprintf(fid,[...
        'facet normal %.7E %.7E %.7E\r\n' ...
        'outer loop\r\n' ...
        'vertex %.7E %.7E %.7E\r\n' ...
        'vertex %.7E %.7E %.7E\r\n' ...
        'vertex %.7E %.7E %.7E\r\n' ...
        'endloop\r\n' ...
        'endfacet\r\n'], facets);
    % Write FOOTER
    fprintf(fid,'endsolid %s\r\n',options.title);

else % BINARY
    % Write HEADER
    fprintf(fid, '%-80s', options.title);             % Title
    fwrite(fid, size(facets, 3), 'uint32');           % Number of facets
    % Write DATA
    % Add one uint16(0) to the end of each facet using a typecasting trick
    facets = reshape(typecast(facets(:), 'uint16'), 12*2, []);
    % Set the last bit to 0 (default) or supplied RGB
    facets(end+1,:) = options.facecolor;
    fwrite(fid, facets, 'uint16');
end

% Close the file
fclose(fid);
fprintf('Wrote %d facets\n',size(facets, 2));


%% Input handling subfunctions
function [faces, vertices, options] = parseInputs(varargin)
% Determine input type
if isstruct(varargin{1}) % stlwrite('file', FVstruct, ...)
    if ~all(isfield(varargin{1},{'vertices','faces'}))
        error( 'Variable p must be a faces/vertices structure' );
    end
    faces = varargin{1}.faces;
    vertices = varargin{1}.vertices;
    options = parseOptions(varargin{2:end});

elseif isnumeric(varargin{1})
    firstNumInput = cellfun(@isnumeric,varargin);
    firstNumInput(find(~firstNumInput,1):end) = 0; % Only consider numerical
input PRIOR to the first non-numeric
    numericInputCnt = nnz(firstNumInput);

    options = parseOptions(varargin{numericInputCnt+1:end});
    switch numericInputCnt
        case 3 % stlwrite('file', X, Y, Z, ...)
            % Extract the matrix Z
            Z = varargin{3};

            % Convert scalar XY to vectors
            ZsizeXY = fliplr(size(Z));
            for i = 1:2
                if isscalar(varargin{i})
                    varargin{i} = (0:ZsizeXY(i)-1) * varargin{i};
                end
            end

            % Extract X and Y
            if isequal(size(Z), size(varargin{1}), size(varargin{2}))
                % X,Y,Z were all provided as matrices
                [X,Y] = varargin{1:2};
            elseif numel(varargin{1})==ZsizeXY(1) &&
numel(varargin{2})==ZsizeXY(2)
                % Convert vector XY to meshgrid
                [X,Y] = meshgrid(varargin{1}, varargin{2});
            else
                error('stlwrite:badinput', 'Unable to resolve X and Y
variables');
            end

            % Convert to faces/vertices
            if strcmp(options.triangulation,'delaunay')
                faces = delaunay(X,Y);
                vertices = [X(:) Y(:) Z(:)];
            else
                if ~exist('mesh2tri','file')
                    error('stlwrite:missing', '"mesh2tri" is required to
convert X,Y,Z matrices to STL. It can be downloaded from:\n%s\n',...

'http://www.mathworks.com/matlabcentral/fileexchange/28327')
                end
                [faces, vertices] = mesh2tri(X, Y, Z, options.triangulation);
            end

        case 2 % stlwrite('file', FACES, VERTICES, ...)
            faces = varargin{1};
            vertices = varargin{2};

        otherwise
            error('stlwrite:badinput', 'Unable to resolve input types.');
    end
end

if ~isempty(options.facecolor) % Handle colour preparation
    facecolor = uint16(options.facecolor);
    %Set the Valid Color bit (bit 15)
    c0 = bitshift(ones(size(faces,1),1,'uint16'),15);
    %Red color (10:15), Blue color (5:9), Green color (0:4)
    c0 = bitor(bitshift(bitand(2^6-1, facecolor(:,1)),10),c0);
    c0 = bitor(bitshift(bitand(2^11-1, facecolor(:,2)),5),c0);
    c0 = bitor(bitand(2^6-1, facecolor(:,3)),c0);
    options.facecolor = c0;
else
    options.facecolor = 0;
end

function options = parseOptions(varargin)
IP = inputParser;
IP.addParamValue('mode', 'binary', @ischar)
IP.addParamValue('title', sprintf('Created by stlwrite.m %s',datestr(now)),
@ischar);
```

```
IP.addParamValue('triangulation', 'delaunay', @ischar);
IP.addParamValue('facecolor',[], @isnumeric)
IP.addParamValue('facecolour',[], @isnumeric)
IP.parse(varargin{:});
options = IP.Results;
if ~isempty(options.facecolour)
    options.facecolor = options.facecolour;
end

function [F,V]=mesh2tri(X,Y,Z,tri_type)
% function [F,V]=mesh2tri(X,Y,Z,tri_type)
%
% Available from http://www.mathworks.com/matlabcentral/fileexchange/28327
% Included here for convenience. Many thanks to Kevin Mattheus Moerman
% kevinmoerman@hotmail.com
% 15/07/2010
%------------------------------------------------------------------------

[J,I]=meshgrid(1:1:size(X,2)-1,1:1:size(X,1)-1);

switch tri_type
    case 'f'%Forward slash
        TRI_I=[I(:),I(:)+1,I(:)+1;  I(:),I(:),I(:)+1];
        TRI_J=[J(:),J(:)+1,J(:);   J(:),J(:)+1,J(:)+1];
        F = sub2ind(size(X),TRI_I,TRI_J);
    case 'b'%Back slash
        TRI_I=[I(:),I(:)+1,I(:);  I(:)+1,I(:)+1,I(:)];
        TRI_J=[J(:)+1,J(:),J(:);   J(:)+1,J(:),J(:)+1];
        F = sub2ind(size(X),TRI_I,TRI_J);
    case 'x'%Cross
        TRI_I=[I(:)+1,I(:);  I(:)+1,I(:)+1;  I(:),I(:)+1;    I(:),I(:)];
        TRI_J=[J(:),J(:);    J(:)+1,J(:);    J(:)+1,J(:)+1;  J(:),J(:)+1];
        IND=((numel(X)+1):numel(X)+prod(size(X)-1))';
        F = sub2ind(size(X),TRI_I,TRI_J);
        F(:,3)=repmat(IND,[4,1]);
        Fe_I=[I(:),I(:)+1,I(:)+1,I(:)]; Fe_J=[J(:),J(:),J(:)+1,J(:)+1];
        Fe = sub2ind(size(X),Fe_I,Fe_J);
        Xe=mean(X(Fe),2); Ye=mean(Y(Fe),2);  Ze=mean(Z(Fe),2);
        X=[X(:);Xe(:)]; Y=[Y(:);Ye(:)]; Z=[Z(:);Ze(:)];
end

V=[X(:),Y(:),Z(:)];
```

SIGNATURE PAGE

This is to certify that the thesis prepared by Seth A. Grinde entitled "Topology Optimization for Additive Manufacturing Using SIMP Method" has been examined and approved for acceptance by the Department of General Engineering, Montana Tech of The University of Montana, on this 25th day of April, 2018.

[illegible]

Bruce Madigan, PhD, Professor and Department Head
Department of General Engineering
Chair, Examination Committee

[illegible]

Peter Lucon, PhD, Associate Professor
Department of General Engineering
Member, Examination Committee

Jerry Downey

Jerry Downey, PhD, Professor
Department of Metallurgical and Materials Engineering
Member, Examination Committee

Scott L Coguill

Scott Coguill, MS, Lab Director
Department of Mechanical and Civil Engineering
Member, Examination Committee