

Clemson University

TigerPrints

All Theses

Theses

August 2020

Topology Optimization of Irregular Shaped Pressure Vessels Using a Level-Set Method

John Michal Kremar

Clemson University, John.Kremar@gmail.com

Follow this and additional works at: https://tigerprints.clemson.edu/all_theses

Recommended Citation

Kremar, John Michal, "Topology Optimization of Irregular Shaped Pressure Vessels Using a Level-Set Method" (2020). *All Theses*. 3379.

https://tigerprints.clemson.edu/all_theses/3379

This Thesis is brought to you for free and open access by the Theses at TigerPrints. It has been accepted for inclusion in All Theses by an authorized administrator of TigerPrints. For more information, please contact kokeefe@clemson.edu.

TOPOLOGY OPTIMIZATION OF IRREGULAR SHAPED PRESSURE VESSELS USING A LEVEL-SET METHOD

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Mechanical Engineering

by
John Michal Kremar
August 2020

Accepted by:
Dr. Georges Fadel, Committee Chair
Dr. Lonny Thompson
Dr. Vincent Ervin

ABSTRACT

Advances in manufacturing capabilities, such as additive manufacturing, have expanded the design freedom given to engineers enabling more efficient designs through the use of complex geometries. However, determining the optimal geometric structure for a given set of performance criteria can be quite challenging when given such design freedom. One technique to do so is with the use of topology optimization methods, in which optimal material distribution within a given design space is determined. Many established topology optimization methods are developed such that a set of boundary conditions are prescribed to the design domain and remain fixed throughout the optimization process of determining the material distribution. This eliminates the ability to implement design dependent loading conditions, such as pressure loading, which requires tracking (following) the pressure surface as the geometry evolves during the optimization process. In this thesis, a level-set topology optimization method is implemented based on voxel elements on design domains in \mathbb{R}^3 subjected to internal pressure loading, such as in the case of a non-spherical or cylindrical pressure vessel.

Following a thorough literature review, a level-set function was chosen to define a crisp material/void boundary for identifying loading conditions caused by the applied pressure. This pressure loading is calculated as an applied traction across all material elements, excluding exterior surface nodes. This results in an equal and opposite cancelation throughout the material domain and leaving forces only at desired nodes along the material/void boundary. This implementation only requires material elements to be meshed, allowing for remeshing throughout the process to increase accuracy while saving computational cost by excluding void regions. Additionally, to improve convergence, the Lagrangian formulation of a penalty is

replaced by a method analogous to PID-control systems as the algorithm hones in on convergence.

To test the effectiveness of the method and the practicality of designing an irregular pressure vessel, the gas storage tanks of the MK-16 rebreather for the US NAVY were redesigned within the current system's geometric constraints in an effort to increase gas storage capacity. To do this, an outside domain geometry of the irregular shaped pressure vessel was defined, and not subject to change, while the optimization code was executed on the interior structure to minimize compliance subjected to an overall volume fraction constraint. This was done at various target volume fractions, and then stresses and compliance values were analyzed and compared to the existing pressure vessel of the MK-16. The findings of this research concluded that designing an irregular shaped pressure vessel is a viable means of increasing storage capacity although future work would need to be executed to manufacture and experimentally validate these findings.

DEDICATION

This thesis is dedicated to my family and friends for all of their love and support throughout this journey. This research would not have been possible without the support of many individuals in particular my parents, Connie and Jason, for their immense love, support, and guidance throughout my entire life. I certainly would not have made it to where I am without them. Additionally, I would like to dedicate this work to my brother, Clark, for his friendship, brilliant young mind, and continual competitiveness. I cannot wait to see what the future has in store for you little brother, even if it means beating me on occasion.

ACKNOWLEDGEMENTS

I would like to thank and express my deepest gratitude to some of the most influential individuals who helped make this research possible. Firstly, I would like to thank my advisor, Dr. Fadel, for his time, wisdom, and guidance throughout this journey. Additionally, I would like to thank the *Science, Mathematics, And Research for Transformation* (SMART) scholarship for service program for providing me with the opportunity to pursue my academic interest and career goals. I would also like to thank my committee members, Dr. Thompson and Dr. Ervin for their time and contributions.

Furthermore, I would like to thank all of the professors that have dedicated their time and efforts towards my education. I would like to recognize Dr. Ferguson, of North Carolina State University, for establishing the academic foundation and sparking the interest in design optimization. Which was then further developed by the teachings of Dr. Fadel and Dr. Wiecek at Clemson University. Additionally, I would like to express my appreciations to Dr. Li and Dr. Thomspon for their knowledge and teachings of the finite element method. The works of this research draws strongly on the principles taught by these professors and without their dedication and passion I would not have been able to accomplish what I have, or developed my interest in the topics.

Additionally, I would like to recognize the support from my sponsoring facility through the SMART program. The mentorship and comradery of my coworkers during my internships at NSWC – Panama City, FL have been an invaluable resource of knowledge and lessons throughout my journey in academics, career, and life. I will forever be grateful for their role in

my journey and the source of inspiration for this project. I look forward to working hand-in-hand with my friends as I begin the next chapter of my life.

Table of Contents

TITLE PAGE.....	i
ABSTRACT	ii
DEDICATION	iv
ACKNOWLEDGEMENTS	v
TABLE OF TABLES.....	xii
TABLE OF FIGURES.....	xiii
NOMENCLATURE	xviii
CHAPTER I: INTRODUCTION.....	1
1.1 Motivation	2
1.2 Research Objectives	6
1.3 Thesis Outline	7
CHAPTER II: LITERATURE REVIEW.....	8
2.1 Topology Optimization History and Overview	8
2.1.1 Ground Structure Approach	11
2.1.2 Homogenization Methods.....	12
2.1.3 Level-Set Methods.....	19
2.1.4 Topology Optimization Conclusion.....	23
2.2 Level-Set Methods Formulation	24
2.2.1 Level-Set Function Parameterization	26

Table of Contents (Continued)

2.2.2 Geometry Mapping	30
2.2.3 Update Procedure	33
2.2.4 Regularization.....	39
2.3 Design Dependent Loading	41
CHAPTER III: Methodology	47
3.1 Finite Element Analysis.....	47
3.2 Level-Set Method Formulation.....	60
3.3 Design Dependent Pressure Loading.....	67
3.3.1 Two-Dimensional Problems with Pressure Loading	67
3.3.2 Three-Dimensional Pressure Vessel Problems	73
Chapter IV: Implementation	78
4.1 Problem Progression	79
4.2 Mesh Generation.....	82
4.3 Optimization Initialization	87
4.4 Optimization Loop	96
4.4.1 Finite Element Analysis.....	97
4.4.2 Postprocessing and Sensitivity Calculations.....	99
4.4.3 Convergence Checks.....	100

Table of Contents (Continued)

4.4.4 Update Procedure	100
4.4.5 Preparation for Subsequent Iterations.....	104
4.5 Conclusion and Appendix Usage	108
Chapter V: Preliminary Results	110
5.1 Constant Loading Conditions.....	110
5.2 Two-Dimensional Pressure Loading	119
5.3 Three-Dimensional Pressure Box	122
5.3.1 Trials and Issues.....	123
5.3.2 Pressure Box Solutions and Results.....	127
Chapter VI: Irregular Shaped Pressure Vessel Results.....	133
6.1 Existing Pressure Vessel.....	133
6.2 Defining the Design Domain.....	137
6.3 Initial Results	141
6.4 Final Results.....	148
CHAPTER VII: CONCLUSION	161
7.1 Thesis Overview.....	161
7.2 Discoveries from Research Objectives	163
7.3 Future Works.....	164

Table of Contents (Continued)

7.3.1 Refined Meshing.....	165
7.3.2 Stress Constraints	165
7.3.3 Designing for Additive Manufacturing	166
7.3.4 Experimental Validation	166
7.4 Final Remarks	167
REFERENCES.....	169
APPENDICES.....	177
Appendix A: Flow Diagram	178
Appendix B: Mesh Generation Code.....	179
Appendix C: Main Code.....	183
Appendix D: Initial Configuration Subfunction	194
Appendix E: Stiffness Matrix Calculation.....	196
Appendix F: FEA Code	199
Appendix G: Update Code.....	202
Appendix H: Remesh Code.....	204
Appendix I: Polynomial Evaluation Code.....	209
Appendix J: Reading STL file Code.....	211
Appendix K: Cross-Section Viewing Code.....	212

Table of Contents (Continued)

Appendix L: Solid Structure Plotting Code221

Appendix M: Transparent Border Plotting Code222

Appendix N: Stress Plotting Code224

Appendix O: Void and STL Plotting Code226

TABLE OF TABLES

Table 1-1:Oxygen Tank Properties	6
Table 3-2: Master Element Node Coordinates.....	52
Table 4-3: ‘elements’ matrix format	90
Table 4-4: ‘nodes’ matrix format	91
Table 6-5: Existing Pressure Vessel Properties.....	135
Table 6-6: Existing Pressure Vessel FEA Results	136
Table 6-7: Meshing Summary.....	141
Table 6-8: Trials at Varying Target Volumes.....	147
Table 6-9: Remeshing Trials at Various Target Volumes	158

TABLE OF FIGURES

Figure 1-1: MK-16 Rebreather Front (left), back (right)	3
Figure 1-2: MK-16 Rebreather Components	4
Figure 1-3: Oxygen Tank	5
Figure 2-1: Ground Structure Approach Illustration	11
Figure 2-2: Typical Classes of Unit Cells. Left: Square with Square Hole,	13
Figure 2-3: Intermediate Density Versus Rigidity	17
Figure 2-4: Converging Corners in Moving Boundaries	20
Figure 2-5: Level-Set Method Visualization [34]	21
Figure 2-6: 1-D Basis Functions	27
Figure 2-7: Ranges of Influence [34]	28
Figure 2-8: Types of LSF Parameterization [34]	29
Figure 2-9: Types of Geometry Mapping [34]	31
Figure 2-10: Types of Update Information [34]	35
Figure 2-11: Effect of Variable LSF Gradients [34]	41
Figure 2-12: Identifying the 1 st (Left) & 2 nd (Right) Iso-Density Points [52]	43
Figure 2-13: Identifying Consecutive Iso-Density Points [52]	44
Figure 3-1: Hexahedral Master Element	52
Figure 3-2: Force Vector Computation from Void	57
Figure 3-3: Force Vector Computation from Material Domain	59
Figure 3-4: Dual LSF Structural Representation [50]	69
Figure 3-5: Example Level-Set Function	72

TABLE OF FIGURES (CONTINUED)

Figure 3-6: Approximate Dirac Function	72
Figure 3-7: Drastic Change in Constraint Violation	74
Figure 3-8: Volume Crosses Target then goes Unstable	75
Figure 3-9: Volume Fraction with PID-type Penalty	77
Figure 4-1: 2-D LSM Input Interface	80
Figure 4-2: 2-D Pressure Problem Definition.....	81
Figure 4-3: 3-D Pressure Problem Definition.....	82
Figure 4-4: STL Projection.....	84
Figure 4-5: YZ-Cross Section Projection.....	85
Figure 4-6: Node Relative Positioning	87
Figure 4-7: Fixed Boundary Conditions	95
Figure 4-8: Basic Flow Chart	96
Figure 5-1: Simply Supported Beam with Distributed Loading	111
Figure 5-2: Distributed Load Optimized Structure	111
Figure 5-3: Distributed Load Deflection Plot	111
Figure 5-4: Distributed Loading Compliance and Volume Fraction Plots.....	112
Figure 5-5: Cantilevered Beam Problem	113
Figure 5-6: RBF Initial Structure.....	113
Figure 5-7: RBF Initial Level-Set Function	114
Figure 5-8: RBF Structure at Iteration 60	115
Figure 5-9: RBF Level-Set Function at Iteration 60.....	115

TABLE OF FIGURES (CONTINUED)

Figure 5-10: RBF Final Structure.....	116
Figure 5-11: RBF Final Level-Set Function	116
Figure 5-12: RBF Compliance Versus Iteration.....	117
Figure 5-13: RBF Volume Fraction Versus Iteration	117
Figure 5-14: 3-D Cantilevered Beam Problem.....	118
Figure 5-15: 3-D Cantilevered Beam Deformed Structure	118
Figure 5-16: Iso-Density Identification During Early Iterations	119
Figure 5-17: Iso-Density Line Errors	120
Figure 5-18: 2-D Pressure Loaded Structure and Deformation.....	121
Figure 5-19: 2-D Pressure Loading Level-Set Functions	121
Figure 5-20: Pressure Box Problem Definition	123
Figure 5-21: Pressure Box 40x20x10 Starting Void and Deformation.....	123
Figure 5-22: Pressure Box 40x20x10 Iteration 75 Void and Deformation	124
Figure 5-23: Pressure Box 60x30x15 Volume and Compliance Versus Iteration	125
Figure 5-24: Pressure Box Intermediate Densities, Iterations 1 and 20.....	125
Figure 5-25: Pressure Box Intermediate Densities, Iterations 40 and 60.....	125
Figure 5-26: Pressure Box Intermediate Densities, Iteration 62	126
Figure 5-27: Pressure Box Intermediate Densities, Iteration 90	126
Figure 5-28: Pressure Box Multiple Starting Voids Compliance and Volume	127
Figure 5-29: One Starting Void Iterations 20 and 30.....	128
Figure 5-30: One Starting Void Iterations 40 and 75.....	128

TABLE OF FIGURES (CONTINUED)

Figure 5-31: One Starting Void Volume and Compliance.....	129
Figure 5-32: One Starting Void PID Terms.....	129
Figure 5-33: One Starting Void Penalty Term.....	129
Figure 5-34: One Starting Void Shape Sensitivity.....	130
Figure 5-35: Multiple Starting Voids Iterations 1 and 68	130
Figure 5-36: Multiple Starting Voids Volume and Compliance	131
Figure 5-37: Real Valued Pressure Box Volume and Compliance	132
Figure 6-1: MK-16 Back Cover Removed	133
Figure 6-2: Existing Pressure Vessel Dimensions	134
Figure 6-3: Existing Sphere Stresses(PSI) Pressure=5,000 PSI and 0.0625" Elements	136
Figure 6-4: Proposed Pressure Vessel Geometry for MK-16.....	137
Figure 6-5: Proposed Pressure Vessel Dimensions	138
Figure 6-6: Proposed Pressure Vessel in MK-16 Assembly	139
Figure 6-7: Mesh with 0.2" Element Size	140
Figure 6-8: Volume and Compliance, Target Volume of 0.45, Pressure Calculation from Void .	142
Figure 6-9: Stress Plot Iteration 76, View Window: X(0,6.75) Y(2,6) Z(0,12.25)	143
Figure 6-10: Top Rib, View Window: X(0,2.75) Y(2.75,5) Z(7,12.75)	144
Figure 6-11: Side Rib, View Window X(0,3.5) Y(3.5,7.25) Z(3.75,6.5)	145
Figure 6-12: Bottom Rib, View Window X(0,2.75) Y(2.75,5) Z(0,3.75)	145
Figure 6-13: Connection Beam, Trial 33, Iteration 25, View Window X(0,6.75) Y(1.5,3.75) Z(0,11.5).....	146

TABLE OF FIGURES (CONTINUED)

Figure 6-14: Design Points Volume Fraction Versus Compliance.....	148
Figure 6-15: Volume and Compliance, $K = 10.50.2$, $\lambda_o = 0.5$, and $\lambda PID = 0.5$	149
Figure 6-16: Volume and Compliance, $K = 0.50.21$, $\lambda_o = 0.5$, and $\lambda PID = 0.5$	150
Figure 6-17: Volume and Compliance, $K = 0.50.21$, $\lambda_o = 0.5$, and $\lambda PID = 1$	151
Figure 6-18: $V_{req}=0.45$ Final, Interior Front View, Window: X(0,6.75) Y(2,6) Z(0,12.25)	152
Figure 6-19: $V_{req}=0.45$ Final, Front View, Window: X(0,6.75) Y(2,6) Z(0,12.25)	153
Figure 6-20: Final Structure from Bottom, Z=8.7" Through Z=12.75"	154
Figure 6-21: Final Structure, Z=7.5" Through Z=8.7"	154
Figure 6-22 Final Structure, Z=5.4" Through Z=7.5"	154
Figure 6-23: Final Structure, Z=3.9" Through Z=5.4"	155
Figure 6-24: Final Structure, Z=0" Through Z=3.9"	155
Figure 6-25: Smoothed Geometry Bottom.....	156
Figure 6-26: Smoothed Geometry Middle	157
Figure 6-27: Smoothed Geometry Top.....	157
Figure 6-28: Remeshing Pareto Front	159
Figure 6-29: Volume Fraction and Compliance Designs from Trial #29	160

NOMENCLATURE

Symbol	Definition	Pages
LSM	Level-Set Method	20, 21, 22, 23, 25, 26, 29, 30, 39, 40, 41, 42, 44, 46, 47, 64, 66, 67, 71, 73, 76, 77, 78, 79, 80, 118, 122, 124, 127
LSF	Level-Set Function	20, 22, 25, 26, 29, 30, 31, 32, 33, 36, 39, 40, 41, 44, 45, 46, 47, 57, 63, 64, 65, 66, 67, 68, 69, 70, 72, 73, 75, 79, 80, 88, 92, 93, 94, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 109, 112, 113, 114, 122, 126, 132, 141, 149, 150, 163
FEA	Finite Element Analysis	47, 63, 71, 79, 104, 105, 107, 136, 149, 163, 167
SIMP	Solid Isotropic Material with Penalization	12, 17, 18, 24, 32, 41, 42, 44, 46, 63, 78, 80, 161
\tilde{A}_{ijkl}	Effective elasticity tensor	15, 16
A_{ijkl}	Elasticity tensor	15, 16, 37, 38, 45, 50
ρ	Density fraction	15, 16
P	Power penalty	15, 16
\tilde{x}_i	Pseudo density at i	18, 93
N_i	List of elements within element i's neighborhood	18, 93, 94
H_{ij}	Radial weighting factor from element i to element j	18, 93, 94
v_j	Element j's volume	18, 93
x_j	Design variable at j	18
r	Radius	18, 37, 94
ϕ	Level-Set Function	20, 21, 26, 36, 37, 38, 39, 41, 63, 64, 66, 93
D	Design Domain	20, 21, 35, 45, 46, 63, 69, 71, 81, 93, 111, 113
Ω	Material Domain	20, 21, 35, 36, 37, 48, 49, 51, 54, 55, 63, 64, 65, 68, 69, 70, 71, 93, 94, 95, 100
Γ	Material/Void interface	20, 21, 24, 35, 36, 37, 49, 51, 56, 58, 59, 63, 68, 71, 93, 95
$D \setminus \Omega$	Void Domain	20, 21, 63, 68, 93
c	Iso-contour plane	21, 30, 36
X	Location within design domain or a collective design state	20, 21, 26, 45, 46, 63, 68, 69, 71, 93
s_i	Design variable	26, 27, 28, 29, 33, 36, 37, 92, 93

Symbol	Definition	Pages
N_i	Shape Function	26, 36, 37, 53, 54, 55, 56
c_i	Centroid of basis function kernel	26, 64
IBT	Immersed Boundary Technique	31, 32, 165
X-FEM	eXtended Finite Element Method	31
$c(x)$	Design's compliance	34, 35, 61, 62, 64, 65, 67, 69, 99
$\{U\}$	Global deformation vector	34, 35, 60, 61, 62, 67, 99
$[K]$	Global stiffness matrix	34, 35, 56, 60, 62, 67, 99
$\{F\}$	Global force vector	34, 35, 46, 59, 60, 62, 67, 113
$\{u_e\}$	Elemental deformation vector	34, 35, 62, 65, 67, 69, 77, 99, 100
$\{k_e\}$	Elemental stiffness matrix	34, 35, 56, 62, 65, 67, 69, 77, 99, 100
N	Total number of elements	34, 35, 62, 67, 99, 100
$V(x)$	Design's volume	34, 35, 37, 61, 62, 64, 65, 67, 70, 74, 76, 77, 100, 101, 168
V_{req}	Required volume goal	34, 35, 61, 62, 64, 65, 67, 70, 74, 76, 77, 100, 101, 152, 153, 168
u_o	Prescribed displacements	34, 35, 61, 62, 67
Γ_D	Dirichlet boundary	35, 45, 59, 61, 62, 67
Γ_N	Neumann boundary	35, 45, 46, 58, 61, 62, 67, 68, 69, 71
Γ_H	Homogeneous boundary	35, 45, 58, 61, 62, 67, 68
R	Arbitrary response	35, 36, 37, 38, 39, 93, 94
$\int dS$	Integral along path	36, 37, 46, 56, 71
$\delta\Omega_n$	Boundary variation in normal direction	35, 36, 41
\mathbf{n}	Normal of material boundary	35, 36, 45, 46, 49, 67, 71
$B(r)$	Hole with radius r	37
λ	Lamé's 1 st parameter	37, 38, 45
μ	Shear modulus	37, 38
$e(u)$	Strain tensor	37, 38, 45
SQP	Sequential Quadratic Programming	38
MMA	Method of Moving Asymptotes	38
CONLIN	CONvex LINearization approximations	38
τ	Pseudo time interval	39, 64, 66
v	LSF design change velocity	38, 39, 45, 64, 65, 66, 70, 77, 100, 103
CFL	Courant-Friedrichs-Lewy condition	39, 66, 88, 103, 150
h	LSF grid spacing	39, 66, 71

Symbol	Definition	Pages
ψ	Pressure LSF	45, 46, 67, 68, 69, 70, 71
ϕ	Free boundary LSF	45, 67, 68, 69, 70
p	Pressure	45, 46
ε	Positive constant for Dirac approximation function	46, 71, 72
\mathbf{u}	Displacement field	38, 47, 48, 49, 61
u_i	Displacement in the i^{th} direction	48, 49, 50, 51, 53, 55, 56, 71
δu	Variational displacement field	48, 49, 55, 56, 60, 71
σ_{ij}	State of stress	35, 48, 49, 50, 51, 62, 66
τ_{ij}	Shear Stress	48, 49, 51
b_i	Body force in the i^{th} direction	48, 49, 51
ε_{ij}	State of strain	50, 51, 55
γ_{ij}	Engineering shear strain	50, 51, 55
$[C]$	Constitutive matrix	50, 51, 55, 56
E	Modulus of elasticity	50, 51
ν	Poisson's ratio	50, 51, 111, 112
t_i	Traction force	49, 51, 56, 62, 71
ξ	'Xi' Relative x direction for master element	52, 53, 54, 55, 56, 58, 59, 90
η	'Eta' Relative y direction for master element	52, 53, 54, 55, 56, 58, 59, 90
ζ	'Zeta' Relative z direction for master element	52, 53, 54, 55, 56, 58, 59, 90
e	Element number	54, 56, 58, 59, 62, 64, 65, 66, 67, 69, 70, 93, 94, 100
a	Node number	53, 54, 55, 57
$[J]$	Jacobian matrix	54, 55, 56
$\{d\}$	Displacement vector	55, 56, 60, 71
$[B]$	Arranged matrix of partial derivatives of shape functions	55, 56
f_i^a	Force at the a^{th} node in the i^{th} direction	56, 57, 58, 59, 111
l_i	Element length in the i^{th} direction	58, 59, 83, 91
L	Lagrangian function	46, 58, 59, 67, 69, 70, 71, 81
λ_i	Lagrange multiplier for the i^{th} iteration	62, 64, 65, 70, 100
α	Lagrange scaling factor	62, 64, 65, 70, 73, 74, 77, 88, 100, 101, 128, 141, 149, 150, 151, 168

Symbol	Definition	Pages
γ	Lagrangian multiplication factor	65, 73, 101, 128, 141, 149
K_P	Proportional gain	74
K_I	Integral gain	76, 77, 88, 101, 102, 128, 141, 149, 150, 151, 168
K_D	Derivative gain	76, 77, 88, 101, 102, 128, 141, 149, 150, 151, 168

CHAPTER I: INTRODUCTION

Since the 1950's [1], computational analysis has been used by engineers to aid in the design process and provide rapid simulation results to support and substitute expensive and time consuming experimental results. Originally, computational analysis tools were primarily used for design confirmation to provide preliminary results before committing to testing, in efforts to limit overall testing time and budget. However, as the capabilities of computational analysis increased, so too did its influence on the design process. Combined with mathematical concepts in optimization, these analysis tools were quickly incorporated into the initial design and component generation phases of the engineering process as the field of computer-aided optimization emerged. Later, the evolution into topology optimization [2] has provided a powerful design tool for determining optimal material distribution for a given domain, conditions and objectives. This allows for structural configurations to be determined as opposed to size optimization determining a finite set of geometric design parameters. Increases in manufacturing capabilities, such as additive manufacturing, have given a practical use for these obscure structural geometries generated by topology optimization, increasing its popularity and usefulness. This in turn led to a growth in popularity and accessibility evident by many Computer-Aided Design (CAD) and computational analysis software tools now providing packages that allow engineers to implement topology optimization. These well-established topology optimization methods require a user to define a design domain with locked, unchangeable features along with static loading conditions. However, in many situations, a component experiences design dependent loading conditions which cause the boundary conditions of the analysis to vary with the material distribution, for example pressure loading. When a component is subjected to pressure loading, the resultant force is exerted in the surface

normal direction with a magnitude proportional to the surface area. Therefore, the locations as well as magnitudes of loading change as the material distribution changes. This thesis explores various ways to account for such loading conditions in topology optimization and provides a method to do so for 3 dimensional domains.

1.1 Motivation

One occurrence of pressure loading is in pressure vessels which act as a storage device to isolate gas or liquid mediums at a differential pressure from its surroundings. Due to its manufacturability and strength in symmetry, the majority of pressure vessels are round or spherical. Pressurized gas storage is common among life support systems to house a supply of breathing gasses to a single user or a group of users in a hostile environment. These systems are customary in the realms of marine diving, aerospace, fire & rescue, and mineral mining. The duration these devices can be used is heavily dependent upon the gas supply quantity. Therefore, it would be extremely advantageous to increase the carrying capacity of a pressure vessel. Evident from the ideal gas law, there are only two ways to accomplish this goal: increase storage volume, or increase storage pressure. Breathing gas pressure vessels store gasses at high pressures, typically ranging from 3,000 to 5,000 PSI [3]. Although research has been done to utilize composite materials to construct pressure vessels capable of holding 10,000-15,000 PSI [4], little research has been done to examine variations in size and shape because solid mechanics provides well-established formulations for hoop and longitudinal stresses in both cylindrical and spherical pressure vessels, the predominant shapes used.

In 2017, diving and life support engineers at the Naval Surface Warfare Center-Panama City Division (NSWC-PCD), introduced a proposal to utilize additive manufacturing to construct

uniquely shaped pressure vessels [5]. Additive manufacturing enables the incorporation of internal supporting features as well as varying wall thicknesses that would be required in an irregular shaped pressure vessel. This development would allow engineers to design gas storage around the available space of a system's geometric constraints.

Because of the recycling of breathing gasses, increased gas capacity has an even more drastic impact on duration when dealing with rebreather systems. One such device heavily used by the US NAVY is the MK-16 Closed Circuit Mixed Gas Rebreather, figure 1-1. This rebreather is worn like a backpack where the face shown in the left image faces the diver's back.

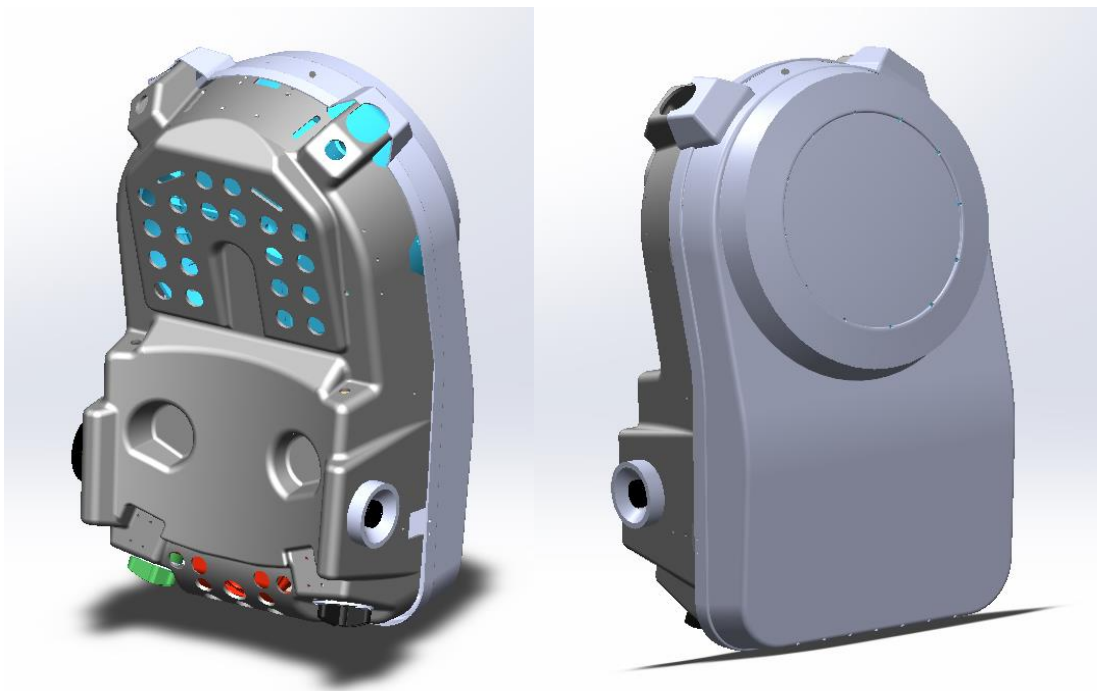


Figure 1-1: MK-16 Rebreather Front (left), back (right)

To provide a real-world example for the design of an irregular shaped pressure vessel, the MK-16 rebreather system, figure 1-1, is used to determine the effectiveness of such a development. Within the housing of the MK-16 backpack, there are four main components: the scrubber, the

diluent tank, the oxygen tank, and the Primary Electronics Assembly (PEA), which are each labeled in figure 1-2. The scrubber houses calcium hydroxide that chemically reacts with exhaled CO_2 to allow for a portion of the exhaled breath to be inhaled. To account for the loss of gas in the breathing loop, or the increased pressure with depth, the PEA determines the appropriate amount of Diluent and Oxygen to add to the breathing loop from their respective storage tanks depending upon the partial pressure of oxygen in the system. Because the diver consumes oxygen based on their work rate, and metabolically requires a specific range of pO_2 , the fraction of O_2 in the breathing loop varies. Thus, the use of both an oxygen tank and a diluent tank is required. [3]

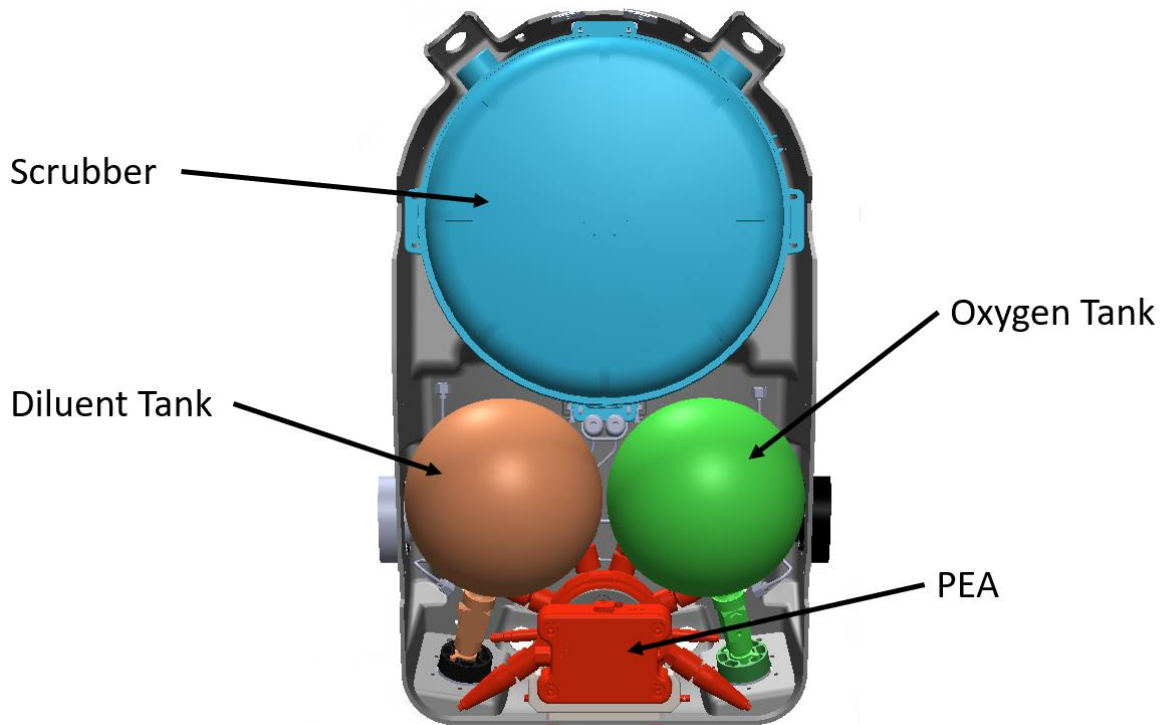


Figure 1-2: MK-16 Rebreather Components

As seen in figure 1-2, there are large regions of unused space around the spherical pressure vessels. Additionally, the internal components of the rebreather are symmetric along the center line. Therefore, if an irregular pressure vessel was designed to replace one of the spherical pressure vessels, as long as it does not cross the centerline, it can be mirrored to replace the other storage tank. Below figure 1-3 shows the dimensions of the spherical oxygen tank and table 1-1 presents some of the important data for this existing pressure vessel that will be needed to compare results of the designed irregular pressure vessel.

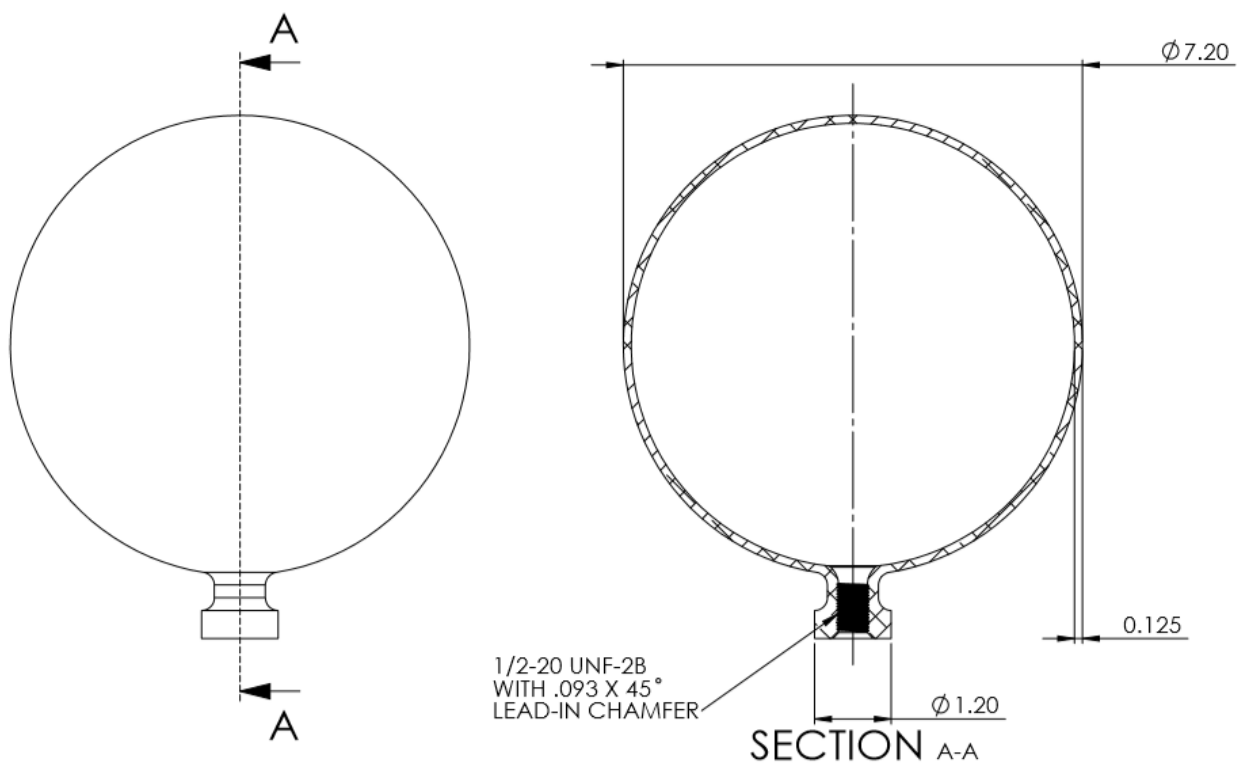


Figure 1-3: Oxygen Tank

Table 1-1: Oxygen Tank Properties

Property	Value
Wet Volume	175±10 in. ³
Outer Diameter	7.2 in.
Working Pressure	3,000 PSI
Material	Inconel 718

With this information, an effective irregular shaped pressure vessel would be one that fits within the geometric constraints of the system provided by the MK-16 rebreather and supports a working pressure of 3,000 PSI while holding a wet volume of at least 175 in³.

1.2 Research Objectives

This thesis focuses on topology optimization with design dependent pressure loading in 3-dimensional space by addressing the following research questions:

RQ1. Can the interior geometry of an irregular shaped pressure vessel, subjected to internal pressure on its surfaces, be designed to efficiently store high pressure gas using topology optimization methods?

A hypothesis is that yes, topology optimization can be used to design the internal structure of such an irregular shaped tank, that could then be manufactured using additive manufacturing.

To solve this research question, a second research question can be identified:

RQ2. Can an efficient method be developed to track (follow) design dependent pressure loading conditions on the interior surface for 3-dimensional spaces for use in a topology optimization algorithm?

A hypothesis is that by adapting a level-set topology optimization approach, it is possible to track changing pressure surfaces as the design evolves during the iterative design process.

1.3 Thesis Outline

With the motivation and objectives introduced, the remainder of this thesis is broken into 5 chapters. Chapter 2 reviews current literature regarding topology optimization methods, their origins as well as possible methods of incorporating design dependent loading. Chapter 3 breaks down the mathematical methodology used to achieve the research objectives. Chapter 4 discusses how this established method was executed in MATLAB. Chapter 5 presents intermediate results that progress the problem from basic topology optimization problems to a simplified pressure vessel problem, then Chapter 6 presents the results from executing these established methods on the real-world design problem involving the MK-16 rebreather. Finally, Chapter 6 concludes the work that was done for this thesis and presents future work to expand upon.

CHAPTER II: LITERATURE REVIEW

In an effort to develop the best approach to solve the design problem and accomplish the research objectives, a review of existing methods and their origin was conducted. This chapter is organized as follows: section 2.1 overviews topology optimization methods and their origins, section 2.2 dives further into the formulation of the level-set method and finally section 2.3 addresses the incorporation of design dependent loading into topology optimization methods.

2.1 Topology Optimization History and Overview

A major limiting factor to an engineer designing a particular component is the manufacturing techniques available and their associated cost. However, with recent advances in manufacturing techniques, notably additive manufacturing, the engineer can be given more design freedom allowing for increasing complexity in components. Naturally, this increased complexity should be justified by serving some benefit and aid the engineer to improve a system's performance. For this, optimization methods have proved to be useful tools to systematically aid engineers in achieving a design that maximizes or minimizes (whichever is desired) the design's performance based on specified criteria. Due to many optimization processes' iterative nature and complex performance criteria, these optimization methods have been coupled with computational analysis techniques into a field known as computer-aided optimization. These computational techniques originally served the purpose of validating and analyzing designs, but, when tied to an optimization algorithm, they form a powerful design improvement and generation tool. Shortly following the establishment of finite element methods by Turner et al. in 1955 [1], Lucien Schmit recognized the potential of coupling

optimization methods with finite-element analysis for structural design in the 1960's [6]. Since then, researchers have developed and refined various methods of executing computer-aided optimization allowing for the development of efficient material distribution directly benefiting the designer's objectives for the component or system.

Computer-aided structural optimization has branched into numerous methods but can be distinguished by two root groups: first being shape and size optimization and the second being topology optimization [2]. Shape and size optimization focus on varying a relatively small number of parameters, such as dimensions or cross-sectional shape, of a design. Thus, shape and size optimization are typically fast and efficient at refining a design to improve its performance, but require an initial close-to-optimal design. Conversely, topology optimization is defined as a computational material distribution method for synthesizing structures without preconceived shape to optimally perform a specific task [7]. This offers innovative and high-performance structures however with increased computational cost and design complexity. Topology optimization itself can be broken into 3 major categories: ground structure [8], homogenization methods [9] and level-set methods [10]. Each of these main categories differ in how they define the structure and thus their assignment of optimization parameters.

As in any field of study, the development of methods to execute topology optimization is spurred by a desire to overcome existing obstacles. In the field of topology optimization, there are several recurring obstacles that constantly are addressed and form the root cause for each of these major categories of optimization to have been developed. Computational limitations have always been an issue but can be mitigated via simplifications, approximations, and creative use of resources. Although, this problem may always exist with the continuing pursuit of higher

accuracy, increased analysis complexity and larger domain sizes, technological advances have seriously aided the ability to push capabilities and allow for the use and development of methods previously thought impractical or even impossible. Other more pressing and challenging hurdles specific to the field include chattering, checkerboarding, mesh dependencies and initial conditions [11]–[13]. Chattering is the result of a large number of regions of a domain flipping back and forth between having material and not among successive iterations of the optimization procedure. This causes oscillating performances, lack of convergence and stalling of the algorithm. Checkerboarding occurs when a large region of the domain contains a patterned occurrence of material and void regions causing the result to become improper and not practical for manufacturing. The existence of these problems occurs from ill-posed problem formulation and implementation. Additionally, many of these algorithms seek to achieve consistent results regardless of starting points and domain meshing. These can be particularly challenging due to the nature of many gradient based optimization algorithms converging to local optima. Heuristic algorithms known for better achieving global minima and not stalling at local minima prove to be inefficient and impractical to use due to the number of design variables and the computational cost of objective analysis. However, with advances in technologies, there have been several uses of these optimization algorithms such as simulated annealing, and genetic algorithms [14]–[17]. On one hand, mesh dependency to some degree will always influence an optimized part's topology as it is known that analysis accuracy is strongly influenced by component meshing. However, at a certain point, there is a diminishing return on accuracy versus mesh refinement, and at this point, topology algorithms seek to mitigate the effect of a mesh on their final results. To counteract all these common issues in

topology optimization, researchers have developed numerous creative means of implementing filtering, penalization, and regularization techniques within algorithms.

2.1.1 Ground Structure Approach

In a ground structure approach to topology optimization, the domain is divided into nodes, then each node is connected to all possible other nodes like a truss structure. That is, all node to node connections that do not directly overlay another node on their path. From here, the optimization algorithm determines which of these trusses are to stay and which are not needed [8], [18]. This has been done both binarily (i.e. on or off) or with continuous variables that represent the cross-section of each member. Figure 2-1 below illustrates this concept for a cantilevered beam, the left depicts the initial setup with all node to node connections being made, then the right shows a later iteration after the optimization algorithm has removed some of the trusses.

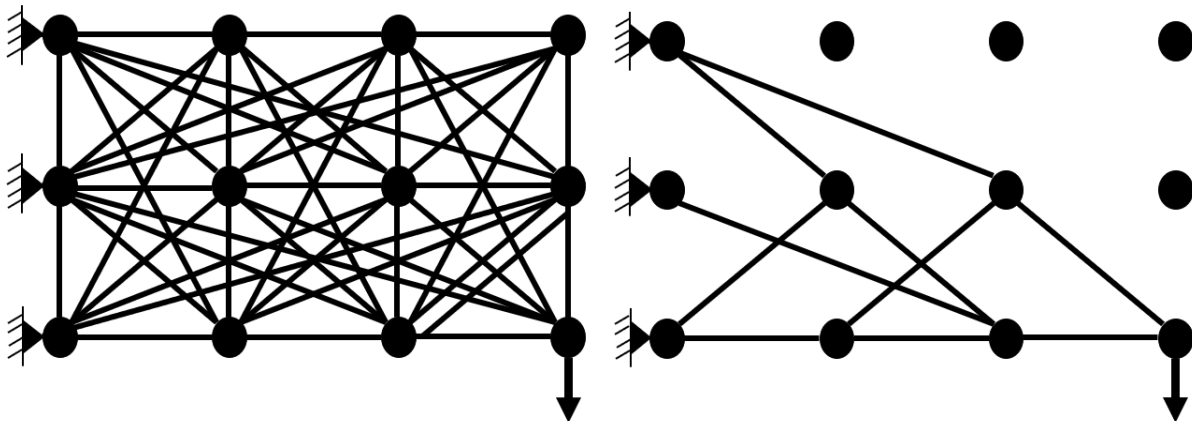


Figure 2-1: Ground Structure Approach Illustration

2.1.2 Homogenization Methods

Introduced in the early 1980's [19], homogenization theory parameterizes the geometry of microstructures within a macrostructure of interest. Recently this method has proven ideal for dealing with composites, lattice structures and any micro-structured materials where anisotropy comes into play [20]. However, homogenization methods were originally developed for periodic structures. They were quickly adapted for the objective of optimizing generic material distribution problems as an alternate to existing ground structure approaches. Upon its conception, the homogenization method did not prove extremely effective or practical due to the need to define and analyze geometry on a microstructure scale [20], [21]. This was the case until its oversimplification into density-based topology optimization which parameterizes the microstructure based solely on density [22]. This density is then directly correlated to the material's modulus of elasticity. The issue with these density-based optimization methods laid in the ill-posed nature of the optimization problem which was overcome by the revolutionizing paper by Bendsoe [23] as the popular *Solid Isotropic Material with Penalization* (SIMP) method began to be formulated. SIMP has since grown to be the most popular form of topology optimization due to its simplistic implementation and ability to generate complex geometries. Due to advances in additive manufacturing's ability to create finely graded microstructures, a resurgence of conventional homogenization methods has occurred as it now has more practical applications [20].

The design of the topology of a structure of interest consists in determining the optimal placement of material (locations of material and locations of void) within a domain of interest. This can be formulated into the 0-1 problem by being interpreted as, at a given spatial location, should there be material or not. This 0-1 problem formulation is the root problem statement of

all topology optimization formulations; however, this problem statement has some drawbacks.

The largest drawbacks of these on-off natured problems is the lack of existence of a solution that satisfies optimality conditions and the results are sensitive to mesh discretization [24].

Researchers recognized that a solution to this problem was the consideration of a heterogeneous material allowing for the use of porous regions at the microscale. This effectively converts the on-off nature of the problem to a continuous design variable problem [21]. These micro-level porous regions are characterized by a chosen class of unit cells, each being defined by an appropriate number of design variables used to describe its specific geometry. Figure 2-2 below depicts typical classes of unit cells used in homogenization methods including square with square holes, square with rectangular holes and rank-2 layered material.

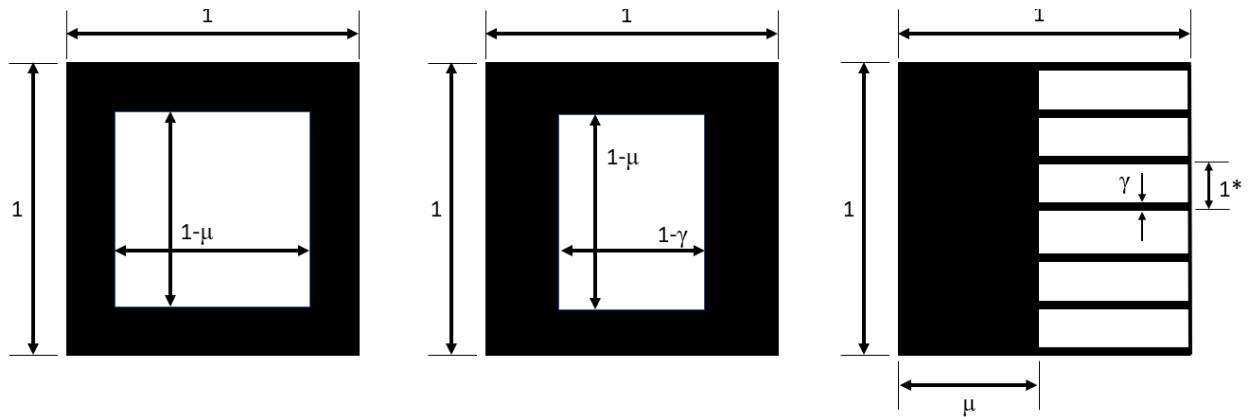


Figure 2-2: Typical Classes of Unit Cells. Left: Square with Square Hole,

Middle: Square with Rectangular Hole, Right: Rank-2 Layered Material

It should be noted that, for analysis purposes, these unit cells are evaluated as if they are infinitely small, but also infinitely many, and thus the microstructures alter the effective material properties of that region based on micromechanics of their geometry and defined parameters [24]. This allows for a correlation between parameters of the microstructure and the macro effective material properties to be formulated. From the figure above, the right

image depicts a rank-2 unit cell. This is defined as a rank-2 classification due to the usage of two scales where the overall unit cell and solid left portion is on one scale and the thickness of the flanges to the right are of another scale. The incorporation of microstructures allows designs to possess intermediate values for material properties allowing for a continuous gradient of performance as parameters change, as opposed to the discontinuous nature of an on-off problem formulation. However, at the final solution of an optimization process, the user typically wishes to have a design of exclusively solid or void regions for manufacturability purposes. Naturally, topology optimization problems are either subjected to a volume fraction constraint or have an objective to limit the volume fraction, both of which result in the seeking of the most efficient use of material. In the homogenization method, the use of microstructures and their effective material properties derived from micromechanics inherently results in intermediate regions between void and solid with microscopic inclusions having less than proportional rigidity [24], [25]. During the optimization process, large regions with porous microstructures could achieve a more efficient use of material distribution by evolving to the necessary subregions being completely solid and the others, completely void. Therefore, it is expected that the homogenization method will result in a solution with the majority of elements completely solid or completely void.

The following process outlines the typical flow of the implementation of the homogenization method for topology optimization. First, the class, or classes, of microstructures to be used must be chosen and then effective material properties can be calculated by forming a functional relation to microlevel design variables. Next, the problem must be formulated by defining the desired objective criteria and constraints, as well as the reference domain, loading

conditions and boundary conditions. Once all of this is established, optimization of the geometry may commence. In this iterative optimization procedure, analysis is run for the current design, the objective is computed, convergence is checked, and design variables are updated before returning to the start of this loop for the subsequent iteration. Finally, once the optimization process converges, post-processing can be done to interpret and evaluate the results [24].

In the late 1980's, Bendsoe explained how to implement a partial relaxation of these methods by restricting the homogenization method to a subclass of microstructures [26]. In this paper, Bendsoe still defines material distribution based on artificial composite material with microstructures just as the original homogenization method does; however, this paper opens the door to simplifications and modifications to the homogenization method to increase its practicality and ease of implementation. Shortly following this progression, researchers realized that, if the type of unit cell microstructure was limited to only one, the microstructures could be parameterized solely based on density as opposed to unit cell relative dimension parameters. The following year, Bendsoe published another paper [23] to further simplify the homogenization method. In this paper, Bendsoe proposes a means of directly relating intermediate density values to an effective modulus of elasticity for analysis via a power law as shown in the equation below. Where \tilde{A}_{ijkl} and A_{ijkl} are the effective elasticity tensor and original solid material's elasticity tensor respectively, while ρ is the density fraction and P the power penalty,

$$\tilde{A}_{ijkl} = A_{ijkl}\rho^P \quad (2.1)$$

with typical power penalties lying between 2 and 7 [11]. To eliminate numerical artifacts and ensure the analysis is well conditioned for optimization, a completely void element is modeled as a very weak (orders of magnitude less) compliant material as opposed to having a modulus of elasticity of 0. This modifies the previous equation into the one shown below. Where A_{min} typically equals something along the lines of $A_{ijkl} * 10^{-9}$.

$$\tilde{A}_{ijkl} = (A_{ijkl} - A_{min})\rho^P + A_{min} \quad (2.2)$$

This expression to model an effective modulus of elasticity implicitly penalizes intermediate densities as it assigns less than proportional rigidity compared to material use. It has the same effect as original homogenization methods by allowing for a continuous function for structural rigidity while forcing converged solutions to possess mainly completely solid or completely void elements [22]. This relation of material cost to structural stiffness can be seen in the figure below where it is clear that intermediate values of density (not 0 or 1) will result in an inefficient use of structural rigidity.

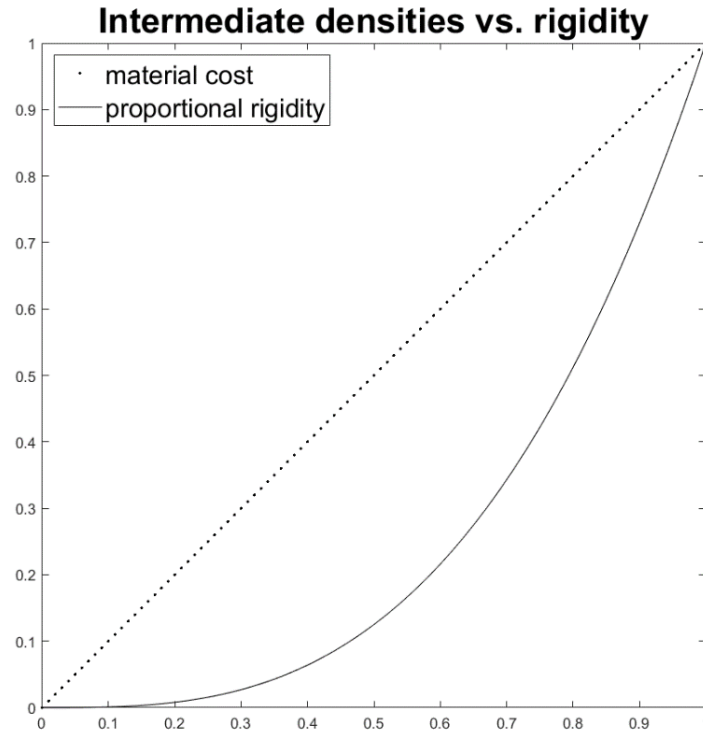


Figure 2-3: Intermediate Density Versus Rigidity

Using this expression for material properties, Bendsoe’s paper [23] validates the concept by comparing these optimization results to those of classical homogenization methods using composites with voids. Simultaneously, Rozvany formulated and tweaked this concept to eliminate and mitigate other undesirable kinks common to all existing topology optimization methods. In his works [27], [28], Rozvany established the ‘*Solid Isotropic Material with Penalization*’ (SIMP) method for topology optimization. Since then, the SIMP method has grown to become the most common, robust, and utilized means of topology optimization. Similar to Bendsoe, Rozvany used intermediate densities to represent porous material modeled via a power law, but Rozvany added regularization techniques formalizing the method.

Although the use of intermediate densities aids the algorithm's characteristics, it remains ill-posed with common problems including checkerboarding and stalling in local minima. Since its inception by Rozvany, a large focus by researchers has been to generate effective regularization techniques. A common approach is the use of a density filter where the optimization variables are no longer directly the density values used in the power law for analysis but instead, a pseudo density is calculated based on the surrounding optimization variables [7]. These density filters take a radially weighted average of density values in a local neighborhood of elements. This type of filter can be implemented by employing the following equation.

$$\tilde{x}_i = \frac{\sum_{j \in N_i} H_{ij} v_j x_j}{\sum_{j \in N_i} H_{ij} v_j} \quad (2.3)$$

Where \tilde{x}_i and x_j are the pseudo densities used in analysis and the optimization design variables respectively, while v_j is the given element's volume. Additionally, N_i identifies the element's neighborhood of other elements and H_{ij} the radial weighting factor of each of those elements. Both can be defined as:

$$N_i = \{j: \text{dist}(i, j) \leq r\} \quad (2.4)$$

$$H_{ij} = r - \text{dist}(i, j) \quad (2.5)$$

Despite being the most common form of topology optimization, the homogenization and SIMP methods may have several drawbacks depending on the specified problem. Since these methods utilize a fictitious intermediate design state throughout the domain during the optimization process, they can make it difficult to identify boundaries. The identification of

boundaries may be important for applications such as geometry control, pressure loading, and component interactions. For these reasons, researchers pursued other methods of topology optimization to overcome these drawbacks for a given application.

2.1.3 Level-Set Methods

The third major category of topology optimization methods is that of level-set methods. In 1988 mathematicians Stanley Osher and James Sethian published a paper [29] proposing a new method to tackle problems of moving boundaries and fronts implicitly. These types of problems were commonly found in the fields of fluid dynamics, computational geometry, and image processing. Prior to this development, many methods for boundary problems proved complex and computationally expensive. They typically involved a Taylor Series formulation or assigned a large number of points along a boundary, moved each point based on a velocity field, and then formed the moved boundary as the spline connecting each of the points' new coordinates [30], [31]. This method proved cumbersome, particularly when boundaries expanded or shrunk, as this would result in the linear distance between defined points either separating and reducing accuracy, or converging, causing computational inefficiencies. This was typically resolved by redefining evenly spaced points along the boundary prior to the following iteration, adding additional computational burden. However, the largest issue was in the event of sharp corners, particularly when the front is moving inwards upon itself [29]. This can be seen in figure 2-4 below where the consecutive points cross and result in a discontinuous or undesired geometrical representation of the boundary.

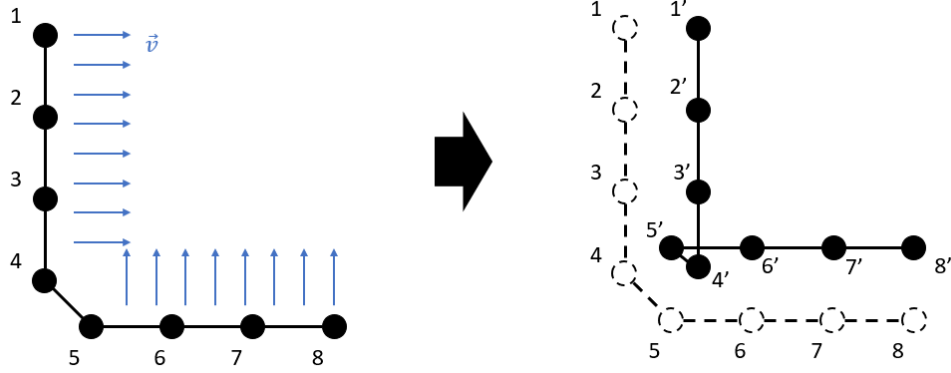
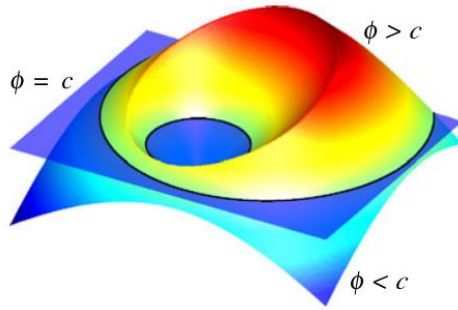


Figure 2-4: Converging Corners in Moving Boundaries

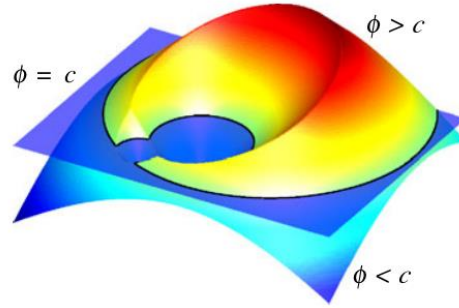
The method proposed by Osher and Sethian avoids this problem by implicitly defining the boundary as opposed to a series of coordinate points splined together. Osher and Sethian termed their method as ‘Propagation of Surfaces under Curvature’ (PSC) in which a scheme was generated to follow an $N-1$ dimensional surface in a N dimensional space via a fixed Eulerian framework [29]. With this, the front no longer needs to be defined as a function nor a series of points. This method formulation evolved into what is known today as the Level-Set Method (LSM); in which a boundary, Γ , in \mathbb{R}^n space is defined as an iso-contour of an evolving function, $\varphi(X)$, in \mathbb{R}^{n+1} space known as the ‘Level-Set Function’ (LSF), where X is the \mathbb{R}^n spatial coordinate [32]. In many cases, such as topology optimization, this boundary delineates the interface between two regions such as those containing material and those being void. To identify two regions using a LSF, one is defined as the region of the function above the iso-contour and the other, the areas below. For shape optimization, the regions of interest in the design domain (D) are the material domain (Ω), void domain (D/Ω), and the interface of the two (Γ) [33]. Mathematically, the relation of these regions to the LSF is represented in equation

2.6 below and visually illustrated in figure 2-5. Note the iso-contour level $\varphi = c$ is held constant throughout the entire optimization process and typically taken as $c = 0$.

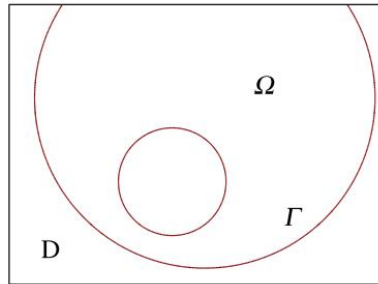
$$\left\{ \begin{array}{lll} \varphi(X) > c & X \in \Omega & \text{'Material'} \\ \varphi(X) = c & X \in \Gamma & \text{'Interface'} \\ \varphi(X) < c & X \in (D \setminus \Omega) & \text{'Void'} \end{array} \right\} \quad (2.6)$$



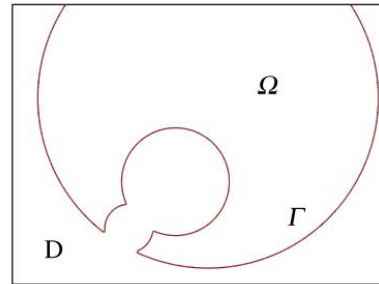
(a) An example of an LSF ϕ .



(b) An updated LSF ϕ .



(c) The material domain Ω corresponding to Fig. 1a.



(d) The updated material domain Ω corresponding to Fig. 1b.

Figure 2-5: Level-Set Method Visualization [34]

The use of level-set functions to define boundaries and regions has expanded to a wide variety of mathematical and engineering problems including fluids, thermal, electro-mechanical and electro-magnetic due to its inherent advantages in crisp boundary descriptions [34]. For these same reasons, in 1998 the LSM was suggested to be used in topology optimization as well [35]. Shortly thereafter, in 2000 two formulations of level-set based topology optimization were

published. Sethian and Wiegmann [36] developed their method using a finite difference mechanical model with evolutionary stress criterion to adjust their LSF throughout the optimization process. And the other paper by De Ruiter and Van Keulen [17], incorporated radial bases functions into their LSF formulation and utilized a genetic algorithm for their optimization. Noticing LSM's close resemblance to shape optimization, Osher and Santosa [37] and Allaire [38] established a shape-sensitivity based framework that has become the most popular approach of a level-set based topology optimization formulation today.

Frequently, the update procedure and evolution of the LSF is done by propagating the front through a pseudo time dependent PDE known as the Hamilton-Jacobi equation. To do this, sensitivities are converted to a velocity field which is applied to the current level-set function to determine the subsequent iteration's level-set function [34], [39] location. Fundamentally, this concept of LSMs only evolves boundaries, they may merge or split, but the method in this form does not allow for the nucleation of new holes, making LSMs merely shape optimization problems. This was the case until [40] where Allaire incorporated the use of topological derivatives via a reaction term to the Hamilton Jacobi Equation allowing for the nucleation of holes throughout the middle of the domain. The update procedure of a level-set based topology optimization algorithm is further explained in section 2.2.3.

As is the case for the homogenization and other topology optimization methods, several intrinsic issues arise when implementing the LSM for topology optimization. These result in the formulation being ill-posed, in the emergence of numerical artifacts, and in improper convergence behavior. To combat this, as in the case with other methods, the LSM requires regularization techniques. Numerous methods of implementing regularization have been

implemented and experimented on all phases and aspects of the LSM formulation for topology optimization and are further discussed in section 2.2. Limited research has been done, but regularization techniques have also been looked at as a means to allow the LSM to account for length-scale control and manufacturing constraints [41], [42].

2.1.4 Topology Optimization Conclusion

Since its inception as rudimentary shape and size design optimization, advances in analysis techniques and computational capabilities have spurred a rapid expansive research field for computer aided design. As these optimization methods were formulated and grew in capability, not only was an optimal size and shape able to be determined, entire structural formulation and part creation was possible as topologies of optimal designs could change [2]. To do so, researchers had to develop regularization methods to overcome the numerical issues related to the initial algorithm formulations which resulted in ill-posed problems, numerical artifacts, mesh dependencies and poor convergence behavior. The resulting topology optimization procedures enabled complex geometrical part creation. Although idealized as optimal, these parts still needed to be manufacturable. This spurred geometrical constraints as well as taking advantage of advances in manufacturing such as additive manufacturing [20], [41].

Over the years, three main branches of topology optimization for structural members have distinguished themselves: ground structure approaches [8], [18], homogenization methods [20], [21], [24], and level-set methods [33], [34], [38], [39]. Ground structure approaches offer the least computationally demanding option with their simple structural representation as trusses connecting nodes. However, this method lacks geometric control compared to its

counterparts. The second approach, the homogenization method, modifies a domain at the microstructure level to allow for continuous values and performance behavior, aided by the well-posed nature of the method. The homogenization method was simplified into a density-based approach by Rozvany [22] as it evolved into the most popular form of topology optimization used and known to date, SIMP. The final category of topology optimization is the Level-Set method which implicitly defines boundaries via a Level-Set function of a higher order domain. This allows for crisp boundary representations which can be advantageous depending upon the nature of the problem at hand. Due to its continuing progression, topology optimization has grown as a viable design tool that takes advantage of computational analysis capabilities as well as manufacturing techniques advances.

It should be noted that each of these methods is developed for applications with constant loading conditions and boundary conditions with respect to the reference domain. Therefore, each would need to be subjected to modifications to be capable of handling situations in which these boundary conditions are changing. For example, in the case of a pressure loading situation, although the magnitude of the pressure may not change and would always be applied perpendicular to the boundary, Γ , the location of this boundary may be unknown and part of the optimization problem. Thus, the nodal force magnitudes and directions will change with every design change by the topology optimization.

2.2 Level-Set Methods Formulation

The nature of topology optimization with pressure loading requires evolving or design dependent loading conditions as opposed to standard constant loading conditions. That is, at every iteration of the optimization process, the locations, directions and magnitudes of the

forces are subject to change based on the current geometric configuration. Due to the ability of the level-set method to provide crisp material boundaries throughout the optimization process and the nature of the current research objectives, a deeper investigation of this method was taken. The following section breaks down the process of implementing a Level Set Method (LSM) for topology optimization and various methods of executing regularization techniques to ensure the algorithm performs as desired.

Implementing a LSM for structural optimization simply just refers to the means of using a Level Set Function (LSF) to define the material/void boundary and distinguish regions of material within the domain. This leaves room for a great deal of flexibility in formulating a complete LSM for topology optimization. Regardless of this formulation, any LSM will be comprised of three major components namely the parameterization of the level-set function, the mechanical model, and the optimization procedure [34]. Each of these three tasks can be accomplished by a variety of means that each influence the performance, speed and effectiveness of the algorithm. It is up to the designer's choice as to how these components are carried out. The following subsections describe further in-depth the methods found in the research that have been used to carry out each of these three components along with the pros and cons of each decision. As stated before, in its base form, any topology optimization method may be ill-posed, contain numerical artifacts and possibly have poor convergence behavior. In order to induce desired results from the formulated algorithm, regularization techniques must be implemented. These regularization techniques can be implemented across all three of these major components.

2.2.1 Level-Set Function Parameterization

The first major component the designer must choose to be able to implement a LSM is determining how the LSF is defined. The LSF defines the material distribution and boundary locations. Therefore, the parameters that define the LSF, \mathbf{s} , become the optimization variables [10]. Thus, the LSF needs to be parameterized in such a way that the update procedure can use design sensitivities to modify the parameters of the LSF resulting in appropriate geometry changes. Typically, this parameterization is done by discretizing the domain and inserting an array of basis kernel functions each subject to a coefficient (design variable) controlling their magnitude. This discretization can position the kernels coincident or independent of analysis node points. The kernel functions are each a function of the spatial distance from the kernel's centroid. The LSF itself is then computed as the summation of these kernels multiplied by their coefficient. This is represented in the equation below where $\varphi(X, \mathbf{s})$ is the LSF value at spatial position ' X ' and current design variables ' \mathbf{s} ' and N_i refer to the particular kernel centered at position ' c_i ' and its associated coefficient ' s_i '. Researchers have used a variety of basis functions for LSMs including bilinear [33], [38], radial [43], [44] and spectral [45] basis functions due to their varying attributes in efforts to improve performance given the specific optimization task.

$$\varphi(X, \vec{s}) = \sum_i N_i(\|X - c_i\|)s_i \quad (2.7)$$

For both linear basis functions and radial basis functions, the kernel equals 1 at its centroid position and goes to zero away from this location. Therefore, when inputted into equation 2.7 above, the design variable s_i assigns the kernel's maximum value, occurring at its centroid location. The difference between a linear basis function and a radial basis function is that, in a linear basis function, the function linearly approaches from 1 at its origin to 0 at an

assigned outer range of influence, whereas a radial basis function decreases non-linearly from 1 based on the distance (or radius in 2D and 3D) from its origin. The range of influence of a radial basis kernel is controlled by a tuning parameter, α [43]. Figure 2-6 shows a 1-D example of both a linear and a radial basis function. Where a segment is discretized into 10 equally spaced sections and 9 kernels are positioned at x values 1 through 9 and, in this case, assigned coefficients $s_i = [1 \ 3 \ 4 \ 5 \ 4.5 \ 5.75 \ 6 \ 5 \ 2]$. In this example, the linear basis function has a range of influence of 2 and the radial basis function has a tuning parameter of $\alpha=1$.

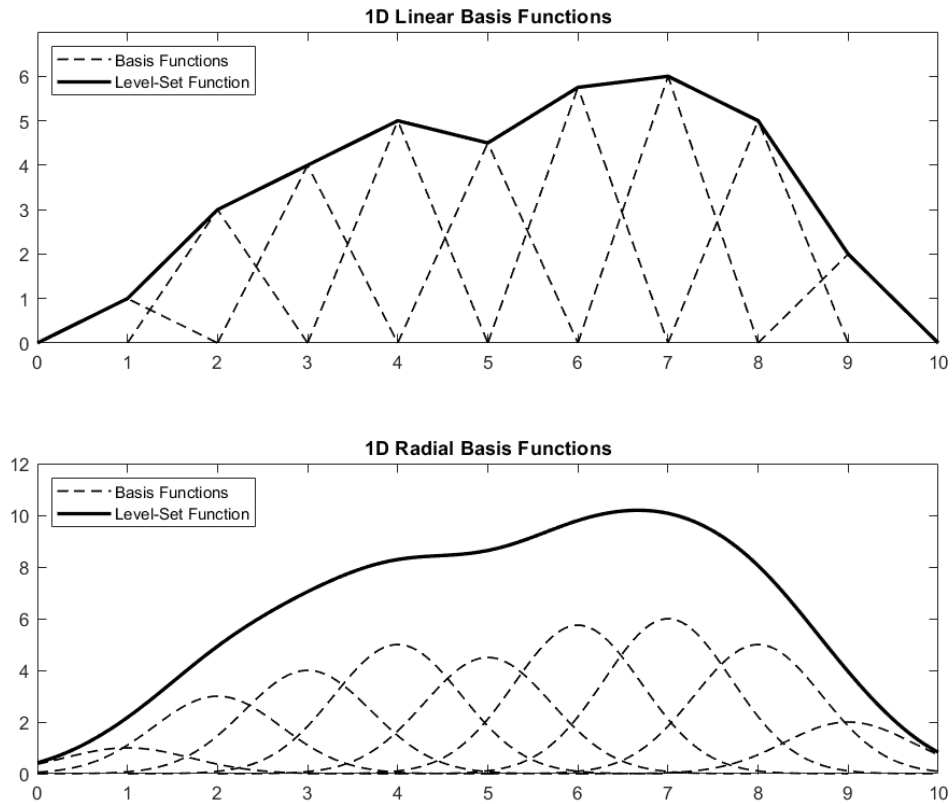


Figure 2-6: 1-D Basis Functions

The range of influence of each basis function determines the size of the domain that is impacted by a given parameter to determine the resulting geometry. This influences the geometric control of the domain as well as the optimization performance. The smaller this influence is, the more control the algorithm will have on the geometry allowing for the creation of smaller feature sizes. However, this also limits the rate of convergence [44]. For example, if each kernel only influences up to the adjacent kernels, as in the 1-D linear basis function in the figure above, then the adjustment of each parameter can only displace the interface by a distance equal to the spacing between each kernel. On the other hand, if these kernels' influences overlap, the algorithm can move the interface more between each iteration but cannot represent small variations and features along the iso-contour. Van Dijk, [34], categorizes the amount of influence into 3 categories: local [13], mid-range [43] and global [44] depicted in the figure below. The black dots show how many kernels, of the diagram, influence each position of the level-set function. Note that typically, in global basis functions, nearly all kernels will influence the entire domain to some degree, despite the figure only showing these four kernels.

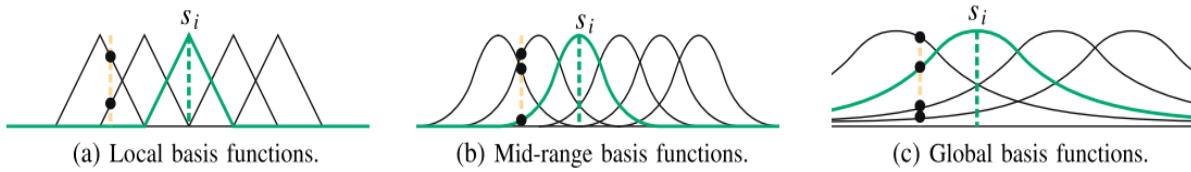


Figure 2-7: Ranges of Influence [34]

Apart from linear and radial basis functions, other methods such as a spectral parameterization or a Boolean combination of moving shapes have been explored as well. A spectral parameterization of the level-set function utilizes a Fourier series where the coefficients

of the Fourier series are the optimization variables [45]. This method has proved beneficial for periodic structures but also results in coarse design resolution. In a Boolean combination of shapes, a series of shapes are scattered throughout the domain and the optimization variables define the positioning and height of each of these shapes, allowing them to translate throughout the domain [46]. Figure 2-8 below provides an illustration of these differing basis functions.

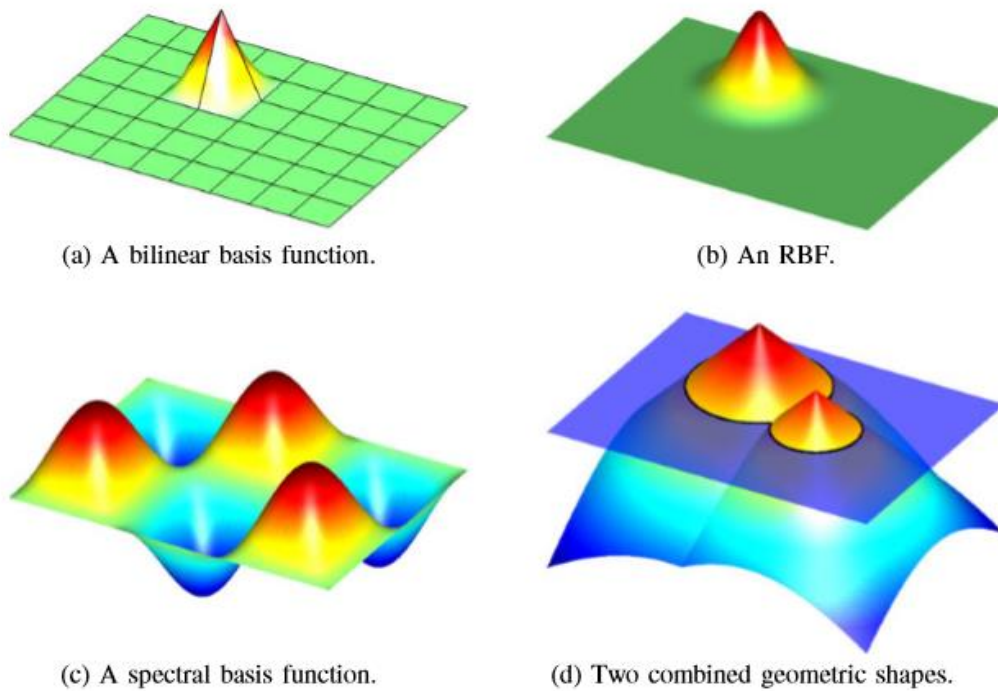


Figure 2-8: Types of LSF Parameterization [34]

When formulating a LSM for topology optimization, there are a great deal of options as to how the level-set function can be parameterized. This choice will determine how the optimization variables are defined, the rate of convergence, and the ability to define small feature sizes. Regularization can be added to the definition of the LSF in several ways. The design variables s_i can be subjected to a filtering or smoothing scheme to prevent drastic jumps

and near discontinuities from forming. Additionally, because the structural geometry is only defined as the intersection of the iso-contour plane and the LSF, there may exist an infinite number of LSFs for a given geometry. Some may be steep, flat, oscillating or a combination in regions of the domain, which would negatively impact the movement of the boundary front. To prevent this from occurring, the LSF can be periodically reinitialized. To do this, the current values of the optimization variables are recalculated such that the geometry and iso-contour of the LSF are maintained [13], [43]. This stabilizes the optimization performance each iteration by maintaining a constant gradient along the boundary. The advantages and need for this will be further explained in the update procedure section (2.2.3).

2.2.2 Geometry Mapping

Once the LSF has been established, its information has to be transferred to the analysis so that sensitivities and updates can then be found. As mentioned previously, a fixed iso-contour (typically $c=0$) of the LSF determines the interface of the geometry, and an assigned convention denotes which phase, material or void, is located above this contour. However, the decision comes in how this geometry is mapped and represented in the mechanical model. These decisions strongly influence the computational cost, accuracy of the structural model and the emergence of numerical artifacts. In Van Dijk's review of LSMs for topology optimization [34], the author covers three major techniques to do this: a conforming mesh [47], [48], an immersed boundary technique [46], [49] and a density-based approach [2], [43]. These three methods are depicted in the following figure 2-9 and explained below.

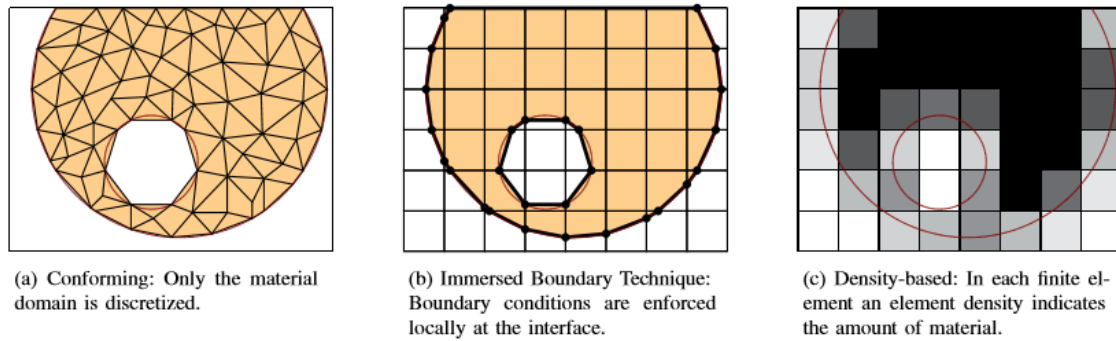


Figure 2-9: Types of Geometry Mapping [34]

The first and most intuitive means to geometrically represent the LSF for structural analysis is to directly take the material region at a given design, discretize it, and re-mesh the geometry every iteration. This method provides the most accurate structural performance prediction for the given design [48]. This has proven to be essential for geometries containing sharp interfaces and optimization problems with stress constraints [39], [50]. However, a major downside to this method and the reason it is not used often is the additional computational burden it creates on the algorithm. Another mild drawback to this technique is the introduction of noise between iterations as the discretization is changing at every iteration.

The second method attempts to maintain as much of this accuracy while reducing the computational burden. This is done via an Immersed Boundary Techniques (IBTs), eliminating the need to completely re-mesh by maintaining a fixed discretization of the domain and only modifying elements along the boundary. As seen in figure 2-9, only elements that would be cut by the interface are reshaped to fit within the iso-contour where material would be. The most common method to implement this technique is to use the eXtended Finite Element Method (X-FEM) [46], [49]. In this method, the integration bounds in computing the stiffness matrix for a boundary finite element are altered to only integrate over the material portions of the element.

This eliminates the void regions from being modeled as an artificially weak material, as is the case with density-based methods such as SIMP. This provides a more accurate model for stress concentrations. Additionally, geometry mapping using an IBT allows for the enforcement of boundary conditions directly along the interface [34]. Drawbacks to this method are that it introduces noise, particularly as an element along the boundary flickers between being on and off in the analysis. Furthermore, the algorithm may attempt to exploit poor discretizations resulting in ill-conditioning of the structural model. This can be remedied with smoothing and filtering of the LSF.

The final and most popular technique to represent a geometry provided by a LSF for analysis purposes is to perform a density-based approach [43]. This is quite similar to the SIMP method described in the Homogenization section (2.1.2), in that the discretization of the design domain is kept fixed and elemental density values are calculated as the fraction of the element within the material domain and then used to compute a proportional stiffness value. However, the major difference from the SIMP method is that only elements cut by the iso-contour of the LSF experience an intermediate density and all the other elements are either represented by the solid or void material. Similarly to SIMP method, the void material is modeled by an artificial extremely weak material as to eliminate numerical issues [2]. This method is significantly more computationally efficient; however, it concedes some analysis accuracy.

Of the three methods described, both the conforming mesh and IBTs techniques result in a model with crisp black-and-white domains and boundaries, resulting in higher structural accuracy (needed for stress or sharp geometries) at the expense of added computational time.

The third method, the density approach, is extremely efficient and easy to implement, making it the most commonly used method.

2.2.3 Update Procedure

Now that the LSF has been parameterized and the resulting geometry has been transferred to the mechanical model for analysis, the update procedure for subsequent iterations must be established so that an optimal design can be achieved. As with the LSF parameterization and geometry mapping, the update procedure can be executed in a variety of ways. The goal is to iteratively compute new optimization variable, \mathbf{s} , such that the objective will improve and eventually converge to an optimal design. As the majority of these procedures do not search globally, most of the time, these optimal designs can only be claimed to be local minima or maxima. To improve the algorithm and prevent it from stalling at a suboptimal local minimum, various regularization and relaxation techniques are implemented.

Within the realm of topology optimization, there are numerous objectives and optimization problems that can be formulated depending on the user's goals. Additionally, there are various types of update information that can be used and tied to a method for calculating new design parameters. Collectively these three aspects form the update procedure, and each is briefly discussed below.

First, the desired objective must be established. This formulates the optimization problem statement which has to be driving the optimizer subject to defined constraints. For topology optimization, the typical objectives may look like: minimize compliance, minimize volume, synthesize a compliant mechanism that achieves some goal, or maximize heat transfer [7], [10]. In designing a compliant mechanism, the goal is to maximize or minimize displacement

values at a given location when the domain is subjected to input forces or deflection values at another specific location. For example, this type of objective could be used in designing some sort of clamping mechanism. A heat transfer objective may be utilized when designing a conductive component such as a heat sink subjected to a boundary condition with a heat source or sink. The other two common objectives, minimum compliance and minimum volume, are typically addressed together to attempt to generate the ‘strongest’ and ‘lightest’ structure. Since cost or weight can be related to amount of material used, an engineer seeks to design a part that accomplishes some goal (such as holding a force) by using the minimum amount of material. Because of this, researchers originally attempted to implement an optimization problem with a minimum volume goal subject to stress constraints to determine the minimum size structure that would not fail [8]. The incorporation of stress constraints proved to be complex and resulted in many numerical errors, so researchers then formulized a minimum compliance objective subject to a volume constraint [11]. This proved to be much simpler and easier to implement, and became the most prevalent formulation for structural problems, and a benchmark for many update algorithms. The compliance of a given design, $c(x)$, is defined as the summation of the elemental strain energies, and the optimization formulation is shown in equation 2.8 below [24] where U and K are the global deformation vector and stiffness matrix respectively and u_e and k_e are the corresponding elemental values for each of the N elements. The constraints on the objective are such that the design’s volume, $V(x)$, is less than the required volume allowed, V_{req} , and the displacements are such that their product with the global stiffness matrix equals the global force vector, F . Additionally, the appropriate boundary conditions must be applied such that assigned displacement values, u_o , are on the Dirichlet

boundary, Γ_D , traction values applied to the Neumann boundary, Γ_N , and zero stress on the homogeneous boundary, Γ_H .

$$\begin{aligned}
\min_x: \quad & c(x) = U^T K U = \sum_{e=1}^N u_e^T k_e u_e \\
S.T.: \quad & V(x) \leq V_{req} \\
& K U = F \\
& u = u_o \text{ on } \Gamma_D \\
& \sigma(u)n = t \text{ on } \Gamma_N \\
& \sigma(u)n = 0 \text{ on } \Gamma_H
\end{aligned} \tag{2.8}$$

Once the objective and constraints are established, the type of update information must be determined. In Van Dijk's review [34], there are three predominant types of update information identified. These include shape sensitivity [37], [38], parameter sensitivity [43], [46], and topological sensitivity [13], [40], which are each depicted in figure 2-10 below from left to right respectively. In the figure, Ω represent the material region of the domain, D represents the void regions, and Γ the boundary between the two. Each of these update informations are then correlated to a generalized change in response, δR , where the response of interest may be an objective or constraint.

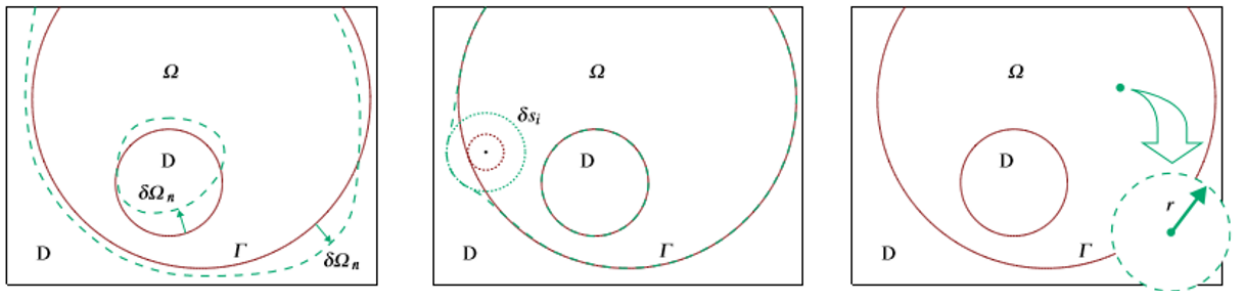


Figure 2-10: Types of Update Information [34]

The left image of figure 2-10 above illustrates variational shape sensitivity, which can be defined as the change in a response function caused by changes in shape of the material domain generated by infinitesimal changes in the normal direction along the boundary [37], [38]. By taking the path integral, $\int dS$, along the boundary, Γ , a generalized 1st order variational response (δR) due to boundary variation in the normal direction ($\delta\Omega_n$), can be modeled as shown in equation 2.9 below where $d_s R$ is the shape gradient of the response, which depends on the particular response definition of interest [34].

$$\delta R = \int_{\Gamma} d_s R \delta\Omega_n dS \quad (2.9)$$

The center image of figure 2-10 depicts sensitivities directly related to the optimization variables \mathbf{s} [34], [46]. Undoubtedly this depends upon the parameterization of the LSF, but taking the popular parameterization as defined in equation 2.7, with N_i and s_i being the individual kernel functions and their associated coefficients, and using an iso-contour of $c = 0$, the variations of optimization variables (δs_i) can be related to variations in material domain ($\delta\Omega$) with the following equation

$$\sum_i N_i \delta s_i + \nabla\varphi \cdot \delta\Omega = 0 \quad (2.10)$$

And defining the outward normal of the material boundary, \mathbf{n} , as:

$$\mathbf{n} = \frac{-\nabla\varphi}{\|\nabla\varphi\|} \quad (2.11)$$

Substituting equations 2.10 and 2.11 into 2.9, the parameterized shape sensitivity can be defined as:

$$\frac{\partial R}{\partial s_i} = \int_{\Gamma} d_s R \frac{N_i}{\|\nabla \varphi\|} dS \quad (2.12)$$

The third primary type of variation that sensitivities are derived from, is that of topological variations. This is exemplified in the right image of figure 2-10 above and can be viewed as the change in a response due to the perforation of an infinitesimal hole [40]. As the previous sensitivities mentioned are merely shape sensitivities, the topological sensitivity is required to alter the interior of the domain and increase the topological complexity of the domain by nucleating new holes. The topological gradient of response ($d_{\tau}R$) can be generically expressed given the equation below where $B(r)$ represents a hole B with radius r and $V(\cdot)$ is a measure of volume.

$$d_{\tau}R = \lim_{r \rightarrow 0} \frac{R(\Omega/B(r)) - R(\Omega)}{V(B(r))} \quad (2.13)$$

Using the minimum compliance objective formulated in equation 2.8, Allaire et al. [40] derives this topological gradient of response of a 2-D domain as:

$$d_{\tau}R = \frac{\pi(\lambda + 2\mu)}{2\mu(\lambda + \mu)} \{4\mu A e(u) \cdot e(u) + (\lambda - \mu) \text{tr}(A e(u)) \text{tr}(e(u))\} \quad (2.14)$$

And for a 3-D domain:

$$d_{\tau}R = \frac{\pi(\lambda + 2\mu)}{\mu(9\lambda + 14\mu)} \{20\mu Ae(u) \cdot e(u) + (3\lambda - 2\mu)tr(Ae(u))tr(e(u))\} \quad (2.15)$$

Where tr is the trace of a matrix, λ is Lamé's 1st parameter, μ is the shear modulus, A is the fourth order stiffness tensor and $e(u)$ is the strain tensor with displacement values u .

Now that various forms of sensitivities have been identified, the specific update procedure method can be established. Apart from heuristic methods [17], there are two main types of update procedures. The first is the use of mathematical programming through well-established optimization methods such as Sequential Quadratic Programming (SQP), Method of Moving Asymptotes (MMA) and CONVex LINearization approximations (CONLIN) [33]. The second, and more popular method, views the update procedure as a quasi-temporal process by advancing the boundaries based on velocity fields [13], [43]. Typically, this is done so by using a partial differential governing equation known as the Hamilton-Jacobi equation, shown in equation 2.16 below with τ representing the pseudo time. Because this equation only uses shape sensitivities, the update procedure does not allow for the nucleation of new holes. The ability to increase topological complexity can be done however by adding a reaction term, $R(\varphi)$, derived from topological sensitivities (equations 2.14 and 2.15) that acts as sink or source term to the PDE [13], [40]. Combining the Hamilton-Jacobi equation with the outward normal definition established in equation 2.11 and adding this reaction term, it can be rewritten as shown in equation 2.17, where v_n is derived from the variational shape sensitivities.

$$\frac{\partial \varphi}{\partial \tau} + \nabla \varphi \cdot v = 0 \quad (2.16)$$

$$\frac{\partial \varphi}{\partial \tau} - v_n \|\nabla \varphi\| - R(\varphi) = 0 \quad (2.17)$$

To determine an appropriate time step, $\Delta \tau$, such that the LSF progresses stably toward an optimum, the Courant-Friedrichs-Lewy (CFL) condition is used, where h is the grid spacing from the discretization of the LSF [13], [34], [43].

$$\Delta \tau \max(v_n) \leq h \quad (2.18)$$

2.2.4 Regularization

As mentioned before and with other topology optimization methods, an original formulation requires regularization techniques to obtain a well posed optimization problem, remove numerical artifacts, improve convergence behavior and control geometric properties. This is no different for LSMs. In fact, regularization can be applied to each of the three components previously discussed based on the nature of the given problem [34]. Many times, these regularization techniques come in the form of penalties or filtering schemes. In the LSF parameterization, the optimization variables themselves may be subjected to filtering techniques or bounded by minimum or maximum values to insure smoothness and consistency. Regularization can be applied to the geometry mapping aspect of LSMs depending on the method of executing the geometry mapping. For example, in the case of using a density-based method, intermediate densities can be penalized to insure black-and-white solutions [51].

The majority of regularization techniques however are applied to the update procedure portion of the LSM. Sensitivity values are often filtered, scaled or mapped to avoid mesh-dependent solutions and to obtain smooth geometric designs [13]. Additionally, the perimeter or length of an iso-contour can be penalized to prevent unnecessary perforations or porosity from forming and ensure smoothness of designs [38], [50]. Perimeter regularization is helpful to achieve a well-posed problem, avoid numerical artifacts and smooth the geometry. However, it may heavily restrict potential designs leading to suboptimal convergence.

As seen in the Hamilton-Jacobi equation (equation 2.16 or 2.17), the gradient of the LSF plays a large role in the update of the parameters from iteration to iteration. The steeper the gradient is, the larger the parameters will be modified and vice-versa. Additionally, if there is a large region close to the intersecting plane forming the iso-contour, there will be a much larger change in the interface and material domain, this is illustrated in figure 2-11 below. Because of how important the gradient of the LSF is, particularly near the iso-contour, many regularization techniques focus here to insure consistent and desired behavior [34]. One method known as Tikhonov regularization adds a penalty term associated with the gradient of the LSF [42]. Another way to handle this issue is to periodically reinitialize the LSF to a signed-distance function, allowing the LSF to evolve appropriately, but establishing a constant gradient before larger variations in gradient or larger regions near the iso-contour can form [13], [43]. When re-initialization is performed, the current LSF is used to map to the given geometry then this geometry is used to calculate LSF parameters such that there exists a constant gradient and the geometry is maintained as best as possible. This concept is very prominent amongst LSM implementations.

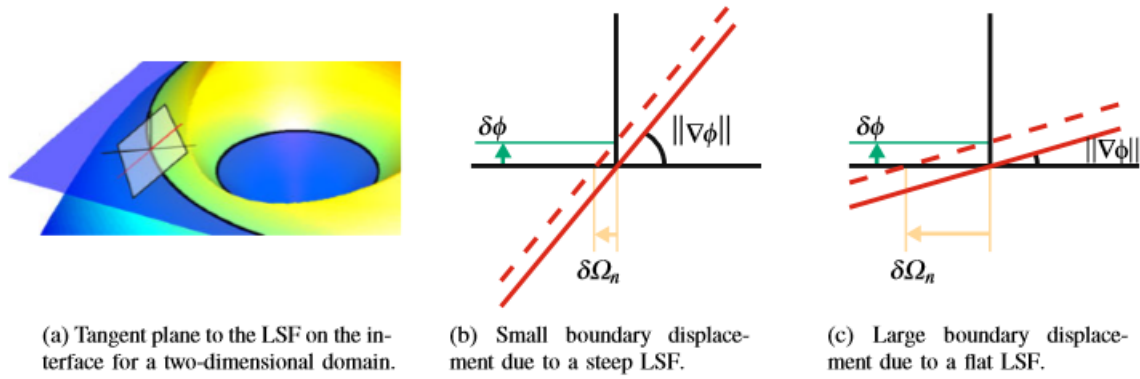


Figure 2-11: Effect of Variable LSF Gradients [34]

2.3 Design Dependent Loading

Often topology optimization is implemented with given boundary conditions and tries to determine the optimal material distribution under a specified objective function. In this case, the boundary conditions are established prior to optimization and maintained constant throughout the process. In elastic analysis, these boundary conditions are in the form of forces and fixed degrees of freedom, however given the nature of the current research objectives in this thesis, this is not the case. The forces acting on the component are pressure forces, applied from inside the part to the material boundary, which changes as the material distribution changes. This raises the need to modify the existing topology optimization methods to allow for design dependent loading. Researchers have implemented means of adapting both the SIMP method [52]–[55] and the LSM [47], [50] for design dependent loading. The two main tasks when adapting for design dependent loading are to effectively diagnose the loading condition given a specific material distribution and to appropriately modify the update procedure to account for the changing loading conditions. Another common form of design dependent loading is in the case of self weight loads. Huang et al. [54] address this for 2-D cases by using a

modified SIMP method known as 'Bi-directional Evolutionary Structural Optimization' (BESO) method with progressive target volume constraints. With pressure loading conditions however it can be quite challenging for the SIMP method to identify the loading condition due to the uses of intermediate densities and gray scale designs throughout the process, whereas with the LSM, the material interface is explicitly defined.

To establish the loading conditions of a pressure load using the SIMP method, a boundary search scheme must be used. Lee and Edmund [52] establish a method to do this for a 2-dimensional domain, which follows these steps:

1. Establish a small region that will always remain void (set densities and sensitivities of these elements to zero)
2. Establish an intermediate density value to apply the pressure loading to (typically start at 0.2 and slowly increase to 0.4 over optimization iterations to limit the formation of islands)
3. Use elemental density values to get nodal density values (average density of all elements containing a given node)
4. Linearly interpolate these nodal densities to identify iso-density points (points with density values equivalent to that established in step 2)
5. Starting from the centroid of the prescribed void and in a user defined search direction, with a minimal tolerance, find an initial iso-density point, figure 2-12 left
6. In another prespecified search direction orthogonal to the first find the second point, using a much larger directional tolerance (wider search cone), figure 2-12 right

7. Identify all consecutive points as an iso-density point within one element length and having the smallest change in segment angle from the previous point to the current point versus the current point to the new point, figure 2-13
8. This is repeated until a loop is established (connecting back to the 1st point) or a domain boundary is hit
9. Pressure force is applied to the line segments between iterative points and equivalent nodal loads are determined for the analysis

The processes of identifying the first two points is depicted in figure 2-12 below and choosing consecutive points in figure 2-13. In figure 2-12 the hashed area represents the predefined void region, the dots represent the iso-density points, and the dashed lines the search direction cone. In determining consecutive points, the angles are compared to one another and need to have a common arbitrary reference, horizontal to the right in this case. Note that in figure 2-13, point C is chosen from B instead of point D because it has the lesser change in angle. This process can be very sensitive and may create islands, stall, or generate numerical artifacts causing the optimization to take advantage of improper boundary identification during one iteration.

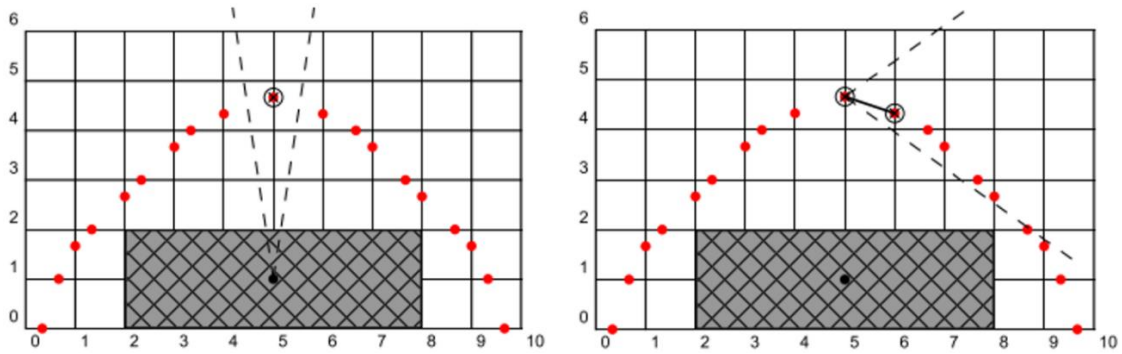


Figure 2-12: Identifying the 1st (Left) & 2nd (Right) Iso-Density Points [52]

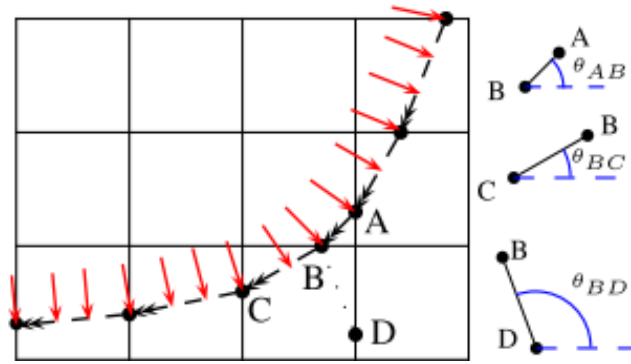


Figure 2-13: Identifying Consecutive Iso-Density Points [52]

Zhang et al. [55] address problems in 3-dimensions with design dependent loading cases while using the SIMP method. This is done by decomposing the 3-D case into a series of 2-D cases and executing a similar 2-D boundary search algorithm to the one mentioned above. Although desired results are achieved, it is noted that this process for 3-D cases is quite computationally expensive and inefficient.

Another proposed boundary identification method by Wang et al. [53] uses image segmentation techniques with a LSF. This method still uses the SIMP method for topology optimization, material distribution and analysis, but to identify the locations of the pressure loading, a LSM is used. In this method, a specific iteration's material distribution generated by the SIMP method is viewed as a gray scale image. Then a Distanced Regularized Level Set Evolution (DRLSE) method is used for image segmentation and the 0-level contour of the LSF is used to represent the pressure locations.

Contrary to the SIMP method's widespread use of intermediate densities, making it difficult to identify the material boundary for pressure loading to be applied, the LSM explicitly defines this interface, making it much simpler to execute topology optimization with design

dependent loading. Any finite element analysis is comprised of 3 boundaries within the domain: a Neumann boundary Γ_N where traction forces are applied, a homogeneous boundary Γ_H that is free of forces and a Dirichlet boundary condition Γ_D where displacement values are prescribed. It is noticed that standard topology optimization maintains fixed Neumann and Dirichlet boundary condition and only modifies the homogeneous boundary. Xia et al. propose a means to modify different types of boundaries by using multiple LSFs [47]. This is done by using separate sensitivities, Hamilton-Jacobi equations and update timesteps. Using a Boolean combination of the separate LSFs, the geometry of the design and designation of boundaries are determined. In a follow-on paper [50], this concept is applied to pressure loading problems. In this method, one LSF, ψ , represents the pressure boundary and another LSF, ϕ , represents the free boundary. The material domain is defined as the regions where both LSFs are below the iso-contour level and this geometry can be represented in the following equation.

$$\Omega = \{x \mid \max(\psi(x), \phi(x)) < 0, x \in D\} \quad (2.19)$$

The update velocities for the Hamilton-Jacobi are then derived as:

$$v^\psi = (2\text{div}(p_o u) + Ae(u) \cdot e(u) - \lambda)n \quad (2.20)$$

$$v^\phi = (Ae(u) \cdot e(u) - \lambda)n \quad (2.21)$$

Where λ is the penalty from the Lagrangian formulation with the volumetric constraint moved into the objective function. A special check and modification to velocities are done to prevent the update procedure to cause the Neumann and Homogeneous boundaries to cross, as this would have no practical meaning and defeat the purpose of the optimization problem with pressure loading.

For the finite element analysis of this method [50], a fixed Eulerian mesh is used with the void modeled by an artificially weak material and the geometry mapping is done with a density-based approach. The pressure load is to be applied on the Neumann boundary defined by $\psi(x) = 0$ which can be written as the line integral along the boundary as shown as the middle equality of equation 2.22. Through the use of a Dirac function, this path integral can be converted to an integral over the full domain, shown on the right side of equation 2.22.

$$\mathbf{F} = \int_{\Gamma_N} p \, ds = - \int_D p_o n \delta_{\Gamma_N} \, dx \quad (2.22)$$

With ε being a small positive constant based on the discretization grid size of the LSF, this Dirac function on the Neumann boundary Γ_N is defined as:

$$\delta_{\Gamma_N} n = \frac{1}{2} \nabla \left(\frac{\psi(x)}{\sqrt{\psi^2(x) + \varepsilon^2}} \right) \quad (2.23)$$

This method utilizes the benefits of a LSM when applied to a topology optimization problem with design dependent pressure loading conditions as opposed to the complex and time consuming methods developed to modify the SIMP method to accomplish the same task.

CHAPTER III: Methodology

Following a literature review, it was decided that the use of a Level-Set method (LSM) would best suit the objective of using topology optimization to determine an ideal material distribution for an irregular shaped pressure vessel. This chapter dives further into the derivation of the methods that were used to accomplish this task. For any optimization procedure, an analysis of the system's response to design variables must be conducted to effectively evaluate performance and implement changes. Here, a linear elastic finite element analysis (FEA) method is used to evaluate the structural response of a given iteration's material distribution. The response field generated by the FEA allows the use of a LSM to effectively modify a Level-Set function (LSF) which is used to implicitly define the material distribution for a subsequent iteration. This optimization process is repeated until an assigned objective is met and all constraints are satisfied. This chapter is organized as follows: section 3.1 summarizes the finite element analysis procedure, section 3.2 covers the generic methodology of using the Level-Set method for topology optimization and finally section 3.3 addresses the modifications of the LSM required for problems with design dependent pressure loading in both \mathbb{R}^2 and \mathbb{R}^3 .

3.1 Finite Element Analysis

As the structure changes every iteration, so too does its response which is used to evaluate the effectiveness of the current structure and assign update information for the following iterations. To evaluate this response of the structure, the finite element method is executed upon each iteration of the optimization. Here, a linear elastic finite element analysis is used, where a structure with defined material properties and boundary conditions is evaluated to identify a displacement field, \mathbf{u} . In order to implement the finite element analysis, a weak

form must be derived from the governing equations then, with a discretized structural domain, a system of equations can be solved to compute this displacement field at each discretized node. For notation, the cartesian components of the displacement vector, \vec{u} , throughout the domain, Ω , can be expressed as:

$$\vec{u} = u_i = [u_x \quad u_y \quad u_z] \quad (3.1)$$

From solid mechanics, in a 3-dimensional domain using a cartesian coordinate system, the equations of equilibrium for a statically elastic problem are:

$$\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z} + b_x = 0 \quad (3.2)$$

$$\frac{\partial \tau_{yx}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} + \frac{\partial \tau_{yz}}{\partial z} + b_y = 0 \quad (3.3)$$

$$\frac{\partial \tau_{zx}}{\partial x} + \frac{\partial \tau_{zy}}{\partial y} + \frac{\partial \sigma_{zz}}{\partial z} + b_z = 0 \quad (3.4)$$

Where σ_{ii} and τ_{ij} represent the normal and shear states of stress respectively, and b_i the body force in the i^{th} direction.

To derive a weak form to be used, the Galerkin weighted residual method is used. This is done by defining the PDEs from the equations of equilibrium as residuals and the variational displacements, δu_i , as the weighting function. In tensor form, the resulting equation is derived for $i = 1, 2, 3$.

$$\int_{\Omega} \delta u_i (\sigma_{ij,j} + b_i) d\Omega = 0 \quad (3.5)$$

Where $\sigma_{ij,j}$ is the partial derivative of the stress σ_{ij} , (where σ_{ij} represents both σ_{ii} and τ_{ij}) with respect to the direction of j and summed over $j = 1,2,3$. Separating the stress components from the body force:

$$\int_{\Omega} (\sigma_{ij,j}) \delta u_i d\Omega + \int_{\Omega} \delta u_i b_i d\Omega = 0 \quad (3.6)$$

Using the identity:

$$\nabla \cdot (\{\sigma_{ij}\} \delta u) = \nabla \cdot \{\sigma_{ij}\} \delta u + \nabla \delta u \cdot \{\sigma_{ij}\} \quad (3.7)$$

And the Divergence Theorem:

$$\int_V (\nabla \cdot \{u\}) dV = \int_{\Gamma} (\{u\} \cdot \{n\}) d\Gamma \quad (3.8)$$

Equation 3.5 can be written as:

$$\int_{\Gamma} \sigma_{ij} \delta u \cdot \vec{n}_i d\Gamma - \int_{\Omega} \nabla \delta u_i \cdot \sigma_{ij} d\Omega + \int_{\Omega} \delta u_i b_i d\Omega = 0 \quad (3.9)$$

Combining equation 3.9 for $i = 1,2,3$ the weak form becomes:

$$\begin{aligned} & \int_{\Gamma} [\delta u_x \quad \delta u_y \quad \delta u_z] \begin{Bmatrix} t_x \\ t_y \\ t_z \end{Bmatrix} d\Gamma \\ & - \int_{\Omega} \left(\left[\frac{\partial \delta u_x}{\partial x} \quad \frac{\partial \delta u_y}{\partial y} \quad \frac{\partial \delta u_z}{\partial y} \quad \frac{\partial \delta u_y}{\partial z} + \frac{\partial \delta u_z}{\partial y} \quad \frac{\partial \delta u_x}{\partial z} + \frac{\partial \delta u_z}{\partial x} \quad \frac{\partial \delta u_x}{\partial y} + \frac{\partial \delta u_y}{\partial x} \right] \begin{Bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{yz} \\ \sigma_{xz} \\ \sigma_{xy} \end{Bmatrix} \right) d\Omega \\ & + \int_{\Omega} [\delta u_x \quad \delta u_y \quad \delta u_z] \begin{Bmatrix} b_x \\ b_y \\ b_z \end{Bmatrix} d\Omega = 0 \end{aligned} \quad (3.10)$$

Where t_i , represents the traction forces along the boundary. The Cauchy strain tensor, ϵ_{ij} , can be defined by displacements, \vec{u} , using the strain-displacement relation:

$$\epsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i}) \quad (3.11)$$

Where $u_{i,j}$ is the partial derivative of the i^{th} component of deflection with respect to the direction of j , allowing the engineering strain to be written as:

$$\{\epsilon\} = \begin{Bmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \epsilon_{zz} \\ \gamma_{yz} \\ \gamma_{xz} \\ \gamma_{xy} \end{Bmatrix} = \begin{Bmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \epsilon_{zz} \\ 2\epsilon_{yz} \\ 2\epsilon_{xz} \\ 2\epsilon_{xy} \end{Bmatrix} = \begin{Bmatrix} \frac{\partial u_x}{\partial x} \\ \frac{\partial u_y}{\partial y} \\ \frac{\partial u_z}{\partial z} \\ \left(\frac{\partial u_y}{\partial z} + \frac{\partial u_z}{\partial y}\right) \\ \left(\frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial x}\right) \\ \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}\right) \end{Bmatrix} \quad (3.12)$$

Using Hooke's law (equation 3.13) stresses and strains can then be related using the 4th order tensor A_{ijkl} .

$$\sigma_{ij} = A_{ijkl}\epsilon_{kl} \quad (3.13)$$

With ν being the Poisson's ratio and E being the Young's modulus of elasticity, Hooke's law can be written into a constitutive matrix $[C]$ for isotropic materials that relates the 6 independent strain components with stress components. Note the use of engineering strain for the shear components.

$$\begin{Bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \tau_{yz} \\ \tau_{xz} \\ \tau_{xy} \end{Bmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{(1-2\nu)}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{(1-2\nu)}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{(1-2\nu)}{2} \end{bmatrix} \begin{Bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \gamma_{yz} \\ \gamma_{xz} \\ \gamma_{xy} \end{Bmatrix} \quad (3.14)$$

To prevent numerical artifacts in the optimization process, the void regions are defined by an artificially weak material, as opposed to not being modeled, by multiplying the modulus of elasticity by 0.0001. Combining equations 3.12 and 3.14 into the weak form of equation 3.10 results in:

$$\begin{aligned} & \int_{\Gamma} [\delta u_x \quad \delta u_y \quad \delta u_z] \begin{Bmatrix} t_x \\ t_y \\ t_z \end{Bmatrix} d\Gamma \\ & - \int_{\Omega} \left([\delta \varepsilon_{xx} \quad \delta \varepsilon_{yy} \quad \delta \varepsilon_{zz} \quad \delta \gamma_{yz} \quad \delta \gamma_{xz} \quad \delta \gamma_{xy}] [C] \begin{Bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \gamma_{yz} \\ \gamma_{xz} \\ \gamma_{xy} \end{Bmatrix} \right) d\Omega \\ & + \int_{\Omega} [\delta u_x \quad \delta u_y \quad \delta u_z] \begin{Bmatrix} b_x \\ b_y \\ b_z \end{Bmatrix} d\Omega = 0 \end{aligned} \quad (3.15)$$

Now that the weak form has been established, in order to evaluate it, the domain must be discretized into elements and nodes. This discretization allows for the use of shape functions within each element to approximate the response field (displacements), which can then be evaluated in equation 3.15 above to develop a system of equations that can then be solved to

determine nodal displacements. To simplify the meshing procedure and the computation of the stiffness matrix, all of the elements are equivalent in shape and size. Here 8-node hexahedral elements are used. A generalized master element shape and node relation can be depicted in figure 3-1 below where ξ , η , and ζ represent the 3 relative coordinate directions for the local element. These coordinates of the master element nodes can be found in table 3-2.

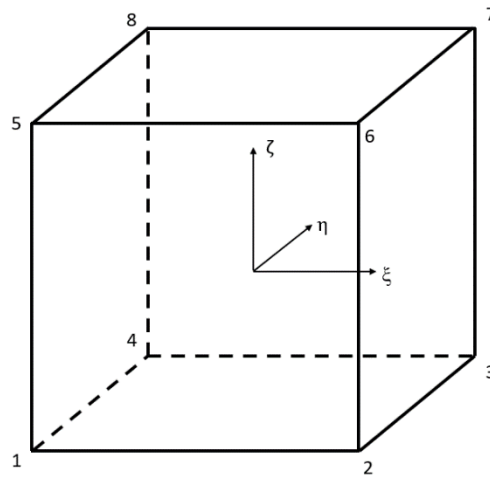


Figure 3-1: Hexahedral Master Element

Table 3-2: Master Element Node Coordinates

Node Number	ξ	η	ζ
1	-1	-1	-1
2	1	-1	-1
3	1	1	-1
4	-1	1	-1
5	-1	-1	1
6	1	-1	1
7	1	1	1
8	-1	1	1

Using this master element definition, at each node ($a = 1:8$) a tri-linear shape function, N_a , can be expressed with the following equation.

$$N_a(\xi, \eta, \zeta) = \frac{1}{8}(1 + \xi_a \xi)(1 + \eta_a \eta)(1 + \zeta_a \zeta), \quad a = 1, 2, \dots, 8 \quad (3.16)$$

These shape functions have a value of 1 at their respective node and a value of 0 at all other nodes allowing a field variable to be approximated throughout the element's domain as the summation of these shape functions multiplied by the respective nodal value of the field variable. Using the displacement vector, u_i , as the field variable, the approximation of displacement throughout the domain within a given element can be expressed as:

$$u_i(\xi, \eta, \zeta) = \sum_{a=1}^n N_a(\xi, \eta, \zeta) u_i^a \quad (3.17)$$

Where u_i^a is the displacement value in the i^{th} direction at node a and n is the number of nodes the element contains, 8 in this case. Similarly, partial derivatives of field variables can be expressed as:

$$\frac{\partial u_i}{\partial x_j}(\xi, \eta, \zeta) = \sum_{a=1}^n \frac{\partial N_a(\xi, \eta, \zeta)}{\partial \xi_j} u_i^a \quad (3.18)$$

Where ξ_j for $j = 1, 2, 3$ are the three relative directions of the master element, $\xi_j = [\xi \quad \eta \quad \zeta]$.

Note, these shape functions are defined over the master element's domain. To transform these equations to the x, y, z domain of the real element, a Jacobian matrix (equation 3.19) is used.

$$\begin{aligned}
[J] &= \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} = \sum_{a=1}^n \frac{\partial N_a(\xi, \eta, \zeta)}{\partial \xi_i} x_j^a \\
&= \begin{bmatrix} \sum_{a=1}^n \frac{\partial N_a(\xi, \eta, \zeta)}{\partial \xi} x_a & \sum_{a=1}^n \frac{\partial N_a(\xi, \eta, \zeta)}{\partial \xi} y_a & \sum_{a=1}^n \frac{\partial N_a(\xi, \eta, \zeta)}{\partial \xi} z_a \\ \sum_{a=1}^n \frac{\partial N_a(\xi, \eta, \zeta)}{\partial \eta} x_a & \sum_{a=1}^n \frac{\partial N_a(\xi, \eta, \zeta)}{\partial \eta} y_a & \sum_{a=1}^n \frac{\partial N_a(\xi, \eta, \zeta)}{\partial \eta} z_a \\ \sum_{a=1}^n \frac{\partial N_a(\xi, \eta, \zeta)}{\partial \zeta} x_a & \sum_{a=1}^n \frac{\partial N_a(\xi, \eta, \zeta)}{\partial \zeta} y_a & \sum_{a=1}^n \frac{\partial N_a(\xi, \eta, \zeta)}{\partial \zeta} z_a \end{bmatrix} \quad (3.19)
\end{aligned}$$

With this relation given by the Jacobian matrix, the shape functions can be written in terms of the domain for the real element (3.20) and integration bounds of the real element can be translated to the master element (3.21).

$$\begin{bmatrix} \frac{\partial N_a(x, y, z)}{\partial x} \\ \frac{\partial N_a(x, y, z)}{\partial y} \\ \frac{\partial N_a(x, y, z)}{\partial z} \end{bmatrix} = [J]^{-1} \begin{bmatrix} \frac{\partial N_a(\xi, \eta, \zeta)}{\partial \xi} \\ \frac{\partial N_a(\xi, \eta, \zeta)}{\partial \eta} \\ \frac{\partial N_a(\xi, \eta, \zeta)}{\partial \zeta} \end{bmatrix} \quad (3.20)$$

$$\int_{\Omega}^e d\Omega \rightarrow \int_{-1}^1 \int_{-1}^1 \int_{-1}^1 \det([J]) d\xi d\eta d\zeta \quad (3.21)$$

Using these shape functions, the strain tensor in equation 3.12 can be expressed as:

$$\{\varepsilon\} = \begin{Bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \gamma_{yz} \\ \gamma_{xz} \\ \gamma_{xy} \end{Bmatrix} = \begin{Bmatrix} u_{1,1} \\ u_{2,2} \\ u_{3,3} \\ u_{2,3} + u_{3,2} \\ u_{1,3} + u_{3,1} \\ u_{1,2} + u_{2,1} \end{Bmatrix} = \sum_{a=1}^n \begin{bmatrix} N_{a,1} & 0 & 0 \\ 0 & N_{a,2} & 0 \\ 0 & 0 & N_{a,3} \\ 0 & N_{a,3} & N_{a,2} \\ N_{a,3} & 0 & N_{a,1} \\ N_{a,2} & N_{a,1} & 0 \end{bmatrix} \begin{Bmatrix} u_1^a \\ u_2^a \\ u_3^a \end{Bmatrix} \quad (3.22)$$

Where $N_{a,i}$ is the partial derivative of the shape function for node a with respect to the i^{th} coordinate direction. Removing the summation by expanding the matrix this can be written as:

$$\{\varepsilon\} = \begin{bmatrix} N_{1,1} & 0 & 0 & N_{2,1} & 0 & 0 & \dots & N_{8,1} & 0 & 0 \\ 0 & N_{1,2} & 0 & 0 & N_{2,2} & 0 & \dots & 0 & N_{8,2} & 0 \\ 0 & 0 & N_{1,3} & 0 & 0 & N_{2,3} & \dots & 0 & 0 & N_{8,3} \\ 0 & N_{1,3} & N_{1,2} & 0 & N_{2,3} & N_{2,2} & \dots & 0 & N_{8,3} & N_{8,2} \\ N_{1,3} & 0 & N_{1,1} & N_{2,3} & 0 & N_{2,1} & \dots & N_{8,3} & 0 & N_{8,1} \\ N_{1,2} & N_{1,1} & 0 & N_{2,2} & N_{2,1} & 0 & \dots & N_{8,2} & N_{8,1} & 0 \end{bmatrix} \begin{Bmatrix} u_1^1 \\ u_2^1 \\ u_3^1 \\ u_1^2 \\ u_2^2 \\ u_3^2 \\ \vdots \end{Bmatrix} = [\mathbf{B}]\{d\} \quad (3.23)$$

Establishing the matrix in equation 3.23 as $[\mathbf{B}]$ and the displacement vector as $\{d\}$, the second integral of the weak form found in equation 3.15 can be rewritten using the relation in equation 3.21 as:

$$\int_{\Omega} \left([\delta\varepsilon_{xx} \quad \delta\varepsilon_{yy} \quad \delta\varepsilon_{zz} \quad \delta\gamma_{yz} \quad \delta\gamma_{xz} \quad \delta\gamma_{xy}] [\mathbf{C}] \begin{Bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \gamma_{yz} \\ \gamma_{xz} \\ \gamma_{xy} \end{Bmatrix} \right) d\Omega \quad (3.24)$$

$$= \{\delta d\}^T \iiint_{-1}^1 [\mathbf{B}]^T [\mathbf{C}] [\mathbf{B}] \det(\mathbf{J}) d\xi d\eta d\zeta \{d\}$$

Numerical integration via Gaussian quadrature is used to evaluate this integral. Because the shape functions are tri-linear, 2 Gauss points are used in each direction for a total of 8 Gauss points. These points have every combination of $\xi_i = \pm 1/\sqrt{3}$ for each of the coordinate directions, ξ , η , and ζ , and an equal weighting of 1. To execute this numerical integration, the function inside the integral is then evaluated at each of these points, multiplied by their respective weighting and summed together. For the 8-node hexahedral, this results in a 24x24 matrix for the element, $[\mathbf{k}_e]$, known as the elemental stiffness matrix.

$$\{\delta d\}^T \iiint_{-1}^1 [\mathbf{B}]^T [\mathbf{C}] [\mathbf{B}] \det([\mathbf{J}]) d\xi d\eta d\zeta \{d\} = \{\delta d\}^T [\mathbf{k}_e]_{24 \times 24} \{d\} \quad (3.25)$$

Because the nodes are shared by multiple elements, a global stiffness matrix for the entire domain can be assembled by correlating common degrees of freedom, deflections of nodes in a particular direction, and summing them together. This global stiffness matrix is denoted by $[\mathbf{K}]$ and is square with dimensions equal to 3 times the total number of nodes.

Following a similar approach to that of the second integral, the first integral can be written as:

$$\int_{\Gamma} [\delta u_x \quad \delta u_y \quad \delta u_z] \begin{Bmatrix} t_x \\ t_y \\ t_z \end{Bmatrix} d\Gamma = \{\delta d\}^T \int_{\Gamma_t} \begin{bmatrix} N_1 & 0 & 0 \\ 0 & N_1 & 0 \\ 0 & 0 & N_1 \\ \hline N_2 & 0 & 0 \\ 0 & N_2 & 0 \\ 0 & 0 & N_2 \\ \vdots & \vdots & \vdots \end{bmatrix} dS = \{\delta d\}^T \begin{Bmatrix} f_1^1 \\ f_2^1 \\ f_3^1 \\ \hline f_1^2 \\ f_2^2 \\ f_3^2 \\ \vdots \end{Bmatrix}_{24 \times 1} \quad (3.26)$$

Where f_i^a is the nodal force value in the i^{th} direction for node a . Pressure is defined as an outward normal force per unit area and is applied to the interior material/void boundary. Although the 2-D application by Xia et al. [50] uses an approximate Dirac-delta function on the LSF to establish this loading condition, here mesh and structural representation of each element is used. If the entire domain is meshed, and elements are simply defined as void or having material, the pressure forces can be calculated as outward normal forces for every void element. Although forces are applied to every void element, due to uniform element sizes, forces within the void region are cancelled out, resulting in only forces being applied to the boundary of void element. Although forces are applied to every void element, due to uniform element sizes, forces within the void region are cancelled out, resulting in only forces being applied to the boundary between void and solid regions. As illustrated in figure 3-2 adjacent void elements will have their outward normal forces cancel out, leaving only desired force components along the boundary.

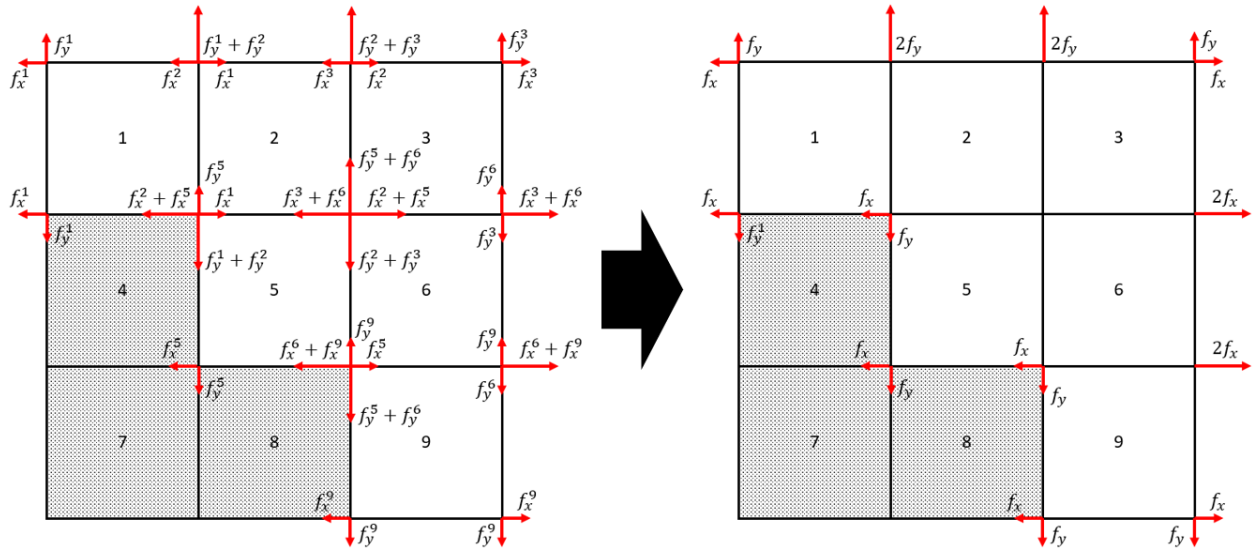


Figure 3-2: Force Vector Computation from Void

The force components for one void element can be calculated as:

$$\{f_e\}_{24 \times 1} = \begin{Bmatrix} f_x^1 \\ f_y^1 \\ f_z^1 \\ f_x^2 \\ f_y^2 \\ f_z^2 \\ \vdots \end{Bmatrix} = \frac{p_o}{4} \begin{Bmatrix} \xi_1 l_y l_z \\ \eta_1 l_x l_z \\ \zeta_1 l_x l_y \\ \xi_2 l_y l_z \\ \eta_2 l_x l_z \\ \zeta_2 l_x l_y \\ \vdots \end{Bmatrix} \quad (3.27)$$

Where p_o is the nominal pressure value and $[l_x \ l_y \ l_z]$ is the edge lengths of the element.

The use of the master element node coordinates in equation 3.27 are simply to denote the sign of the force to ensure it is outward normal and only works because the node coordinates are ± 1 .

If a different master element is used, this relation would be inaccurate.

In a similar manner, if regions of the void are not meshed, the material domain can be used to formulate the force vector by applying an inward normal force to every solid element. As in the case with using the void elements, adjacent solid elements will result in the cancelling of forces at shared nodes. This leaves only force on the material boundary, including both the Homogeneous, Γ_H , and Neuman, Γ_N , boundary. Because the pressure force should only be applied to the Neuman boundary which is along the interior boundary, nodes along the exterior, or homogeneous boundary, are stored and set to zero following the assembly of the force vector. This concept is shown in figure 3-3 below where the left image shows all of the force components and the right shows the resultant forces following the global assembly process and zeroing out the homogeneous boundary, represented by the circles and labeled with Γ .

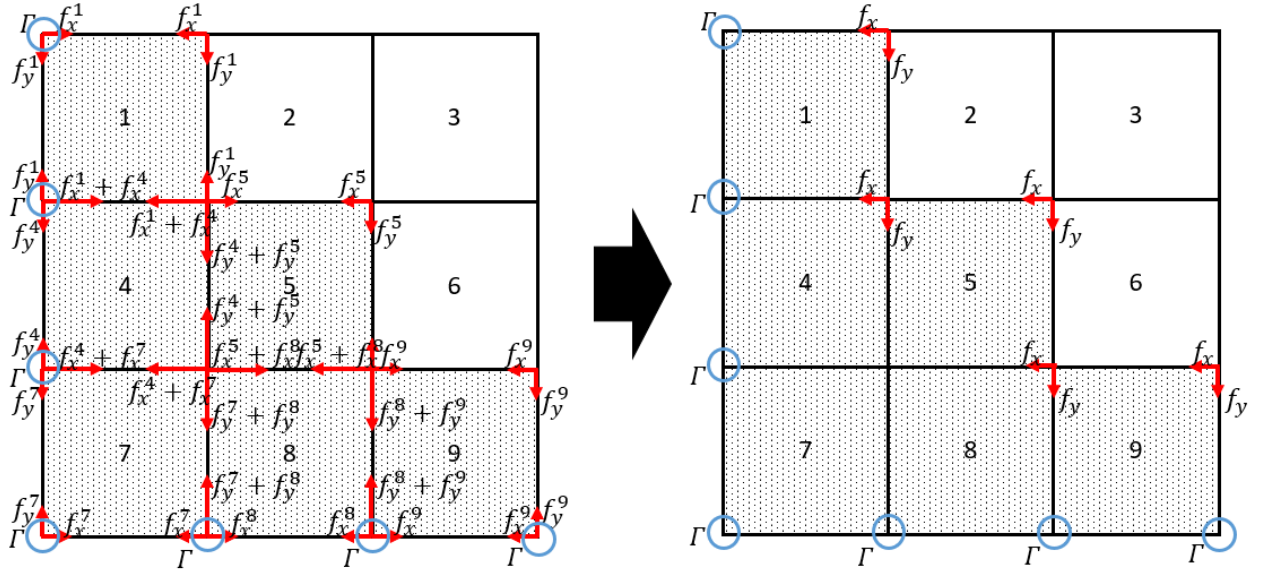


Figure 3-3: Force Vector Computation from Material Domain

As with equation 3.27 for the elemental outward force vector for the void elements, the negative provides the inward force vector for the solid element, as seen in equation 3.28.

$$\{f_e\}_{24 \times 1} = \begin{Bmatrix} f_x^1 \\ f_y^1 \\ f_z^1 \\ f_x^2 \\ f_y^2 \\ f_z^2 \\ \vdots \end{Bmatrix} = \frac{-p_o}{4} \begin{Bmatrix} \xi_1 l_y l_z \\ \eta_1 l_x l_z \\ \zeta_1 l_x l_y \\ \xi_2 l_y l_z \\ \eta_2 l_x l_z \\ \zeta_2 l_x l_y \\ \vdots \end{Bmatrix} \quad (3.28)$$

Similar to the assembly of the global stiffness matrix, a global force vector, $\{F\}$, can be assembled by summing common global degrees of freedom between elements. This is where adjacent void elements cancel out their force.

The third integral of the weak form (equation 3.15) can be cancelled out because the body forces are neglected in the current circumstances. This leaves only the first (equation 3.26) and second (equation 3.25) integrals, which can be expressed with the global force vector and stiffness matrix respectively. Using the assembly process previously mentioned for these and exchanging the elemental displacements, $\{d\}$, with the global displacement vector, $\{u\}$, equation 3.29 is achieved.

$$\{\delta u\}^T \{F\} - \{\delta u\}^T [K] \{u\} = 0 \quad (3.29)$$

Rearranging and cancelling out the variational deflections, the equation can be re-written as a system of equations.

$$[K] \{u\} = \{F\} \quad (3.30)$$

Before the system of equations can be solved, Dirichlet boundary conditions must be applied. In the current implementation, these boundary conditions come in the form of fixed degrees of freedom and therefore the partitioning method can be used. To ensure the displacement values of the fixed degrees of freedom are set to zero, the corresponding rows and columns of the stiffness matrix and force vector are removed, allowing the remaining system of equations to be solved to achieve a displacement vector field for all of the nodes within the domain.

3.2 Level-Set Method Formulation

Now that a given structural configuration can be analyzed to determine its response (deflection vector field) through the use of the finite element analysis method, this structure

needs to be iteratively modified to optimize a particular objective function subject to a set of constraints. As mentioned earlier, the Level-Set method has been chosen to execute this optimization procedure due to its inherent benefits from the implicit boundary representation, aiding in pressure loading application. This section provides a detailed mathematical explanation of the Level-Set method. The modifications that had to be made to this method in order to implement the topology optimization with design dependent pressure loads are explained in the following section, 3.3.

Ideally the objective for such a problem would be to maximize the internal void volume such that the part does not fail due to stress criteria. However, as mentioned in section 2.2.3, this type of objective formulation gives rise to various difficulties. Therefore, similarly to the development of many topology optimization methods, the problem has been rewritten into a minimum compliance objective. Although it should be mentioned that, in future works, it would be desired to revert back to the original maximum void volume objective, see chapter 7:

Conclusion. This being said, the objective for the works of this project has been set to minimize the compliance, $c(x)$, or total strain energy of the system. This objective formulation is then subject to constraints such that the design's material volume fraction, $V(x)$, is equal to the required volume fraction, V_{req} (chosen such that the void volume satisfies the wet volume requirement of the pressure vessel), the displacement field, $\{u\}$, is such that the finite element analysis equation (equation 3.30) is satisfied, and the appropriate boundary conditions are applied. These boundary conditions ensure that the assigned displacement values, u_o , are on the Dirichlet boundary, Γ_D , traction values applied to the Neumann boundary, Γ_N , and zero stress on the homogeneous boundary, Γ_H . This optimization formulation is shown in equation

3.31. Note, V refers to the volume fraction that the material takes (used volume divided by design space) up and the volume fraction of the void could be expressed as $1 - V$.

$$\begin{aligned}
\min_x: \quad & c(x) = U^T K U = \sum_{e=1}^N u_e^T k_e u_e \\
S.T.: \quad & V(x) = V_{req} \\
& [K]\{u\} = \{F\} \\
& u = u_o \text{ on } \Gamma_D \\
& \sigma(u)n = t \text{ on } \Gamma_N \\
& \sigma(u)n = 0 \text{ on } \Gamma_H
\end{aligned} \tag{3.31}$$

In order to effectively treat the volume constraint, it must be moved into the objective and a penalty must be applied, corresponding to the violation of the constraint to drive the problem towards an optimal solution that also satisfies the constraints. To do this, a Lagrangian is used and the resultant objective function can be seen in equation 3.32.

$$L = c(x) + Penalty = \left(\sum_{i=1}^N \mathbf{u}_e^T \mathbf{k}_e \mathbf{u}_e \right) + \lambda_i (V(x) - V_{req})^2 \tag{3.32}$$

Where λ_i is the Lagrange multiplier for the i^{th} iteration. The constraint derived term, $V(x) - V_{req}$, is squared to ensure a smooth application of the penalty due to its slope of zero when the volume is equivalent to the target volume. Additionally, as seen from equation 3.32, the penalty is applied to violations both above and below the target volume. This is done intentionally as any structure could always reduce its compliance by adding material and therefore the optimal solution to the original optimization formulation, equation 3.31, will be one such that the volume is equivalent to the target volume.

Now that the objective and constraints have been formulated, an optimization process needs to be executed to determine the ideal material distribution. Unlike other topology

optimization methods such as the SIMP method, the Level-Set method does not directly modify this material distribution within the domain. Instead the Level-Set method modifies a function, the level-set function, φ , that then implicitly defines the structure based on its zero-level contour. As shown in equation 3.32, for the works of this research, negative LSF values are defined as material in the structure, and positive values as void.

$$\left\{ \begin{array}{lll} \varphi(X) < 0 & X \in \Omega & 'Material' \\ \varphi(X) = 0 & X \in \Gamma & 'Interface' \\ \varphi(X) > 0 & X \in (D \setminus \Omega) & 'Void' \end{array} \right\} \quad (3.33)$$

The level-set function itself is defined by tri-linear basis functions discretized throughout the domain, see section 2.2.1. Upon running each optimization, an initial structure is defined by the user and the starting LSF is defined as a signed distance function based on this structure. That is, the magnitude of the LSF at a given location is the Euclidean distance to the nearest location in the structure of the opposite phase (solid or void), and the sign of the LSF is such that it satisfies equation 3.33.

Now that the LSF have been defined, its relationship to the structure for analysis purposes has to be established as per section 2.2.2 ‘Geometry Mapping’. Here a fixed Eulerian field is used to ease and accelerate the response calculations by the FEA method. However, instead of using an intermediate density for the structural representation, the process is further simplified, and the elements are only evaluated as completely void or solid (note void elements still have an artificially weak material property). To allow for improved geometric representation without invoking extensive computational burdens, the domain’s discretization is periodically re-meshed, exempting void regions as the algorithm converges. Following re-meshing, a subsequent iteration can add material into these void regions and the appropriate elements will

be added to the mesh. This allows for fast and easy conversion of the LSF to a structural representation for response and volume analysis. Because the elements of the structure are merely “on” or “off”, the volume fraction can be evaluated as the number of “on” elements multiplied by one element’s volume and divided by the total volume of the design domain.

As many LSM currently do, the evolution of the LSF is done by viewing the update procedure as a quasi-temporal, τ , process through the use of a Hamilton-Jacobi equation.

$$\frac{\partial \phi}{\partial \tau} + \nabla \phi \cdot v = 0 \quad (3.34)$$

Where v is a scalar velocity field based on shape derivatives. Note the absence of the reaction term derived from topological derivatives found in the Hamilton-Jacobi equation of section 2.2.3. This is due to the nature of the internal pressure vessel problem, where one continuous void is desired, therefore, sink and source terms to nucleate voids are removed.

These velocities, v , are derived from sensitivity analysis and chosen as a descent direction for the Lagrangian, equation 3.32. The sensitivity for a given element is defined as the change in response with respect to a change in domain. Taking the partial derivative of the Lagrangian in equation 3.32 for a particular element, e , results in the following equation.

$$\frac{\partial L}{\partial \Omega} |_e = \frac{\partial c}{\partial \Omega} |_e + 2\lambda_i(V(x) - V_{req}) \frac{\partial V}{\partial \Omega} |_e \quad (3.35)$$

As shown in equation 3.35, there are only two terms that contain a response from the material distribution. These are the sensitivity of compliance, $\frac{\partial c}{\partial \Omega} |_e$, and the sensitivity of the volume, $\frac{\partial V}{\partial \Omega} |_e$. The shape sensitivity of an element for the compliance term of the objective is shown in equation 3.36 [13], [38]. Because the volume response has a direct correlation to a

change in the domain, the shape sensitivities for the volume response are 1 and uniform across the entire domain, equation 3.37.

$$\frac{\partial c}{\partial \Omega}|_e = -\mathbf{u}_e^T \mathbf{k}_e \mathbf{u}_e \quad (3.36)$$

$$\frac{\partial V}{\partial \Omega}|_e = 1 \quad (3.37)$$

Plugging both equations 3.36 and 3.37 into the partial derivative of the Lagrangian, equation 3.35, and establishing the Hamilton-Jacobi velocity field as the decent direction, elemental velocities can be expressed as:

$$v|_e = -\frac{\partial L}{\partial \Omega}|_e = \mathbf{u}_e^T \mathbf{k}_e \mathbf{u}_e - \lambda_i (V(x) - V_{req}) \quad (3.38)$$

Recalling that λ_i is the Lagrange multiplier for the penalty and therefore the coefficient of '2' on the second term in equation 3.35 can be absorbed into this multiplier. This Lagrange multiplier needs to start small as to allow for the structure's update to be dominated by the compliance sensitivity to achieve an optimum solution and not fall into a local minimum. However, as the iteration procedure hones in upon the final solution, this multiplier needs to increase to ensure that the volume constraint is satisfied. This update of the Lagrange multiplier is done by a factor, α , every iteration, as seen in equation 3.39. The physical values used for λ_0 and α are discussed in chapter 4.

$$\lambda_i = \alpha \lambda_{i-1} \quad (3.39)$$

Once the velocities are found, the Hamilton-Jacobi equation (equation 3.34) can be used to update the LSF accordingly. However, prior to this update, the velocities are filtered to

smooth them so that mesh-dependent solutions are avoided and to obtain smooth geometric designs. Additionally, locations that are to remain a particular structural phase (solid or void) have their corresponding velocities set to 0. This includes the boundary of the pressure vessel, as it is desired for those to remain solid. Finally, the Hamilton-Jacobi equation itself is solved using an upwind finite difference scheme.

$$\varphi_{i+1} = \varphi_i - \Delta t(\nabla \varphi \cdot v) \quad (3.40)$$

Where Δt is the timestep of each modification of the LSF. In order to effectively modify the LSF, this time step must satisfy the 'Courant-Friedrichs-Lewy' (CFL) condition [10], [34], shown in equation 3.41, with h being the distance between grid-points of the LSF and v being the velocities

$$\Delta t \leq \frac{h}{\max|v|} \quad (3.41)$$

The gradient of the LSF, $\nabla \varphi$, is evaluated using a central difference scheme. Due to the generally poor accuracy of an explicit method to calculate the gradient, it is advised that this time step be much smaller than this stability limit [34]. However, multiple time steps can be executed with a single finite element analysis, allowing for reasonable shape changes to occur despite the small timestep.

As mentioned in section 2.2.3 and evident in the update procedure of the LSF, the gradient of the LSF plays a crucial role in the effectiveness of the LSM. Because of this, the LSF is periodically reinitialized to a signed distance function based on the current structure. This ensures a consistent gradient and prevents large regions near the zero-level contour. The same process as establishing the first LSF function is used to do this.

3.3 Design Dependent Pressure Loading

The above explained Level-Set method is a generic implementation that is designed for topology optimization with static loading and boundary conditions. That is, the user defines the design space, fixed boundary conditions and loading conditions which all remain constant throughout the entire process of optimizing the material distribution. However, the case of optimizing a pressure vessel falls under the umbrella of design-dependent loading, because, as the structure changes, so do the loading conditions. Because of this, the LSM described in section 3.2 needs to be modified. In this research, the LSM was first modified to mimic the work of Xia et al. [50] for 2-dimensional cases. Then it was further modified to allow for the topology optimization of 3-dimensional pressure vessels. Defining p_o as the pressure value, for both \mathbb{R}^2 and \mathbb{R}^3 , the optimization problem can be formulated as:

$$\begin{aligned}
 \min_x: \quad & c(x) = U^T K U = \sum_{e=1}^N u_e^T k_e u_e \\
 S.T.: \quad & V(x) \leq V_{req} \\
 & [K]\{u\} = \{F\} \\
 & u = u_o \text{ on } \Gamma_D \\
 & \sigma(u)n = p_o \text{ on } \Gamma_N \\
 & \sigma(u)n = 0 \text{ on } \Gamma_H
 \end{aligned} \tag{3.42}$$

3.3.1 Two-Dimensional Problems with Pressure Loading

Before the end goal of optimizing a 3-D pressure vessel is done, the LSM procedure is modified for a 2-D domain. To do this, the works of Xia et al. [50] were followed and implemented. The first and major modification to the method is the use of two LSFs, ϕ & ψ , to define both the ‘free’ homogeneous boundary, Γ_H , and the ‘pressure’ Neumann boundary, Γ_N , respectively. Because the structure is now defined by two LSFs, each defining a boundary, the

material phase of the domain is defined as the region where both LSFs are below the zero-level iso-contour. The structural implicit relation between the LSF and the structure is then defined as:

$$\left\{ \begin{array}{lll} \Phi(X) < 0 \text{ and } \psi(X) < 0 & X \in \Omega & \text{'Material'} \\ \Phi(X) > 0 \text{ or } \psi(X) > 0 & X \in (D \setminus \Omega) & \text{'Void'} \\ \Phi(X) = 0 \text{ and } \psi(X) < 0 & X \in \Gamma_H & \text{'Free Interface'} \\ \psi(X) = 0 \text{ and } \Phi(X) < 0 & X \in \Gamma_N & \text{'Pressure Interface'} \end{array} \right\} \quad (3.43)$$

In the implementation, the material can be defined as the locations where the maximum of the two LSFs is less than zero. Furthermore, because in many 2-D applications the pressure loading is applied at the edge of the domain, even if ψ 's zero-level contour extends past this boundary of the domain, the pressure force should still be applied. If ψ 's zero-level contour would extend past the domain, without modification, it would result in segmentation and a non-continuous boundary for the force to be applied to. This modification is done by defining a LSF, ψ_o , such that its zero-level contour is congruent to the edge of the domain that the force is applied from. Then, following an update, ψ for the subsequent iterations is taken as the maximum between this updated LSF, $\psi_{updated}$, and ψ_o , equation 3.44. This ensures that the zero-level contour for ψ is either within the domain, or congruent to the desired boundary creating a continuous boundary for the pressure to be applied on. These concepts for the structural representation are illustrated in figure 3-4 where the force is intended to be applied from the top edge of the domain and the structure is fixed on the left and right sides. Here an updated ψ shown in 3-4.c would cross the top edge of the domain causing portions to not have pressure forces applied, but when taken as the maximum between ψ in 3-4.c and ψ_o in 3-4.d, these portions have their zero-level contour converted to be congruent to the edge of the domain as shown by the red

line in 3-4.a representing the pressure boundary, Γ_N . Additionally, defining the solid regions as locations where the maximum of Φ , shown in 3-3.b, and ψ is less than zero, the appropriate material distribution is achieved as shown in 3-4.a.

$$\psi^{i+1}(x) = \max\{\psi_{updated}(x), \psi_o(x)\} \quad (3.44)$$

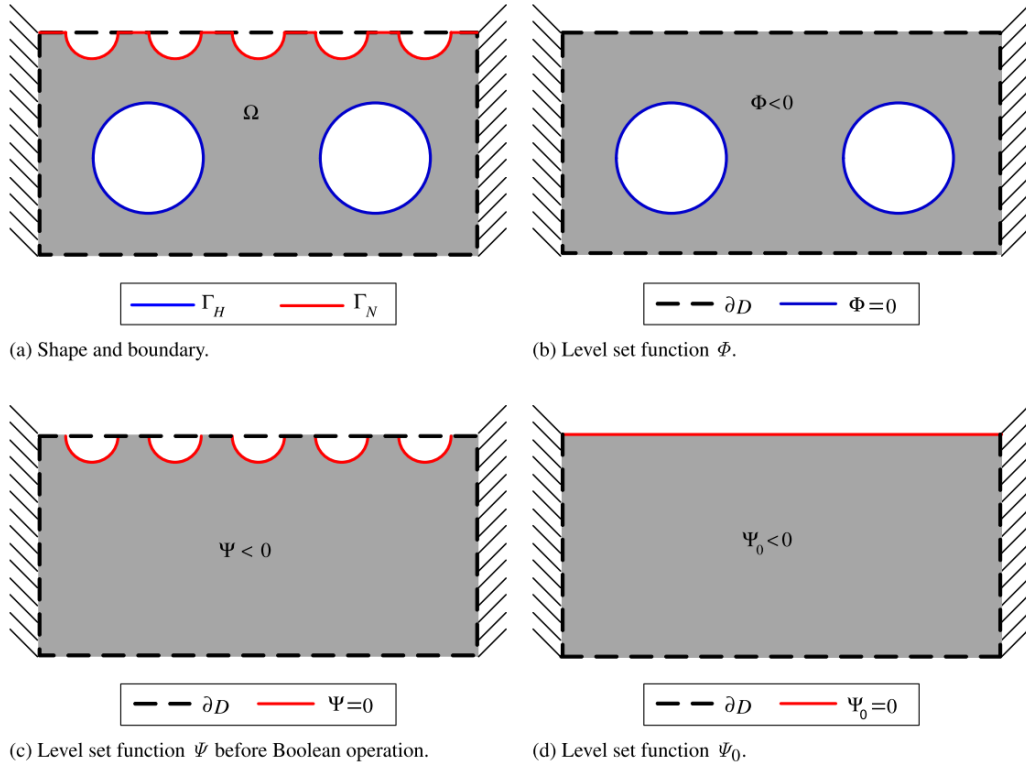


Figure 3-4: Dual LSF Structural Representation [50]

To update the structure, each LSF is subjected to its own Hamilton-Jacobi equation and shape sensitivity. The compliance sensitivity analyses for both of these LSFs are defined as [50]:

$$\frac{\partial C_\Phi}{\partial \Omega} |_e = -\mathbf{u}_e^T \mathbf{k}_e \mathbf{u}_e \quad (3.45)$$

$$\frac{\partial C_\psi}{\partial \Omega} |_e = -\mathbf{u}_e^T \mathbf{k}_e \mathbf{u}_e - 2\nabla \cdot (p_o \mathbf{u}_e) \quad (3.46)$$

Where p_o is the magnitude of the pressure load. The volume sensitivity for both Hamilton-Jacobi equations is a constant, 1, the same as defined in equation 3.37. As in equation 3.32, the volume constraint can be brought into the objective to formulate a Lagrangian. Following the same process for equation 3.38, by using the sensitivities for the compliance and volume, and choosing the descent direction for the Lagrangian, the velocities for both Hamilton-Jacobi equations can be expressed as shown in equations 3.47 and 3.48.

$$v_\phi|_e = -\frac{\partial L}{\partial \Omega}|_e = \mathbf{u}_e^T \mathbf{k}_e \mathbf{u}_e - \lambda_i(V(x) - V_{req}) \quad (3.47)$$

$$v_\psi|_e = -\frac{\partial L}{\partial \Omega}|_e = \mathbf{u}_e^T \mathbf{k}_e \mathbf{u}_e + 2\nabla \cdot (p_o \mathbf{u}_e) - \lambda_i(V(x) - V_{req}) \quad (3.48)$$

In many real-world applications of this optimization objective, the structural component is used to isolate a pressurized area from a non-pressurized area. Therefore, the boundaries of these two interfaces must not intersect, as this would defeat the purpose and have no physical meaning. To ensure this is the case, an additional procedure is applied with a prescribed minimum thickness, t . First for all locations along each border the shortest distance to the opposite border is found. Then for any distance value less than or equal to prescribed thickness requires a modified velocity. First, the largest magnitude of the two sensitivities at their respective border is determined. This is done to ensure a continued decent of the objective. Then, the magnitude of this velocity is assigned to the opposite border such that both boundaries progress along the same direction, keeping the thickness equal. Finally, to ensure smooth updates to the LSF, the change in velocity is diffused radially amongst the velocity field the change took place on. In using an upwind finite difference scheme (equation 3.40) to update the LSFs, the two timesteps must be equal to maintain a minimum thickness where velocity

modifications were applied. This timestep is defined as the minimum of the two that would be established individually.

The last modification to the LSM for this 2-D pressure case is in the force application within the FEA procedure. Standard topology optimization algorithms with static loading cases establish a global force vector prior to the optimization loop that is held constant. This cannot be done for the case of design dependent loading. For pressure loading cases, the first integral of the weak form (equation 3.15) can be expressed with the pressure value, p_o , as shown in the middle equality of equation 3.49 which can then be converted from a surface integral to an integral over the full domain using the divergence theorem (equation 3.8), shown in the right equality of 3.49.

$$\int_{\Gamma} [\delta u_x \quad \delta u_y \quad \delta u_z] \begin{Bmatrix} t_x \\ t_y \\ t_z \end{Bmatrix} d\Gamma = \{\delta d\}^T \int_{\Gamma_N} p_o dS = -\{\delta d\}^T \int_D p_o n \delta_{\Gamma_N} d\Omega \quad (3.49)$$

Where n is the outward normal direction of the boundary and δ_{Γ_N} is the Dirac function for the Neumann boundary, Γ_N . With the ψ 's zero-contour defining this boundary, this Dirac function can be approximated as:

$$\delta_{\Gamma_N} n \approx \frac{1}{2} \nabla \left(\frac{\psi(x)}{\sqrt{\psi^2(x) + \varepsilon^2}} \right) \quad (3.50)$$

Where ε is a small positive parameter recommended to be between $h/10$ and $h/2$, with h being the elemental grid size [50]. Smaller values of ε will result in the approximate force being applied to a tighter band along the iso-contour but with coarse directionality, while larger values

will diffuse the applied force but offer smoother application directions. An example of equation 3.50 with $\varepsilon = 0.2$ is shown in figure 3-6 for the example LSF in figure 3-5 with a grid size of 0.4.

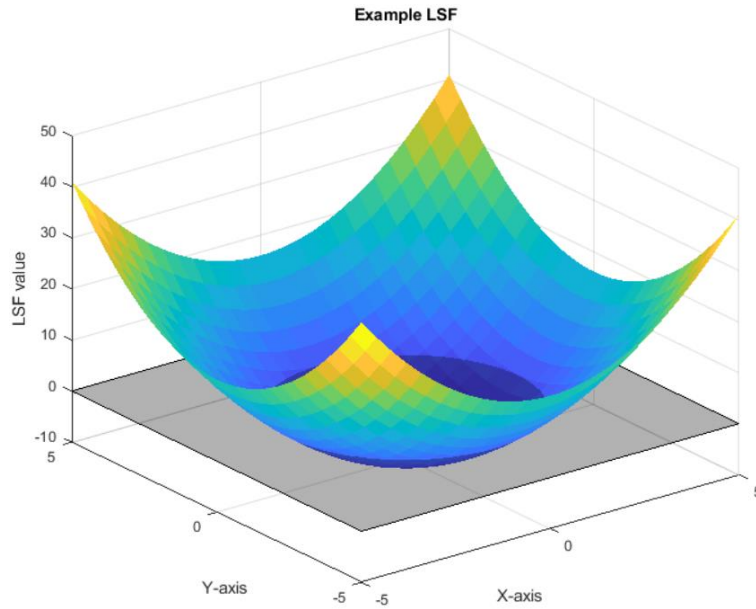


Figure 3-5: Example Level-Set Function

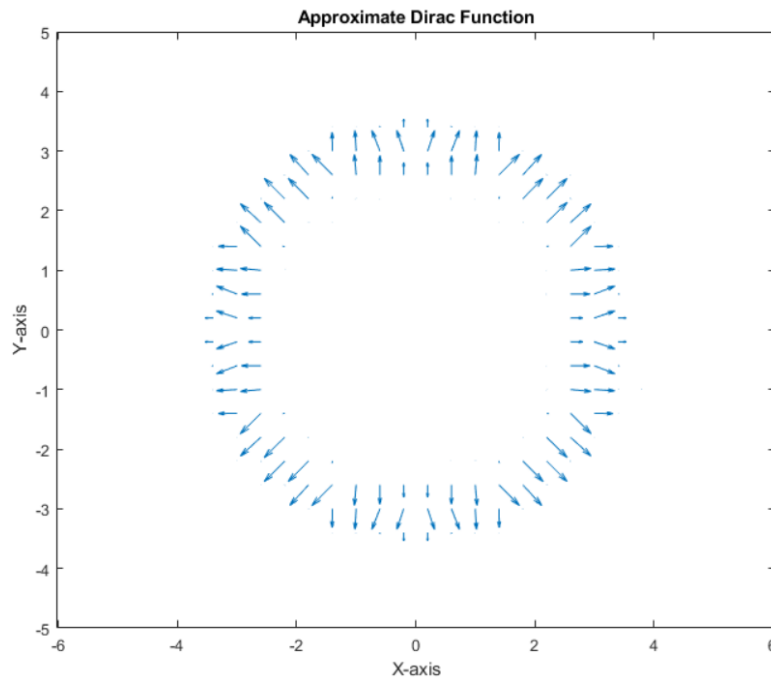


Figure 3-6: Approximate Dirac Function

3.3.2 Three-Dimensional Pressure Vessel Problems

Once the LSM was modified for pressure loading cases in 2-D it was then modified for 3-dimensional internal pressure loading cases. The process that was developed for this is quite different from the 2-D case. Due to the nature of the problem, pressure loading on all internal surfaces, the material distribution can be defined by one LSF. This greatly simplifies the 2-D process by only requiring one Hamilton-Jacobi equation and set of velocities, which are set to the original design velocity of the compliance minimization problem (equation 3.38). In fact, the original LSM formulation performs well initially. However, it has convergence and unstable oscillation issues as the volume fraction nears the target volume fraction, giving rise to a need to modify the LSM to overcome these issues.

The two terms that lead to the velocity field of equation 3.38 stem from the elemental strain energy and the penalty from the volume constraint violation. In the case of internal pressure loading, this compliance component of the velocity will almost always be positive, correlating to adding material in the Hamilton-Jacobi equation. As for the penalty component, the magnitude will reach zero as the volume approaches the target volume fraction. This occurrence is amplified if the relative change in constraint violation between iterations is larger than the scaling factor, α , on the Lagrange multiplier, λ , equation 3.39. Thus, this combination of events leads to the update procedure adding too much material suddenly as the penalty term decreases. Once this occurs, the subsequent iteration will have a much higher than necessary penalty term as the Lagrange multiplier is much larger compared to when the algorithm initially hit that relative volume fraction due to the continuous increased scaling, equation 3.39. This then causes the algorithm to drastically remove material and this process is repeated as the algorithm oscillates and becomes unstable. This concept is illustrated in the volume versus

iteration plot shown in figure 3-7. Intuitively, the solution to this problem requires the penalty term to only be modified based on the constraint violation as opposed to being completely recalculated based solely on the current volume fraction. One penalty application method found in the literature that acts as such can be found in the works of Wei et al. [43]. Here an increasing multiplication factor, γ , multiplies the constraint violation, which is then added to the previous iteration's Lagrange multiplier.

$$\lambda_i = \lambda_{i-1} + \gamma_i(V_i(x) - V_{req}) \quad (3.51)$$

$$\gamma_{i+1} = \max(\gamma_i + 0.05, 5) \quad (3.52)$$

Where the increase of γ is linear and capped to a value of 5.

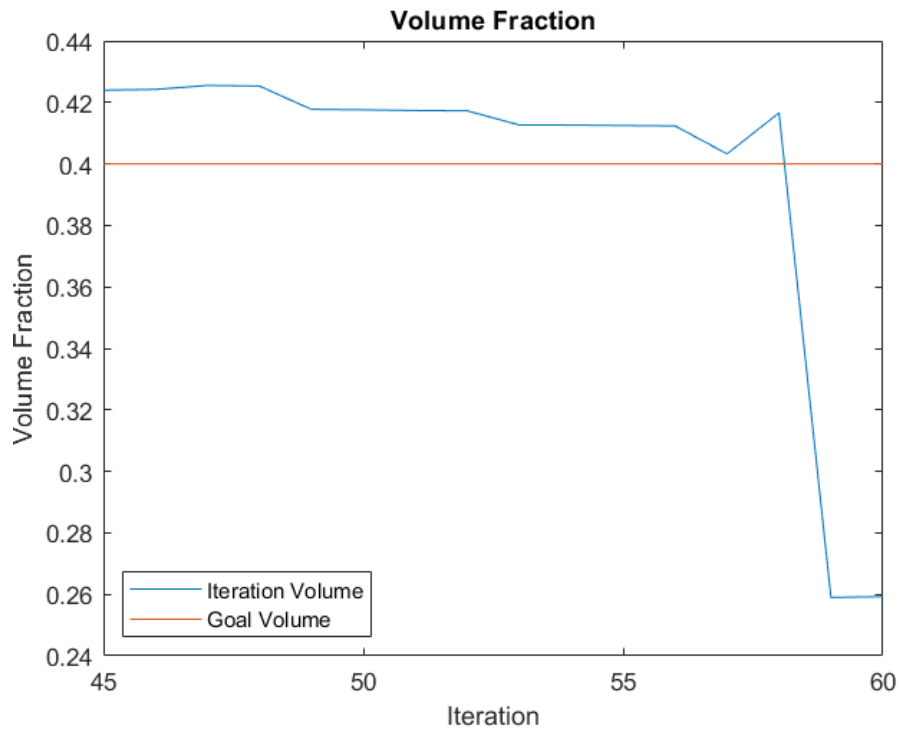


Figure 3-7: Drastic Change in Constraint Violation

A second issue commonly found in the 3-D pressure vessel problem is in the event the volume constraint flips signs. For example, if the structure has a volume fraction near the target value and a small change in the structure occurs such that the volume fraction is just on the other side of the target volume. With regards to the velocity calculation, equation 3.38, for this example, this event causes the ensuing iteration to have similar values for the first term in the velocity equation, derived from the compliance, while having a drastic change in the penalty term. This situation leads to an undesired update of the LSF. This occurrence can be shown in the plot of volume versus iteration found in figure 3-8. Although the use of a penalty formulation such as the one in equations 3.51 and 3.52 mitigates this situation from occurring, there still arises convergence issues, particularly if the volume fraction rapidly approaches the target as the ‘momentum’ tends to continue removing material when not desired.

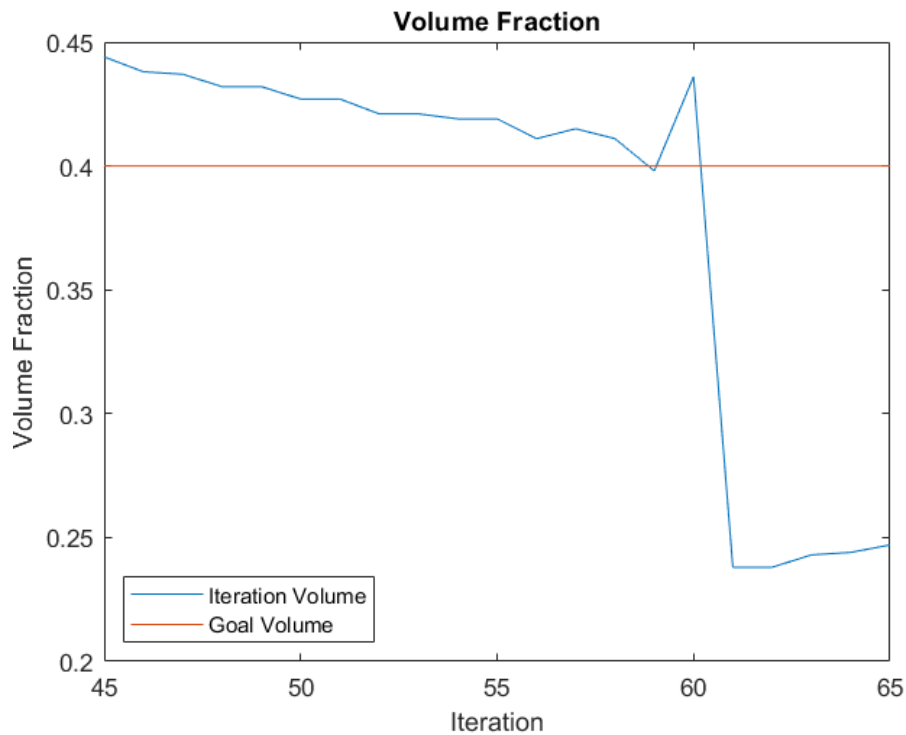


Figure 3-8: Volume Crosses Target then goes Unstable

To solve the convergence issues when optimizing a 3-D pressure vessel, a correlation was made between the different methods of applying penalties and the concept of ‘Proportional, Integral and Derivative control’ (PID). The original penalty term defined for the LSM in equation 3.38 closely resembles that of proportional control as the difference in current volume fraction and target volume fraction are multiplied by the Lagrange multiplier. The penalty formulation by Wei et al. [43] can be viewed as the summation over the iterations of the volume constraint violation multiplied by the iteration’s Lagrange multiplier. This mirrors the definition of integral control. Finally, an intuitive method to aid against the second common issue mentioned above is to add a predictive term, echoing derivative control. All of these are combined to form a PID-type penalty formulation to be used as the volume fraction approaches the target volume. The three terms for the proportional, integral, and derivative violations can be written as shown in equations 3.53-3.55.

$$Proportional = V_i - V_{req} \quad (3.53)$$

$$Integral = \frac{1}{n} \sum_{a=0}^{n-1} V_{i-a} - V_{req} \quad (3.54)$$

$$Derivative = 2V_i - V_{i-1} - V_{req} \quad (3.55)$$

Where n is a positive integer defining the number of previous iterations the integral term uses. Here the derivative term uses a finite difference approximation with the previous iteration and forecast one iteration using this slope. Each of these terms is then applied to a unique scaling factor, K_P , K_I , and K_D , which can be modified for tuning purposes. Then these are summed to make the total control term for the iteration. Similarly to equation 3.51, this control term is then

added to the previous iteration's penalty to determine the penalty term that is to be applied to the current iteration.

(3.56)

$$Penalty_i = Penalty_{i-1} + \lambda_i \left[K_P(V_i - V_{req}) + K_I \left(\frac{1}{n} \sum_{a=0}^{n-1} V_{i-a} - V_{req} \right) + K_D(2V_i - V_{i-1} - V_{req}) \right]$$

Combined with the compliance sensitivity found in equation 3.36, the design update velocities from equation 3.38 can be written as:

$$v|_e = \mathbf{u}_e^T \mathbf{k}_e \mathbf{u}_e - Penalty_i \quad (3.57)$$

The use of this method has prevented unstable oscillations and improper convergence behavior in implementing the LSM to topologically optimize a 3-dimensional pressure vessel. This effect can be seen in the volume versus iteration plot found in figure 3-9. Note the use of the original penalty formulation for the initial iterations until the volume nears the target volume. This change is apparent where the volume fraction levels off and has a slight bump up as the integral and derivative control kick in.

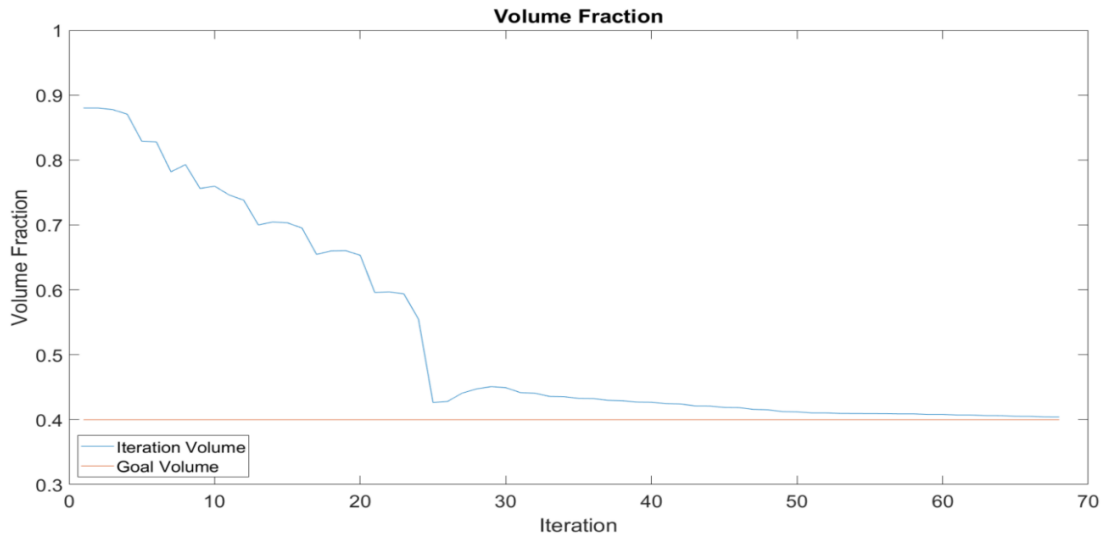


Figure 3-9: Volume Fraction with PID-type Penalty

Chapter IV: Implementation

This chapter discusses the physical implementation of how the methods previously established have been executed resulting in the optimization of an irregular shaped pressure vessel. To ensure understanding of the Level-Set Method, first a standard \mathbb{R}^2 and \mathbb{R}^3 static loading topology optimization code was created. Then, a \mathbb{R}^2 code to handle design dependent pressure loading was written before expanding it to a generic rectangular prism in \mathbb{R}^3 . Here is where the procedure for optimizing a pressure vessel was established while maintaining a simple geometry. Finally, this code was modified to handle an irregularly shaped domain. This chapter discusses this progression along with the details of this final phase. To address such a problem, two codes have been developed. The first code interprets an STL file to develop a mesh for the domain. Once the mesh has been created, it is used in a second code that implements the Level-Set method to determine a structural design that minimizes compliance while achieving a specific volume fraction.

The optimization code can be broken down into two main parts: an initialization phase, and an optimization loop that executes the topology optimization itself. The code strongly follows the 2-D discrete implementation of the Level-Set method in MATLAB by Challis [13], with occasional references to a LSM using Radial Basis Functions by Wei et al. [43]. Extensions to 3-dimensions were aided by references from the 3-dimensional SIMP implementation in MATLAB by Kai et al. [7]. These codes were built upon to be able to handle a random domain as well as internal pressure loading.

This chapter is organized as follows: section 4.1 summarizes the progression of the research as the problem was broken down, section 4.2 overviews the mesh generation code,

section 4.3 covers the initialization phase of the optimization code, section 4.4 addresses the implementation of the LSM within the optimization loop of the code, and finally section 4.5 summarizes the code and introduces the appendices.

4.1 Problem Progression

Once it was determined to utilize the Level-Set method to optimize an irregular pressure vessel, a basic understanding was developed by developing code to optimize a 2-D domain subjected to static loading conditions. This was done mirroring the works of Challis [13] and Wei et al. [43]. Two separate LSF parameterizations and geometry mapping were done to compare and deepen the understanding of the method. The first utilized linear basis functions where the discretization of LSF control points coincided with the FEA mesh, thus resulting in a discrete level-set method, limiting the elements to merely 'on' or 'off'. Increasing complexity, the second used radial basis functions and a density-based geometry mapping. This allows for a smoother structural representation and changes in response. To test and ensure robustness across a variety of structures, a user interface was developed to input problem parameters and adjust LSF parameters. This interface can be seen below in figure 4-1 where the number of elements in both x and y direction, the loading conditions, the Dirichlet boundary conditions along with the LSM parameters of step length, topological weighting factor and volume fraction constraint can all be defined.

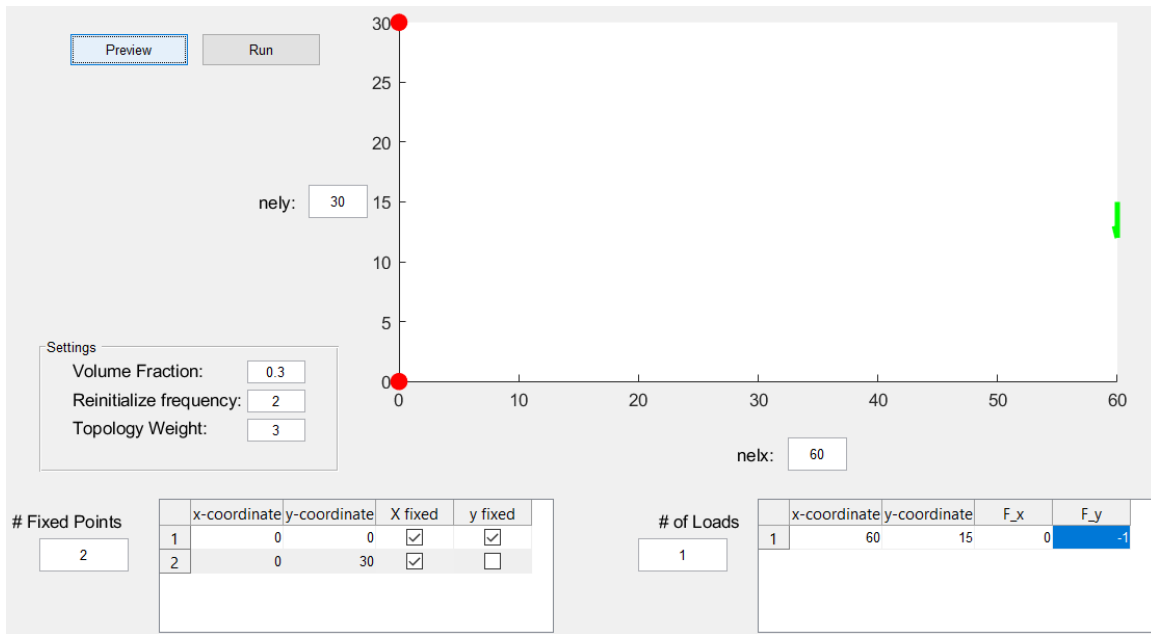


Figure 4-1: 2-D LSM Input Interface

After establishing this generic 2-dimensional topology optimization, the problem was further developed to account for design dependent pressure loads. Two methods were used to do this. The first uses SIMP, a density-based topology optimization, by following the works of Edmund and Lee [52] as explained in section 2.3. For the second method, the works of Xia et al. [47], [50] as discussed in sections 2.3 and 3.3.1 were mirrored. Here two separate LSFs were implemented to model both the homogeneous boundary and the Neumann boundary, each of which were subjected to their own Hamilton-Jacobi equations. To prevent boundary crossing, the velocity modification method discussed in 3.3.1 was used. Additionally, the pressure force was calculated based on an approximate Dirac function for the level-set function as established in equations 3.49 and 3.50. For this phase, a rectangular domain was used with a pressure loading applied from the bottom side and pinned boundary conditions applied on both the lower left and right corners. Figure 4-2 illustrates this design problem.

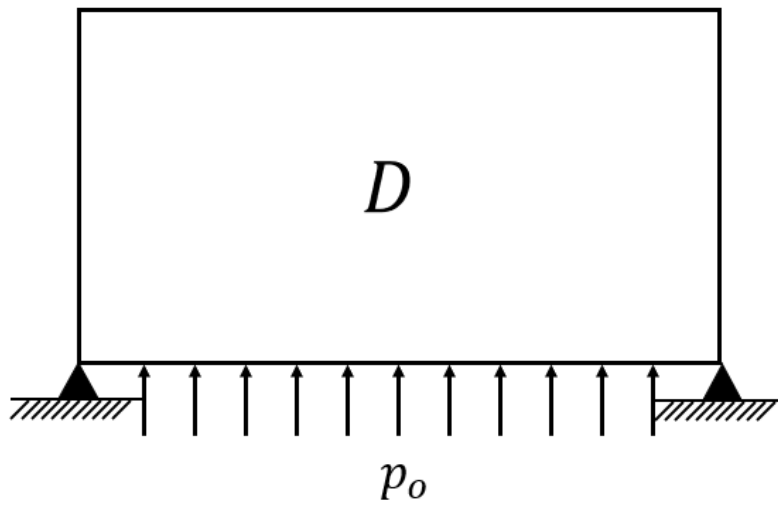


Figure 4-2: 2-D Pressure Problem Definition

At this point, the domains were shifted to 3 dimensions. As in the 2-dimension problems, first problems with constant loading conditions were solved then the progression to pressure loading was done. Initially, internal pressure loading was applied to a rectangular cuboid domain, as shown in figure 4-3 below where the left image shows the initial geometry and the right shows the deformation plot of this geometry. In the left image, the outer boundary is shown by a transparent orange so that the structure itself can be visualized by the void elements plotted as purple.

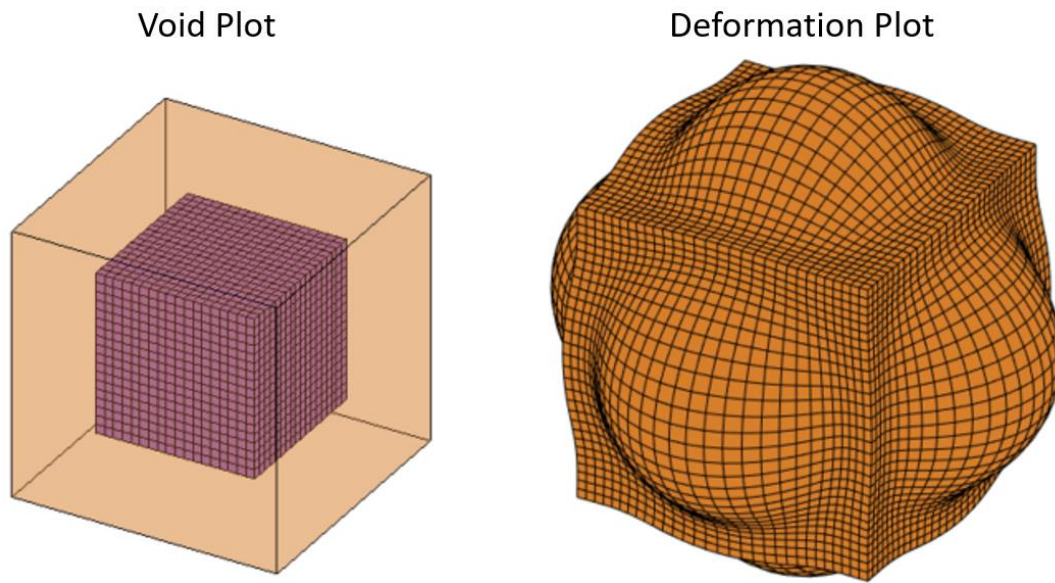


Figure 4-3: 3-D Pressure Problem Definition

During this phase, the formulation of the PID-type penalty, discussed in section 3.3.2, proved to be effective. Here a fixed Eulerian mesh is used where all elements are of equal cuboid size and remain as such throughout the optimization. The next progression involved implementing the methods established here towards an irregularly shaped design domain as opposed to the rectangular cuboid shown in figure 4-3. This final phase is discussed in further detail throughout the remainder of this chapter as it involves the conversion of an STL file into a voxelated mesh, discussed in section 4.2 and the implementation of the level-set method for optimization discussed in section 4.3.

4.2 Mesh Generation

To establish the irregular shape that is to be optimized, an STL file of the part is converted to a finite element mesh that can be used by the optimization and finite element analysis codes. An STL part is defined by a series of triangles that form the outer boundary of the

component. Each corner of a triangle is defined by a node with x, y, and z coordinates. In order to differentiate the interior from the exterior of the part, an outward normal vector for each triangular shape is also provided. An example of how this information is presented in the ASCII STL file is shown below.

```
...
facet normal 9.753949e-02 8.394471e-01 5.346163e-01
  outer loop
    vertex 1.472648e+02 2.135673e+02 1.981356e+02
    vertex 1.472694e+02 2.134192e+02 1.983673e+02
    vertex 1.472710e+02 2.133868e+02 1.984179e+02
  endloop
endfacet
facet normal 8.069924e-02 8.404727e-01 5.358109e-01
  outer loop
    vertex 1.472710e+02 2.133868e+02 1.984179e+02
    vertex 1.472694e+02 2.134192e+02 1.983673e+02
    vertex 1.472699e+02 2.134033e+02 1.983922e+02
  endloop
endfacet
...
```

Here the 'facet normal' defines the x, y, z components of the outward normal for the triangle that is defined by the 3 'vertex' below. Each of these vertices then provide their x, y, z coordinates. This section from 'facet normal' to 'endfacet' is then repeated for each of the triangle surfaces that define the part's outer geometry.

The code 'MakeMesh.m' converts the assigned STL file into an array of elements each with the lengths [l_x l_y l_z] defined by the user in the variable 'voxelsize'. Using this element size along with the maximum and minimum vertex values in each direction, 'ranges', the maximum possible number of elements in each direction is defined. With this information, a 3-D matrix, 'cells', of size equal to the maximum elements in each direction is constructed and filled with the value of -1. The indices of this matrix represent each possible hexahedral element of the mesh. This matrix will be modified such that each element of the matrix

designates the on-off nature of the element in the part. A value of 1 refers to there being material and thus the element will become part of the mesh, whereas a value of -1 denotes void regions that will be excluded from the mesh. Additionally, centroid coordinates for each of these possible elements is established.

To formulate this 'cells' matrix, each triangular face of the STL with a z component in its normal is evaluated to determine which x and y centroids lie within the triangular face projected onto the xy -plane. Each of the centroid coordinates that intersects this projection is then evaluated to determine the z -value the face has at that particular x, y coordinate. As illustrated in figure 4-4, this process identifies the x, y centroid coordinates, represented by the red arrows, that would intersect the given STL triangle, represented by the red triangle, projected in the z -direction. Then the z -value of this intersection is determined.

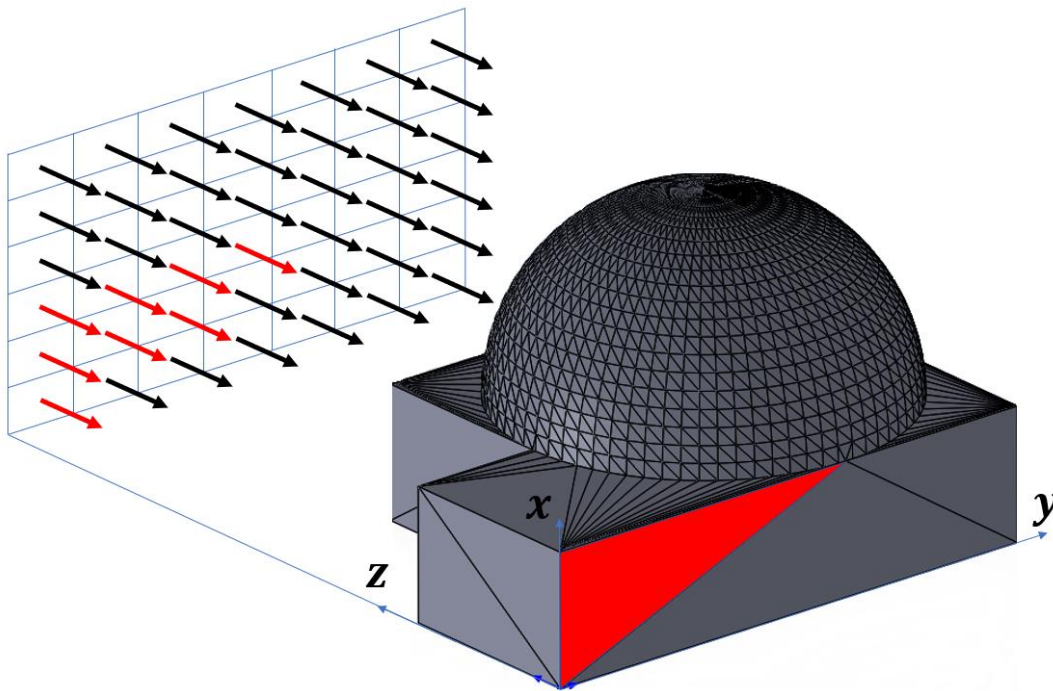


Figure 4-4: STL Projection

From here, all indices of the 'cells' matrix corresponding to these x and y coordinates with z-values greater than this intersection are multiplied by -1. Therefore, if the face is the 1st face crossed in the projection, all cells after it will be turned on, 1, and all prior cells will remain off, -1. Then if a particular face is the second or final face crossed, all the prior cells will remain the same, and the following cells will be turned back off. This concept for an arbitrary yz-cross section at any x value is illustrated in figure 4-5 below, where the boxes represent the indices of the 'cells' matrix and the red lines represent the intersection of the STL surfaces and the cross-section.

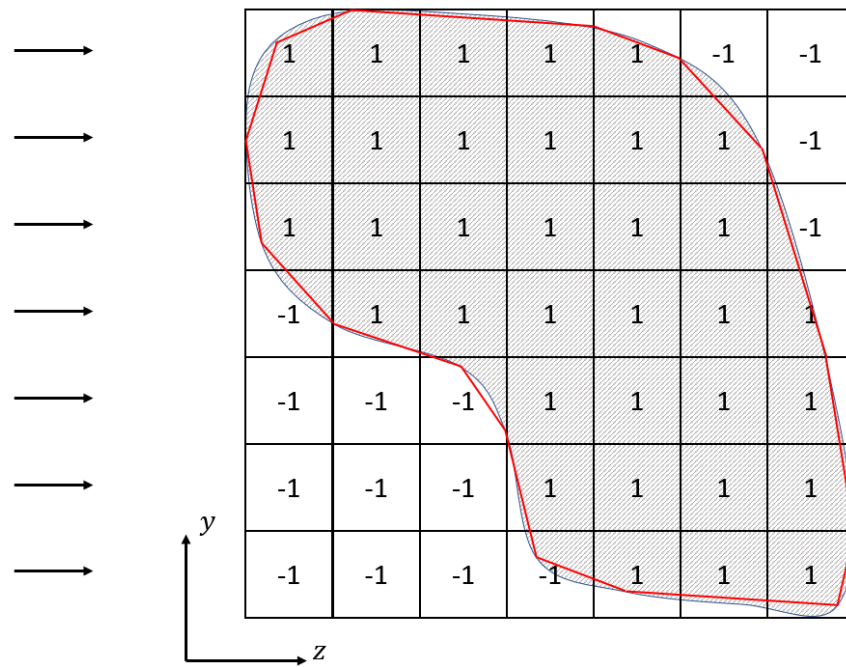


Figure 4-5: YZ-Cross Section Projection

This process of generating the appropriate 'cells' matrix is done between lines 14 and 50 of the script 'MakeMesh.m', found in appendix B.

In optimizing a pressure vessel, it is critical to keep the outer boundary solid, as to maintain the component's ability to be a pressure vessel. This is done immediately following the completion of the 'cells' matrix (lines 55 through 58) by summing all of the neighboring indices of the matrix via a convolution with a 3x3x3 matrix of ones, therefore if all values of 'cells' are positive one within a 3x3x3 matrix centered at a given location, this convolution would produce a value of 27. Then boundary elements can be defined as values of this convolution less than 27, with a 'cells' matrix value of 1, as shown in the code below.

```
cells=permute(cells,[2,1,3]);
outer=(cells==1).* (convn(cells,ones(3,3,3),'same')<27);
outer(cells(:)==-1)=[];
Boundary=nonzeros(outer(:)'.*(1:nnz(cells==1)));
```

Once each possible element is deemed on or off via the 'cells' matrix, the actual list of elements and nodes has to be generated, characterized in the variables 'elements' and 'nodes'. Here 'elements' contains a row for each element of the mesh and 8 columns for each node number of the given element. These node numbers are ordered such that for the given element they follow the relative positioning as shown in figure 4-6. As for the 'nodes' variable, each row correlates to the particular node number referenced in 'elements', and each of these rows contains three columns for the x, y, and z coordinates of the node. Despite the physical translation of the geometry in the STL file, these nodes start at the origin, such that if the first index of 'cells' was 1, its first node would have the coordinates (0,0,0).

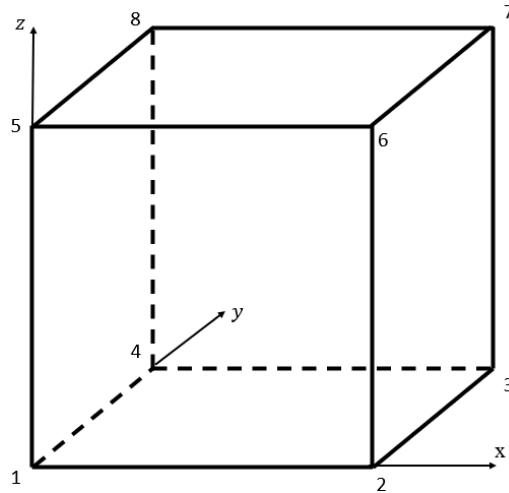


Figure 4-6: Node Relative Positioning

To generate these lists of nodes and elements, the indices of 'cells' that have a value of 1 are iteratively considered to make an element. The nodes for the element are then determined, and if the node already exists in the 'nodes' matrix, that node number is used in the 'elements' matrix, otherwise a new entry to the 'nodes' matrix is made and used. This process is executed in lines 73 through 89. Once this is complete, the mesh is plotted to confirm correct operation and that the chosen 'voxelsize' sufficiently captures geometric features for the user. Then the variables 'elements', 'nodes' and 'boundary' are saved to be used in the topology optimization of the domain.

4.3 Optimization Initialization

During the initialization phase, the material parameters, optimization parameters, and initial geometry are setup along with a few 'book-keeping' items. The material properties for the modulus of elasticity and poisson's ratio are saved as variables 'E' and 'nu' with values of 29.5×10^6 and 0.29 respectively as these are the material properties for Inconel718, a known

3D printed metal for high pressure and life support devices. When doing nominal runs ‘E’ and ‘nu’ were saved as 1 and 0.3 respectively. The level-set parameters that are defined here are:

- `volReq=0.45`: The volume fraction goal for the topology optimization
- `stepLength=2`: The number of ‘Courant-Friedrichs-Lewy’ (CFL) time steps the evolution equation is solved at each iteration, this is explained further in the update procedure section
- `numReinit=2`: The frequency at which the LSF is reinitialized, a value of 2 refers to the LSF being reinitialized every other iteration
- `max_itr=200`: The maximum number of iterations that will be executed before the program aborts the loop if a convergence criterion has not been established yet
- `La=1/2`: The initial Lagrange multiplier for the first phase of the optimization, λ_1 from equation 3.38
- `La2=1/10`: The initial Lagrange multiplier for the second phase of the optimization once the penalty formulation has switched from just proportional to a PID-type penalty, λ_i from equation 3.56
- `alpha=1/0.95`: Scaling factor for Lagrange multipliers, equation 3.39
- `PID=[1, 0.5, 0.1]`: The gains applied to each portion of the PID-type penalty, $[K_p \ K_I \ K_D]$ from equation 3.56

From the paper presenting the 2-D discrete LS topology optimization code written in MATLAB by V.J. Challis [13], the suggested values for the frequency of reinitialization are between 2 and 6. The justification behind the range is that if the number is too small, no new holes can nucleate in the design, and if too large, the LSF becomes very steep, leading to poor accuracy when

solving the evolution equation. This same paper [13], suggest that the 'stepLength' variable be set between the minimum number of element in a coordinate direction divided by 10, and the maximum number of elements divided by 5. These recommendations were for a 2-dimensional case, claiming that if the number was too low, the design change will be slow and converge to a poor local minimum, and if the number was too large, the design will change rapidly with the possibility of removing material from important supporting features. During trials of the 3-D pressure box, numbers on the upper end of this range led to severe oscillations and therefore the number was set very low, which seemed to help the convergence. The use of the Lagrange multipliers and the 'PID' variables to better control some of these issues is further explained in the update procedure.

For both cases in 3-D, the geometry is defined by a series of hexahedral (box) elements comprised of 8 nodes each. For irregular shapes, a discretization of the domain converts geometry from an STL file into defined elements and nodes, 'MakeMesh.m' section 4.2, otherwise a patterned discretization is established prior to the optimization. Because this geometry is voxelated so that every element is exactly of the same size and shape, the elemental stiffness matrices, established in equation 3.25, are all equivalent and can be calculated once prior to the optimization loop in the subfunction 'stiff3D(E, nu, Esize)'. Using the relative local element node order illustrated in figure 3-1 used for the finite element method and the mesh generation process from section 4.2, the 'elements' matrix has one row for every element and 8 columns for each node corresponding to the element's local node positioning. An example of the first few elements are shown in table 4-3 below. The 'nodes'

matrix contains a row for each node of the mesh and 3 columns for the x, y, and z coordinates of the node, an example of the first few nodes are shown in table 4-4.

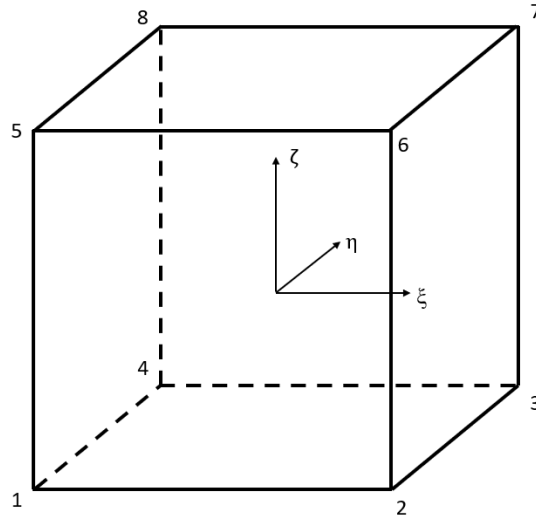


Figure 3-1: Hexahedral Master Element

Table 4-3: 'elements' matrix format

Element #	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6	Node 7	Node 8
1	1	2	3	4	5	6	7	8
2	2	9	10	3	6	11	12	7
3	13	4	14	15	16	8	17	18
4	4	3	19	14	8	7	20	17
5	3	10	21	19	7	12	22	20
6	15	14	23	24	18	17	25	26

Table 4-4: 'nodes' matrix format

Node #	X-coordinate	Y-coordinate	Z-coordinate
1	2	2	0
2	2.25	2	0
3	2.25	2.25	0
4	2	2.25	0
5	2	2	0.25
6	2.25	2	0.25
7	2.25	2.25	0.25
8	2	2.25	0.25
9	2.5	2	0
10	2.5	2.25	0
11	2.5	2.25	0
12	2.5	2.25	0.25

Once the mesh is imported, an initial geometry with an array of voids can be formulated, using the subfunction 'InitialStruc'. This array of voids is determined by a user defined 3x3 matrix 'init' , which for example can be set to $[4, 4, 4; 3, 3, 3; 10, 10, 10]$. The first row of this matrix, $[4, 4, 4]$, denotes the size of each initial void in the three coordinate directions. The second row of this matrix, $[3, 3, 3]$, denotes the gap between each initial void in each direction, and the final row, $[10, 10, 10]$ represents how many times this void and structure pattern are repeated in each direction. This subfunction 'InitialStruc' uses the imported mesh data of 'elements', 'nodes', and 'boundary' along with the variable 'init' to generate the following outputs:

- **struc**: True/false matrix of material distribution
- **Esize**: $[l_x \quad l_y \quad l_z]$ size of the elements in the mesh
- **map**: A vector specifying the index of the level-set function belonging to each element

- `noF`: List of degrees of freedom belonging to the homogeneous boundary where no forces should be applied to
- `exterior`: List of indices of '`struc`' that lie outside of the design domain

In implementing these initial voids, this array is centered and trimmed to ensure that all boundary elements start as solid. With all of this, the initial geometry is created and stored into a 3-D matrix saved as '`struc`'. This matrix has the size of the maximum number of elements in each direction and has a value of '1' if the element has material and a '0' if the element is void. Indices of the '`struc`' matrix that are outside of the imported geometry are set to '1' as well and will stay as such throughout the entire process. Despite not containing material, '`struc`' indices outside the domain are set to a value of 1 to ensure proper LSF values along the outer edges of the pressure vessel when initializing to a signed distance function. Following the execution of the subfunction '`InitialStruc`', a meshed grid of the centroid coordinate for each of the indices of '`struc`' are stored into the variables '`sX`', '`sY`', and '`sZ`' for the three coordinate directions respectively. Additionally, the total volume, stored as '`volTot`', is calculated as the product of the components of '`Esize`' multiplied by the number of elements.

Once the initial structure has been established, the initial level-set function can be computed as a signed distance function. Unlike in the situation with the rectangular cuboid design domain or the initial phases of the irregular shaped domain where the level-set function coincides with the discretization of the meshed finite elements, here the LSF is disjointed and spaced at 1.5 times the size of the elements. LSF kernel values or design variables, ' s_i ' in equation 2.7, are stored in a matrix, '`lsf`'. Shown in equation 4.1, the convention of the level-set function defines any positive value as void and any negative as solid. With this convention and utilizing the image processing toolbox and its '`bwdist`' function, the initialization of the LSF

to a signed distance function can be done by first finding a signed distance function of the structure then linearly interpolating to the grid points of the LSF discretization, shown in the two lines of code below. The function 'bwdist' evaluates the Euclidean distance from each element to the nearest non-zero element. Therefore, the first half of the first line of code evaluates the void regions of the LSF (positive values), and the second term evaluates the LSF for the solid regions which are negative values, thus the subtraction of the terms. The built-in function 'griddata' is used to execute this linear interpolation.

```
lsf=(~struc).*bwdist(struc)-struc.*bwdist(struc-1);
lsf=griddata(sX,sY,sZ,double(sdf),lsfX,lsfY,lsfZ);
```

$$\left\{ \begin{array}{lll} \varphi(X) < 0 & X \in \Omega & 'Material' \\ \varphi(X) = 0 & X \in \Gamma & 'Interface' \\ \varphi(X) > 0 & X \in (D \setminus \Omega) & 'Void' \end{array} \right\} \quad (4.1)$$

Once the level-set function is defined, a list of the LSF indices that lie on the boundary or outside the design domain are saved in the variable 'bearing', which is used later to set the Hamilton-Jacobi velocities of these indices to zero. There are indices of the 'struc' matrix that lie outside of the design domain because the size of the structure matrix is squared off to the maximum number of elements in each direction. Additionally, the matrix 'Hie' is defined to be used during filtering sensitivities from the elements. The conversion from elemental sensitivities to LSF sensitivities is done using the basic filter defined in equation 4.2, similar to density filters used in density-based topology optimization methods.

$$\frac{\partial \widetilde{R}}{\partial s_i} = \frac{\sum_{e \in N_i} H_{ie} V_e \frac{\partial R}{\partial \Omega} |_e}{\sum_{e \in N_i} H_{ie} V_e} \quad (4.2)$$

Where H_{ie} are weighting factors and N_i defines the neighborhood of elements, e , for a particular LSF index, i . Each element having its own volume, V_e , and computed response sensitivity, $\frac{\partial R}{\partial \Omega} \big|_e$. These neighborhoods are defined as:

$$N_i = \{e : \text{dist}(i, e) \leq r\} \quad (4.3)$$

Here the operator $\text{dist}(i, e)$ refers to the Euclidean distance between the center of the e^{th} element and the i^{th} index of the LSF, and r is the size of the neighborhood or filter, set to 1.25 times the LSF discretization spacing. The weighting factor, H_{ie} , is then defined as:

$$H_{ie} = r - \text{dist}(i, e) \quad (4.4)$$

The final portion of the code prior to the optimization loop determines the loading conditions, fixed boundary conditions and the elemental stiffness matrix. Because the topology optimization problem contains pressure loading, the force vector for the finite element analysis has to be computed every iteration as the design changes. Despite this, a value for the nodal force component magnitude of each element that is void is defined as the perpendicular surface area times the nominal pressure value and divided evenly amongst each node on the respective surface of the element, equation 3.27 and 3.28. This is computed as

`'Po=circshift(Esize,1).*circshift(Esize,-1)*Pressure/4'` during this initialization phase, where 'Esize' is a 1x3 vector of the size of each element in the x, y, and z direction respectively and 'Pressure' is the nominal PSI value of the internal pressurized gas. The Dirichlet boundary condition, or fixed degrees of freedom, are determined to be 4 nodes with one fixed in all directions and the other three having a roller boundary condition in each of the three coordinate directions. These three points with roller boundary conditions are each

projected along from the pinned node along the direction they are allowed to deform in. A 2-dimensional representation of this is shown in figure 4-7 below. Because the nodes of the Dirichlet boundary need to belong to elements that contain material, only nodes of elements belonging to the border are considered. The chosen set of nodes and their constraints are shown on the figure and selected as the set that has the maximum distance from the pinned point. This search is done in lines 86 through 102 of the code, but are only executed if the values are not already saved in the loaded mesh file. Once the 4 nodes are determined for the 3-D problem, they are converted to a list of 9 fixed degrees of freedom for the finite element process, 3 for the pinned node and 2 for each of the other three elements with roller conditions, saved as 'fixeddofs'.

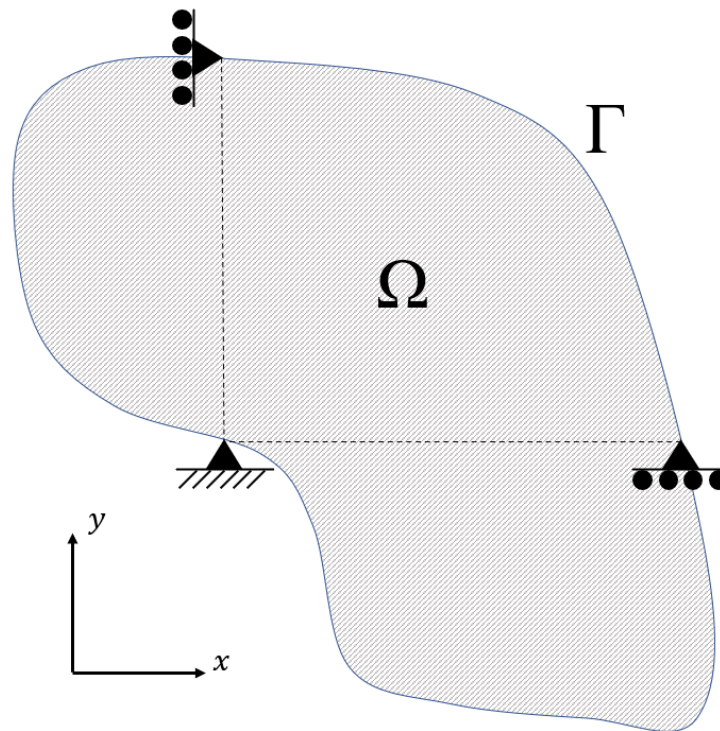


Figure 4-7: Fixed Boundary Conditions

4.4 Optimization Loop

Following the initialization, the code enters the optimization loop to determine the optimal internal geometry. This is done in a while loop until a variable, 'flag', no longer equals to 0. The current iteration number is stored as the counter 'i'. This loop follows the basic flow chart shown below in figure 4-8 and can be broken down into 5 parts: 1 finite element analysis, 2 postprocessing and sensitivity calculations, 3 convergence check, 4 update procedure, and 5 preparation for the subsequent iteration. Each of these will be divided into their respective subsection and explained further in detail. Additionally, a more in-depth flowchart can be found in appendix [A].

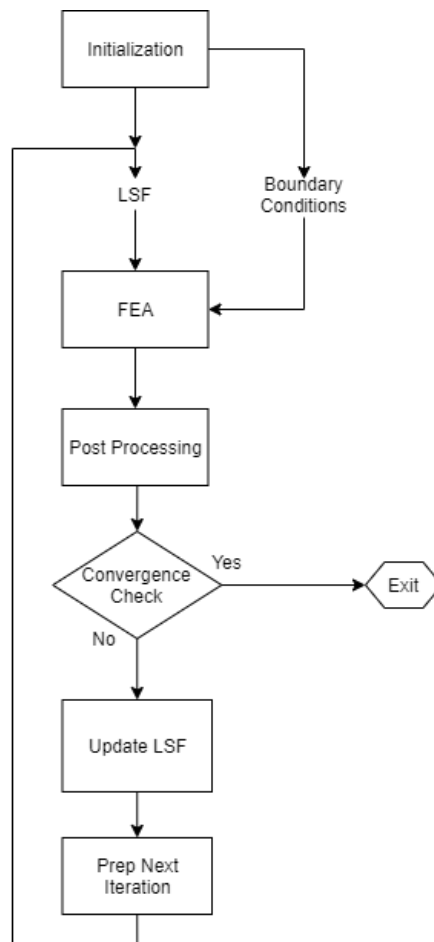


Figure 4-8: Basic Flow Chart

4.4.1 Finite Element Analysis

At the beginning of each loop of the optimization, finite element analysis is run to determine the displacement values for all of the nodes which will be used to calculate the strain energy densities and sensitivities of each element. To aid in organization, this is all done in a subfunction `'[U, K, F]=FEA_3DP5(struc, elements, map, KE, Po, noF, fixeddofs, oldstruc, oldK, oldF)'`. This function has the following inputs:

- `struc`: The true false matrix of material distribution
- `elements`: The matrix of elements and their corresponding nodes
- `map`: A vector specifying the index of the level-set function belonging to each element
- `KE`: The elemental stiffness matrix computed previously from `'stiff3D'`
- `Po`: The force components applied to each void element's nodes
- `noF`: The degrees of freedom that are on the outer boundary
- `fixeddofs`: The degrees of freedom that are to have no deflection
- `oldstruc`: The previous iteration's structure
- `oldK`: The previous iteration's global K matrix
- `oldF`: The previous iteration's global force vector

and the following outputs:

- `U`: Deflection values for each degree of freedom
- `K`: The global stiffness matrix
- `F`: The global force vector

Within the subfunction, the global stiffness matrix and force vector, 'K' and 'F', are initialized as the previous iteration's, 'oldK' and 'oldF', and then only need to be modified accordingly as opposed to completely recalculated every iteration. First the current structure is compared to the previous iteration's structure (input 'oldstruc') to identify the indices of the LSF that have changed since the previous iteration. Using the 'map' vector, these indices are set to correspond to the elements that have changed, 'ele'. Next, for each element that has changed, the previous iteration's elemental stiffness matrix (Ke_old) and force vector (Fe_old) are computed along with the current iteration's elemental stiffness matrix (Ke) and force vector (Fe). Then at the appropriate indices (dof (ele (i) , :)) of the global stiffness matrix (K) and force vector (F), the previous iteration's elemental stiffness matrix is subtracted out and the current one's added in. This process can be found in lines 54 through 63 of the subfunction and has proved to save orders of magnitude in computational time every iteration that uses the same mesh from the previous iteration, because it eliminates looping through every element of the domain each time during the assembly process. Following the assembly process, the force vector components along the homogeneous boundary, the exterior surface of the pressure vessel, are set to zero with the 'noF' index list, 'F (noF) =0'. Once the global stiffness matrix and force vectors are computed, the fixed degrees of freedom are applied via the partitioning method, using the following lines of code. The free degrees of freedom 'freedofs' can be computed as:

```
freedofs=setdiff(1:3*numnodes,fixeddofs)
```

Then the remaining system of equations is computed using the standard MATLAB backslash operator as:

```
U(freedofs,:)=K(freedofs,freedofs)\F(freedofs,:)
```


The function outputs the displacement values along with the new global stiffness matrix and global force vector that are to be saved as a starting point for the following iteration.

4.4.2 Postprocessing and Sensitivity Calculations

Directly following the computation of the nodal displacements, the strain energies of each element are computed. As defined in section 3.23.2, the strain energy, C , of the system is equivalent to the summation of the elemental strain energies, as shown in equation 4.5 below, where \mathbf{U} is the deflection vector, \mathbf{K} is the global stiffness matrix, N is the number of elements, \mathbf{u}_e is the elemental deflections, and \mathbf{k}_e is the elemental stiffness matrix.

$$C = \mathbf{U}^T \mathbf{K} \mathbf{U} = \sum_{i=1}^N \mathbf{u}_e^T \mathbf{k}_e \mathbf{u}_e \quad (4.5)$$

Because of the defined LSF to structure relation, equation 4.1, and the sensitivity of compliance computed in equation 3.36, the negative of the strain energies for each element are saved into 'CompE'. This is done via the following lines of code (lines 123-125):

```
for (e=1:numelem)
    CompE(e) = -
    max(struc(map(e)), 0.0001) * U(dof(e,:))' * ke * U(dof(e,:));
end
```

After this, the overall objective, i.e. system compliance, 'obj(i)' is computed as the summation of 'CompE'. Because the structure matrix is defined as 1 where material is and 0 where void, the current volume fraction 'vol(i)' is computed as the sum of the structure matrix that is meshed multiplied by the volume of one element and divided by the total volume calculated during the initialization phase as 'volTot'. This phase of the loop is also where the iteration data is printed to the command window, and, if desired, plots are created and saved into video files.

4.4.3 Convergence Checks

Immediately following this, a quick check for convergence is done if the optimization has done at least 5 iterations. There are two checks for convergence. The first being: is the volume within a specified tolerance (0.003 for this problem) of the target volume fraction, and the previous 5 iterations are all within 5% compliance of the current iteration's? The second check for convergence is if the iteration counter, 'i' has reached the maximum allowed iterations, 'max_itr'. These convergence checks are done between lines 134 and 141.

4.4.4 Update Procedure

Once convergence is checked and it is determined that the optimization procedure needs to continue, the LSF is updated for the following iteration. To do this, design velocities, equations 3.38 and 3.56, are computed based on the objective function and constraints. Then the LSF can be updated via the Hamilton-Jacobi equation (equation 3.34) and these velocities. The mathematical description of a minimum compliance structure subjected to pressure loading can be found in equation 3.42. Following the process in section 3.2, the Lagrangian derived in equation 3.32 and associated design update velocities for the Hamilton-Jacobi equation, found in equations 3.38 and 3.57, are restated as:

$$L = \left(\sum_{i=1}^N \mathbf{u}_e^T \mathbf{k}_e \mathbf{u}_e \right) + \lambda_i (V(x) - V_{req})^2 \quad (3.32)$$

$$v|_e = -\frac{\partial L}{\partial \Omega}|_e = \mathbf{u}_e^T \mathbf{k}_e \mathbf{u}_e - \lambda_i (V(x) - V_{req}) \quad (3.38)$$

$$v|_e = \mathbf{u}_e^T \mathbf{k}_e \mathbf{u}_e - Penalty_i \quad (3.57)$$

Using the filtering scheme from equation 4.2 with the weighting factors in 'Hie', the elemental velocities can be converted to LSF velocities. As discussed in section 3.3.2, equation 3.38 is used

initially, and then 3.57 is used once the optimization approaches the target volume. Recall this term ' $Penalty_i$ ' in equation 3.57 is defined as:

(3.56)

$$Penalty_i = Penalty_{i-1} + \lambda_i \left[K_P(V_i - V_{req}) + K_I \left(\frac{1}{n} \sum_{a=0}^{n-1} V_{i-a} - V_{req} \right) + K_D(2V_i - V_{i-1} - V_{req}) \right]$$

The Lagrange multipliers for the i^{th} iteration, λ_i , are stored as 'La' and 'La2' for equations 3.38 and 3.56 respectively because once the penalty method switches from the original formulation a separate Lagrange multiplier is used. Following equation 3.39, the Lagrange multipliers starts small, 0.25 and 0.1 respectively, and are updated by a factor α , stored as 'alpha', set to 1.05. Although the Lagrange multiplier for the original formulation, 'La', is updated every iteration, the second Lagrange multiplier, 'La2', is increased by the same factor, α , only upon the volume stalling for 5 iterations. This stalling is defined as 5 consecutive iterations with less than a 0.005 change in volume fraction.

The first term of the design velocities for both equations 3.38 and 3.57 come from the compliance term, being the individual components of the 'CompE' calculated in the post processing section as shown in equation 4.5. Then these terms are converted for the LSF using equation 4.2 and stored as 'shapeSens'. As discussed in section 3.3.2, the second term for these velocities serves as a penalty based on the volume constraint, with an original formulation shown in equation 3.38 and the PID-type scheme in equation 3.56. This original scheme is utilized until the volume enters a specified range of the required volume. This range is set to ± 0.05 . From this point on, the penalty term is computed following the PID-type scheme. The final penalty term prior to entering this range is saved to a vector 'Control'. Then each

iteration attempts to track the progression of the volume fraction and modify this control value accordingly. The gains, $[K_p \ K_I \ K_D]$, serve as scaling factors between the proportional, integral, and derivative terms respectively. Within the code, these factors are stored in the vector 'PID' and set to $[1, 0.5, 0.1]$. The computation of the design update velocities is done in lines 144 through 164, which are shown below.

```
%Update Procedure-----
if(abs(vol(i)-volReq)<0.05)
    relax=1;      %Stop relaxed penalty if within volume band (0.1)
end
if(relax==0)      %Execute relaxed penalty
    La=alpha*La;
    Penalty=La*(vol(i)-volReq);
    Control=[];
    Control(i)=Penalty;
else
    if(max(vol(max(1,i-5):i))-min(vol(max(1,i-5):i))<0.002&&i>5)
        La2=alpha*La2;%Update Lagrange mult on PID if volume
hasn't changed
    end
    Control(i)=La2*PID*[ (vol(i)-volReq); ...
        ((sum(vol(max(1,i-4):i))/numel(max(1,i-4):i))-volReq); ..
        (2*vol(i)-vol(max(1,i-1))-volReq) ];
    Penalty=sum(Control);
End
shapeSens=reshape((Hij*CompE)./max(sum(Hij,2),0.0001),LSFsize);
SensTotal=(shapeSens/max(abs(shapeSens(:))))+Penalty;
```

Here 'SensTotal' contains the design velocities for the Hamilton-Jacobi equation. Note that the 'shapeSens' values are normalized by dividing them by their largest absolute value. Also, the 'Control' term is saved as a vector to allow for its analysis following the optimization to aid in debugging and tuning.

The physical update of the LSF is done in a subfunction 'updatestep3'. In this function, the LSF, sensitivities, step length, element size, and list of elements that cannot change are passed as inputs and the updated LSF along with the new structure serve as the function's

outputs. The first thing that is done in this subfunction is the smoothing or filtering of the velocities. This is done by a 3-D convolution with the matrix ' $[C]$ ' defined as:

$$\begin{aligned} C(:, :, 1) &= \begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} / 27 \\ C(:, :, 2) &= \begin{bmatrix} 1 & 2 & 1 \\ 2 & 3 & 2 \\ 1 & 2 & 1 \end{bmatrix} / 27 \\ C(:, :, 3) &= \begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} / 27 \end{aligned} \quad (4.6)$$

In effect, this takes each term as the weighted average of itself and the neighboring indices that would form a 3x3x3 matrix around it. This ' $[C]$ ' matrix weights the center element with a value of 3, and all of the indices ± 1 in the i, j, k directions a weight of 2. The ' $[C]$ ' matrix is divided by 27 so that the sum of all of the indices equals 1 to make it a true weighted average.

From here, the sensitivities for the locations where the LSF are not supposed to change are set to 0, this list of elements that are locked is found in the variable '`bearing`'. The final step of the update is to apply the Hamilton-Jacobi equation (equation 3.34), which is done in the subfunction '`[struct, lsf] = evolve(v, g, lsf, stepLength, w)`'. Here the inputs '`v`', '`g`' and '`w`' refer to the terms v , g , and w of the Hamilton-Jacobi equation and '`stepLength`' is the number of 'Courant-Friedrichs-Lewy' (CFL), equation 3.41, time steps the evolution equation is solved at each iteration. Note this subfunction incorporates the topological derivative, g , and its weighting term, w , to formulate the reaction term of the Hamilton-Jacobi equation, here these terms are set to 0 for the optimization of pressure vessels. In the code, the time step found in equation 3.40 is stored as '`dt`', and is calculated to be 10% of the stability condition. Then 10 of these timesteps are done for the prescribed value of

'stepLength'. The gradient of the LSF for the Hamilton-Jacobi equation is evaluated with a finite difference scheme. To prevent errors along the perimeter, LSF values are extended to form a border. Then using the 'circshift' command both a positive and negative finite difference in each coordinate direction can be computed. To calculate the update for one iteration, the following lines of code repeat the computation of the Hamilton-Jacobi equation based on the value of 'stepLength'.

```
for(i=1:(10*stepLength))
    dpx=circshift(lsf, [-1,0,0])-lsf;    %Find derivatives on the
grid
    dmx=lsf-circshift(lsf, [1,0,0]);
    dpy=circshift(lsf, [0,-1,0])-lsf;
    dmy=lsf-circshift(lsf, [0,1,0]);
    dpz=circshift(lsf, [0,0,-1])-lsf;
    dmz=lsf-circshift(lsf, [0,0,1]);
    %Update LSF
    lsf=lsf-dt*min(vFull,0).*sqrt(min(dmx,0).^2+max(dpx,0).^2+...
        min(dmy,0).^2+max(dpy,0).^2+min(dmz,0).^2+max(dpz,0).^
        2)...
        -dt*max(vFull,0).*sqrt(max(dmx,0).^2+min(dpx,0).^2+...
        max(dmy,0).^2+min(dpy,0).^2+max(dmz,0).^2+min(dpz,0).^
        2);
end
```

4.4.5 Preparation for Subsequent Iterations

Following the update of the LSF, a few book-keeping items are taken care of in preparation for the following iteration. The majority of this comes in the form of mesh consideration and determining if the structure should be re-meshed or if elements need to be added to the existing mesh. However, prior to this the old structure is saved as 'oldstruc' for the FEA to compare to the new structure. Then a re-mesh determination is considered. This determination is stored in the iterative counter 'mesh' and re-meshing occurs when this counter is zero. To be set to zero the following if statements are considered:

```

if (mesh>=5)
    if (mesh>=8 && max(abs(vol(i-4:i)-volReq))<band)
        mesh=0;    disp('option1');
        band=0.8*band;
        Esize=0.85*Esize;

elseif (numel(setdiff(find((strucoldstruc)==1),[map;exterior]))>=...
        0.3*numelem)

    mesh=0;    disp('option2');
    band=0.15;
    Esize=repelem((prod(Esize)*(sum(struc(map)+...
        numel(setdiff(find((strucoldstruc)==1),[map;exterior]...
        )))/(1.2*numelem))^(1/3),3);
elseif (numnodes>100000)
    mesh=0;    disp('option3');
    Esize=repelem((prod(Esize)*sum(struc(map))/...
        (0.75*numelem))^(1/3),3);
end
end

```

This first ensures that the domain is not re-meshed within 5 iterations of remeshing.

Then three checks are conducted to determine if the domain should be re-meshed. The first of these checks is on the convergence behavior of the volume fraction. Here if the volume fraction is consistently within a range of the goal volume for the past 5 iterations, the algorithm determines remeshing should occur by setting 'mesh' to 0 and assigning a new element size at 85% of the existing size. Additionally, this tolerance band is initialized to 0.15 and is reduced each time the algorithm re-meshes via this criterion. A second criteria for remeshing is if a large number of elements would need to be added to the mesh based on the last evolution of the LSF. This large number of elements is considered to be 30% of the existing number of elements. Here the element size is chosen such that there would be roughly a 10% increase in the number of elements. Finally, the last criteria deeming the need to re-mesh is in the event of too many nodes as this causes the FEA procedure to be too computationally expensive. In this event the element size is chosen such that there would be roughly a 25% reduction in the number of elements.

Following this logic flow to determine if re-meshing should be done, one of three possible processes is executed. These include: 1 re-meshing the domain, 2 reverting to original structure, 3 reinitializing the LSF and adding appropriate elements to the mesh. The process of re-meshing occurs if the variable 'mesh' is zero and is executed in the subfunction 'remesh'. The flow of this subfunction closely resembles the script 'Make_Mesh.m' discussed in section 4.2. However, in this case once the 'cells' matrix is formulated, the LSF is used to generate the 'struc' matrix based on the new element size. Then only the 'cells' that also correspond to material domain in 'struc' are meshed into elements. This process omits void regions from being part of the mesh and having to be modeled as artificially weak material, similar to a conforming mesh or an immersed boundary technique discussed in section 2.2.2. The outputs of the subfunction 'remesh' include new values for:

- `struc`: The new material distribution representation based on the new element size
- `elements`: The new global node to element relations
- `nodes`: The new coordinates for each of the nodes in the mesh
- `map`: The new relation from elements to 'struc' indices
- `boundary`: The new list of elements along the border of the design domain
- `noF`: The new degrees of freedom that are on the outer boundary
- `[sX, sY and sZ]`: The new meshgrid of the coordinate centroids for the indices of 'struc'
- `exterior`: The new indices of the structure that lie outside of the design domain

After the subfunction 'remesh' is called, if there are too many nodes (greater than 100,000) the function is called again with a slightly larger element size. Finally, once the appropriate mesh is conducted, new values are computed for the remaining variables that need to be updated. These variables include: the weighting terms 'Hie', nodal pressure value 'Po', the degrees of freedom matrix 'dof', elemental stiffness matrix 'ke', and the fixed degrees of freedom 'fixeddofs'.

If re-meshing does not occur, the next thing that is checked is if the structure is completely solid. Since the compliance sensitivity is always negative, if the Lagrange multiplier starts too small and the initial void region is too thin, there is a rare chance that in the initial iterations, the update may remove all void elements, making the structure completely solid. This makes the FEA analysis meaningless since there is no force applied on the inside, and results in no displacements nor shape sensitivities, causing the algorithm to never recover or add any void back. To prevent this from occurring, if the structure is completely solid, the code reverts back to the initial geometry and once the Lagrange multiplier is large enough this problem would not happen again, thus highlighting the importance of starting with an appropriate Lagrange multiplier to limit wasted iterations that result in a completely solid structure.

If neither remeshing nor restarting occurred, the LSF is periodically reinitialized and elements are added to the mesh if needed. This periodicity of reinitialization is defined by the user defined variable 'numReinit' and is done so with the following lines of code.

```

if(~mod(i,numReinit))    %reinitialize LSF
    sdf=(~struc).*bwdist(struc)-struc.*bwdist(struc-1);
    lsf=griddata(sX,sY,sZ,double(sdf),lsfX,lsfY,lsfZ);
    lsf(Nanind)=sdf(map(id))-d./Esize(1);
    struc=griddata(lsfX,lsfY,lsfZ,lsf,sX,sY,sZ)<=0;
    struc(map(boundary))=1; struc(exterior)=1;
    clear sdf
end

```

To determine the indices of any elements that need to be added to the mesh for finite element analysis during the subsequent iteration, the following line is used:

```
add=setdiff(find((struc-oldstruc)==1),[map;exterior]);
```

For each of these indices, the new elements and nodes are appended to their respective variables. Additionally, the appropriate entries to 'map', 'dof', and 'Hie' are appended to their stored variables. Finally, to prevent numerical errors with the time saving method of reusing and modifying the previous iteration's global stiffness matrix, the new elements are assembled into 'oldK' as the artificially weak material to simulate the element having already been part of the mesh and modeled as void.

4.5 Conclusion and Appendix Usage

To optimize an irregular shaped pressure vessel defined by an STL file, the geometry is first converted to a voxelated mesh ideal for topology optimization, done so in the 'Make_Mesh.m' script. Then the optimization code takes this meshed domain, applies user defined parameters to establish a topology optimization problem statement to be solved using the Level-Set method and generates an initial void geometry. During the optimization, this void is modified such that a prescribed volume fraction goal is achieved while the overall compliance of the structure is minimized. The first phases allow the user to define various level-set and problem parameters, then prepares the code to enter the optimization loop. The main portion

of the code is done during the second phase, the optimization loop, where the optimal structure is found. This loop is comprised of 5 main components to analyze the response of the structure, evaluate the response, check for convergence, update the LSF, and prepare for the subsequent iteration. As the algorithm converges, the domain is re-meshed to smaller element sizes, omitting the void to reduce computational expenses of the increased number of nodes. This is a viable solution due to the forces being calculated as inward normal throughout the solid domain as opposed to outward normal throughout the void. Additionally, regions omitted during the remeshing procedure can be added to the solid domain by appending the appropriate elements to the mesh as needed.

A detailed flow diagram of this optimization loop can be found in appendix A where each of these main components is identified by dashed boxes. Following this flow chart, the code itself is presented in appendices B-O. Appendix B contains the script for generating the mesh, followed by appendix C containing the main optimization code. Appendices D through J contain all of the subfunctions necessary to execute these two scripts. Appendices K through O contain the script and associated subfunctions to view the structure and stress distribution at chosen cross-sections and iterations. Each of these sections start with a table that lists all the associated variables along with their size and a brief description. In regard to the variables' size, the notation of 'r', 'c', and 'p' refer to an arbitrary number of rows, columns, or pages that may be different on each run of the code. Also, the notation of NSx, NSy, and NSz refer to the number of structural elements in the x, y, and z directions respectively, while NLx, NLy, and NLz refer to the number of LSF kernels in each direction. Additionally, the codes can be found online at: https://github.com/JKremar/Irregular_Pressure_Vessel_Topology_Optimization.

Chapter V: Preliminary Results

This chapter discusses the results that were produced throughout the various phases of the problem evolution prior to solving the topology optimization problem of an irregular shaped pressure vessel. This is done in hopes to provide insight to the development and formulation of the final product and justification for the methodologies and implementation procedures discussed in chapters 3 and 4. Mirroring the problem progression established in section 4.1, this chapter is organized as follows: section 5.1 discusses results from constant loading conditions in both \mathbb{R}^2 and \mathbb{R}^3 , section 5.2 covers the findings from 2-dimensional design dependent loading trials, and section 5.3 introduces the 3-dimensional pressure cases using a rectangular cuboid design domain. Note, for the entirety of this chapter, all deformation plots are magnified for clarity and visibility.

5.1 Constant Loading Conditions

To develop an understanding of the level-set method, standard codes were made to optimize 2-dimensional structures with constant loading conditions. The first of these codes uses a discrete material representation with the discretization of the level-set function coinciding with the finite element mesh. This code was tied to the user interface shown in figure 4-1 to allow for testing of the algorithm and multiple trials at various parameters and starting conditions. One run of the code was to optimize a simply supported beam subjected to a distributed load from the bottom edge, as shown in figure 5-1.

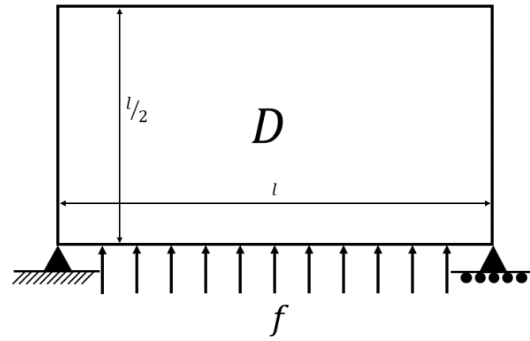


Figure 5-1: Simply Supported Beam with Distributed Loading

When optimizing this structure, nominal values were used for the modulus of elasticity and force loading. A Poisson's ratio of $\nu = 0.3$ was used and the domain was discretized into 100 elements in the x-direction and 50 in the y-direction. The following level-set parameters were used: step length of 3, reinitialization frequency of 2, and a topological sensitivity weighting of 3 for the reaction term on the Hamilton-Jacobi equation. With these parameters and a volume fraction goal of 30%, the structure and deformed structure shown in figures 5-2 and 5-3 were achieved.



Figure 5-2: Distributed Load Optimized Structure

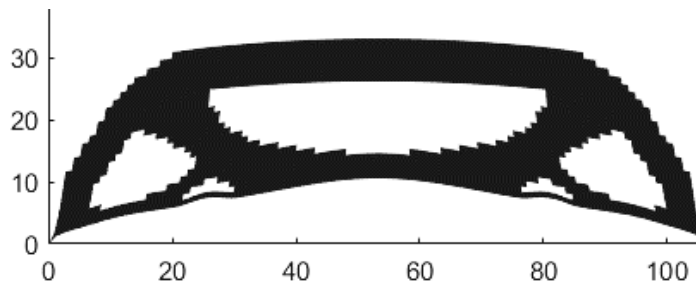


Figure 5-3: Distributed Load Deflection Plot

Figure 5-4 shows the volume fraction and compliance by iteration plots.

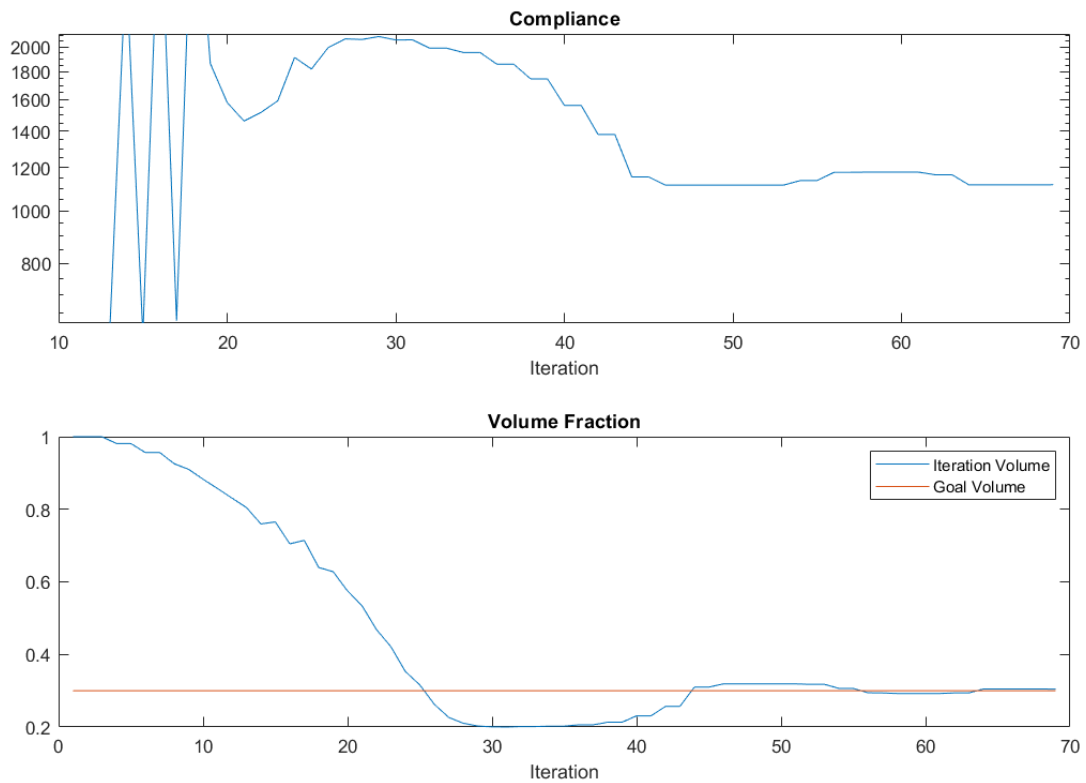


Figure 5-4: Distributed Loading Compliance and Volume Fraction Plots

The second code for 2-dimensional static loading conditions increased the complexity by utilizing radial basis function (RBF) to parameterize the LSF (see section 2.2.1) and a density-based geometry mapping (see section 2.2.3) to allow for the use of intermediate densities for each element cut by the cross-section of the iso-contour of the LSF. This allows for the use of a contour map with much smoother representation of the geometry as opposed to the pixelated results from the discrete implementation shown in figures 5-2 and 5-3. A cantilevered beam, figure 5-5, was optimized using this method with nominal values for the force value and modulus of elasticity, a Poisson's ratio of $\nu = 0.3$, and a volume goal of 35% material.

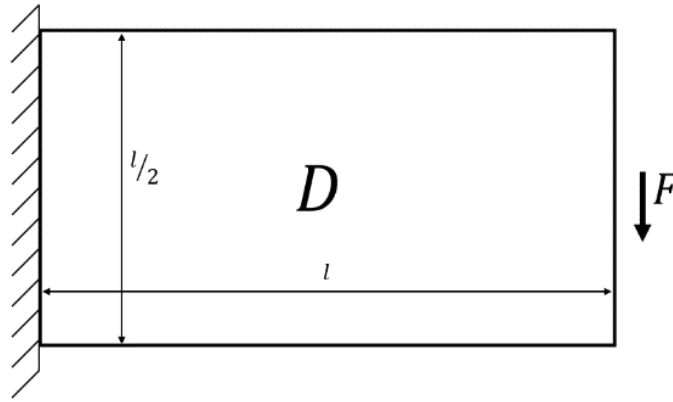


Figure 5-5: Cantilevered Beam Problem

For the implementation, the domain was discretized into a 30x60 square element mesh. In this example there was not a reaction term derived from topological sensitivities used in the Hamilton-Jacobi equation to update the LSF. Because of this, the initial structure, figures 5-6 and 5-7, had a series of holes due to the lack of ability to add holes throughout the optimization.

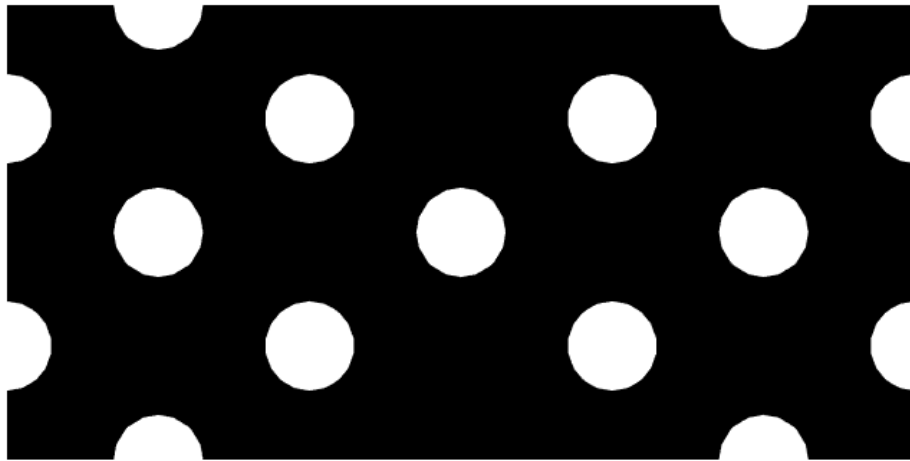


Figure 5-6: RBF Initial Structure

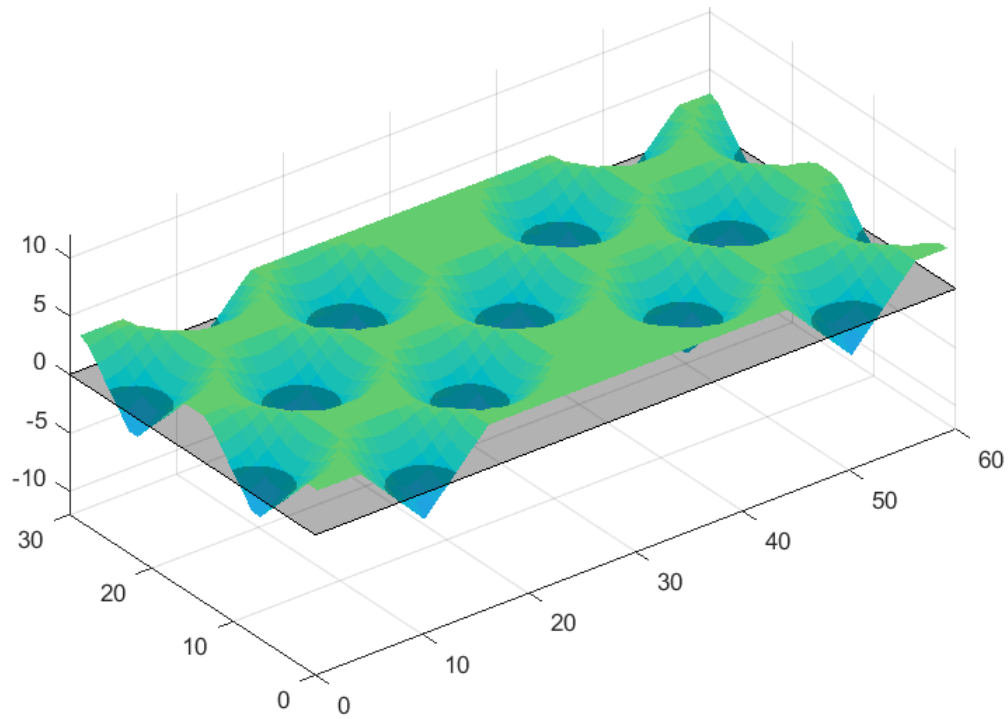


Figure 5-7: RBF Initial Level-Set Function

Additionally, the method of updating the Lagrange multiplier was modified from the method used in the discrete example. Instead, the update method shown in equations 3.51 and 3.52 was used. An intermediate structure and LSF at iteration 60 are shown below in figures 5-8 and 5-9.

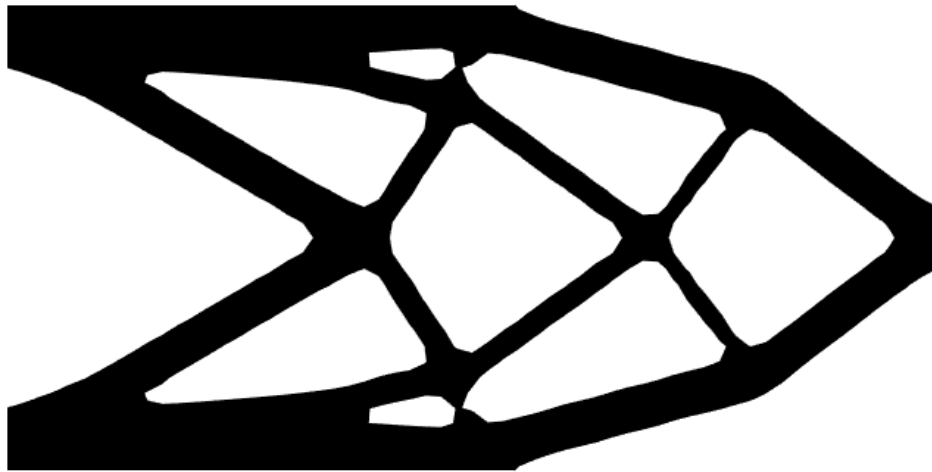


Figure 5-8: RBF Structure at Iteration 60

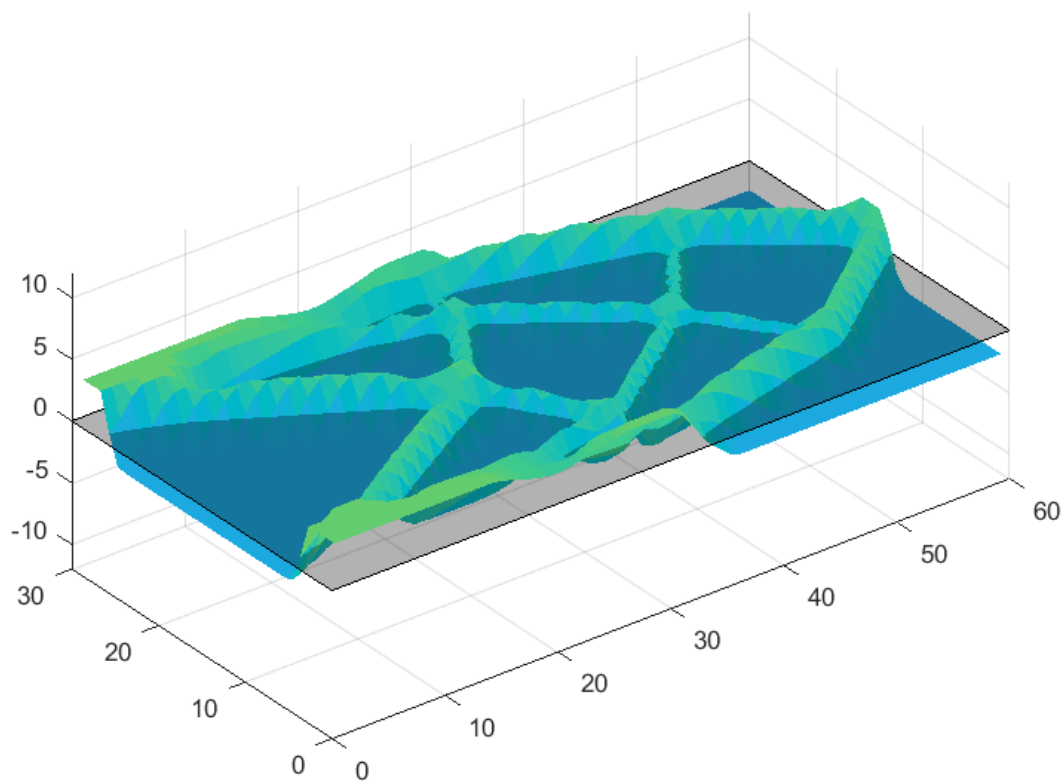


Figure 5-9: RBF Level-Set Function at Iteration 60

Throughout the optimization, the ‘truss’ members on the right side of the figure were phased out as the algorithm added material to the remaining structural members, as shown in the final structure shown in figure 5-10.

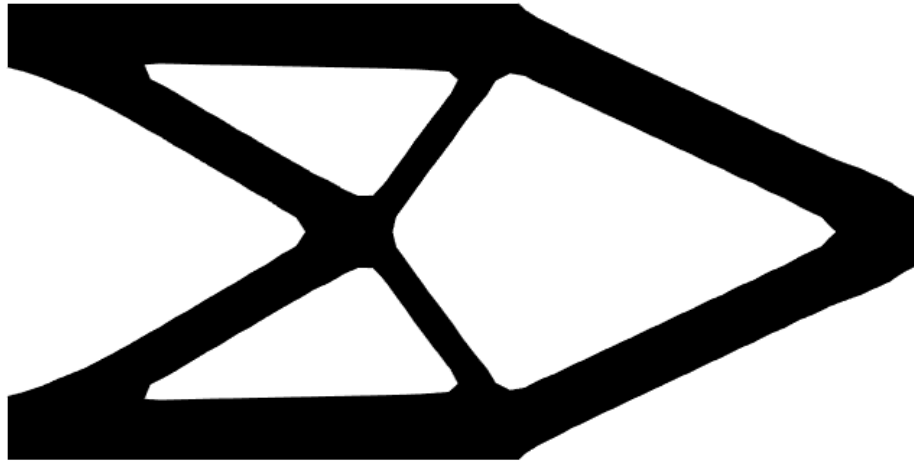


Figure 5-10: RBF Final Structure

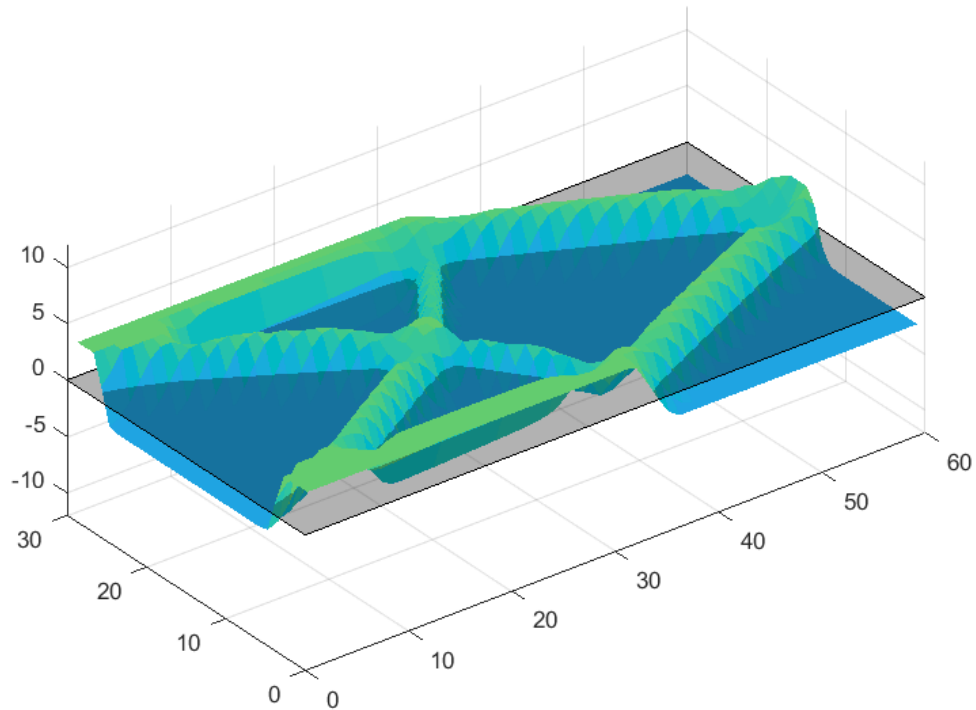


Figure 5-11: RBF Final Level-Set Function

This can be seen in the volume fraction and compliance versus iteration plots shown in figures 5-12 and 5-13 where there are evident spikes between iterations 65 and 75 as these sections were phased out.

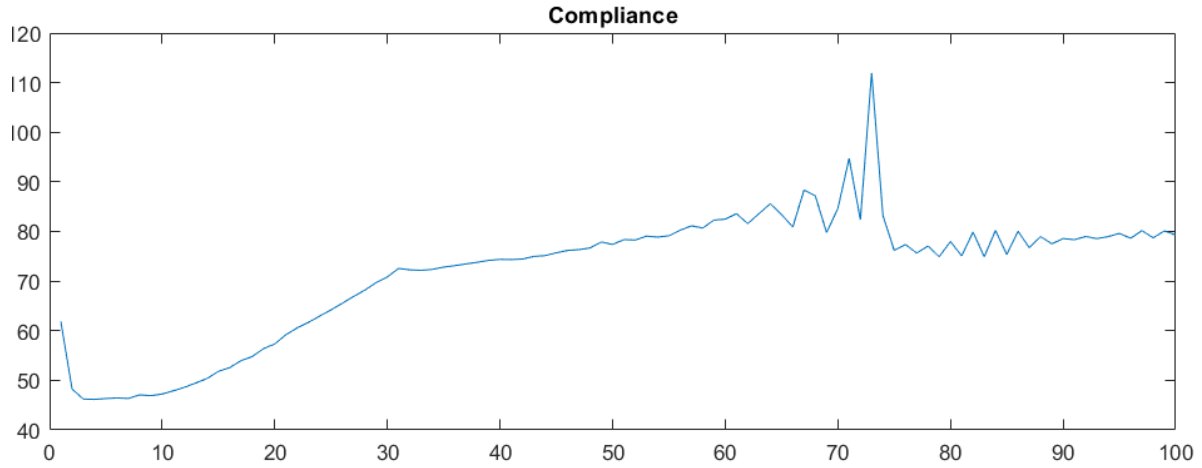


Figure 5-12: RBF Compliance Versus Iteration

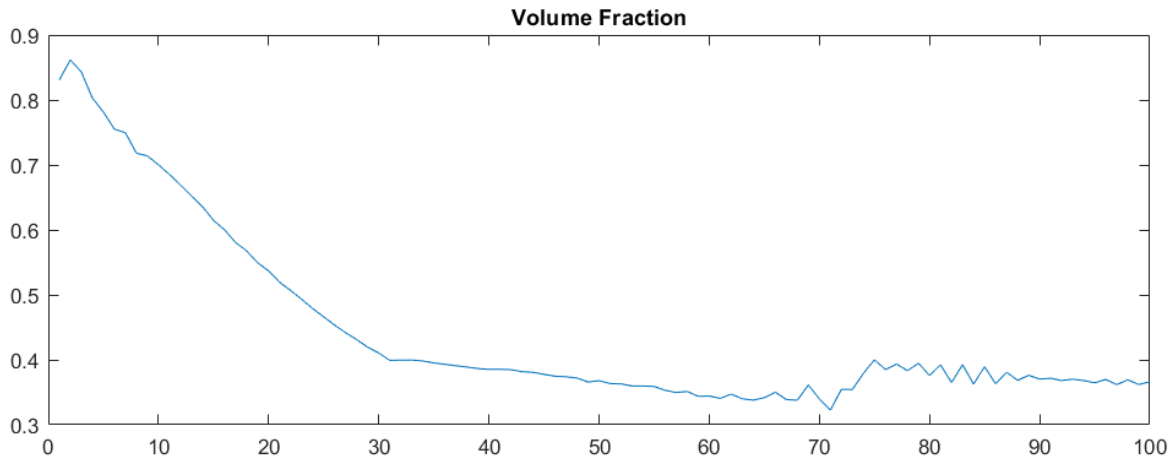


Figure 5-13: RBF Volume Fraction Versus Iteration

It should be noted that this implementation struggled to completely converge depending on the target volume fraction, where it would become unstable, oscillate, and fail to converge. Both of these level-set methods in \mathbb{R}^2 , provided results comparable to the literature [13], [43].

Following the implementation of the LSM for static loading cases in \mathbb{R}^2 , the method was expanded to \mathbb{R}^3 . Here a cantilevered beam, shown in figure 5-14 was discretized into 60x4x30 cubic elements. As in the first 2-dimensional case, a discrete geometry representation was used. The deformation of the final solution can be seen in figure 5-15.

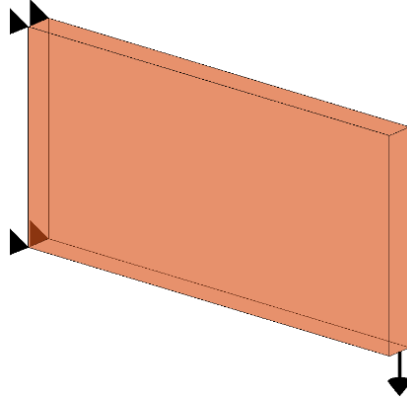


Figure 5-14: 3-D Cantilevered Beam Problem

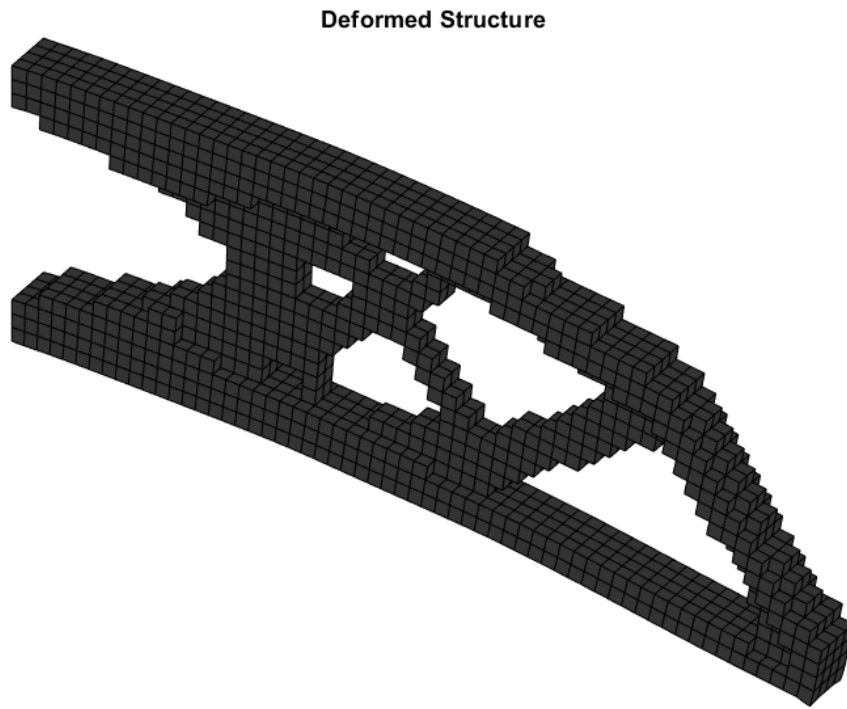


Figure 5-15: 3-D Cantilevered Beam Deformed Structure

5.2 Two-Dimensional Pressure Loading

The first phase of the progression toward topology optimization of an irregular shaped pressure vessel is to solve problems in \mathbb{R}^2 with design dependent pressure loading. Two methods were attempted during this phase, which led to the decision to pursue using the Level-Set Method. The first of these methods modeled the approaches developed by Lee and Edmund [52] using a density-based topology optimization, identifying an iso-density line as described in section 2.3. The second method modeled the work of Xia et al. [47], [50] by using a level-set method with two level-set functions to model the pressure and free boundary independently.

This first method proved effective in identifying the pressure boundary when the density distribution is crisp, as so with initial geometries, shown in figure 5-16. Here the intent is to simulate a pressure vessel in \mathbb{R}^2 by optimizing a structure with internal voids. Figure 5-16 shows the density distribution on a grey scale of a square structure with 2 initial voids, cropped vertically for visibility. Iso-density points are located and marked with blue circles and the pressure boundary is illustrated by the orange and yellow lines connecting these points.

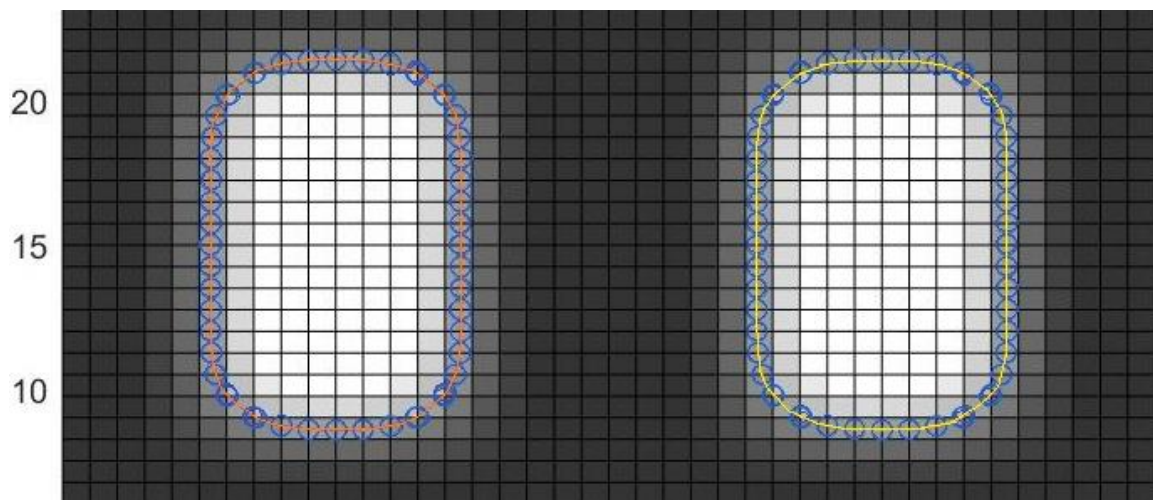


Figure 5-16: Iso-Density Identification During Early Iterations

However, congruent with density-based optimizations, the intermediate iterations tend to blur the density variation throughout the design domain as the algorithm determines the optimal material distribution. This causes issues with the current method of identifying the pressure boundary. As shown in figure 5-17, as this density distribution gradient flattens with more intermediate densities at a later iteration of the optimization, the method struggles to identify the appropriate locations to apply the pressure forces. The geometry of figure 5-17 started as one centrally located void and here it can be seen that there are unnecessary iso-density points forming islands. This causes the pressure loading path to be improperly determined as it loops over itself, X=8, Y=25 in figure 5-17.

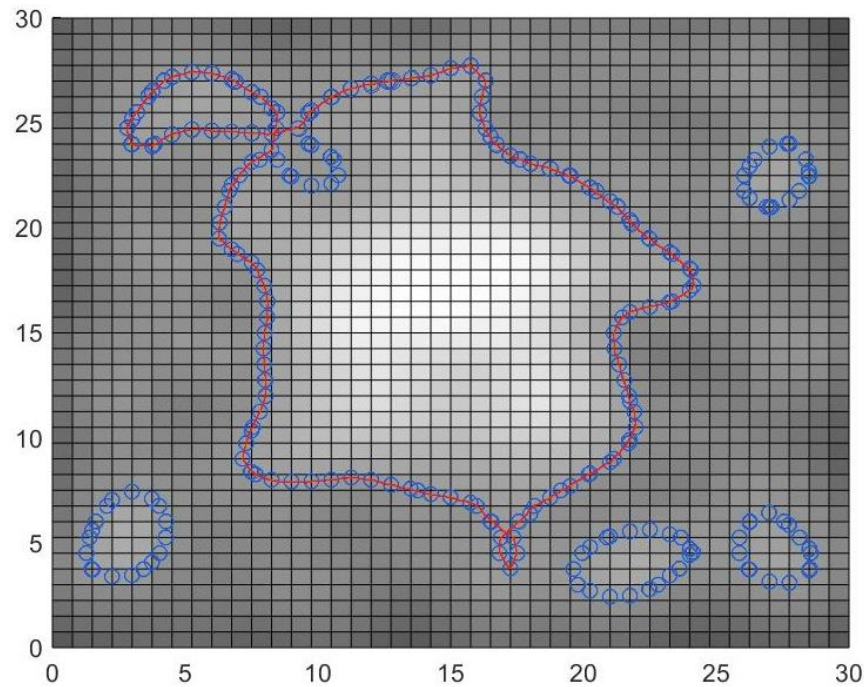


Figure 5-17: Iso-Density Line Errors

Additionally, this method does not translate to \mathbb{R}^3 as this would cause the algorithm to perform this operation at every cross-section and slicing them together.

The second method implemented for design dependent pressure loading problems in \mathbb{R}^2 followed the works of Xia et al. [47], [50] described in sections 2.3 and 3.3.1. To determine the effectiveness of this concept, it was used to solve the problem as defined in figure 4-2. Here a rectangular domain is pinned on both bottom edges and subjected to an upward pressure loading from the bottom edge. This closely resembles the problem described and solved in figure 5-1 with the exception of the forces now being design dependent and following the material boundary as it moves.

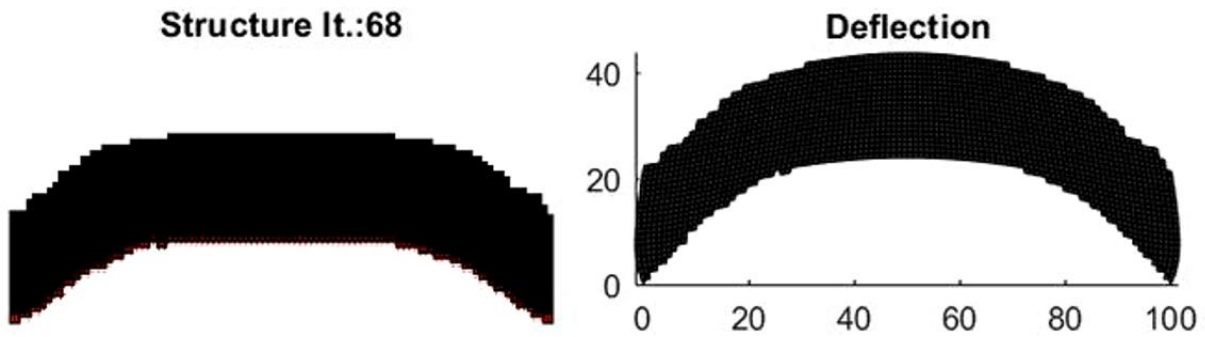


Figure 5-18: 2-D Pressure Loaded Structure and Deformation

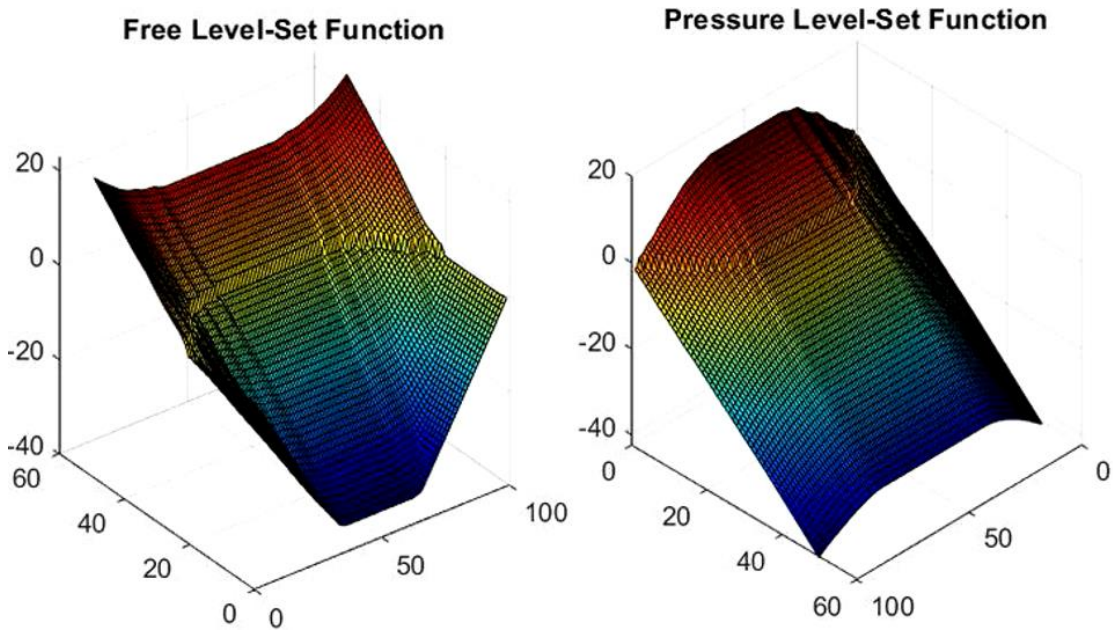


Figure 5-19: 2-D Pressure Loading Level-Set Functions

As to be expected, figure 5-19 shows the algorithm optimized the structure to form an arch, converging in 68 iterations. The use of a LSM proved to be effective in allowing the material domain to remain solid (no intermediate densities) and identifying the locations and magnitudes of the design dependent pressure loads. With these discoveries it was determined that the use of a level-set method would be ideal for a pressure vessel.

5.3 Three-Dimensional Pressure Box

After using a level-set method to solve design dependent pressure loading problems in \mathbb{R}^2 (section 5.2) and static loading problems in \mathbb{R}^3 (section 5.1), these two concepts were combined to optimize 3-dimensional structures with design dependent pressure loads. Before attempting to solve irregular shapes, however, the design domain was simplified to a box with internal pressure at the void/material interface as shown in figure 4-3. Similar to the first implementation of static loading in \mathbb{R}^2 , a discrete material representation was used with LSF nodes coinciding with element centers. As discussed in the explanation of defining fixed boundary conditions for 3-dimensional problems (section 4.3 and figure 4-7), the pressure boxes were fixed at the origin and given roller boundary conditions at the corners located on the coordinate axes allowing for deflection in their respective direction. This is shown in figure 5-20. As in a pressure vessel, the outer boundary is forced to stay solid and the interior void is subject to change during the optimization. As the method was developed and trials were run, nominal values were used for the modulus of elasticity and pressure.

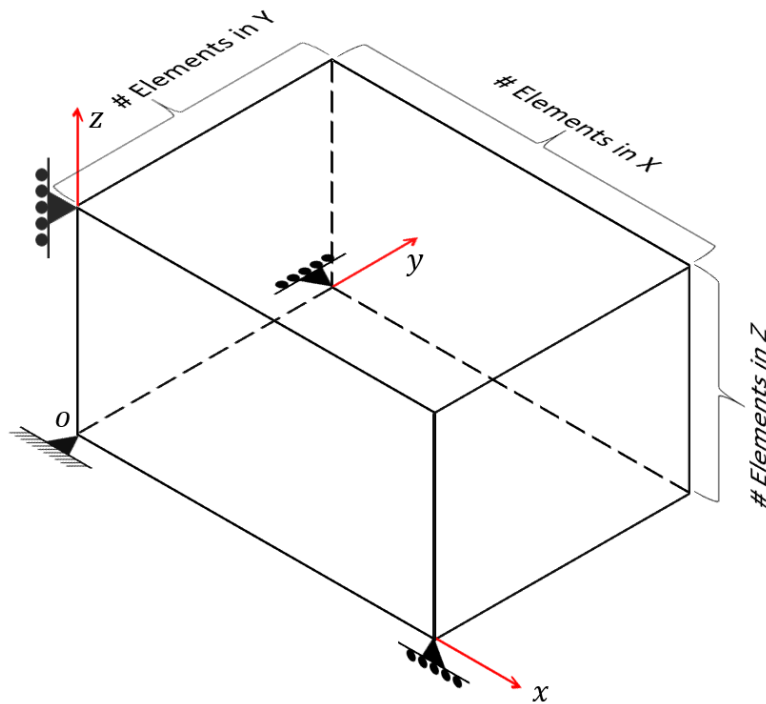


Figure 5-20: Pressure Box Problem Definition

5.3.1 Trials and Issues

At first, this problem was executed with 40 elements in the x-direction, 20 elements in the y-direction and 10 in the z-direction and an initial condition with one centrally located void as seen in figure 5-21. Note, for all void plots the outer surface of the pressure vessel is modeled as a transparent orange and the interior void elements a solid purple.

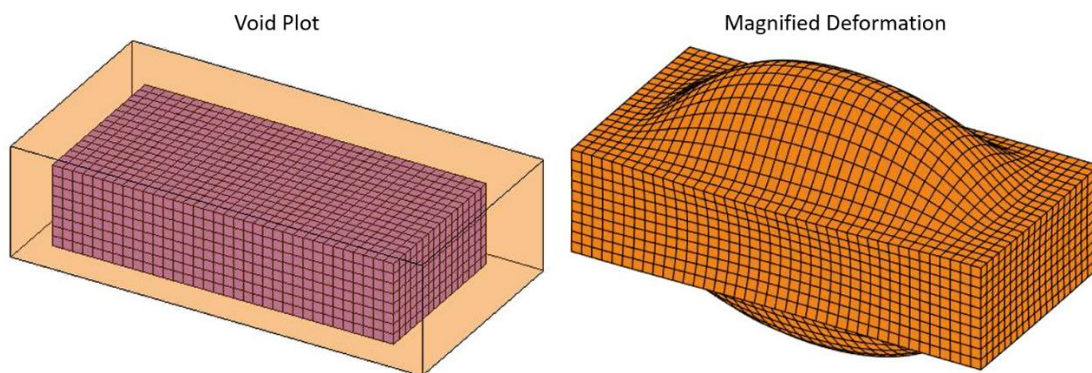


Figure 5-21: Pressure Box 40x20x10 Starting Void and Deformation

The algorithm performs well initially, as seen in figure 5-22 with ribs and other internal support structures being generated at iteration 75.

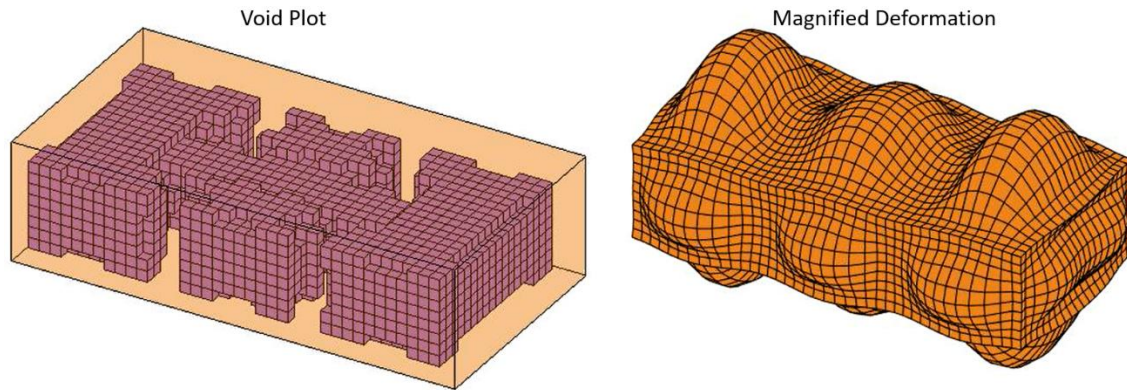


Figure 5-22: Pressure Box 40x20x10 Iteration 75 Void and Deformation

Despite appropriate moves during initial iterations, unfortunately the optimization fails to converge on a solution and instead begins adding too much material until it eventually is a solid structure. At this point the LSM will never be able to remove material because of the lack of a reaction term from topological derivatives in the Hamilton-Jacobi equation.

It was recognized that the supporting structures approached elemental thickness at the near convergence states before rapidly diverging. This could prove particularly troublesome especially when combined with the discrete material nature of the LSM's current formulation. A couple of the experimented solutions to this were to 1) maintain the structure's aspect ratios and refine the mesh and 2) Incorporate the use of intermediate densities. The volume fraction and compliance versus iteration for a trial with a mesh size of 60x30x15 can be seen in figure 5-23, and the void structure and deformations of the trial with intermediate densities can be seen in figures 5-24 through 5-27. Here the deflection plots for iterations 1, 20, 40 and 60 are neglected because the deflections are unrecognizable at a consistent magnification factor used for iterations 62 and 90.

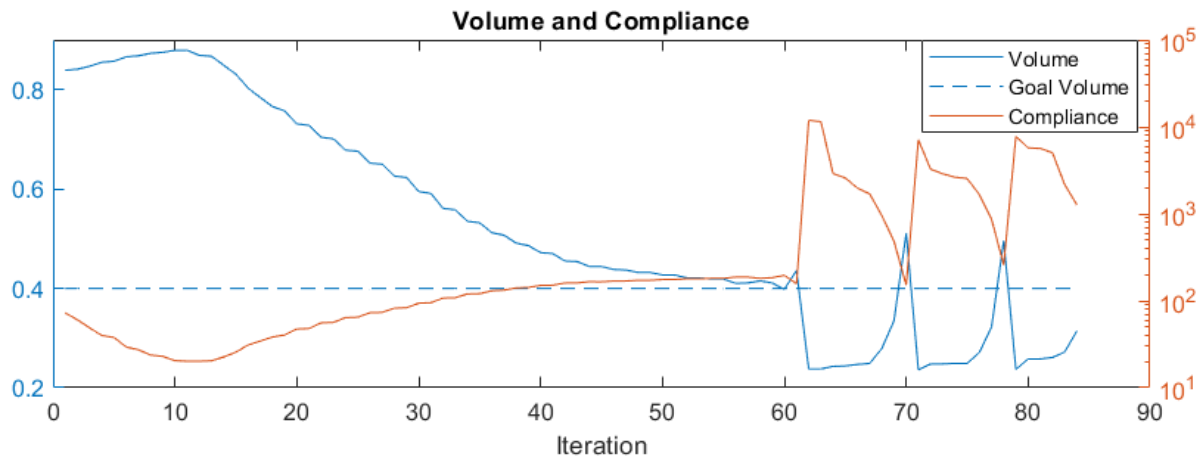


Figure 5-23: Pressure Box 60x30x15 Volume and Compliance Versus Iteration

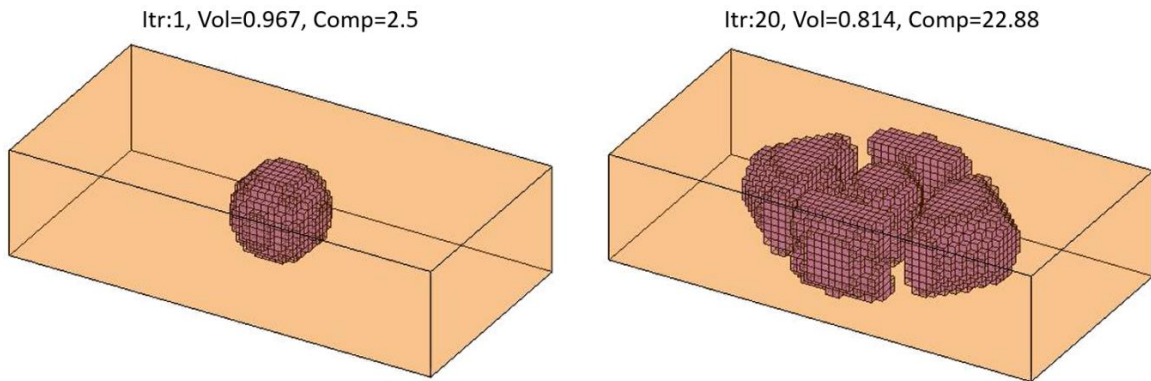


Figure 5-24: Pressure Box Intermediate Densities, Iterations 1 and 20

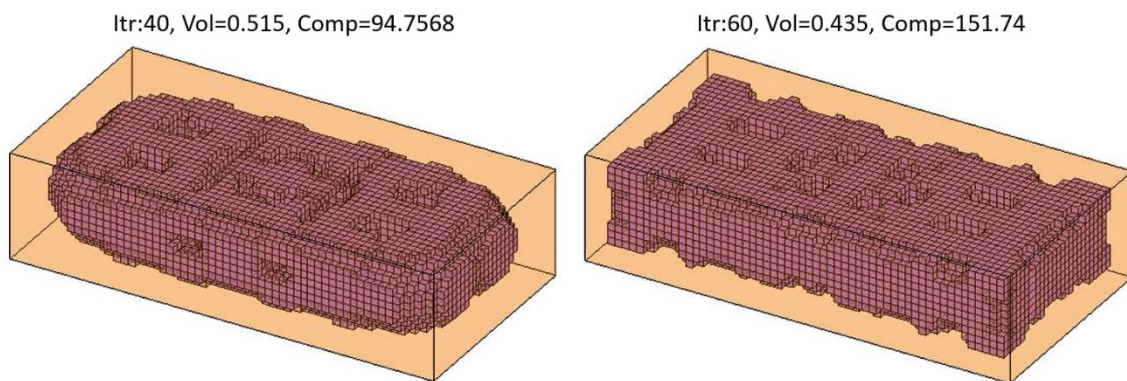


Figure 5-25: Pressure Box Intermediate Densities, Iterations 40 and 60

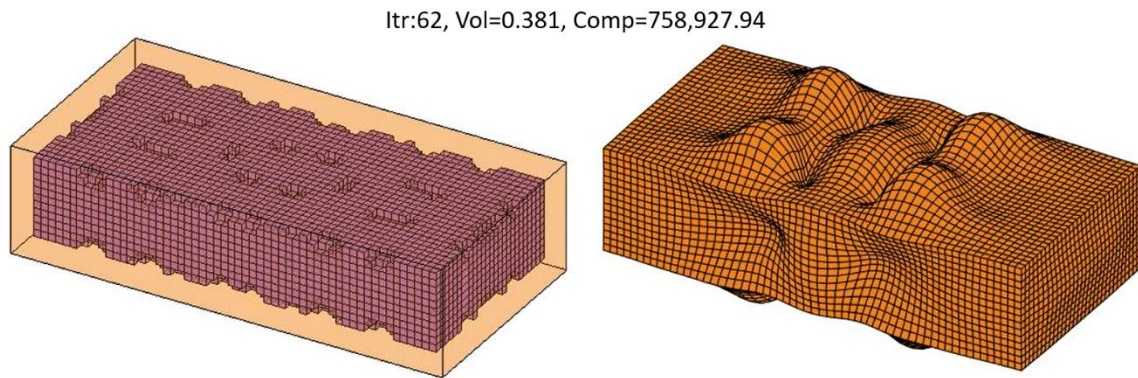


Figure 5-26: Pressure Box Intermediate Densities, Iteration 62

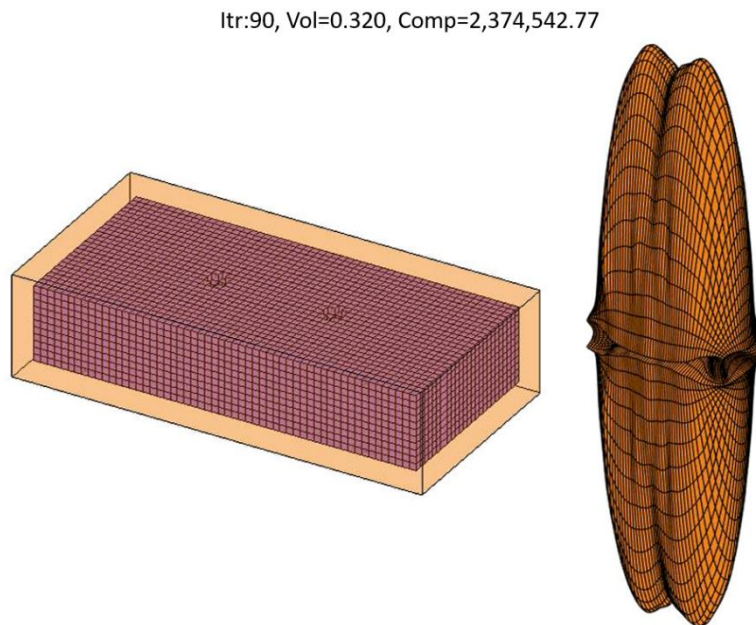


Figure 5-27: Pressure Box Intermediate Densities, Iteration 90

Both changes seemed to help by giving more geometric control compared to the initial results, however the algorithm continued to experience improper behavior close to the target volume, seen in figures 3-7, 3-8, and 5-23.

To attempt to solve these issues, other modifications were tried. In an effort to aid the starting condition and flexibility of the LSF evolution, the initial structure was subjected to a

variety of void shapes and multi-void array patterns, mimicking what was done with the radial basis functions in \mathbb{R}^2 . These multi void starting conditions provided improved results upon initially reaching the target volume fraction, however failed to solve the issues of final convergence. Figure 5-28 shows both the compliance and volume fraction versus iteration plots for a trial with 5x5x5 voids spaced 2 elements apart in all directions and patterned 8, 4, and 2 times in each coordinate direction, respectively. As seen in figure 5-28, although many times the LSM will attempt to recover from this improper performance, it fails to reach objective values previously found and has unstable behavior. Other attempted solutions involved flipping the objective and constraint, and modifying the Lagrange multiplier update, all with limited success.

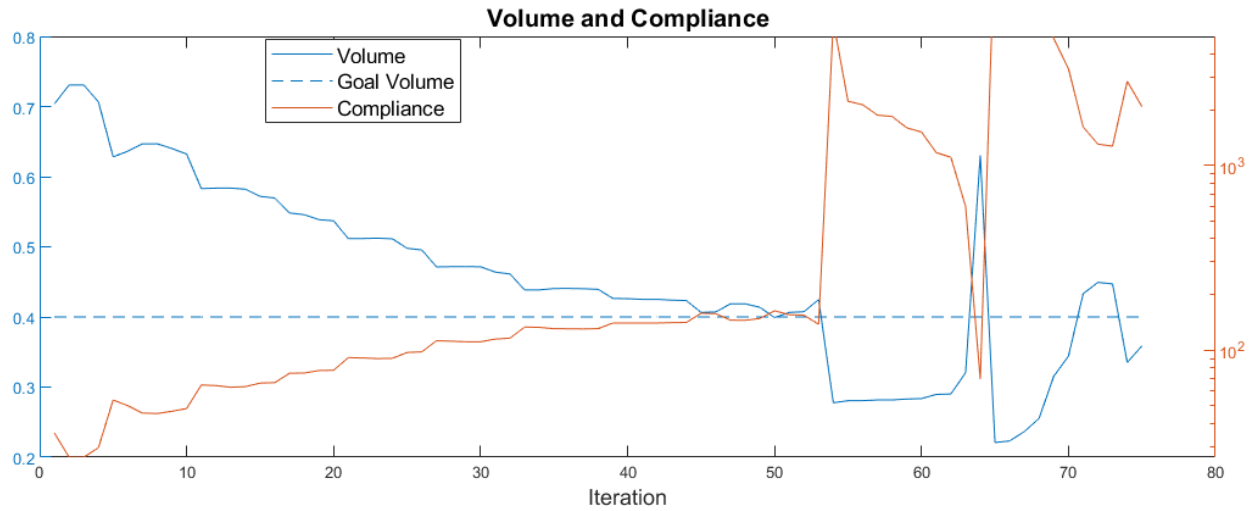


Figure 5-28: Pressure Box Multiple Starting Voids Compliance and Volume

5.3.2 Pressure Box Solutions and Results

As explained in sections 3.3.2 and 4.4.4, it was recognized that the various Lagrange multiplier update and penalty schemes from the literature harbored great similarities to concepts of PID controls. With this in mind, the penalty formulation was modified to act as a PID controller would (equation 3.56). The figures below show the results from two runs of the

algorithm with this modification. The first having an initial condition of one large void (figures 5-29 through 5-34) and the second an array of smaller voids (figures 5-35 and 5-36). In both cases, the problem is unitless with a domain of 60x30x15, the material properties have the modulus of elasticity set to 1 and Poisson's ratio to 0.3, and there is a pressure force of 1 from all interior voids. The Lagrange multipliers were initialized to $\lambda_o = 0.001$ and $\lambda_{PID} = 0.001$ with an update factor of $\alpha = 1.11$. Additionally, it was discovered that PID gains of $K_p = 0.5$, $K_I = 1$, and $K_D = 0.25$ provided stable convergence behavior across most starting conditions.

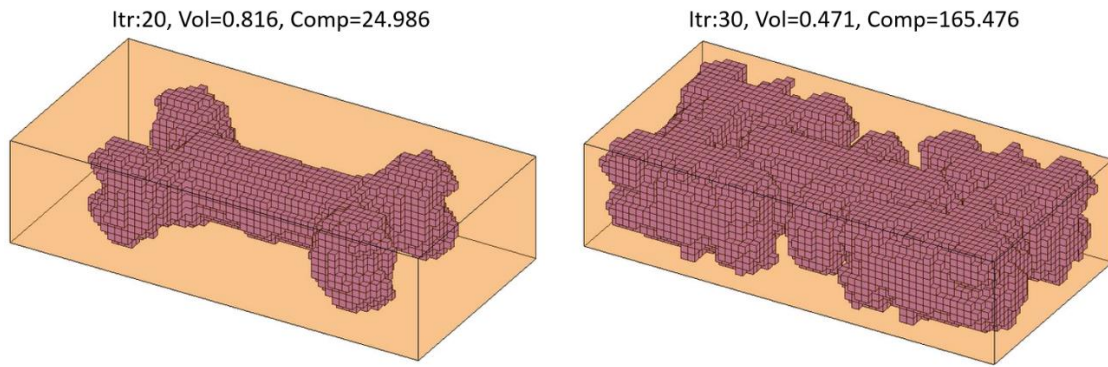


Figure 5-29: One Starting Void Iterations 20 and 30

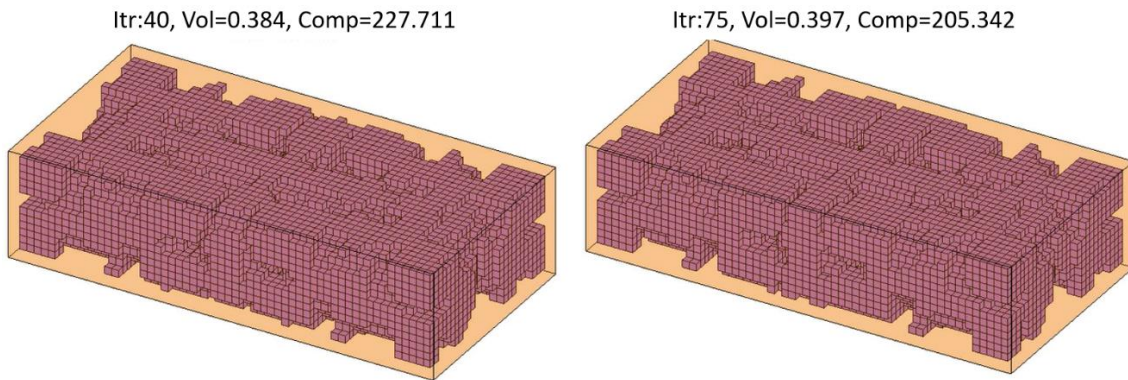


Figure 5-30: One Starting Void Iterations 40 and 75

The following plots show the iteration data for this run. Figure 5-32 shows the individual terms found in equations 3.53, 3.54 and 3.55 multiplied by their respective controller gains. Figure

5-33, shows the difference between the penalty term used in this implementation and what would have been used with the original Lagrangian penalty method. Note the non-zero value at the end, and the 2 lines coinciding for the first 20 iterations because the original formulation is used initially. The justification for this non-zero penalty can be seen in figure 5-34 where the average shape sensitivity is clearly negative, and the maximum is zero.

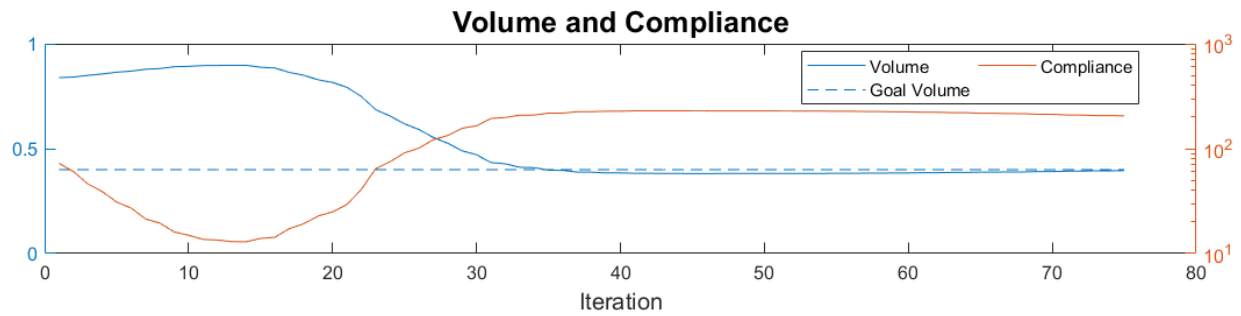


Figure 5-31: One Starting Void Volume and Compliance

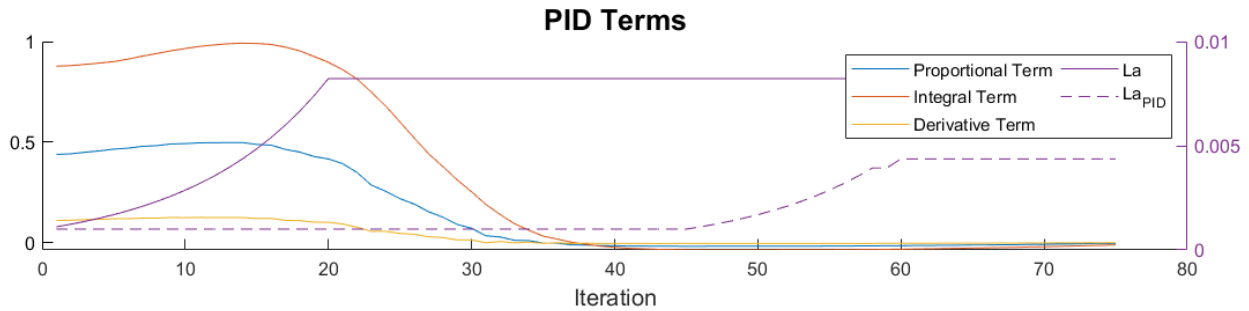


Figure 5-32: One Starting Void PID Terms

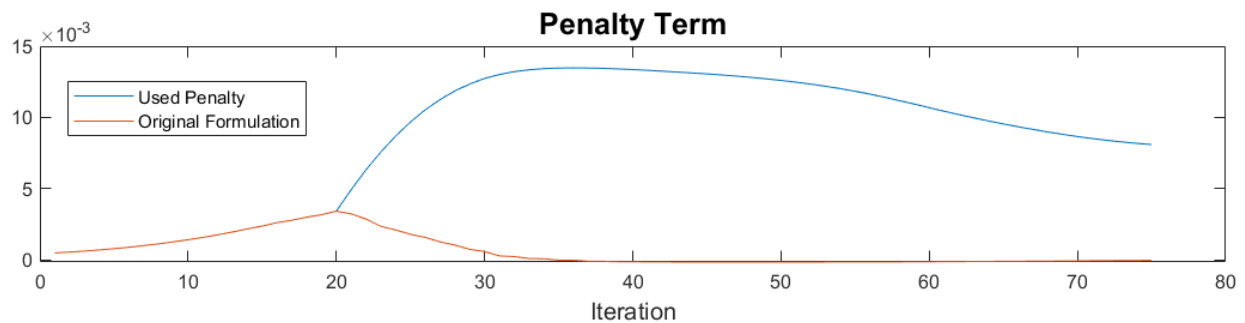


Figure 5-33: One Starting Void Penalty Term

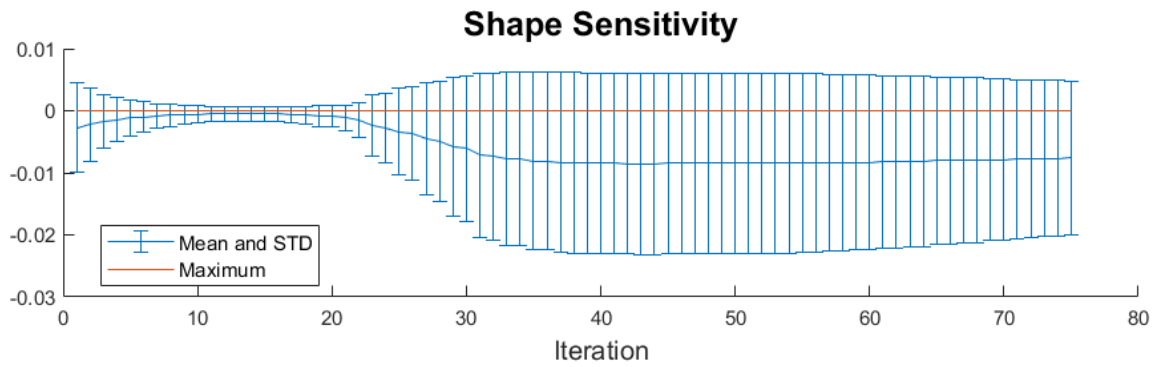


Figure 5-34: One Starting Void Shape Sensitivity

Figure 5-35 and 5-36 show the initial and final void structures and the volume and compliance versus iteration plot for a trial with multiple starting voids by using an 'init' matrix of [3,3,3;2,2,2;10,6,2] (explained in section 4.3). The one starting void trial converged in 75 iterations with a compliance of 205.342, whereas the trial with multiple voids converged in 68 iterations with a compliance of 181.628.

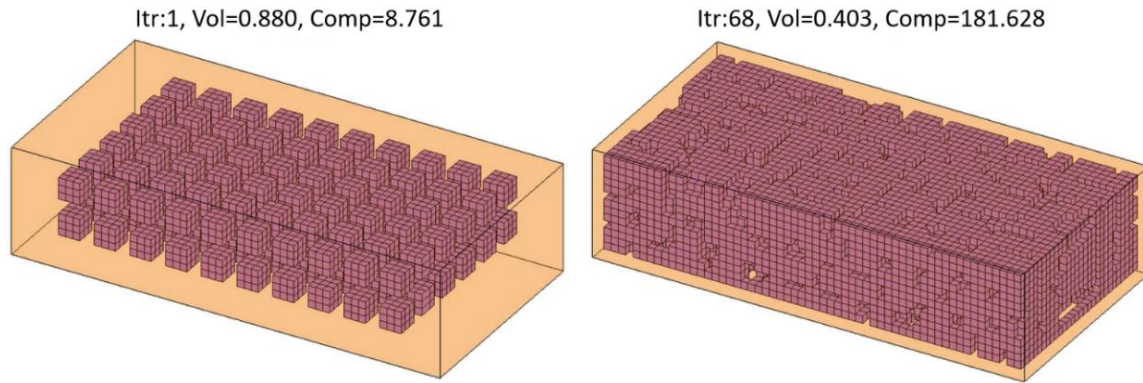


Figure 5-35: Multiple Starting Voids Iterations 1 and 68

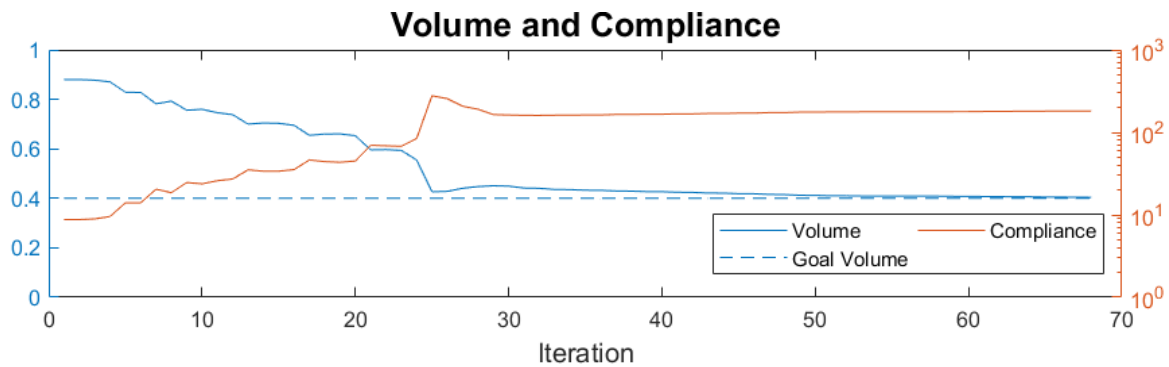


Figure 5-36: Multiple Starting Voids Volume and Compliance

Once the method was established and working correctly, trials were done with practical material and pressure values as opposed to the prior trials using nominal values. Here the domain used maintained the aspect ratios of the previous trials but was given dimensions of 11.6"x5.8"x2.9", resulting in the same outer volume as the existing pressure vessel for the MK-16. The domain was again discretized into 60 elements in x, 30 in y, and 15 in the z-direction, making each element a cube of length 0.1934". Because the existing pressure vessel is made from Inconel718, the assigned material properties were set to 29.5×10^6 PSI for the modulus of elasticity and 0.29 for the Poisson's ratio. The void was modeled with a pressure of 5,000 PSI. The volume and compliance versus iteration is shown below in figure 5-37. Here the importance of not having too small of a starting Lagrange multiplier was discovered, as it would cause the structure to turn completely solid, eliminating the ability to recover and add voids. Thus, the check for a solid structure and reverting back to the initial condition as explained in the preparation for the subsequent iteration found in section 4.4.5 was implemented.

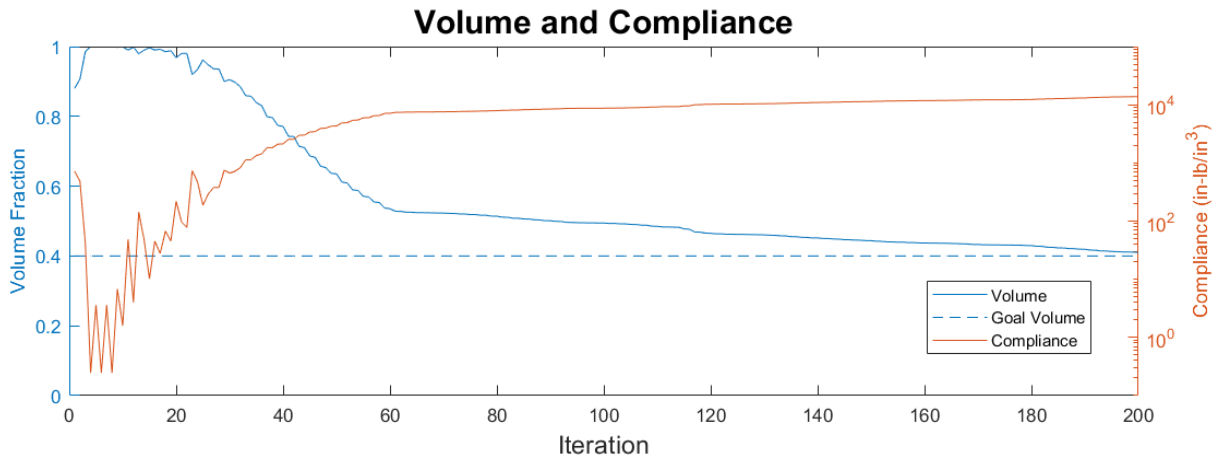


Figure 5-37: Real Valued Pressure Box Volume and Compliance

Having dealt with regular shapes and identified the various issues with the algorithm and possible ways to address convergence issues, the next chapter describes the application of the LSF approach to an irregular shaped pressure vessel.

Chapter VI: Irregular Shaped Pressure Vessel Results

Using the methods established in chapter 3, the implementation procedures laid out in chapter 4, and the incremental progression of the problem executed in chapter 5, this chapter discusses the results and findings when applying these principles to determine a geometric structure for an irregularly shaped pressure vessel. This chapter is organized as follows: section 6.1 overviews the existing pressure vessel, section 6.2 defines the design space for the irregular shaped pressure vessel, section 6.3 covers the initial results of using the formulation directly as established for the pressure box, and section 6.4 presents the results after applying the re-meshing method and disjointing the level-set function from the finite element mesh.

6.1 Existing Pressure Vessel

As discussed in the introduction in chapter 1, the ability to utilize topology optimization to determine a geometric structure for an irregular shaped pressure vessel would be tested on an existing NAVY diving rebreather, the MK-16. This system can be seen in figures 1-1, 1-2 and 6-1 with the back cover removed.

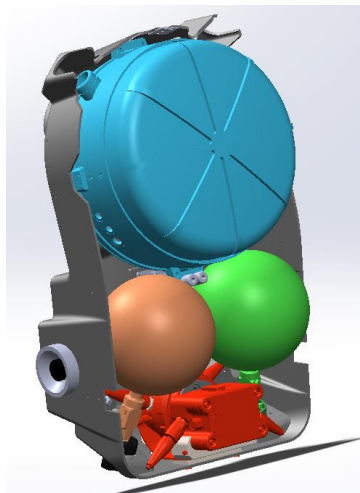


Figure 6-1: MK-16 Back Cover Removed

As seen in figure 6-1 and labeled in figure 1-2, there are two spherical gas storage pressure vessels in the rig. The left side houses the diluent, shown in orange, and the right-side stores oxygen, shown in green. Due to symmetry and assuming a desire to store an equal quantity of gas in each, only one of these pressure vessels needs to be considered. The dimensions of the pressure vessel can be seen in figure 6-2 in the cross-section drawing view, and relevant properties can be seen in table 6-5.

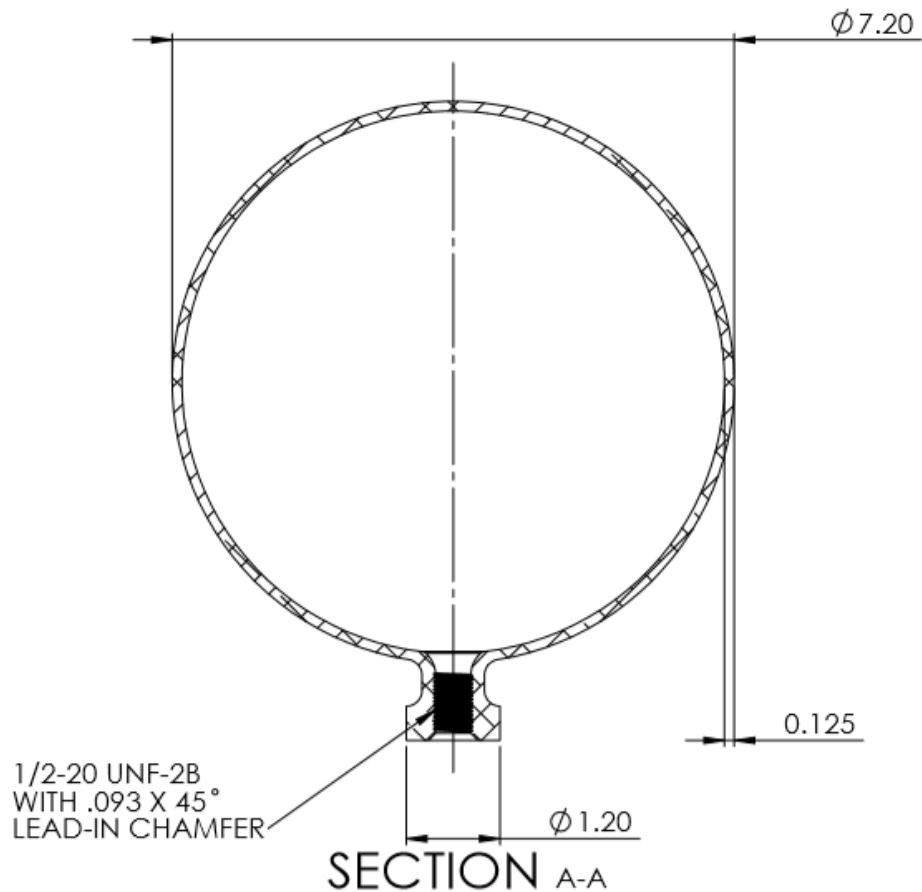


Figure 6-2: Existing Pressure Vessel Dimensions

Table 6-5: Existing Pressure Vessel Properties

Property	Value
Material	Inconel 718
Modulus of Elasticity	29.5×10^6 PSI
Poisson's Ratio	0.29
Yield Strength	150 KSI
Outer Diameter	7.20 in.
Thickness	0.13 in.
Displaced Volume	195.43 in. ³
Wet Volume	175±10 in. ³
Working Pressure	3,000 PSI

In order to compare the results of the irregular pressure vessel to the existing spherical design, this component was exported to an STL, meshed by 'MakeMesh.m' (section 4.2, Appendix B) and analyzed with the same finite element analysis procedure used during the topology optimization procedure. This meshing was done with voxel elements just as the irregular shaped pressure vessel will be processed, and to visualize convergence behavior, the meshing was carried out with 0.125", 0.1", and 0.0625" element sizes. To compare the linearity in the results, the finite element analysis was executed with an internal pressure of 3,000 PSI, 5,000 PSI, and 12,000 PSI. The results from this analysis can be found in table 6-6 and a stress plot of the 0.0625 mesh subjected to 5,000 PSI is shown below in figure 6-3. Note the use of voxel elements leads to increased stress concentrations, particularly with larger elements.

Table 6-6: Existing Pressure Vessel FEA Results

Mesh Size	# of Elements	# of Nodes	Pressure (PSI)	Max Deflection (in.)	Compliance (in-lb/in ³)	Max VonMises Stress (PSI)
0.0625"	82852	144159	3000	0.0225	372.692	58528.05
			5000	0.0372	1035.256	97546.75
			12000	0.09	5963.075	234112.2
0.1"	20195	44111	3000	0.0231	415.626	52974.3
			5000	0.0385	1154.517	88290.5
			12000	0.0923	6650.015	211897.2
0.125"	10341	25664	3000	0.0266	453.5255	56591.95
			5000	0.0444	1259.794	94319.92
			12000	0.1066	7256.41	226367.8

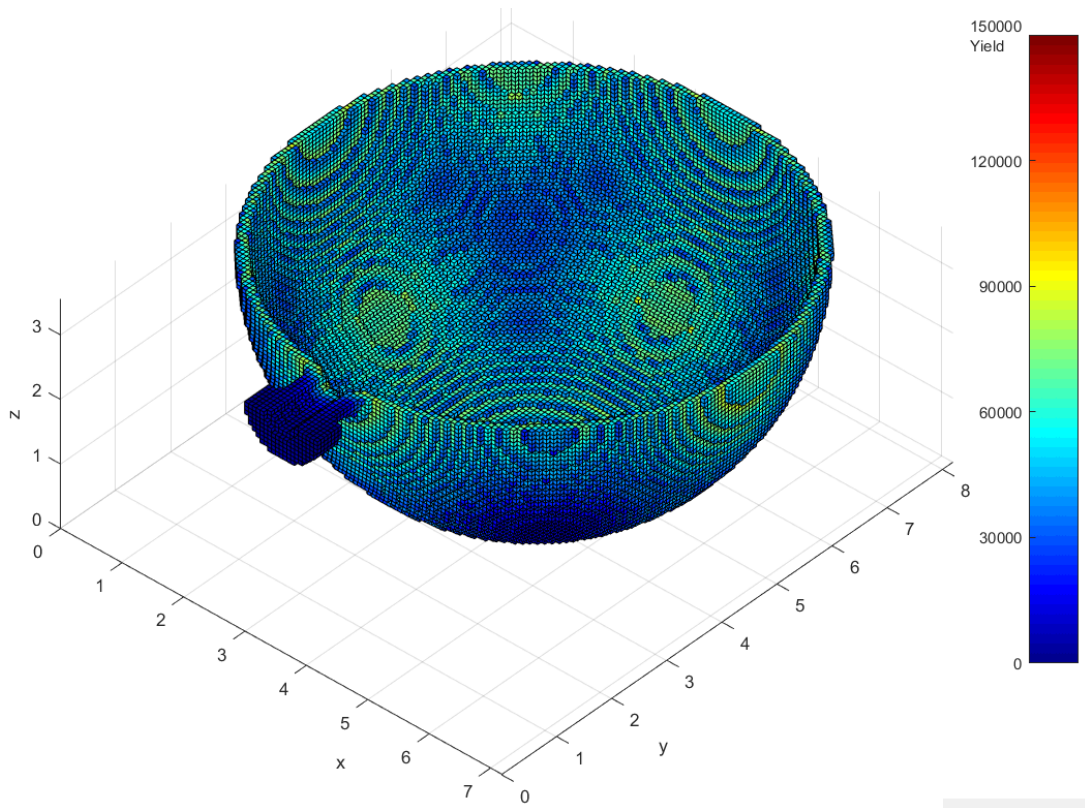


Figure 6-3: Existing Sphere Stresses(PSI) Pressure=5,000 PSI and 0.0625" Elements

6.2 Defining the Design Domain

Once the existing system was chosen and defined, the design space for the irregular shaped pressure vessel needed to be established. Because the method for optimizing a pressure vessel used in this thesis fixed the outer boundary of the pressure vessel during optimization, defining the design domain meant determining the outer geometry of the pressure vessel. Without changing the rest of the MK-16 and due to the symmetry of the MK-16 and the assumption of wanting equal oxygen and diluent storage capacity, this meant expanding the sphere such that it does not interfere with other components, leaving a tolerance, nor cross the center line of the MK-16. This was done in SolidWorks and the resulting part can be seen in figure 6-4. For reference, later in the chapter, sides of this geometry are labeled.

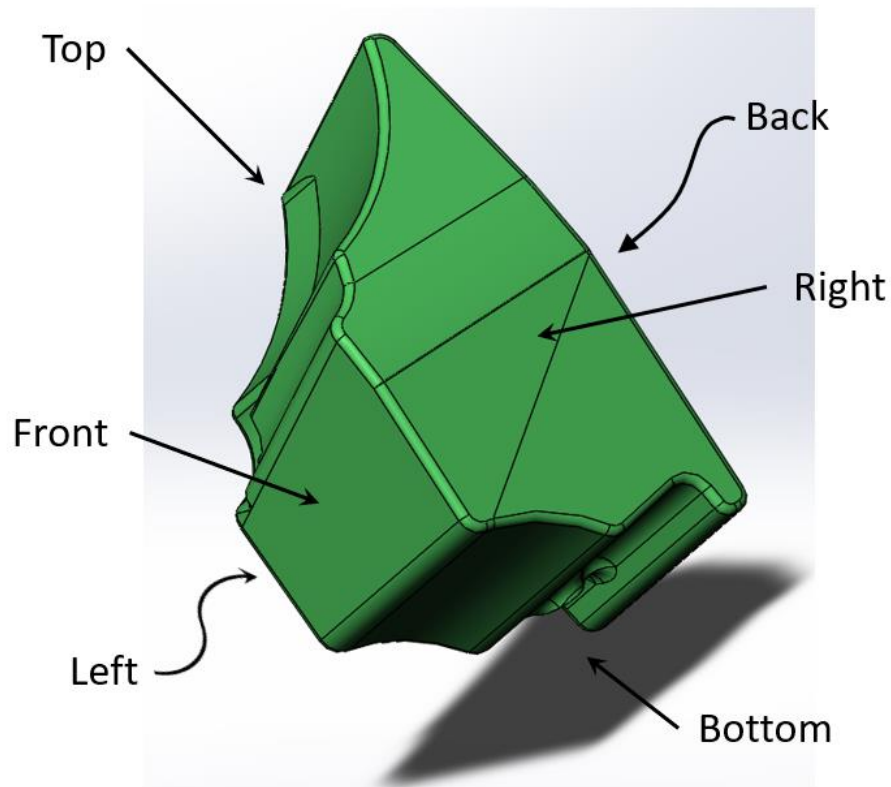


Figure 6-4: Proposed Pressure Vessel Geometry for MK-16

This geometry has a displaced volume of 369.89 in.³, an 89% increase from the existing spherical pressure vessel. For an idea of size, several of the dimensions are displayed in figure 6-5 and the geometry can be seen in place of the oxygen tank within the MK-16 assembly in figure 6-6.

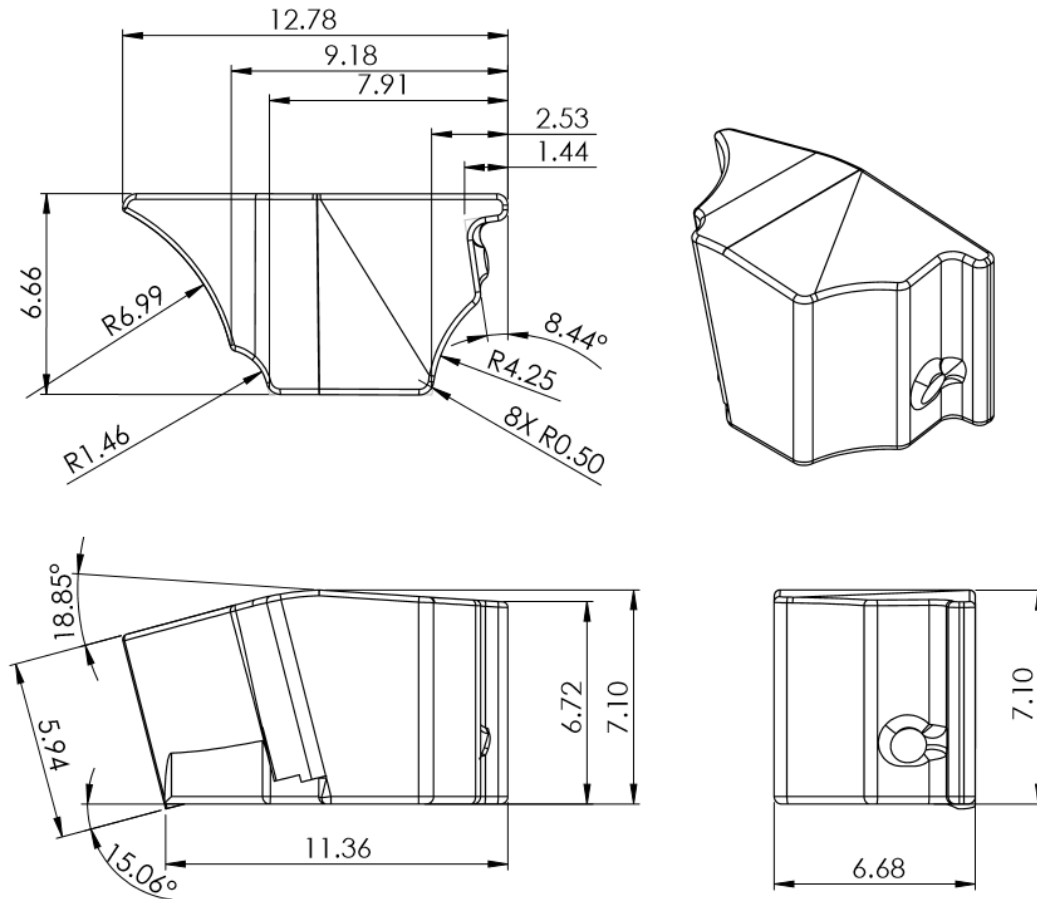


Figure 6-5: Proposed Pressure Vessel Dimensions

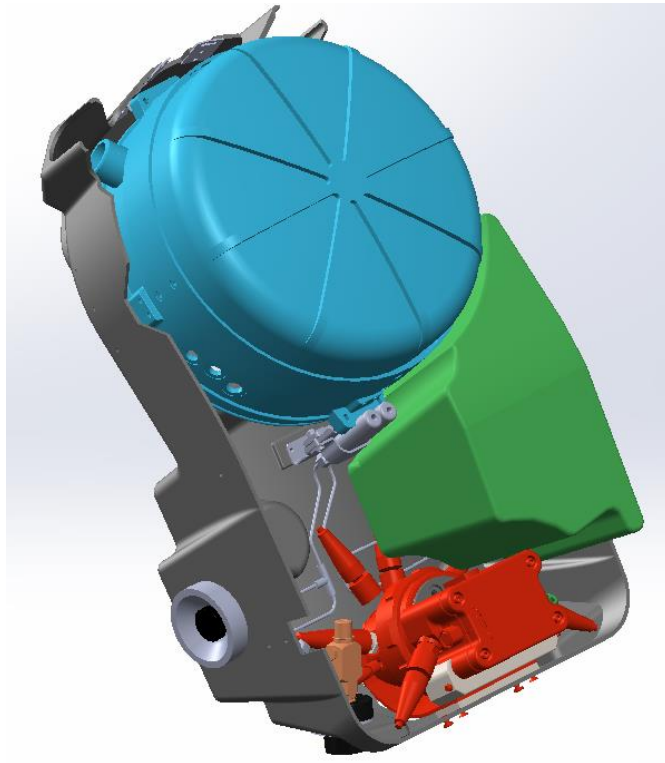


Figure 6-6: Proposed Pressure Vessel in MK-16 Assembly

In order to establish a viable mesh for the finite element method to use during optimization, the geometry needs to be converted to and exported as an STL file to be read by the 'MakeMesh.m' script developed in MATLAB, section 4.2 and appendix B. The generated STL file resulted in 40,448 triangular faces to define the geometry of this proposed pressure vessel. Using the 'MakeMesh.m' script, the geometry was discretized into numerous meshes with varying element sizes. Figure 6-7 shows this meshing executed with 0.2" voxel element size.

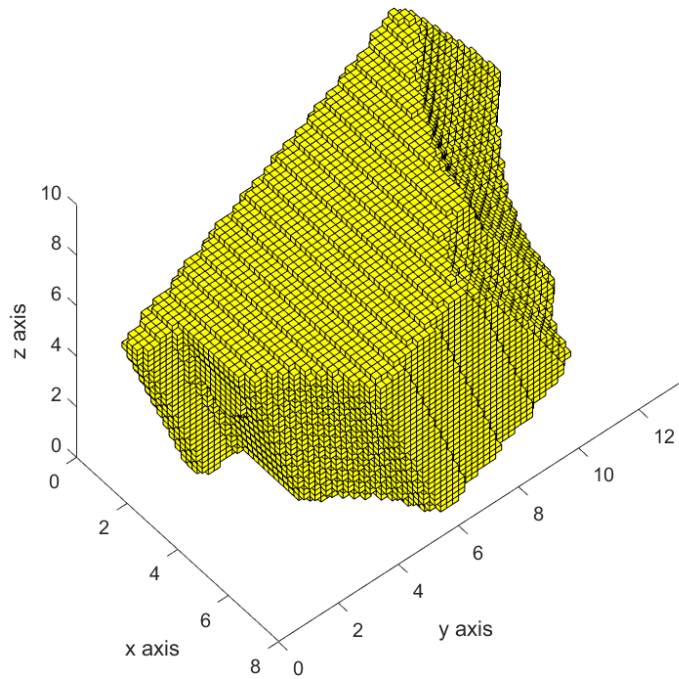


Figure 6-7: Mesh with 0.2" Element Size

Upon meshing this geometry, it was discovered that to achieve an ideal response from the finite element analysis, the geometry should be rotated such that as many surfaces as possible are parallel to a cartesian plane. Thus, the geometry was rotated and re-meshed. Table 6-7 summarizes the results from meshing the geometry with different element sizes and orientations. Note that during the optimization, boundary elements are not subject to change and are forced 'on', thus the last two columns denote the number of boundary elements and volume fraction of the interior, respectively.

Table 6-7: Meshing Summary

No.	Voxel Size	Orientation	# of Elements	# of Nodes	# Degrees of Freedom	Boundary Elements	Center Fraction
1	0.25"	Original	23,782	27,399	82,197	6,440	0.729
2	0.25"	Rotated	23,593	26,730	80,190	5,618	0.762
3	0.2"	Original	46,308	51,913	155,739	10,223	0.779
4	0.2"	Rotated	45,768	50,591	151,773	8,831	0.807
5	0.1875"	Rotated	56,326	102,120	306,360	10,233	0.818
6	0.15"	Rotated	109,738	118,377	355,131	16,183	0.853
7	0.125"	Original	189,514	203,710	611,130	26,825	0.858
8	0.1"	Original	370,160	392,267	1,176,801	42,260	0.886
9	0.1"	Rotated	369,417	388,694	1,116,082	36,918	0.900

6.3 Initial Results

This section presents the data from directly using the method and implementation procedure that provided stable convergence for the rectangular cuboid design domain problems, section 5.3.2. That is, a discrete material representation was used with the LSF discretization coinciding with the element mesh and the pressure being calculated as outward normal from every void element, figure 3-2 and equation 3.27. Due to the computational expenses and limitations of MATLAB, it was determined to start with the 0.25" mesh, No.2 in table 6-3, to optimize the interior structure of the proposed pressure vessel. Considering the proposed geometry's total volume, the mesh's interior volume fraction, and desiring a void volume larger than the existing pressure vessel, the initial target volume fraction was set to 0.45, equating to a void volume of 203 in³. The PID gains, $[K_p \ K_I \ K_D]$, for best convergence were determined to be $[1 \ 0.5 \ 0.2]$. Initial Lagrange multipliers were set to $\lambda_o = 0.1$ and $\lambda_{PID} = 0.1$ with an update factor of $\alpha = 1.11$. The material properties were set to that of Inconel718, and a pressure value of 3,000 PSI. With these conditions the optimization converged

in 76 iterations with a compliance of 3538.95 in-lb/in³, max stress of 54,128 PSI as seen in the plots shown in figure 6-8.

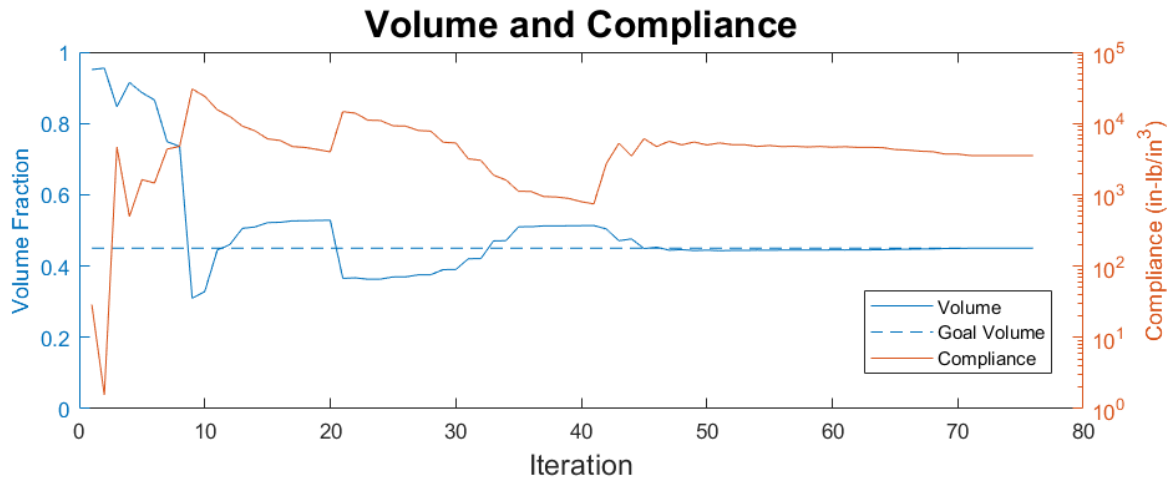


Figure 6-8: Volume and Compliance, Target Volume of 0.45, Pressure Calculation from Void

As seen in figure 6-8, as with many of the trials, there were frequently large oscillations, iterations 9 and 20 here, as opposed to the steady convergence seen in the pressure box optimizations, figure 5-37. Many variations of PID gains, starting Lagrange multipliers, and penalty formulations were attempted to correct this, however the aforementioned conditions proved the best. Once the structure itself was analyzed, another issue became apparent, checkerboarding. Different element sizes and filtering techniques were attempted to solve this issue with little success. Despite this, several structural geometric features were able to be distinguished, circled in red on figure 6-9 and highlighted in figures 6-10 through 6-12. The maximum Von Mises Stress of this final structure was 54,128 PSI with an average of 11,596 PSI, although it should be noted with such large element sizes these peaks are caused by exaggerated stress concentrations.

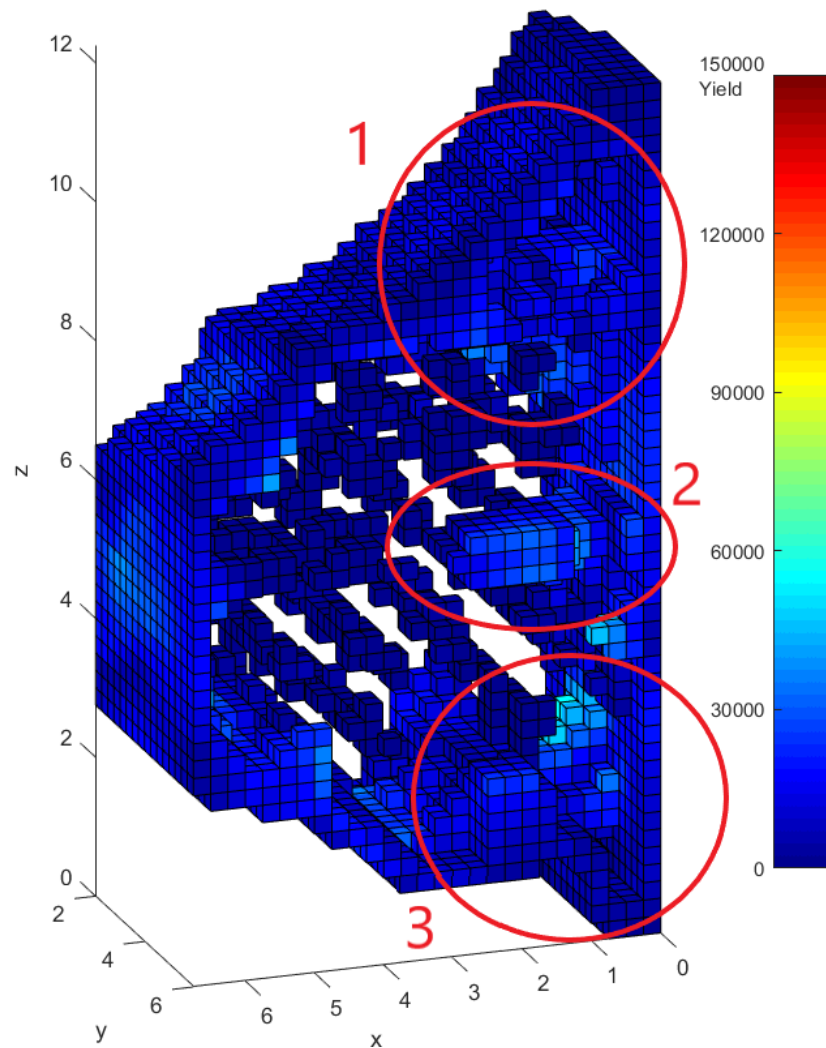


Figure 6-9: Stress Plot Iteration 76, View Window: X(0,6.75) Y(2,6) Z(0,12.25)

Marked by the red “1” in figure 6-9, the structure contains a rib connecting the back face with the angled top face. Figure 6-10 shows this feature closer from both the left and right views. Here it can be seen that the feature connects the two surfaces like a rib but leaves the upper portion hollow for more volume.

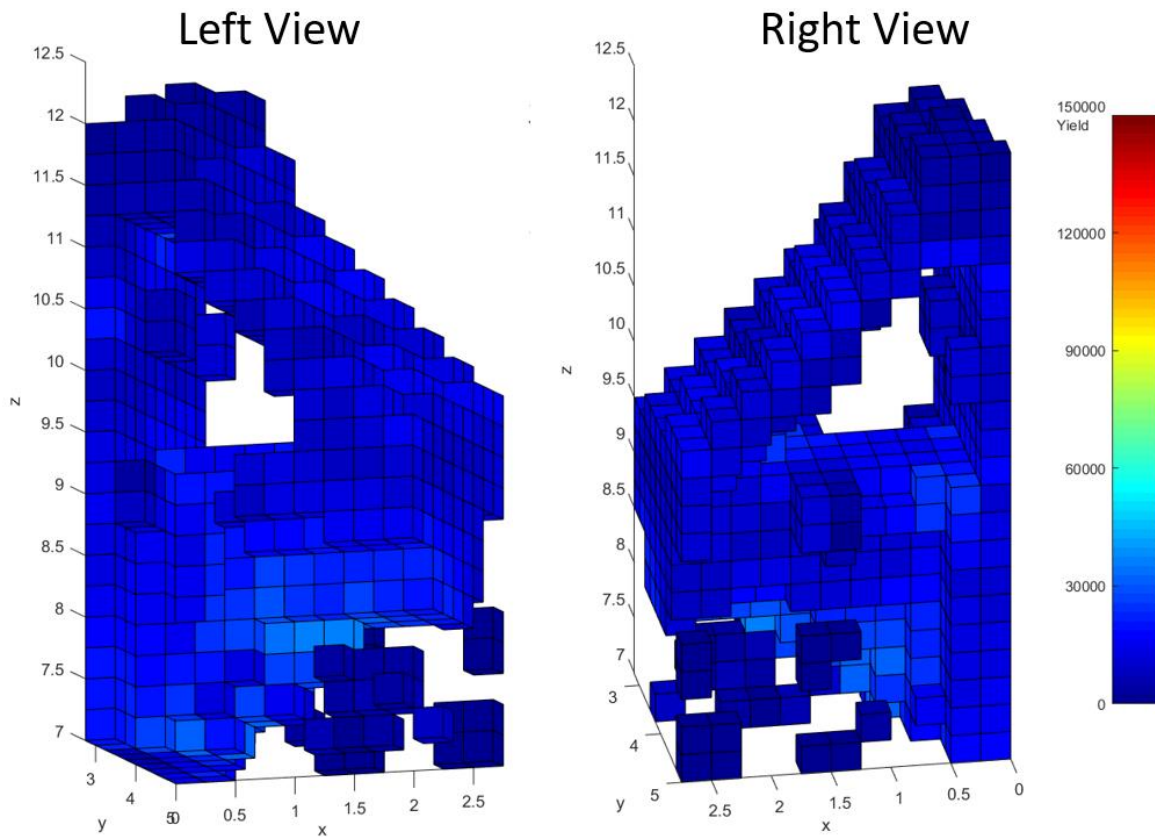


Figure 6-10: Top Rib, View Window: $X(0,2.75)$ $Y(2.75,5)$ $Z(7,12.75)$

The second label in figure 6-9 shows another rib located midway up the connection between the near surface (out of window to view interior) and the back surface. As with the top rib, there is a small opening at its center. This is shown in figure 6-11. Finally, the third label in figure 6-9 shows a rib located along the bottom of the pressure vessel, this is shown closer in figure 6-12.

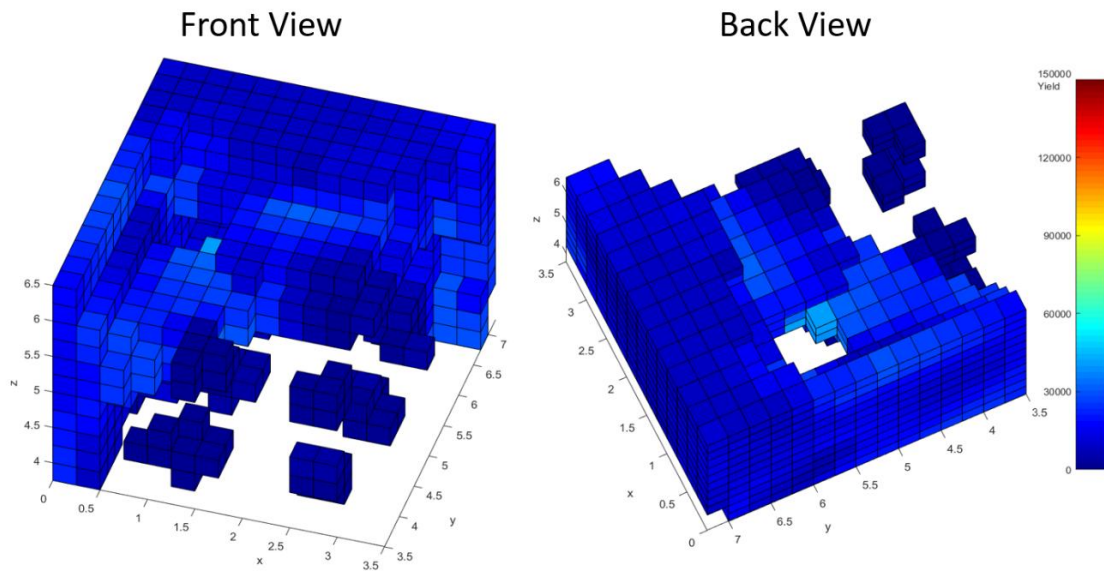


Figure 6-11: Side Rib, View Window $X(0,3.5) Y(3.5,7.25) Z(3.75,6.5)$

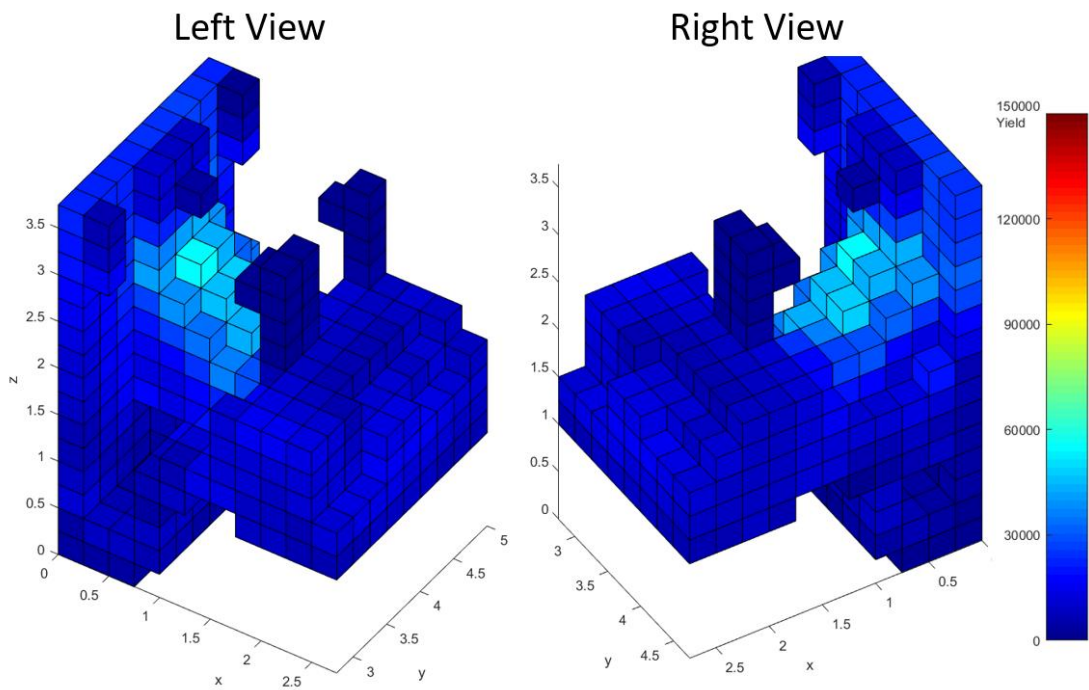


Figure 6-12: Bottom Rib, View Window $X(0,2.75) Y(2.75,5) Z(0,3.75)$

Another common feature found in many of the results from optimizing this domain was a connection beam from the front to the back surfaces. An example of this can be seen in an intermediate design at iteration 25 of 93 for trial #33, shown below in figure 6-13. Here the optimization has over-shot the target volume and is currently at a volume fraction of 0.345.

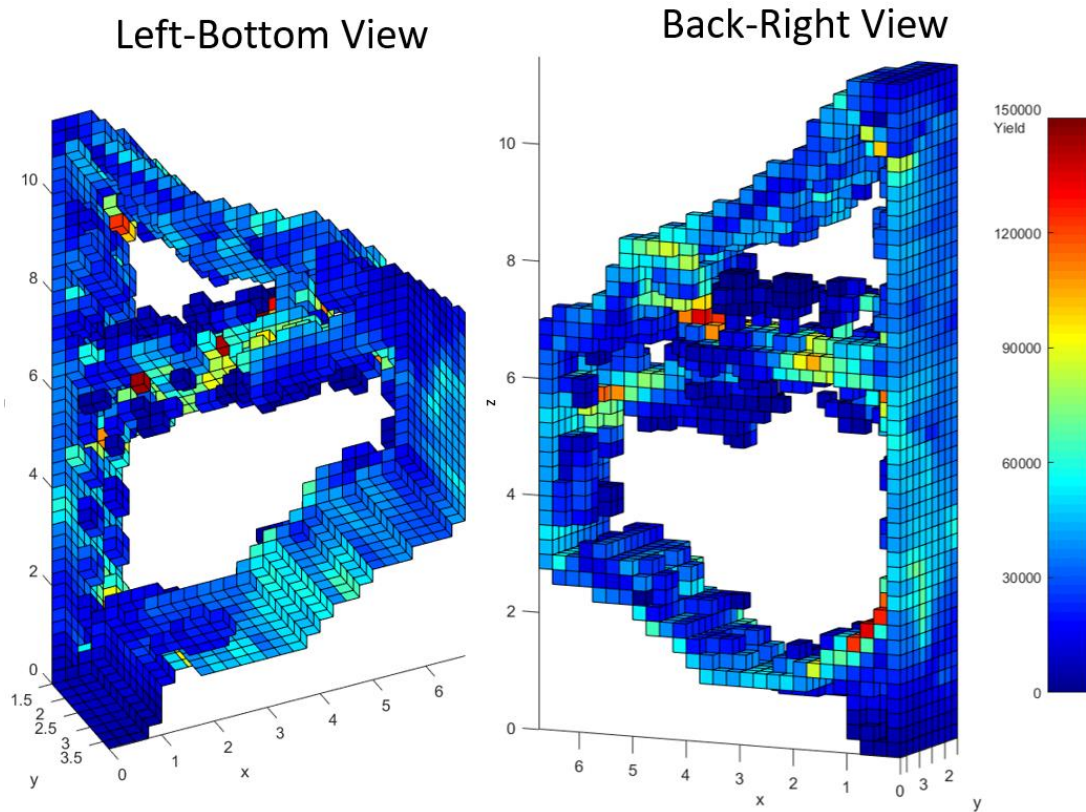


Figure 6-13: Connection Beam, Trial 33, Iteration 25, View Window $X(0,6.75)$ $Y(1.5,3.75)$ $Z(0,11.5)$

Once parameters that provided stable convergence were determined, the optimization was run with varying target volumes to determine a Pareto front. The results from these optimizations can be seen in table 6-8. As expected, this shows that as material volume increases, the compliance and stresses decrease. This was the case for all target volumes, with the exception of 50% material, where it is believed that the algorithm stalled at a local

minimum. Note the maximum iterations was set to 200, and three of these trials reached this criterion, despite having final volumes extremely close to their target volume.

Table 6-8: Trials at Varying Target Volumes

Target Volume Fraction	Final Volume Fraction	Void Volume (in.3)	Final Compliance (in-lb/in3)	Max VonMises Stress (PSI)	Average VonMises Stress (PSI)	Iterations
0.25	0.2400	282.4	40,204	320,266	36,268	200
0.30	0.2993	260.4	11,077	143,465	22,953	60
0.35	0.3527	240.5	8,988	143,300	19,233	58
0.40	0.3999	223.0	5,392	73,471	14,812	200
0.45	0.4502	204.3	3,539	54,128	11,596	76
0.50	0.5013	185.3	3,737	82,753	14,368	200

Additionally, each iteration from all of the executed trials were compiled into a scatter plot, figure 6-14, to grasp an idea of the design space available with the given domain along with a visual for the Pareto front. Final values from the runs shown in table 6-8 are highlighted and labeled. Note, the majority of the trials were done with a target volume fraction of 0.45, thus the heavy clustering in this region. The fact that some of the volume specific trials were not along the Pareto front compared to other design points is indicative of the algorithm finding local minima. Additionally, it should be noted that these points that would be pareto optimal to the volume specific results were found mid-way through highly unstable trials that never converged.

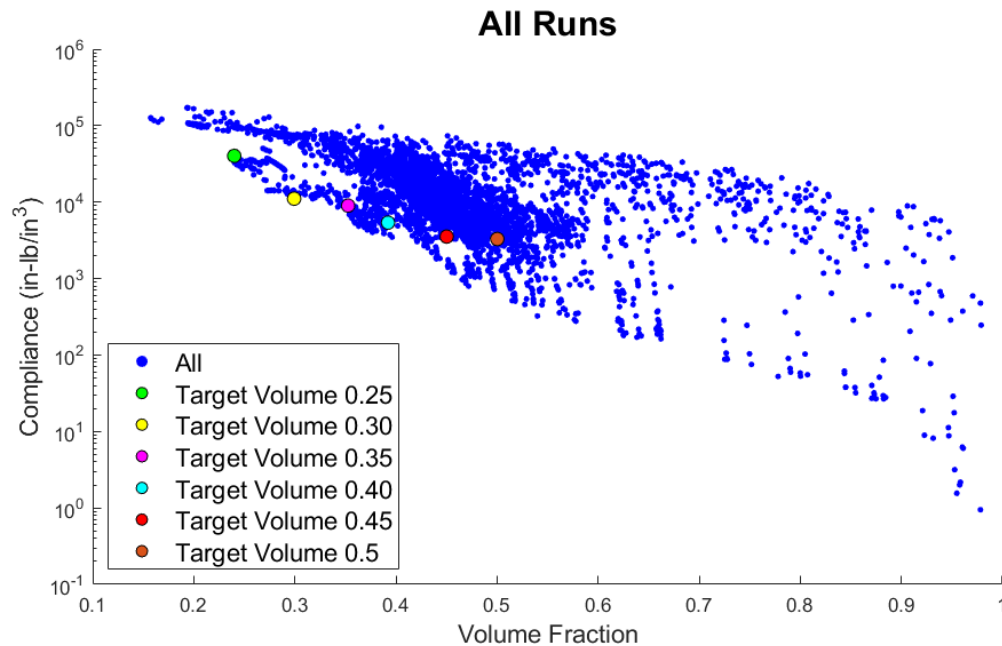


Figure 6-14: Design Points Volume Fraction Versus Compliance

6.4 Final Results

After analyzing the results with the implementation used for the 3-D pressure box, section 5.3.2, it was determined that the structures were sub-optimal, based on the apparent pareto-front with some solutions dominating it, the non-conclusive convergence behavior, and the presence of checkerboarding. It was believed that the meshed elements' size, along with the filtering techniques, were the main contributors to these undesired results. The use of a finer mesh proved impractical due to the computational burdens caused by the increase in the number of degrees of freedom, as shown in table 6-7. Trials were executed with 0.2" voxel elements, however, any smaller elements were infeasible. Here, is where it was conceived to change the definition of the pressure loading from the void region to the material domain, figure 3-3 and equation 3.28, allowing for the mesh to begin the optimization with 0.25" voxel elements and re-mesh excluding void regions as the voids increased in size. This would in turn

allow for increased geometric definition while remaining under the computational limits. This schematic change also addressed the filtering technique. Because the elements would be re-meshed to a smaller size, the LSF had to be decoupled from the FEA mesh. To correlate the elements' sensitivity to the LSF kernels, the weighted neighborhood filtering scheme is implemented, equation 4.2. These changes proved to improve structural performance and aid against checkerboarding. The results of this final implementation are shown here in this section.

Immediately following this schematic change, the identical starting structure and PID gains from section 6.3 were tried, $\vec{K} = [1 \quad 0.5 \quad 0.2]$. Additionally, it was observed that initial Lagrange multipliers of $\lambda_o = 0.5$ and $\lambda_{PID} = 0.5$ were required to prevent the structure from turning completely solid, section 4.4.5. To slow the increase of the penalty, a Lagrangian update factor of $\alpha = 1.05$ was used. Figure 6-15 shows the volume and compliance versus iteration from this execution. As seen in the figure, following the start of the PID implementation, the system experiences unstable oscillation. However, prior to this, the compliance values can be seen as far superior than the previous implementation, an indication of the removal of checkerboarding (section 6.3).

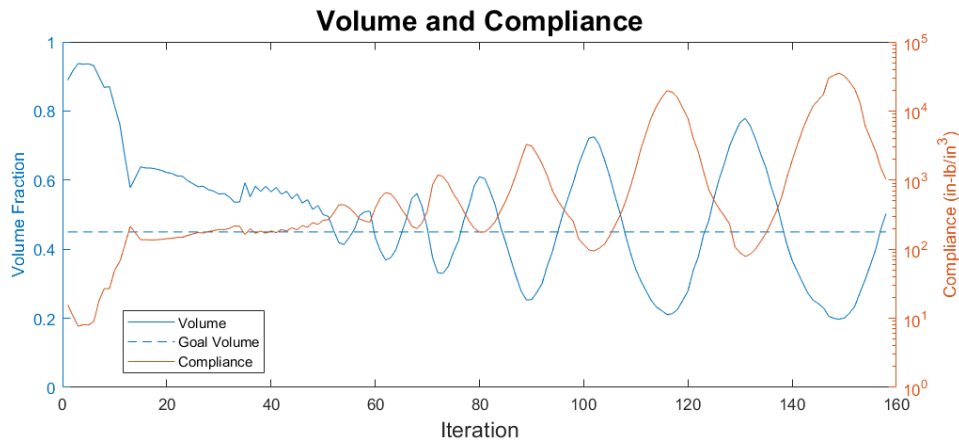


Figure 6-15: Volume and Compliance, $\vec{K} = [1 \quad 0.5 \quad 0.2]$, $\lambda_o = 0.5$, and $\lambda_{PID} = 0.5$

To prevent this unstable oscillation, the proportional gain was reduced, and the derivative gain was increased, to a new set of gain values of $\vec{K} = [0.5 \quad 0.2 \quad 1]$. The results of this trial can be seen in figure 6-16, where the stable convergence behavior can be observed after 194 iterations.

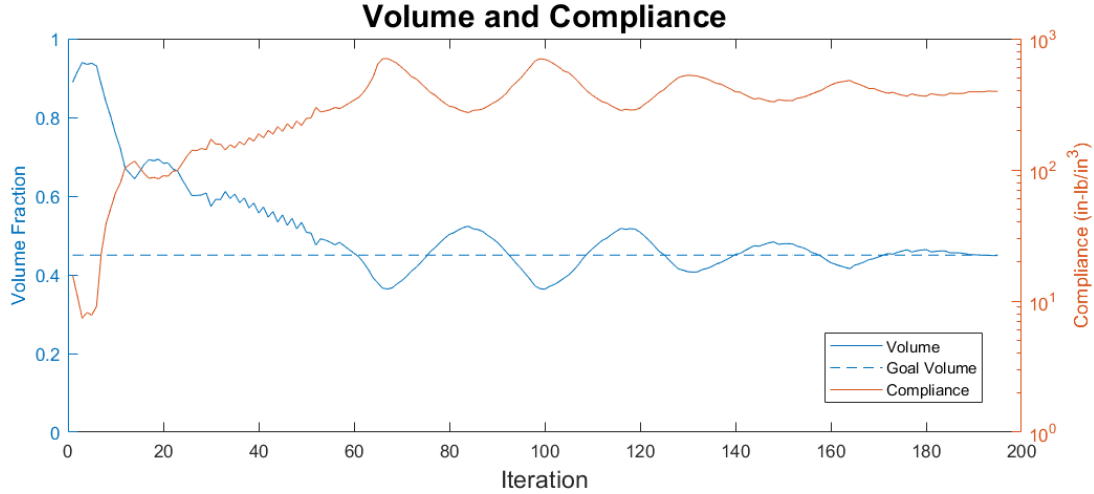


Figure 6-16: Volume and Compliance, $\vec{K} = [0.5 \quad 0.2 \quad 1]$, $\lambda_o = 0.5$, and $\lambda_{PID} = 0.5$

After observing the results, it was determined that there were several geometric features with a thickness of one element as the volume first crosses the target volume. Then upon the first undershoot of the target volume, some of these features are removed and once the LSF reinitializes it would not be able to add these features back. To minimize the overshoot of the optimization once the PID terms kick in, the Lagrange multiplier was increased to $\lambda_{PID} = 1$, for a faster response in control change. Additionally, the Hamilton-Jacobi time step was reduced to 30% of the CFL condition, equation 3.41. As seen in figure 6-17, this resulted in less initial undershoot and a faster convergence.

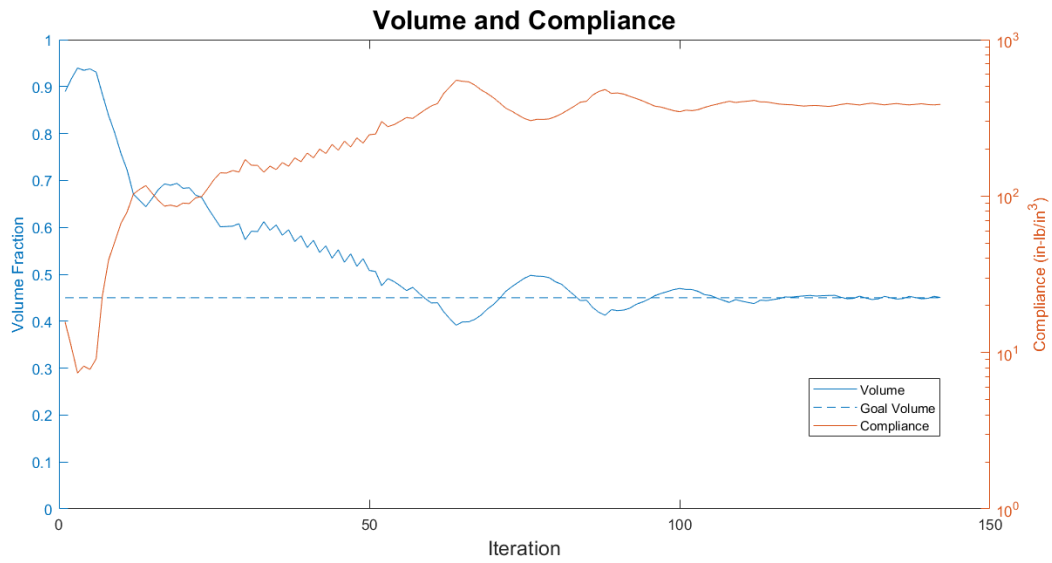


Figure 6-17: Volume and Compliance, $\vec{K} = [0.5 \quad 0.2 \quad 1]$, $\lambda_o = 0.5$, and $\lambda_{PID} = 1$

These algorithm parameters resulted in the optimization converging in 142 iterations with a final compliance of 385.81 in-lb/in³, maximum VonMises stress of 49,179 PSI and an average stress of 8,231 PSI. The final structure was comprised of 0.15" voxel elements. The stress plots with the four sides' outer walls removed, leaving the top and bottom uncropped, are shown in figures 6-18 with front views and in 6-19 with rear views. Refer to figure 6-4 for orientation and face descriptions.

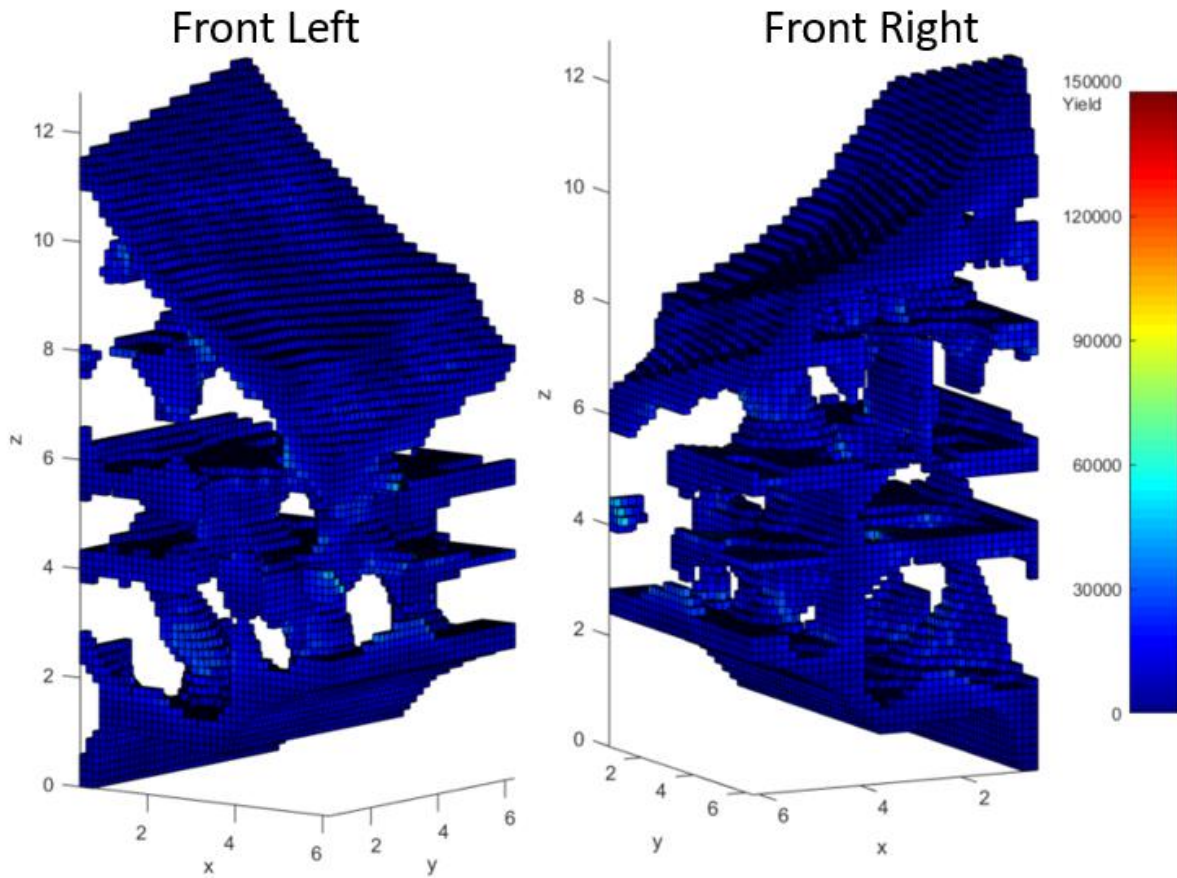


Figure 6-18: $V_{req}=0.45$ Final, Interior Front View, Window: $X(0,6.75)$ $Y(2,6)$ $Z(0,12.25)$

As seen in figure 6-18, it is evident that this implementation provided improved results from section 6.3 as there are numerous structural features present and no signs of checkerboarding. It is apparent that the algorithm developed several layers to subdivide the domain vertically and support the larger exterior surfaces by joining them together. Despite this, there are sufficient openings such that the geometry provides one continuous void region.

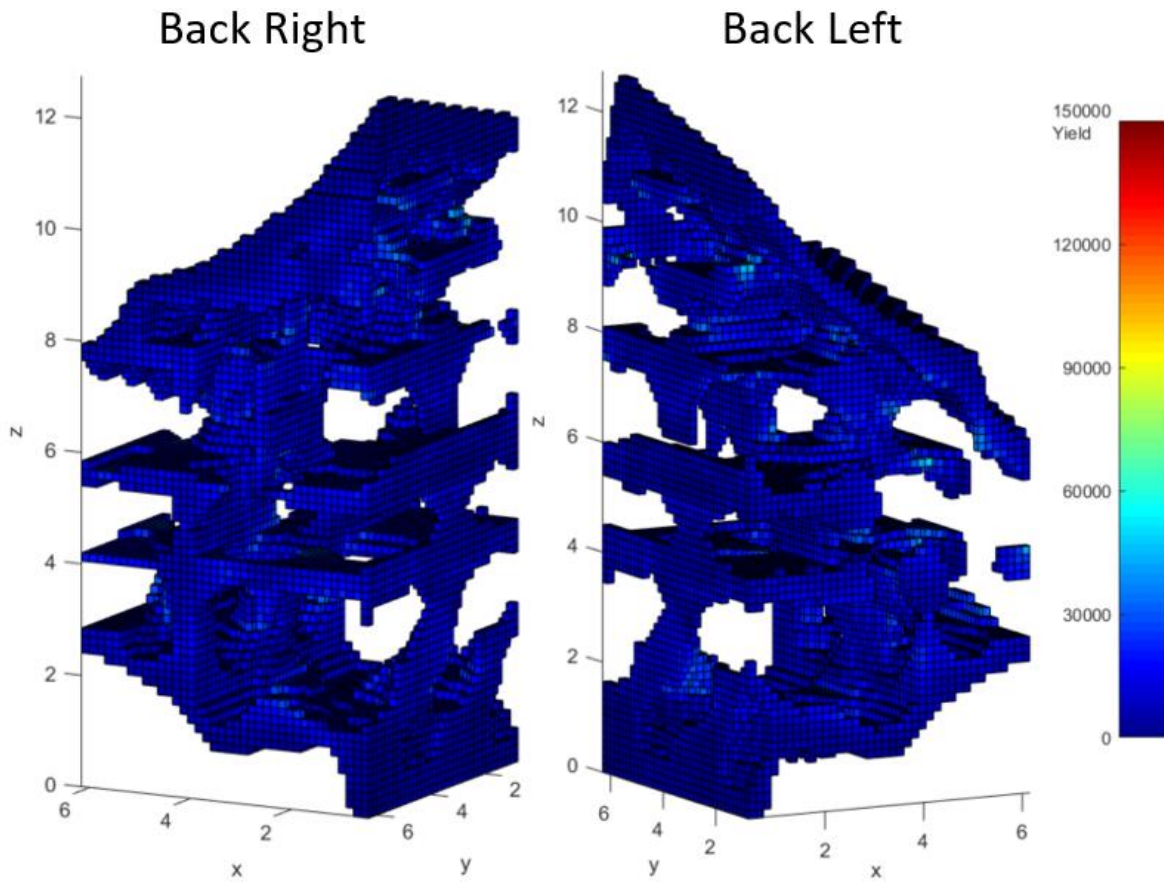


Figure 6-19: $V_{req}=0.45$ Final, Front View, Window: $X(0,6.75)$ $Y(2,6)$ $Z(0,12.25)$

To better convey the resulting structure, each of these 'layers' of the final geometry are shown in the following figures as cross-sections in the z-direction and the full domain shown in the x and y directions. Figure 6-20 shows the upper cross-section from $z=8.7''$ to $z=12.75''$ while 6-21 through 6-24 show the cross-sections between $8.7''$, $7.5''$, $5.4''$, $3.9''$, and $0''$ respectively.

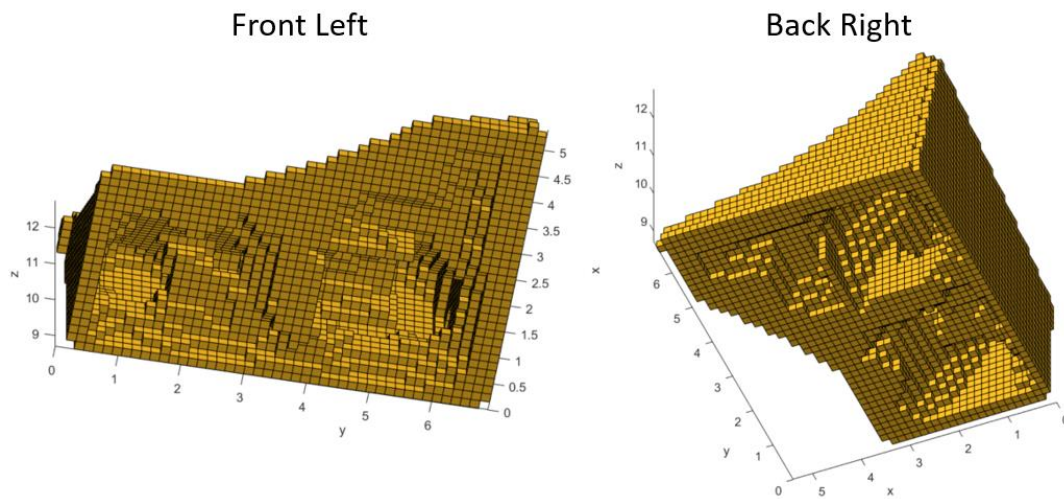


Figure 6-20: Final Structure from Bottom, Z=8.7" Through Z=12.75"

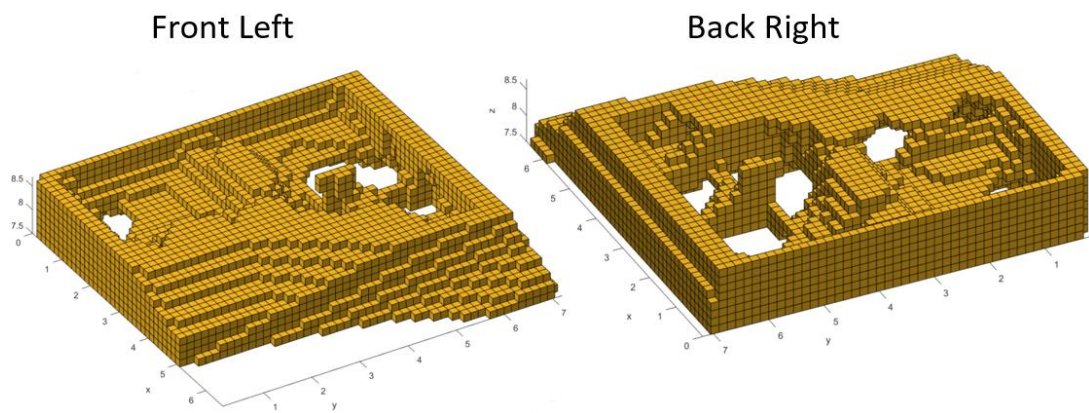


Figure 6-21: Final Structure, Z=7.5" Through Z=8.7"

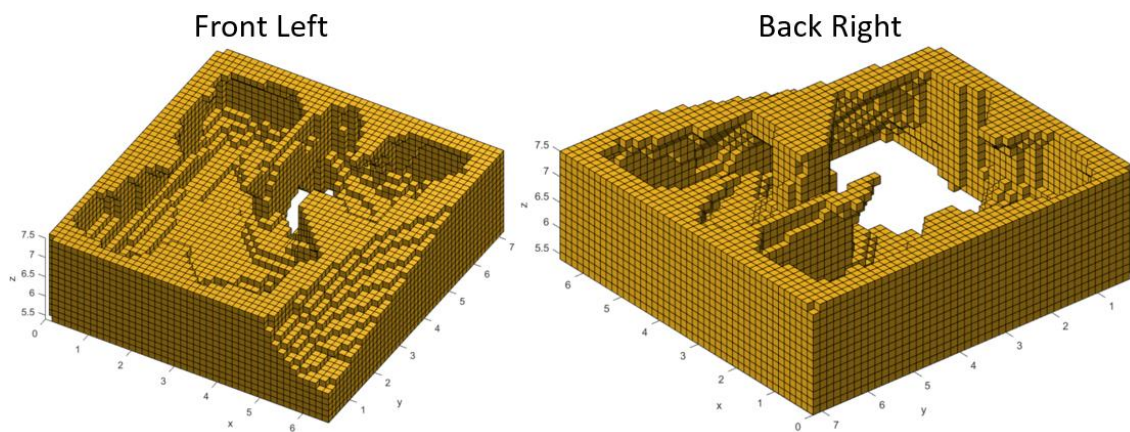


Figure 6-22 Final Structure, Z=5.4" Through Z=7.5"

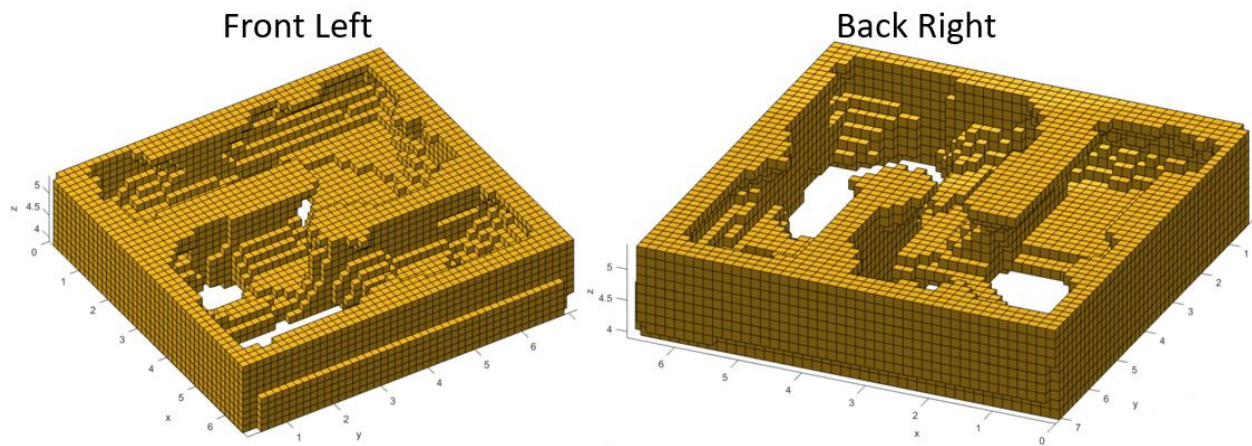


Figure 6-23: Final Structure, $Z=3.9''$ Through $Z=5.4''$

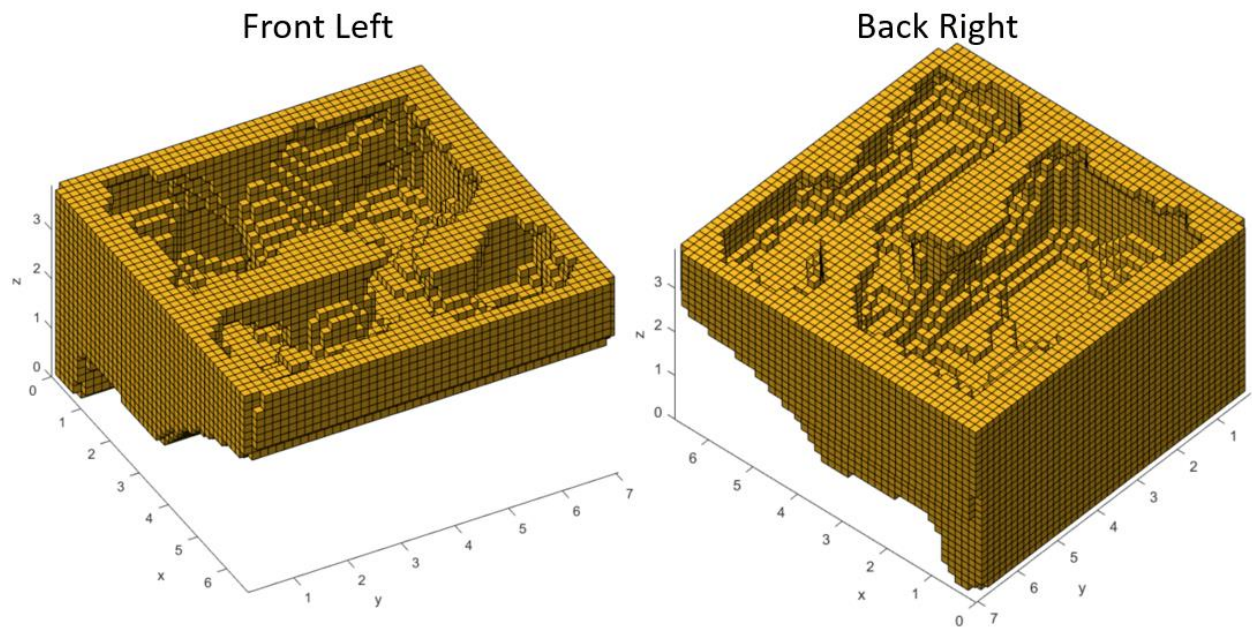


Figure 6-24: Final Structure, $Z=0''$ Through $Z=3.9''$

To get a smoother representation of the structure, the level-set function itself was converted to an STL file to model the void. Then this void structure was combined with the original exterior model of the irregular pressure vessel, as shown in figure 6-4, and subtracted using a Boolean operation. This results in a final smoothed geometry by eliminating voxelated structure. These smoothed results can be seen at various cross sections throughout the geometry in figures 6-25, 6-26, and 6-27.

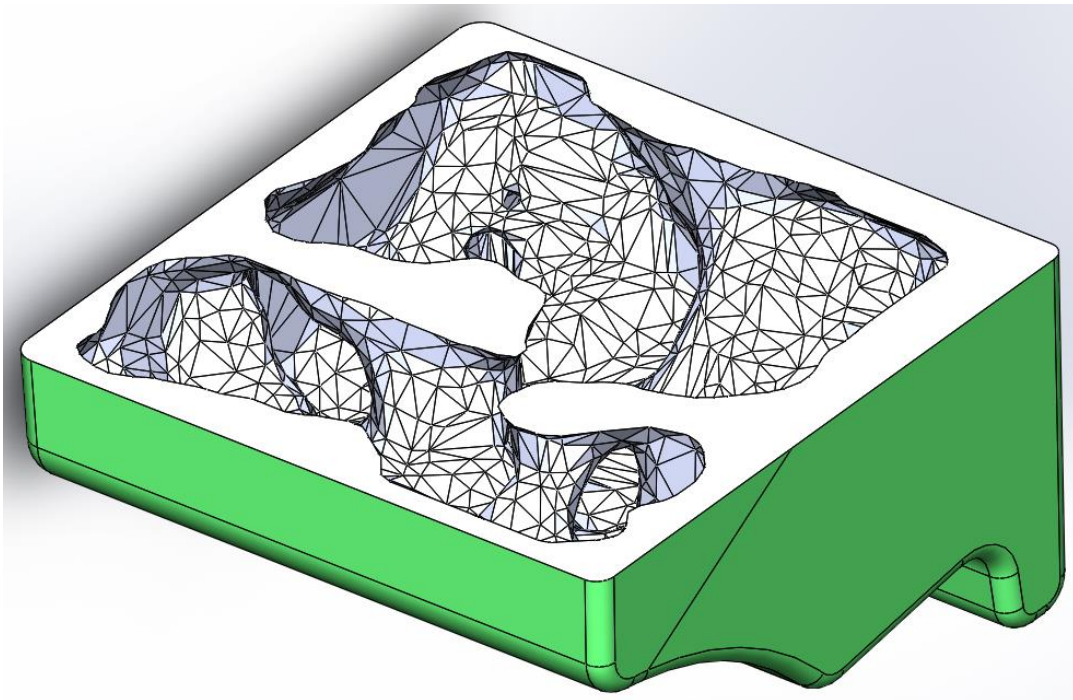


Figure 6-25: Smoothed Geometry Bottom

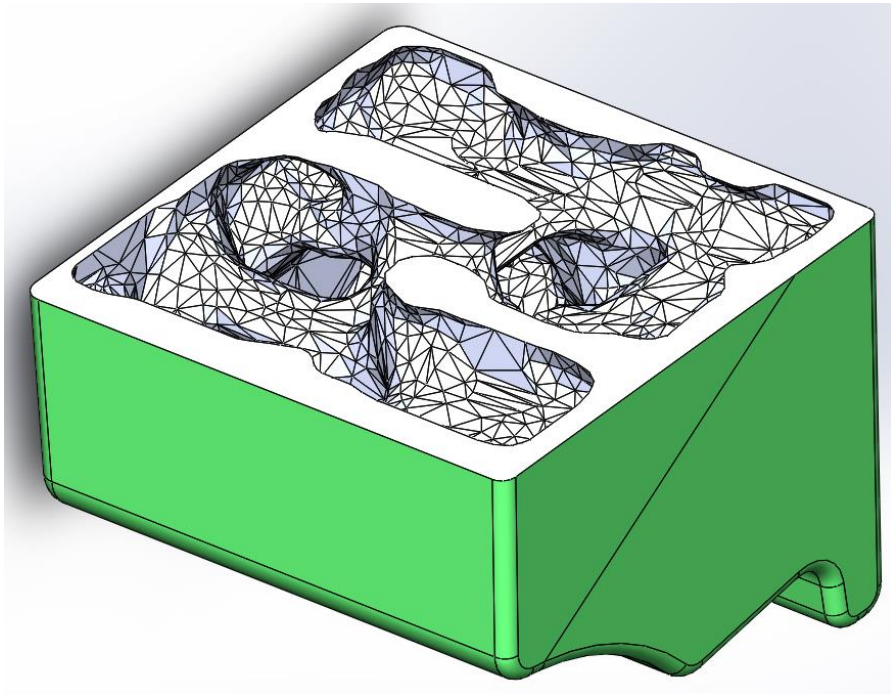


Figure 6-26: Smoothed Geometry Middle

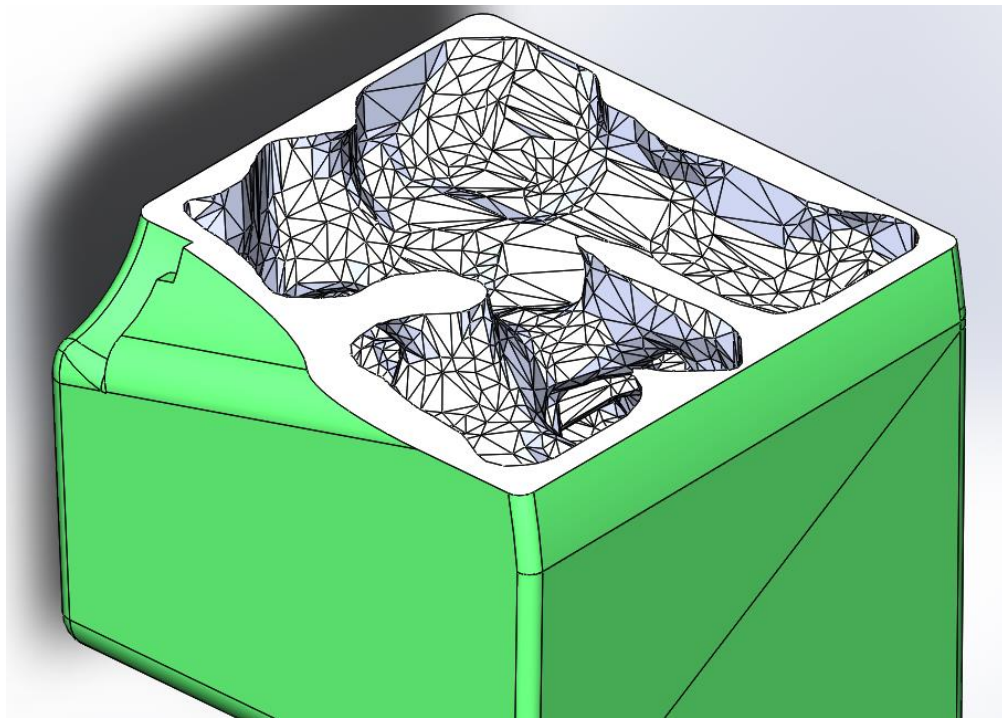


Figure 6-27: Smoothed Geometry Top

Similar to section 6.3, once the algorithm was performing properly for a volume fraction of 45%, multiple trials were executed at varying volume fractions to develop a Pareto curve for the objective design space. The results from these trials can be seen in table 6-9. Recall the existing spherical pressure vessel having a void volume of 175 in.³, thus each of these volume fractions would achieve an improvement in storage capacity. The compliance achieved by these various volume fractions is not linear and shows drastic increases with volume fractions less than 40%. Although this is to be expected as the boundary elements are required to remain solid, the use of discrete voxel elements of such large size would have a much more drastic effect on this. This is because at 0.25" elements the boundary consist of 23.8% of the total elements and 14.7% with 0.15" elements, table 6-7, therefore the remaining available volume to generate support features is drastically limited. Additionally, the use of voxel elements has an impact on the maximum stresses as it causes stress concentrations that would be able to be reduced with a conforming mesh allowing for the smoothing of surfaces. Thus, the high peak stresses yet low average stresses.

Table 6-9: Remeshing Trials at Various Target Volumes

Target Volume Fraction	Achieved Volume Fraction	Void Volume (in.3)	Achieved Compliance (in-lb/in3)	Max VonMises Stress (PSI)	Average VonMises Stress (PSI)	Iterations
0.25	0.246	278.10	5,619.19	462,720	24,341	91
0.30	0.297	259.01	1,127.85	178,185	12,931	115
0.35	0.351	239.04	860.33	115,611	12,310	200
0.40	0.394	223.28	564.49	86,540	7,465	170
0.45	0.450	202.60	385.81	49,179	8,231	142
0.50	0.504	182.857	264.45	52,086	5,877	95

Additionally, all of the iterations from every trial executed were plotted to visualize this Pareto front, as seen in figure 6-28. This figure provides a clear understanding of the minimum achievable compliance at a specific volume fraction. The tightness of the scatter points,

compared to figure 6-14, indicated the improved performance and removal of checkerboard. Figure 6-29 shows when visualizing the points for one trial with a large initial overshoot, such as the one shown in figure 6-16, it was apparent that the designs follow the pareto front, then after the overshoot compliance values were offset from the pareto front as the volume fraction increased back towards the target volume. This hysteresis type behavior indicates the inability to return to previous designs once support features have been removed. Thus, the motivation to reduce the amount of overshoot, as discussed at the beginning of this section. Additionally, these reasons combined with the lack of available material are believed to be the cause of both trials at 25% and 30% volume to not be along the pareto front, as they are dominated by other designs at their volume fraction.

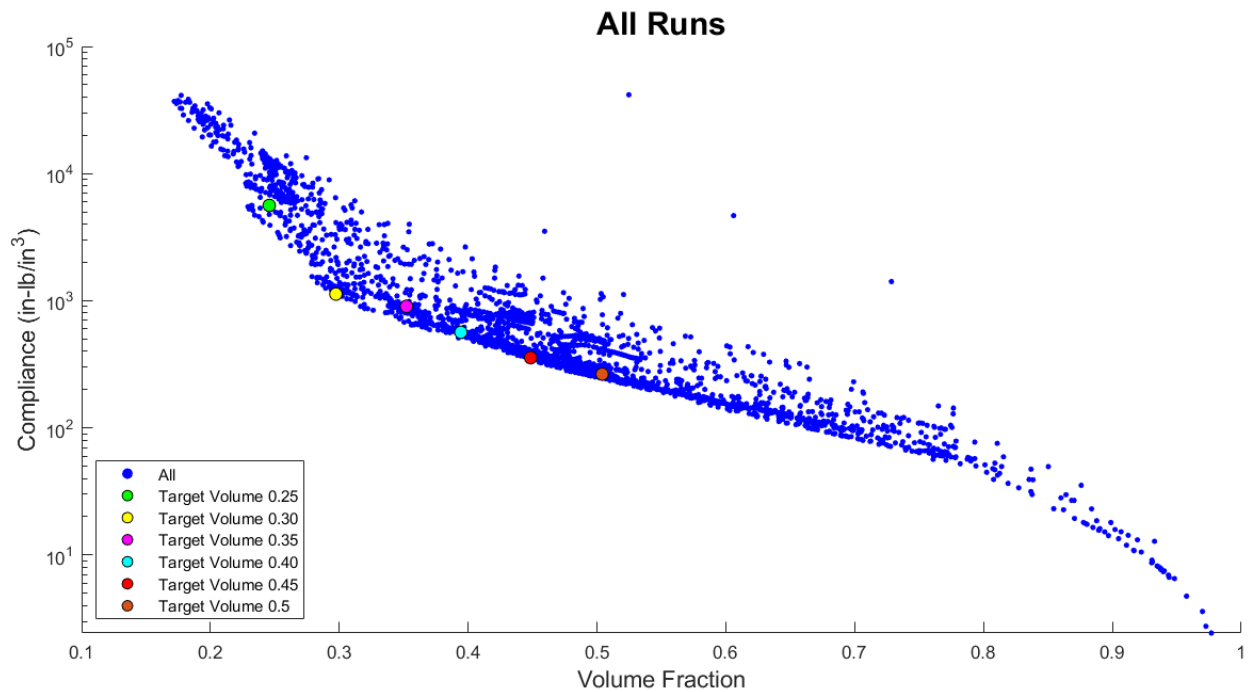


Figure 6-28: Remeshing Pareto Front

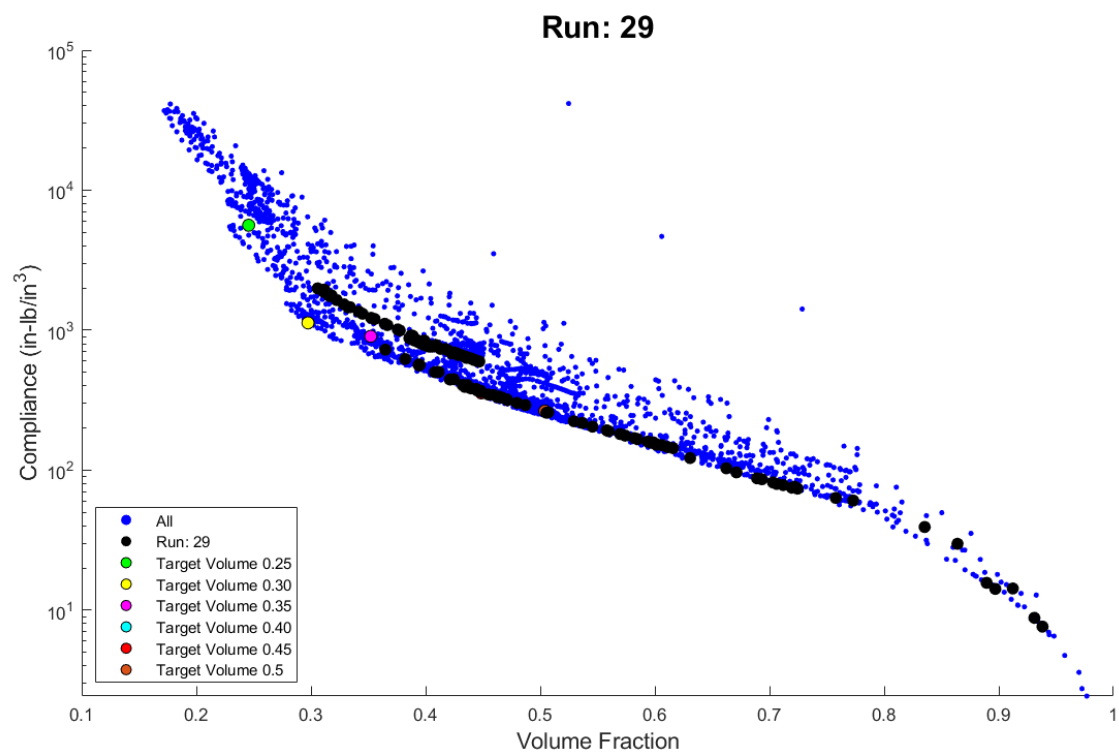


Figure 6-29: Volume Fraction and Compliance Designs from Trial #29

CHAPTER VII: CONCLUSION

Overall, the research presented in this thesis establishes a means to implement topology optimization on an irregularly shaped pressure vessel. Chapter 1 discusses the motivation behind conducting such research and lays down research objectives and questions. Discussed further in section 7.2, the results of this research have deemed conclusive to both research questions proposed in section 1.2. In pursuit of these research questions, a theoretical irregular shaped pressure vessel was designed for the MK-16 rebreather, in which an increased gas storage capacity was achieved. This chapter is organized as follows: section 7.1 summarizes the work that was done for this research by following the chapters outlined in this thesis, section 7.2 addresses the research questions posed at the beginning of this research, and sections 7.3 discusses future works that should be done to further the developments made by this research. Section 7.4 wraps up final remarks and main takeaways from this research.

7.1 Thesis Overview

Following the introduction to the research objectives and motivation, an extensive literature review was conducted, chapter 2. Here, the foundations and variations of topology optimization were examined. The field of topology optimization was broken into 3 major methods: a ground structure approach, homogenization methods, and level-set methods. Additional research was conducted to observe any current methods of tackling design dependent pressure loading problems, where two main methods were found for problems in \mathbb{R}^2 . The first utilized a modification of the SIMP method, a branch of homogenization methods, where iso-density points throughout the design domain are consecutively connected to establish a location to apply the pressure forces. This method proved ineffective and would be

computationally demanding when extrapolated to 3-dimensional problems, figures 2-12, 2-13, and 5-17. The second method implements the level-set method with two independent level-set functions, one to define the homogeneous free boundary and the other to define the Neumann pressure boundary, figures 3-4, 5-18, and 5-19. With the knowledge gathered from the literature review it was determined that the use of a level-set method would be best suited to optimize a pressure vessel in \mathbb{R}^3 .

Chapter 3 then goes on to lay out the governing equations and mathematical derivation of the methods that were used to conduct this optimization. Ensuing these derivations, chapter 4 dives into the practical implementation procedure used to execute these methods. For the extent of this research, all computations were performed in MATLAB, and the developed source code can be viewed in the appendices or online at: https://github.com/JKremar/Irregular_Pressure_Vessel_Topology_Optimization. Section 4.1 also lays out the progression of the research problem as it evolved step by step, incorporating additional complexities until reaching the overall research objectives.

Chapter 5 presents the preliminary results of the initial phases of this progression. Here, standard level-set methods in \mathbb{R}^2 were developed for constant loading conditions before expanding to \mathbb{R}^3 . Then, design dependent pressure loading was explored in \mathbb{R}^2 , followed by a simplified pressure loading problem in \mathbb{R}^3 , where the design domain was restricted to rectangular cuboid shapes. As explained in section 3.3.2, at this phase of the progression a PID-type penalty scheme was developed and implemented to improve convergence behavior. After the groundwork had been established, the developed method was evaluated for redesigning the existing spherical pressure vessels used to store breathing gases in a MK-16 rebreather into an

irregular shaped pressure vessel for increased storage capacity. The results of this can be seen in chapter 6. In order to achieve proper optimization performance and convergence behavior, the LSF was decoupled from the FEA mesh and the structural domain was allowed to re-mesh to a smaller element size as the material volume fraction was decreased.

7.2 Discoveries from Research Objectives

The beginning of this research, section 1.2, established 2 research questions and their associated hypothesis. To discuss the discoveries of this thesis in regard to each question, they are restated here for reference:

RQ1: “Can the interior geometry of an irregular shaped pressure vessel, subjected to internal pressure on its surfaces, be designed to efficiently store high pressure gas using topology optimization methods?”

H1: “Yes, topology optimization can be used to design the internal structure of such an irregular shaped tank, that could then it can be manufactured using additive manufacturing.”

Discovery: Yes, topology optimization can be used to design an irregular shaped pressure vessel.

It was determined that the use and modification of a level-set method proved beneficial in doing so by providing crisp material and void distinctions throughout the optimization process. Manufacturing and testing of the resulting structures were not conducted within the scopes of this research and are further expanded upon in the future works of section 7.4.3.

RQ2: “Can an efficient method be developed to track (follow) design dependent pressure loading conditions on the interior surface for 3-dimensional spaces for use in a topology optimization algorithm?”

H2: “By adapting a level-set topology optimization approach, it is possible to track changing pressure surfaces as the design evolves during the iterative design process.”

Discovery: Although a level-set method explicitly defines the material/void boundary of the design domain, it was determined that the simplest and most effective means of applying design dependent pressure loading was to apply an inward normal stress on each element containing material and zeroing nodes along the homogeneous boundary. This resulted in an equal and opposite cancelation of forces throughout the interior of the material domain where elements were adjacent to each other. The cancellation of these adjacent forces, left only desired forces along the Dirichlet boundary, located at the interior material/void boundary of the pressure vessel.

7.3 Future Works

Although this research effectively addressed each of the research questions proposed at the beginning of this work, there remains a great deal of tasks that would need to be completed before an irregular shaped pressure vessel could be efficiently designed, manufactured, and used. The majority of these tasks fall into the categories of further refining the mesh, utilizing stress constraints, properties of additive manufacturing, and experimental evaluation.

7.3.1 Refined Meshing

Although the final implementation used here in this thesis refines the mesh as the volume fraction decreases and the structure begins to converge, it still utilizes a grid of voxel elements. To more appropriately represent the structure and evaluate stresses, the finite element method should utilize a much more refined mesh. This is primarily due to the presence of stress concentrations caused by the jagged structural representation of the gridded voxel elements. Additionally, an improved meshing technique would allow for a much better structural representation and control of design changes throughout the optimization, as it is believed this was the cause of many of the difficulties experienced during this research. Two possible means of accomplishing this task could be the adoption of a conforming mesh or an immersed boundary technique, section 2.2.2. Although to effectively utilize either of these would result in an enormous increase in computational costs.

7.3.2 Stress Constraints

As mentioned in section 3.2, ideally the optimization problem would be posed as to minimize structural volume (maximizing void volume) subject to a maximum stress constraint. However, as found during the literature review (section 2.2.3), this causes added difficulty in developing proper update sensitivities for optimization. Additionally, as mentioned in section 2.2.2, to properly impose stress constraints, a conforming mesh is required, thus the future work mentioned in 7.4.1. Because of these reasons, the use of stress constraints was deemed out of the scopes of this research, and the minimum compliance formulation was used with post process stress evaluations.

7.3.3 Designing for Additive Manufacturing

Due to the internal features of an irregular shaped pressure vessel, the components would need to be created using additive manufacturing techniques. The material properties of additive manufacturing are known to be non-isotropic based on print direction and heavily dependent on a wide variety of print parameters. Because of this, printed samples with known and controllable design parameters should be created and tested to accurately establish material parameters to be used during the analysis phase of the optimization. Additionally, because of the completely enclosed nature of a pressure vessel, support materials could not be manually removed, and the part would need to be printed at an appropriate angle to allow for a proper build without the need of support material. Due to the non-isotropic behavior and the inability to use support material, it could prove advantageous to have print direction as an additional optimization parameter.

Furthermore, the results from the optimization would need to go through a post-processing phase, during which features such as the connecting ports would need to be added.

7.3.4 Experimental Validation

Due to the high values of strain energy within an in-use pressure vessel, they can be extremely dangerous upon failure. This combined with the life supporting functionality they often possess; pressure vessels are subjected to extensive validation and safety testing. For diving and life support purposes, this involves surviving a hydrostatic burst test at four times working pressure. This gives the system a factor of safety of 4, suitable for proper use and safe handling. Additionally, visual, and ultra-sonic inspections are regularly performed on pressure vessels to detect any defects or deformities. This would prove quite difficult to accomplish

based on the compartmentalization of the irregular shaped pressure vessels derived in this thesis and some other means of inspection would have to be executed.

7.4 Final Remarks

This thesis investigates various topology optimization methods and concludes that the modification of a Level-Set Method best suits the design problem of optimizing an irregular shaped pressure vessel. Throughout the development of this modified Level-Set Method for pressure vessels, 3 main innovations were discovered and implemented:

1. FEA assembly consolidation:

Discussed in section 4.4.1, during the optimization loop, the previous iteration's structure, stiffness matrix, and force vector are stored. Then during the finite element analysis procedure, the current and previous structures are compared, and the assembly procedure is executed only on the changed elements and modified from the previous stiffness matrix and force vector accordingly. This reduces the computational time from an estimated 175 hours to less than 10 hours. Note, the entire assembly process has to be executed following remeshing, therefore excessive re-meshing would result in added computational time.

2. PID-type optimization penalty:

When testing various methods found in the literature of implementing the volume constraint penalty to solve unstable convergence issues, similarities to control concepts of proportional, integral, and derivative controllers were

recognized. This inspired the implementation of a PID-type penalty, converting the penalty from equation 3.38 to equation 3.56.

$$Penalty_i = \lambda_i (V(x) - V_{req}) \quad (3.38)$$

$$Penalty_i = Penalty_{i-1} + \lambda_i \left[K_P (V_i - V_{req}) + K_I \left(\frac{1}{n} \sum_{a=0}^{n-1} V_{i-a} - V_{req} \right) + K_D (2V_i - V_{i-1} - V_{req}) \right] \quad (3.56)$$

3. Inward normal pressure calculations:

Switching the pressure loading from being calculated as outward normal, as shown in figure 3-2, to being defined as inward normal, figure 3-3, removes the need to have the entire design domain meshed. This allows void regions to be excluded from finite element meshing, reducing computational costs, or allowing for mesh refinement while maintaining computational cost.

This work establishes a strong foundation on which additional work can be built upon to finalize a thorough and robust procedure to design an optimal irregular shaped pressure vessel to exploit the expanding design space that additive manufacturing provides.

REFERENCES

- [1] M. J. TURNER, R. W. CLOUGH, H. C. MARTIN, and L. J. TOPP, "Stiffness and Deflection Analysis of Complex Structures," *J. Aeronaut. Sci.*, vol. 23, no. 9, pp. 805–823, Sep. 1956, doi: 10.2514/8.3664.
- [2] M. P. Bendsoe and O. Sigmund, *Topology Optimization Theory, Methods and Applications - Second Edition*, 2nd ed. Springer-Verlag Berlin Heidelberg, 2004.
- [3] M. L. Nuckols, W. C. Tucker, and A. J. Sarich, *Life Support Systems Design: Diving and Hyperbaric Applications*. Needham Heights, Mass: Simon & Schuster Custom Publishing, 1996.
- [4] McLaughlan P. B. and Grimes-Ledesma L. R., "Composite Overwrapped Pressure Vessels," *Nasa/Sp-2011-573*, no. March, pp. 1–20, 2011.
- [5] J. Cornman, "Additive Manufacturing of Pressure Vessels." Naval Surface Warfare Center, Panama City Division, Panama City, FL, p. 1, 2017.
- [6] L. A. Schmit and R. L. Fox, "An integrated approach to structural synthesis and analysis," *AIAA J.*, vol. 3, no. 6, pp. 1104–1112, 1965, doi: 10.2514/3.3062.
- [7] L. Kai and T. Andres, "An efficient 3D topology optimization code written in Matlab," *Springer*, vol. 50, no. 6, pp. 1175–1196, 2014, doi: 10.1007/s00158-014-1107-x.
- [8] B. H. V. Topping, "Shape optimization of skeletal structures: A review," *J. Struct. Eng. (United States)*, vol. 109, no. 8, pp. 1933–1951, 1983, doi: 10.1061/(ASCE)0733-9445(1983)109:8(1933).

- [9] G. Allaire, E. Bonnetier, G. Francfort, and F. Jouve, "Shape optimization by the homogenization method," *Numer. Math.*, vol. 76, no. 1, pp. 27–68, 1997, doi: 10.1007/s002110050253.
- [10] J. A. Sethian, *Level Set Methods and Fast Marching Methods Evolving Interfaces in Computational Geometry*, vol. 39, no. 1. Cambridge University Press, 1999.
- [11] G. I. N. Rozvany, "Aims, scope, methods, history and unified terminology of computer-aided topology optimization in structural mechanics," *Structural and Multidisciplinary Optimization*, vol. 21, no. 2, pp. 90–108, Apr-2001, doi: 10.1007/s001580050174.
- [12] O. Sigmund, "A 99 line topology optimization code written in matlab," *Struct. Multidiscip. Optim.*, vol. 21, no. 2, pp. 120–127, Apr. 2001, doi: 10.1007/s001580050176.
- [13] V. J. Challis, "A discrete level-set topology optimization code written in Matlab," *Springer*, vol. 41, no. 3, pp. 453–464, 2010, doi: 10.1007/s00158-009-0430-0.
- [14] M. Ohsaki, "Genetic algorithm for topology optimization of trusses," *Comput. Struct.*, vol. 57, no. 2, pp. 219–225, Oct. 1995, doi: 10.1016/0045-7949(94)00617-C.
- [15] S. Y. Wang, K. Tai, and M. Y. Wang, "An enhanced genetic algorithm for structural topology optimization," *Int. J. Numer. Methods Eng.*, vol. 65, no. 1, pp. 18–44, Jan. 2006, doi: 10.1002/nme.1435.
- [16] E. Biyikli and A. C. To, "Proportional topology optimization: A new non-sensitivity method for solving stress constrained and minimum compliance problems and its implementation in MATLAB," *PLoS One*, vol. 10, no. 12, Dec. 2015, doi: 10.1371/journal.pone.0145041.

- [17] M. J. de Ruiter and F. van Keulen, "Topology Optimization: Approaching the Material Distribution Problem using a Topological Function Description," in *Computational Techniques for Materials, Composites and Composite Structures*, 2000, doi: 10.4203/ccp.67.1.13.
- [18] T. Hagishita and M. Ohsaki, "Topology optimization of trusses by growing ground structure method," *Struct. Multidiscip. Optim.*, vol. 37, no. 4, pp. 377–393, Jan. 2009, doi: 10.1007/s00158-008-0237-4.
- [19] F. Mignot, J. P. Puel, and P. M. Suquet, "Homogenization and bifurcation of perforated plates," *Int. J. Eng. Sci.*, vol. 18, no. 2, pp. 409–414, 1980, doi: 10.1016/0020-7225(80)90060-9.
- [20] G. ALLAIRE, L. CAVALLINA, N. MIYAKE, T. OKA, and T. YACHIMURA, "The Homogenization Method for Topology Optimization of Structures: Old and New," *Interdiscip. Inf. Sci.*, vol. 25, no. 2, pp. 75–146, 2019, doi: 10.4036/iis.2019.b.01.
- [21] G. Allaire, "Periodic homogenization - Asintotic expansion homogeneization (Master lect. 1)," no. December, pp. 13–16, 2010.
- [22] G. I. N. Rozvany, M. Zhou, and T. Birker, "Generalized shape optimization without homogenization," *Struct. Optim.*, vol. 4, no. 3–4, pp. 250–252, Sep. 1992, doi: 10.1007/bf01742754.
- [23] M. P. Bendsøe, "Optimal shape design as a material distribution problem," *Struct. Optim.*, vol. 1, no. 4, pp. 193–202, Dec. 1989, doi: 10.1007/BF01650949.
- [24] M. P. Bendsøe, *Optimization of Structural Topology, Shape, and Material*. Springer-Verlag

Berlin Heidelberg, 1995.

- [25] K. Suzuki and N. Kikuchi, "A homogenization method for shape and topology optimization," *Comput. Methods Appl. Mech. Eng.*, vol. 93, no. 3, pp. 291–318, 1991, doi: 10.1016/0045-7825(91)90245-2.
- [26] M. P. Bendsøe and N. Kikuchi, "Generating optimal topologies in structural design using a homogenization method," *Comput. Methods Appl. Mech. Eng.*, vol. 71, no. 2, pp. 197–224, 1988, doi: 10.1016/0045-7825(88)90086-2.
- [27] G. I. N. Rozvany and M. Zhou, "The COC algorithm, part I: Cross-section optimization or sizing," *Comput. Methods Appl. Mech. Eng.*, vol. 89, no. 1–3, pp. 281–308, 1991, doi: 10.1016/0045-7825(91)90045-8.
- [28] M. Zhou and G. I. N. Rozvany, "The COC algorithm, Part II: Topological, geometrical and generalized shape optimization," *Comput. Methods Appl. Mech. Eng.*, vol. 89, no. 1–3, pp. 309–336, 1991, doi: 10.1016/0045-7825(91)90046-9.
- [29] S. Osher and J. A. Sethian, "Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations," *J. Comput. Phys.*, vol. 79, no. 1, pp. 12–49, 1988, doi: 10.1016/0021-9991(88)90002-2.
- [30] W. F. Noh and P. Woodward, "SLIC (Simple Line Interface Calculation)," 1976, pp. 330–340.
- [31] H. M. Wu, E. A. Overman, and N. J. Zabusky, "Steady-state solutions of the euler equations in two dimensions: Rotating and translating V-states with limiting cases. I. Numerical algorithms and results," *J. Comput. Phys.*, vol. 53, no. 1, pp. 42–71, 1984, doi:

10.1016/0021-9991(84)90051-2.

- [32] J. A. Sethian, "Evolution, Implementation, and Application of Level Set and Fast Marching Methods for Advancing Fronts," *J. Comput. Phys.*, vol. 169, no. 2, pp. 503–555, May 2001, doi: 10.1006/jcph.2000.6657.
- [33] M. Y. Wang, X. Wang, and D. Guo, "A level set method for structural topology optimization," *Comput. Methods Appl. Mech. Eng.*, vol. 192, no. 1–2, pp. 227–246, Jan. 2003, doi: 10.1016/S0045-7825(02)00559-5.
- [34] N. P. Van Dijk, K. Maute, M. Langelaar, and F. Van Keulen, "Level-set methods for structural topology optimization : a review," *Struct. Multidiscip. Optim.*, vol. 48, no. 3, pp. 437–472, 2013, doi: 10.1007/s00158-013-0912-y.
- [35] R. B. Haber and M. P. Bendsøe, "Problem formulation, solution procedures and geometric modeling: Key issues in variable-topology optimization," in *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 1998, doi: 10.2514/6.1998-4948.
- [36] J. A. Sethian and A. Wiegmann, "Structural Boundary Design via Level Set and Immersed Interface Methods," *J. Comput. Phys.*, 2000, doi: 10.1006/jcph.2000.6581.
- [37] S. J. Osher and F. Santosa, "Level Set Methods for Optimization Problems Involving Geometry and Constraints," *J. Comput. Phys.*, 2001, doi: 10.1006/jcph.2001.6789.
- [38] G. Allaire, F. Jouve, and A. M. Toader, "Structural optimization using sensitivity analysis and a level-set method," *J. Comput. Phys.*, vol. 194, no. 1, pp. 363–393, Feb. 2004, doi: 10.1016/j.jcp.2003.09.032.

- [39] S. Osher and R. P. Fedkiw, "Level Set Methods: An Overview and Some Recent Results," *J. Comput. Phys.*, vol. 169, no. 2, pp. 463–502, May 2001, doi: 10.1006/jcph.2000.6636.
- [40] G. Allaire, F. De Gournay, F. Jouve, and A. M. Toader, "Structural optimization using topological and shape sensitivity via a level set method," *Control Cybern.*, vol. 34, no. 1, pp. 59–81, 2005.
- [41] L. Fernandez, "Topology Optimization Using a Level Set Penalization With Constrained Topology Features," Graduate School of Clemson University, 2013.
- [42] T. Yamada, K. Izui, S. Nishiwaki, and A. Takezawa, "A topology optimization method based on the level set method incorporating a fictitious interface energy," *Comput. Methods Appl. Mech. Eng.*, vol. 199, no. 45–48, pp. 2876–2891, Nov. 2010, doi: 10.1016/j.cma.2010.05.013.
- [43] P. Wei, Z. Li, X. Li, and M. Y. Wang, "An 88-line MATLAB code for the parameterized level set method based topology optimization using radial basis functions," *Springer*, vol. 58, no. 2, pp. 831–849, 2018.
- [44] S. Wang and M. Y. Wang, "Radial basis functions and level set method for structural topology optimization," *Int. J. Numer. Methods Eng.*, vol. 65, no. 12, pp. 2060–2090, Mar. 2006, doi: 10.1002/nme.1536.
- [45] A. A. Gomes and A. Suleman, "Application of spectral level set methodology in topology optimization," *Struct. Multidiscip. Optim.*, vol. 31, no. 6, pp. 430–443, Jun. 2006, doi: 10.1007/s00158-006-0005-2.
- [46] L. Van Miegroet and P. Duysinx, "Stress concentration minimization of 2D filets using X-

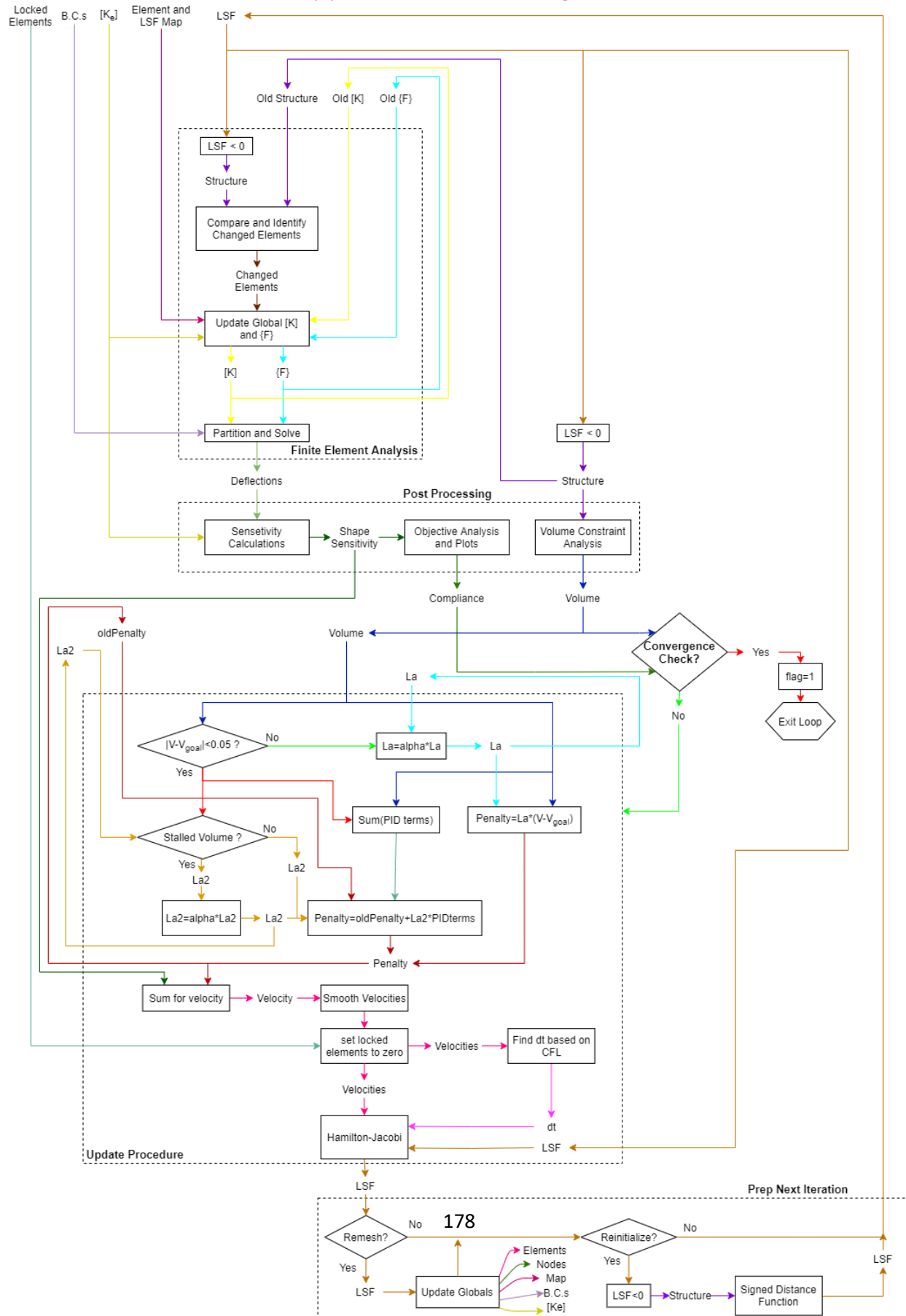
- FEM and level set description,” *Struct. Multidiscip. Optim.*, 2007, doi: 10.1007/s00158-006-0091-1.
- [47] Q. Xia, M. Y. Wang, and T. Shi, “A level set method for shape and topology optimization of both structure and support of continuum structures,” *Comput. Methods Appl. Mech. Eng.*, vol. 272, pp. 340–353, Apr. 2014, doi: 10.1016/j.cma.2014.01.014.
- [48] S. H. Ha and S. Cho, “Level set based topological shape optimization of geometrically nonlinear structures using unstructured mesh,” *Comput. Struct.*, vol. 86, no. 13–14, pp. 1447–1455, Jul. 2008, doi: 10.1016/j.compstruc.2007.05.025.
- [49] T. P. Fries and T. Belytschko, “The extended/generalized finite element method: An overview of the method and its applications,” *Int. J. Numer. Methods Eng.*, vol. 84, no. 3, pp. 253–304, Oct. 2010, doi: 10.1002/nme.2904.
- [50] Q. Xia, M. Wang, and T. Shi, “Topology optimization with pressure load through a level set method,” *Comput. Methods Appl. Mech. Engrg.*, vol. 283, pp. 177–195, 2015, doi: 10.1016/j.cma.2014.09.022.
- [51] N. P. Van Dijk, M. Langelaar, and F. Van Keulen, “Explicit level-set-based topology optimization using an exact Heaviside function and consistent sensitivity analysis,” *Int. J. Numer. Methods Eng.*, vol. 91, no. 1, pp. 67–97, Jul. 2012, doi: 10.1002/nme.4258.
- [52] E. Lee and J. R. R. A. Martins, “Structural topology optimization with design-dependent pressure loads,” *Comput. Methods Appl. Mech. Eng.*, 2012, doi: 10.1016/j.cma.2012.04.007.
- [53] C. Wang, M. Zhao, and T. Ge, “Structural topology optimization with design-dependent

pressure loads,” *Struct. Multidiscip. Optim.*, vol. 53, no. 5, pp. 1005–1018, 2016, doi: 10.1007/s00158-015-1376-z.

- [54] X. Huang and Y. M. Xie, “Evolutionary topology optimization of continuum structures including design-dependent self-weight loads,” *Finite Elem. Anal. Des.*, vol. 47, no. 8, pp. 942–948, 2011, doi: 10.1016/j.finel.2011.03.008.
- [55] H. Zhang, S. T. Liu, and X. Zhang, “Topology optimization of 3D structures with design-dependent loads,” *Acta Mech. Sin. Xuebao*, vol. 26, no. 5, pp. 767–775, Oct. 2010, doi: 10.1007/s10409-010-0370-3.

APPENDICES

Appendix A: Flow Diagram



Appendix B: Mesh Generation Code

Variable Name	Variable Size	Description
voxelsize	1x3	length of meshed elements in x, y and z directions
faces	(#triangles)x3	Outward normal direction components of STL faces
vertices	3x3x(#triangles)	(each node of the triangle)x(x, y, z coordinate of the node)x(each STL triangle)
ranges	2x3	minimum STL node coordinates over the maximum
x_centroids	1x(c)	x-coordinates of the centroid for each cell
y_centroids	1x(c)	y-coordinates of the centroid for each cell
z_centroids	1x(c)	z-coordinates of the centroid for each cell
xyfaces	(r)x1	List of STL faces that have a z component
X	(r)x(c)	X values from a meshgrid of x and y centroids
Y	(r)x(c)	X values from a meshgrid of x and y centroids
numxyzf	1x4	[number of cell elements in x, y, and z then the number of xy-faces]
cells	(r)x(c)x(p)	Logic representation of if a cell has material or not (-1=void, 1=solid)
p	3x2	x and y coordinates of the 3 nodes that make up the given triangle
intersects	(r)x(c)	Logic value of which x and y centroids are within the given triangle
A	scalar	Area of given triangle
up	scalar	1,2,3 index value for cross product to make shape function
down	scalar	1,2,3 index value for cross product to make shape function
Plane	2x2	Coefficient representation of shape function for triangle
I	(r)x1	Indices of x and y centroids that are within the given triangle
z_int	(r)x1	Z values of where the STL triangle intersects each x, y centroid
mult	1x(c)	Numbers to multiply along z to change 'cells'
outer	1x(c)	Indices of cell that are not completely surrounded by elements
Boundary	1x(c)	Elements that are part of the border

Variable Name	Variable Size	Description
nelx	scalar	Number of elements in x
nely	scalar	Number of elements in y
nelz	scalar	Number of elements in z
Elements	(r)x(c)	List of element numbers
nlz	(r)x1	List of node one z-coordinates for each element
nlx	(r)x1	List of node one x-coordinates for each element
nly	(r)x1	List of node one y-coordinates for each element
Relative	8x1	Relative node numberings from node 1 for each element
Nodes	(r)x3	List of all possible node coordinates
ind	(r)x1	Indices for repeated nodes
Nodes	(r)x3	List of all possible node coordinates
elements	(r)x8	Final element list for mesh
nodes	(r)x3	Final node list for mesh
N	scalar	Next node number index

Make_Mesh.m

```

1  close all
2  clear all
3  clc
4
5  voxelsize=[0.5,0.5,0.5];      %element voxelsize in the x, y, and z direction
6
7  if(numel(voxelsize)==1)
8      voxelsize(1:3)= voxelsize;
9  end
10
11
12  [faces,vertices] = readSTL('Irregular Pressure Vessel.STL','inches');
13
14  ranges=[min(min(vertices),[],3);max(max(vertices),[],3)];
15  x_centroids=ranges(1,1)+0.5*voxelsize(1): voxelsize(1):ranges(2,1);
16  y_centroids=ranges(1,2)+0.5*voxelsize(2): voxelsize(2):ranges(2,2);
17  z_centroids=ranges(1,3)+0.5*voxelsize(3): voxelsize(3):ranges(2,3);
18  xyfaces=find(faces(:,3)~=0);
19  disp('STL file read')
20
21  [X,Y]=meshgrid(x_centroids,y_centroids);
22  numxyzf=[flip1r(size(X)),numel(z_centroids),numel(xyfaces)];
23  cells=-1*ones(numxyzf([2,1,3]));
24  for(f=1:numxyzf(4))
25      p=vertices(:,1:2,xyfaces(f));

```

```

26 intersects=inpolygon(X,Y,p(:,1),p(:,2));
27 A=0.5*det([1;1;1],p);
28 Plane=zeros(2);
29 for(i=1:3)
30     up=rem(i,3)+1;
31     down=3-rem(4-i,3);
32     Plane=Plane+0.5*vertices(i,3,xyfaces(f))*[0,p(down,1)-p(up,1);p(up,2)-
33 p(down,2),p(up,1)*p(down,2)-p(down,1)*p(up,2)]/A;
34 end
35 I=find(intersects);
36 if(~isempty(I))
37     z_int=poly2Deval(Plane,[X(I),Y(I)]);
38     mult=-1*(z_centroids>=z_int)+(z_centroids<z_int);
39     %[z_int,mult]
40     t=0;
41     for(i=1:numel(I))
42         %mult=-1*(z_centroids>=z_int(i))+(z_centroids<z_int(i))
43         [r,c]=ind2sub([numxyzf(2),numxyzf(3)],I(i));
44         cells(r,c,:)=cells(r,c,:).*permute(mult(i,:),[1,3,2]);
45         %cells(r,c,:)=cells(r,c,:).*permute(mult,[1,3,2]);
46     end
47 end
48 if(~mod(f,250))
49     fprintf('evaluated %d of %d faces \n',f,numxyzf(4));
50 end
51 end
52
53
54
55 disp('generating mesh')
56 cells=permute(cells,[2,1,3]);
57 outer=(cells==1).*(convn(cells,ones(3,3,3),'same')<27);
58 outer(cells(:)==-1)=[];
59 Boundary=nonzeros(outer(:)).*(1:nz(cells==1));
60
61
62 nelx=numxyzf(1);    nely=numxyzf(2);    nelz=numxyzf(3);
63 Elements=1:nelx*nely*nelz;
64 nlz=floor((Elements-1)/(nelx*nely));
65 nlx=rem((Elements-(nelx*nely)).*floor((Elements-1)/(nelx*nely))-1,nelx);
66 nly=floor((Elements-(nelx*nely)).*floor((Elements-1)/(nelx*nely))-1)/nelx;
67 Relative=[0;1;nelx+2;nelx+1;...
68 (nelx+1)*(nely+1);(nelx+1)*(nely+1)+1;(nelx+1)*(nely+1)+nelx+2;(nelx+1)*(nely+1)+nelx+1];
69 Elements=(1+nlx+nly*(nelx+1)+nlz*(nelx+1)*(nely+1))'+Relative';
70 [Nodes(:,1),Nodes(:,2),Nodes(:,3)]=ind2sub([nelx+1,nely+1,nelz+1],1:(nelx+1)*(nely+1)*(nelz+1));
71 Nodes=voxelsize.*(Nodes-[1,1,1]);
72
73
74 Elements=(cells(:)==1).*Elements;    %find on elements
75 Elements((Elements(:,1)==0),:)=[];    %remove off elements
76

```

```

77 elements=zeros(size(Elements));
78 nodes=zeros(size(Nodes));
79 N=1;
80 while(sum(elements(:)==0)>0)
81     [c,r]=find(elements'==0,1);
82     ind=find(Elements==Elements(r,c));
83     elements(ind)=N;
84     nodes(N,:)=Nodes(Elements(r,c),:);
85     N=N+1;
86     if(~mod(N,1000))
87         fprintf('meshing node %d of %d \n',sum(elements(:)~=0),numel(Elements));
88     end
89 end
90 nodes(N:end,:)=[];
91
92 disp('plotting')
93 meshplot(elements, nodes, Boundary);    axis equal
94 xlabel('x axis')
95 ylabel('y axis')
96 zlabel('z axis')
97 %meshplotlayer(elements, nodes, Boundary);
98 disp('done')

```

[Published with MATLAB® R2018a](#)

Appendix C: Main Code

Variable Name	Variable Size	Description
E	scalar	Modulus of Elasticity 29.5×10^6 PSI
nu	scalar	Poisson's Ratio 0.29
Yield	scalar	Yield Strength 150×10^3 PSI
Pressure	scalar	Pressure in PSI applied to the interior set to 5000
elements	(numelem)x8	Row for each element and a column for each of the element's node numbers
nodes	(numnodes)x3	[x,y,z] coordinates for each node
boundary	(r)x1	List of elements that are on the boundary of the geometry
Title	character	Folder name that all of the data will be saved to
volReq	scalar	Volumetric constraint goal, between 0 and 1
stepLength	scalar	Number of CFL time steps the evolution equation is solved every iteration
numReinit	scalar	Frequency the LSF is reinitialized
topWeight	scalar	Weighting factor for topological derivative's influence in Hamilton-Jacobi equation, set to 0 for pressure vessels
max_itr	scalar	The maximum number of iterations the code will run before it forces it to exit the loop
LSFspacing	scalar	Distance between LSF kernels
init	3x3	Defines the initial void geometry. 1st row defines edge lengths of voids, 2nd row defines x,y,z gap between voids and 3rd row the number of voids in the x,y,and z directions
La	scalar	Lagrange multiplier for the first portion of the optimization, initialized to 0.5
La2	scalar	Lagrange multiplier for the second portion of the optimization, initialized to 1
alpha	scalar	Multiplication factor for the Lagrange multipliers, set to 1/0.95
PID	1x3	Scaling factors for each of the PID terms, set to [0.5,0.2,1]
relax	scalar	State of which type of penalty should be executed, 0 for proportional only, 1 for PID-type formulation
i	scalar	Iteration counter
flag	scalar	Loop termination state, 0 to continue optimization, anything else to stop

Variable Name	Variable Size	Description
mesh	scalar	Iteration counter for remeshing, set to 0 when remeshing should occur
band	scalar	Remeshing tolerance
s	Scalar	Percentage of CFL condition time step used
Domain	2x3	1st row is the minimum coordinates in x,y, and z of the geometry and the second row is the maximum
struc	(NSx)x(NSy)x(NSz)	Material distribution, 0 for void 1 for material
Esize	1x3	x, y, and z edge lengths of each element
map	(numelem)x1	Index positions of each element in the LSF
noF	1x(c)	Degrees of freedom on homogeneous boundary for FEA
exterior	(r)x1	List of indices of 'struc' that are exterior to the design domain
sX, sY, & sZ	(NSx)x(NSy)x(NSz)	Meshgrid of coordinates of the elements' centroids
numnodes	scalar	Number of nodes in the FEA analysis
numelem	scalar	Number of elements in the FEA analysis
CompE	(numelem)x1	2*Compliance of each element
volTot	Scalar	Total initial volume
LSFsize	1x3	Number of elements in the LSF in the x, y, and z directions
cent	1x3	Center coordinates of the domain
lsfX, lsfY, & lsfZ	(NLx)x(NLy)x(NLz)	Meshgrid of coordinates of the LSF kernels
sdf	(NSx)x(NSy)x(NSz)	Signed distance function of the structure
lsf	(NLx)x(NLy)x(NLz)	Level-Set Function values
Nanind	(r)x1	List of LSF kernels outside of 'struc'
LSF2EleDist	(numLSF)x(numelem)	Distance from each LSF kernel to each centroid of 'struc'
id	(r)x1	List of 'struc' indices that are closest to 'Nanind' of the LSF
d	(r)x1	List of Euclidean distances corresponding to 'id'
R	Scalar	Maximum filter distance for equation 4.3
Hij	(numLSF)x(numelem)	Weighting factors of filter for each LSF kernel and each element
inside	1x(c)	List of LSF indices inside of the design domain
bearing	(r)x1	List of indices of the LSF that are to remain constant and not change (regions outside the domain and the boundary elements)
Po	1x3	Outward normal force each void element's nodes experience

Variable Name	Variable Size	Description
dof	(numelem)x24	Degrees of freedom for each element of the mesh
ke	24x24	Elemental stiffness matrix
B	6x24	B-matrix in elemental stiffness matrix calculation, used for stress calculations
C	6x6	Constitutive relation for the material, used for stress calculations
oldstruc	(NSx)x(NSy)x(NSz)	Previous iteration's 'struc' matrix used to compare for changes in the FEA global matrices
oldK	(#dofs)x(#dofs)	Previous iteration's global stiffness matrix
oldF	(#dofs)x1	Previous iteration's global force vector
fix	4*3	Node coordinates of the pinned node then each of the roller conditioned nodes in x, y, and z respectively
Nborder	(#border elements*8)*3	Node coordinates of each node of a border element
D_best	scalar	Current maximum sum of squares for 'dx', 'dy', and 'dz'
N1	1x3	Node coordinates of pinned node
Nx	(r)x3	Node coordinates of all border nodes along the x-direction of the pinned node
Ny	(r)x3	Node coordinates of all border nodes along the y-direction of the pinned node
Nz	(r)x3	Node coordinates of all border nodes along the z-direction of the pinned node
dx	scalar	Maximum distance from pinned node to any 'Nx' node
ix	scalar	Index of 'Nx' that the maximum distanced node is
dy	scalar	Maximum distance from pinned node to any 'Ny' node
iy	scalar	Index of 'Ny' that the maximum distanced node is
dz	scalar	Maximum distance from pinned node to any 'Nz' node
iz	scalar	Index of 'Nz' that the maximum distanced node is
tf	scalar	Number of node coordinates that match 'fix' nodes, should always equal 4
ind (line 105)	(r)x1	Node numbers for 'fix' nodes
fixeddofs	9x1	List of fixed degrees of freedom
folder	character	Folder name that all of the data will be saved to, title with timestamp appended to it
U	(#dofs)x1	Deflection values for each degree of freedom
e	scalar	Loop counter for each element

Variable Name	Variable Size	Description
obj	1x(i)	Vector of all iteration's compliance
vol	1x(i)	Vector of all iteration's volume fraction
shapeSens	(NLx)x(NLy)x(NLz)	Shape sensitivity calculated from elemental strain energy densities
SensTotal	(NLx)x(NLy)x(NLz)	Shape sensitivity plus penalty
Con	1x(c)	Initial control values to test
V	(r)x1	Resulting volume fractions after control values
newlsf	(NLx)x(NLy)x(NLz)	Level-Set Function values after update
newstruc	(NSx)x(NSy)x(NSz)	'struc' values after update
add	(r)x1	List of 'struc' indices that need to be added to the mesh
newmap	(numelem)x1	Index positions of each element in the LSF after update
Control	1x(i)	Vector of all of the previous control terms
Penalty	scalar	The volume constraint penalty added to the shape sensitivity to create velocities
ind (line243)	4x1	Nodes that are closed the initial fixed nodes
mult	8x3	Matrix to multiply to 'Esize' to get a master element
Nfull	8x3x(p)	Nodes of elements to be added
Nall	(r)x3	Nodes of elements to be added
Nnum	(r)x1	Node numbers '#' if already exist '0' if a new node is needed
L2ED	(numLSF)x(numelem)	Distance from each LSF kernel to each centroid of 'struc'

Main Code

```

1  close all
2  clear all
3  clc
4  addpath([pwd, '\IrregularShapeSubfunctions'])
5  addpath([pwd, '\MakeMeshSubfunctions'])
6  %Attempt to implement level-set topology optimization on a 3D structure with a
7  %pressure load being applied from a void in the center
8  %Calculates forces as outward normal for all void elements
9  disp('running...')
10 Title='RemeshIPVo10.45';
11 %Material Parameters and Working Pressure-----
12 %Inconel718
13 E=29.5*10^6;    %psi

```



```

14 nu=0.29;
15 Yield=150*10^3; %psi
16 Pressure=3000; %PSI
17 %-----
18
19 %Geometry and Loading-----
20 load('RotatedIPVmesh25.mat') %imports 'elements' 'nodes' and 'boundary' from saved mesh file
21 load('RemeshStart25.mat')%load values of oldStruct,OldK,OldF to compare to
22 %-----
23
24 % Establish Level-Set parameters-----
25 volReq=0.25;
26 stepLength=2;
27 numReinit=3;
28 topweight=0;
29 max_itr=200;
30 LSFspacing=0.375;
31 init=[3,3,3;2,2,2;10,10,10]; %edge length of initial void; gap between; repeated
32 maxNodes=75000;
33 La=1/2; La2=1; alpha=1/0.95;
34 PID=[0.5,0.2,1]; relax=0;
35 %-----
36
37 %Initialization-----
38 i=1; flag=0; mesh=1; band=0.15; s=1;
39 %Initialize Struc-----
40 Domain=[min(nodes);max(nodes)];
41 [struc,Esize,map,noF,exterior]=InitialStruc(elements,nodes,boundary,init); %map is a list for
42 each element, which struc index is used
43 [sX,sY,sZ]=meshgrid(Esize(1)/2:Esize(1):Domain(2,1),...
44 Esize(2)/2:Esize(2):Domain(2,2),Esize(3)/2:Esize(3):Domain(2,3));
45 sX=permute(sX,[2,1,3]); sY=permute(sY,[2,1,3]); sZ=permute(sZ,[2,1,3]);
46 numElem=size(elements,1); numNodes=size(nodes,1);
47 CompE=zeros(numElem,1);
48 volTot=prod(Esize)*numElem;
49 %Initialize LSF-----
50 LSFsize=ceil(Domain(2,:)/LSFspacing)+1;
51 cent=mean(Domain);
52 lsfx=LSFspacing*(LSFsize(1)-1)*linspace(-0.5,0.5,LSFsize(1))+cent(1);
53 lsfy=LSFspacing*(LSFsize(2)-1)*linspace(-0.5,0.5,LSFsize(2))+cent(2);
54 lsfz=LSFspacing*(LSFsize(3)-1)*linspace(-0.5,0.5,LSFsize(3))+cent(3);
55 [lsfx,lsfy,lsfz]=meshgrid(lsfX,lsfY,lsfZ);
56 lsfx=permute(lsfX,[2,1,3]); lsfy=permute(lsfY,[2,1,3]); lsfz=permute(lsfZ,[2,1,3]);
57 sdf=(~struc).*bwdist(struc)-struc.*bwdist(struc-1); %reinitialize LSF
58 lsf=griddata(sX,sY,sZ,double(sdf),lsfx,lsfy,lsfz);
59 Nanind=find(isnan(lsf));
60 LSF2EleDist=(nodes(elements(:,1),1)+Esize(1)/2'-lsfx(:)).^2+...
61 (nodes(elements(:,1),2)+Esize(2)/2'-lsfy(:)).^2+...
62 (nodes(elements(:,1),3)+Esize(3)/2'-lsfz(:)).^2;
63 %LSF2StrucDist=(sX(:)'-lsfx(:)).^2+(sY(:)'-lsfy(:)).^2+(sZ(:)'-lsfz(:)).^2;
64 [d,id]=min(LSF2EleDist(Nanind,:),[],2);

```

```

65  lsf(Nanind)=sdf(map(id))-d./Esize(1);
66  struc=griddata(lsfX,lsfY,lsfZ,lsf,sX,sY,sZ)<=0;
67  %Filter and Update Prep-----
68  R=1.25*LSFspacing;
69  Hij=max(R-LSF2EleDist,0);
70
71  inside=setdiff(1:numelem,boundary);
72  bearing=find(sum(lsfX(:)'>=nodes(elements(inside,1),1) &...
73      lsfY(:)'>=nodes(elements(inside,1),2) &...
74      lsfZ(:)'>=nodes(elements(inside,1),3) &...
75      lsfX(:)'<=nodes(elements(inside,7),1) &...
76      lsfY(:)'<=nodes(elements(inside,7),2) &...
77      lsfZ(:)'<=nodes(elements(inside,7),3))==0);
78  %-----
79
80  %Loading and Boundary Conditions-----
81  Po=circshift(Esize,1).*circshift(Esize,-1)*Pressure/4;
82  dof=3*repelem(elements,1,3)-repmat([2,1,0],1,8);
83  [ke,B,C]=stiff3D(E,nu,Esize);
84  if(~isequal(struc,oldstruc))
85      oldstruc=[]; oldk=[]; oldF=[];
86  end
87  if(~exist('fix'))
88      Nborder=nodes(elements(boundary,:),:);
89      D_best=0;
90      for(b=1:size(Nborder,1))    %finds border points that would be best for coordinate oriented
91  B.C.s
92          N1=Nborder(b,:);
93          Nx=Nborder(Nborder(:,2)==N1(2) & Nborder(:,3)==N1(3),:);
94          Ny=Nborder(Nborder(:,1)==N1(1) & Nborder(:,3)==N1(3),:);
95          Nz=Nborder(Nborder(:,1)==N1(1) & Nborder(:,2)==N1(2),:);
96          [dx,ix]=max(abs(sum(N1-Nx,2)));
97          [dy,iy]=max(abs(sum(N1-Ny,2)));
98          [dz,iz]=max(abs(sum(N1-Nz,2)));
99          if(dx^2+dy^2+dz^2>D_best)
100              D_best=dx^2+dy^2+dz^2;
101              fix=[N1;Nx(ix,:);Ny(iy,:);Nz(iz,:)];
102          end
103      end
104  end
105  [tf,ind]=ismember(fix,nodes,'rows');
106  if(sum(tf)~=4)
107      disp('constraint error')
108  end
109  fixeddofs=nonzeros(reshape((3*ind-[2,1,0]).*[1,1,1;~eye(3)],[],1));
110  %-----
111
112  %Save Initial-----
113  folder=strcat(Title,strrep(datestr(datetime),':',''));
114  mkdir(folder);
115  save([pwd,'\ ',folder,'\ ', 'Iteration0'], 'lsf', 'struc', 'La', 'alpha', ...

```

```

116 'init','volReq','ke','bearing','elements','nodes','boundary','map',...
117 'max_itr','numReinit','Po','stepLength','PID','volTot','lsfX','lsfY','lsfZ')
118 clear LSF2EleDist;
119 disp(['Starting ',Title])
120 %-----
121
122 while(flag==0)
123     [U,oldk,oldF]=FEA_3DP6(struc,elements,map,ke,Po,noF,fixeddofs,oldstruc,oldk,oldF);
124     %evaluate sensitivities of each element-----
125     for(e=1: numel)
126         CompE(e)=-max(struc(map(e)),0.0001)*U(dof(e,:))'*ke*U(dof(e,:));
127     end
128
129     %Post Processing and Plotting-----
130     obj(i)=-sum(CompE(:));
131     vol(i)=prod(Esize)*sum(struc(map))/volTot;
132     disp(['It.: ' num2str(i) ' Compl.: ' sprintf('%10.4f',obj(i)) ' vol.: '
133     sprintf('%6.3f',vol(i))...
134     ' La: ' sprintf('%10.3f',La) ' LaPID: ' sprintf('%10.5f',La2)])
135
136     %check for convergence-----
137     if(i>5)
138         if((abs(vol(i)-volReq)<0.005) && all(abs(obj(end)-obj(end-5:end-1))<0.03*abs(obj(end))))
139             flag=1;
140         end
141         if(i>=max_itr)
142             flag=2;
143         end
144     end
145
146     %Update Procedure-----
147     if(relax==0 && abs(vol(i)-volReq)<=0.035) % (max(abs(vol(i-4:i)-
148     volReq))<0.05 && relax==0) % (max(abs(vol(i-4:i)-
149         relax=1; %Stop relaxed penalty if within volume band (0.15)
150         s=0.3;
151         shapeSens=reshape((Hij*CompE)./max(sum(Hij,2),0.0001),LSFsize);
152         SensTotal=(shapeSens/max(abs(shapeSens(:)))));
153         Con=-0.75:0.005:0.75;
154         V=zeros(numel(Con),1);
155         for(c=1: numel(Con))
156             [newlsf]=updatestep3(lsf,SensTotal+Con(c),stepLength,bearing,Esize(1));
157             newstruc=(griddata(lsfX,lsfY,lsfZ,newlsf,sx,sY,sZ)<=0);
158             newstruc(map(boundary))=1; newstruc(exterior)=1;
159             add=setdiff(find((newstruc-struc)==1),[map;exterior]);
160             newmap=[map;add];
161             V(c)=prod(Esize)*(sum(newstruc(newmap)))/volTot;
162         end
163         Control(i-1)=Con(find(V>=vol(i),1,'last'));
164     end
165     if(relax==0) %Execute relaxed penalty
166         La=alpha*La;

```

```

167     Penalty=La*(vol(i)-volReq);
168 %     Control=[];
169 %     Control(i)=Penalty;
170 else
171     if(max(vol(max(1,i-5):i))-min(vol(max(1,i-5):i))<0.005 && i>5)
172         La2=(alpha^2)*La2; %Update Lagrange multiplier on PID if volume hasn't changed
173     end
174     Control(i)=La2*PID*[(vol(i)-volReq);...
175         ((sum(vol(max(1,i-4):i))/numel(max(1,i-4):i))-volReq);...
176         (2*vol(i)-vol(max(1,i-1))-volReq)];
177     Penalty=sum(Control);
178 end
179 shapeSens=reshape((Hij*CompE)./max(sum(Hij,2),0.0001),LSFsize);
180 SensTotal=(shapeSens/max(abs(shapeSens(:))))+Penalty;
181
182 %Save values every iteration-----
183 save([pwd,'\',folder,'\','Iteration',num2str(i)],'lsf','struc','U',...
184     'La','La2','shapeSens','SensTotal','Penalty','oldstruc',...
185     'oldk','oldF','nodes','elements','map','CompE','exterior','boundary')
186 %-----
187 oldlsf=lsf;
188 [lsf]=updatestep3(lsf,SensTotal,stepLength,bearing,s*Esize(1));
189 oldstruc=struc;
190 struc=griddata(lsfX,lsfY,lsfZ,lsf,sx,sY,sZ)<=0);
191 struc(map(boundary))=1; struc(exterior)=1;
192 %-----
193
194 add=setdiff(find((struc-oldstruc)==1),[map;exterior]);
195 newmap=[map;add];
196 if((prod(Esize)*(sum(struc(newmap)))/volTot)<(volReq-0.04))
197     disp('Stepped Back')
198     lsf=oldlsf;
199     struc=griddata(lsfX,lsfY,lsfZ,lsf,sx,sY,sZ)<=0);
200     struc(map(boundary))=1; struc(exterior)=1;
201 end
202
203 %Prep next iteration-----
204 if(mesh>=5)
205     if(mesh>=8 && max(abs(vol(i-4:i)-volReq))<band && Esize(1)>(3/32))
206         mesh=0; disp('option1');
207         band=0.8*band;
208         Esize=max(0.75*Esize,1/8);
209     elseif(sum(struc(map))==0)>0.5*numelem || numnodes>150000)
210         mesh=0; disp('option2');
211         band=0.15;
212         Esize=repelem((prod(Esize)*(sum(struc(map))+...
213             numel(setdiff(find((struc-
214 oldstruc)==1),[map;exterior])))/(1.2*numelem))^(1/3),3);
215     end
216 end
217

```

```

218
219     if(mesh==0) %Remesh
220         fprintf('remeshing with element size: %2.5f\n',Esize(1));
221         [struc,elements,nodes,map,boundary,noF,sX,sY,sZ,exterior]=remesh(lsf,...
222             [lsfX(:),lsfY(:),lsfZ(:)],Esize,'Rotated Irregular Pressure Vessel.STL');
223         numelem=size(elements,1);    numnodes=size(nodes,1);
224         fprintf('meshing complete with %d elements and %d nodes\n',numelem,numnodes);
225         while(numnodes>160000) %retry if too many nodes
226             Esize=Esize*1.05;
227             fprintf('remeshing with element size: %2.5f\n',Esize(1));
228             [struc,elements,nodes,map,boundary,noF,sX,sY,sZ,exterior]=...
229                 remesh(lsf,[lsfX(:),lsfY(:),lsfZ(:)],Esize,'Rotated Irregular Pressure
230 vessel.STL');
231             numelem=size(elements,1);    numnodes=size(nodes,1);
232             fprintf('meshing complete with %d elements and %d nodes\n',numelem,numnodes);
233         end
234         LSF2EledDist=(nodes(elements(:,1),1)+Esize(1)/2'-lsfX(:)).^2+...
235             (nodes(elements(:,1),2)+Esize(2)/2'-lsfY(:)).^2+...
236             (nodes(elements(:,1),3)+Esize(3)/2'-lsfZ(:)).^2;
237         [d,id]=min(LSF2EledDist(Nanind,:),[],2);
238         Hij=max(R-LSF2EledDist,0);
239         Po=circshift(Esize,1).*circshift(Esize,-1)*Pressure/4;
240         dof=3*repelem(elements,1,3)-repmat([2,1,0],1,8);
241         [ke,B,C]=stiff3D(E,nu,Esize);
242         oldstruc=[]; oldK=[]; oldF=[];
243         [~,ind]=min((fix(:,1)-nodes(:,1')).^2+(fix(:,2)-nodes(:,2')).^2+(fix(:,3)-
244 nodes(:,3')).^2,[],2);
245         fixeddofs=nonzeros(reshape((3*ind-[2,1,0]).*[1,1,1;~eye(3)],[],1));
246         clear LSF2EledDist;
247         mesh=1;
248         elseif((prod(Esize)*sum(struc(map))/volTot)>0.98) %If entire domain becomes solid revert back
249 to initial configuration
250             disp('Domain solid reverting to original discretization')
251             La=(alpha)^5*La; %Take a large step in La
252             load('RotatedIPVmesh25.mat')
253             load('RemeshStart25.mat')
254             [struc,Esize,map,noF]=InitialStruc(elements,nodes,boundary,init);
255             sdf=(~struc).*bwdist(struc)-struc.*bwdist(struc-1); %reinitialize LSF
256             lsf=griddata(sX,sY,sZ,double(sdf),lsfX,lsfY,lsfZ);
257             Nanind=find(isnan(lsf));
258             LSF2EledDist=(nodes(elements(:,1),1)+Esize(1)/2'-lsfX(:)).^2+...
259                 (nodes(elements(:,1),2)+Esize(2)/2'-lsfY(:)).^2+...
260                 (nodes(elements(:,1),3)+Esize(3)/2'-lsfZ(:)).^2;
261             [d,id]=min(LSF2EledDist(Nanind,:),[],2);
262             lsf(Nanind)=sdf(map(id))-d./Esize(1);
263             struc=griddata(lsfX,lsfY,lsfZ,lsf,sX,sY,sZ)<=0;
264             Hij=max(R-LSF2EledDist,0);
265             Po=circshift(Esize,1).*circshift(Esize,-1)*Pressure/4;
266             dof=3*repelem(elements,1,3)-repmat([2,1,0],1,8);
267             [ke,B,C]=stiff3D(E,nu,Esize);
268             if(~isequal(struc,oldstruc))

```

```

269         oldstruc=[]; oldk=[]; oldF=[];
270     end
271     clear LSF2EleDist;
272     mesh=1;
273 else
274     if(~mod(i,numReinit)) %reinitialize LSF
275         sdf=(~struc).*bwdist(struc)-struc.*bwdist(struc-1); %reinitialize LSF
276         lsf=griddata(sX,sY,sZ,double(sdf),lsfX,lsfY,lsfZ);
277         lsf(Nanind)=sdf(map(id))-d./Esize(1);
278         struc=griddata(lsfX,lsfY,lsfZ,lsf,sX,sY,sZ)<=0;
279         struc(map(boundary))=1; struc(exterior)=1;
280         clear sdf
281     end
282     mesh=mesh+1;
283     %Add elements if needed
284     add=setdiff(find((struc-oldstruc)==1),[map;exterior]);
285     if(~isempty(add))
286         oldnumN=numnodes;
287         mult=[-1,-1,-1;1,-1,-1;1,1,-1;-1,1,-1;-1,-1,1;1,-1,1;1,1,1;-1,1,1];
288         Nfull=permute([sX(add),sY(add),sZ(add)],[3,2,1])+(Esize/2).*mult;
289         Nall=reshape(permute(Nfull,[1,3,2]),[],3);
290         [~,Nnum]=ismembertol(Nall,nodes,0.01*Esize(1),'ByRows',true);
291         nodes=[nodes;Nall(Nnum==0,:)];
292         Nnum(Nnum==0)=(numnodes+1):(numnodes+sum(Nnum==0));
293         elements=[elements;reshape(Nnum,8,[])'];
294         numnodes=size(nodes,1);
295         numelem=size(elements,1);
296         map=[map;add];
297         dof=[dof;3*repelem(elements(end-numel(add)+1:end,:),1,3)-repmat([2,1,0],1,8)];
298         L2ED=(nodes(elements(end-numel(add)+1:end,1),1)+Esize(1)/2'-lsfX(:)).^2+...
299             (nodes(elements(end-numel(add)+1:end,1),2)+Esize(2)/2'-lsfX(:)).^2+...
300             (nodes(elements(end-numel(add)+1:end,1),3)+Esize(3)/2'-lsfX(:)).^2;
301         Hij=[Hij,max(R-L2ED,0)];
302         K=oldK; F=oldF;
303         oldF=[oldF;zeros(3*(numnodes-oldnumN),1)];
304         oldK=sparse(3*numnodes,3*numnodes);
305         oldK(1:3*oldnumN,1:3*oldnumN)=K;
306         for(a=1:numel(add))
307             oldK(dof(end+1-a,:),dof(end+1-a,:))=oldK(dof(end+1-a,:),dof(end+1-
308 a,:))+0.0001*ke;
309         end
310         fprintf('Added %d elements, new node total:%d\n',numel(add),numnodes)
311         clear K a N Nnum add L2ED
312     end
313 end
314 CompE=zeros(numelem,1);
315 i=i+1;
316 %-----
317 end
318 %End of Optimization
319

```

```
320 %Show Final Values
321 disp('done')
```

Published with MATLAB® R2018a

Appendix D: Initial Configuration Subfunction

Variable Name	Variable Size	Description
elements	(numelem)x8	Row for each element and a column for each of the element's node numbers
nodes	(numnodes)x3	[x,y,z] coordinates for each node
boundary	(r)x1	List of elements that are on the boundary of the geometry
init	3x3	Defines the initial void geometry. 1st row defines edge lengths of voids, 2nd row defines x,y,z gap between voids and 3rd row the number of voids in the x,y, and z directions
struc	(Nx)x(Ny)x(Nz)	Material distribution, 0 for void 1 for material
Esize	1x3	x, y, and z edge lengths of each element
map	(numelem)x1	Index positions of each element in the LSF
noF	1x(c)	Degrees of freedom on homogeneous boundary for FEA
exterior	(r)x1	List of indices of 'struc' that are exterior to the design domain
Domain	2x3	1st row is the minimum coordinates in x,y, and z of the geometry and the second row is the maximum
StrucSize	(r)x(c)x(p)	Number of elements in the structure in each direction
ind	(r)x1	'struc' indices for each meshed element
void	(r)x(c)x(p)	Structural representation of the initial void based on 'init'
vs	1x3	Initial size of the initial void based on 'init'
bounds	2x3	Start and end position in i,j,k indices of the LSF for the initial void centered in the structure
bn	(r)x8	Nodes that are on a boundary element
noFnodes	(r)x1	Nodes that are on the boundary

InitialStruc.m

```

1  function [struc,Esize,map,noF,exterior] =InitialStruc(elements,nodes,boundary,init)
2  %Evaluates initial values prior to optimization loop
3  %
4
5  Domain=[min(nodes);max(nodes)];
6  Esize=max(nodes(elements(1,:),:))-min(nodes(elements(1,:),:));

```



```

7   StrucSize=round((Domain(2,:)-Domain(1,:))./Esize);
8   struc=ones(StrucSize);
9   ind=round(nodes(elements(:,1),:)./Esize+1-Domain(1,:)./Esize);
10  map=sub2ind(StrucSize,ind(:,1),ind(:,2),ind(:,3)); %map is a list for each element, which struc
11  index is used
12  void=zeros(init(1,1)*init(3,1)+init(2,1)*(init(3,1)-1),...
13      init(1,2)*init(3,2)+init(2,2)*(init(3,2)-1),...
14      init(1,3)*init(3,3)+init(2,3)*(init(3,3)-1));
15  void([0:init(3,1)-1]*(init(1,1)+init(2,1))+[1:init(1,1)],...
16      [0:init(3,2)-1]*(init(1,2)+init(2,2))+[1:init(1,2)],...
17      [0:init(3,3)-1]*(init(1,3)+init(2,3))+[1:init(1,3)])=1;
18  vs=size(void);
19  bounds=round(mean(nodes)./Esize-vs/2);
20  void=void(max(1,2-bounds(1)):min(vs(1),StrucSize(1)-bounds(1)-1),...
21      max(1,2-bounds(2)):min(vs(2),StrucSize(2)-bounds(2)-1),...
22      max(1,2-bounds(3)):min(vs(3),StrucSize(3)-bounds(3)-1));
23  bounds=[max(2,bounds);max(2,bounds)+size(void)-1];
24  struc(bounds(1,1):bounds(2,1),bounds(1,2):bounds(2,2),bounds(1,3):bounds(2,3))=...
25      max(0,struc(bounds(1,1):bounds(2,1),bounds(1,2):bounds(2,2),bounds(1,3):bounds(2,3))-void);
26  struc(map(boundary))=1;
27  struc(setdiff(1:prod(StrucSize),map))=1;
28
29
30  bn=elements(boundary,:);
31  bn=unique(bn(:));
32  noFnodes=bn(find(sum(bn'==elements(:))<8));
33  noF=reshape(3*noFnodes'-[2;1;0],1,[]);
34  exterior=(setdiff(1:numel(struc),map))';
35
36  end

```

[Published with MATLAB® R2018a](#)

Appendix E: Stiffness Matrix Calculation

Variable Name	Size	Description
E	scalar	Modulus of Elasticity
v	scalar	Poison's Ratio
lx	scalar	Length of each element in the x-direction
ly	scalar	Length of each element in the y-direction
lz	scalar	Length of each element in the z-direction
Ke	24x24	Elemental stiffness matrix
C	6*6	Constitutive relation for the material, used for stress calculations
num_nodes	scalar	Number of nodes for each element
J	3*3	Jacobian matrix
dN	8x3 cell	Derivatives of shape functions
n	2x2	1-D shape functions
dn	2x1	1-D derivative of shape functions
xy	2x2	2-D shape function in x and y for the given node
dxy	2x1	Partial derivative in x for 2-D shape function in x and y for the given node
xdy	1x2	Partial derivative in y for 2-D shape function in x and y for the given node
P_1D	1x2	Gauss Points in 1-D
W	1x8	Weighting factor for each Gauss point
Gpts	3x8	Master element Gauss points
Ng	8x3x8	Derivatives of shape functions evaluated at each Gauss point
D	scalar	Number of directions (3 for 3-D)
G	scalar	Number of Gauss points (8)
delN	8x3	Derivative of shape function on real element
B	6x24	B-matrix in elemental stiffness matrix calculation, used for stress calculations

stiff3D.m

```

1  function [Ke,B,C] = stiff3D(E,v,lx,ly,lz)
2  %Calculates the elemental stiffness matrices
3
4  %Inputs:    -E:modulus of elasticity
5  %          -v:Poison's ratio
6  %Outputs:   -Ke:elemental stiffness matrix
7  %          -Ktr:element matrix for trace tensor
8  %          -lamda:Lame Constant

```

```

9
10 c=(E/((1+v)*(1-2*v)))*[(1-v)*eye(3)+v*~eye(3)],zeros(3);zeros(3),((1-2*v)/2)*eye(3)];
11
12
13
14 if(nargin==2)
15     %ke=[(3-v)/6 , (1+v)/8 , (-3-v)/12 , (3*v-1)/8 , (v-3)/12 , (-1-v)/8 , v/6 , (1-3*v)/8];
16     kp=[-(3*v-2)/9,1/24,-1/18,-(4*v-1)/24,(4*v-1)/24,1/36,1/48,-1/24,(6*v-5)/72,-(4*v-1)/48,-
17     1/48,(4*v-1)/48,(3*v-1)/36,(3*v-2)/36];
18     k1=kp([1,2,2,3,5,5;2,1,2,4,6,7;2,2,1,4,7,6;3,4,4,1,8,8;5,6,7,8,1,2;5,7,6,8,2,1]);
19     k2=kp([9,8,12,6,4,7;8,9,12,5,3,5;10,10,13,7,4,6;6,5,11,9,2,10;4,3,5,2,9,12;11,4,6,12,10,13]);
20     k3=kp([6,7,4,9,12,8;7,6,4,10,13,10;5,5,3,8,12,9;9,10,2,6,11,5;12,13,10,11,6,4;2,12,9,4,5,3]);
21
22 k4=kp([14,11,11,13,10,10;11,14,11,12,9,8;11,11,14,12,8,9;13,12,12,14,7,7;10,9,8,7,14,11;10,8,9,7,
23 11,14]);
24 k5=kp([1,2,8,3,5,4;2,1,8,4,6,11;8,8,1,5,11,6;3,4,5,1,8,2;5,6,11,8,1,8;4,11,6,2,8,1]);
25
26 k6=kp([14,11,7,13,10,12;11,14,7,12,9,2;7,7,14,10,2,9;13,12,10,14,7,11;10,9,2,7,14,7;12,2,9,11,7,1
27 4]);
28
29 ke=(E/((1+v)*(1-2*v)))*[k1,k2,k3,k4;k2',k5,k6,k3';k3',k6,k5',k2';k4,k3,k2,k1'];
30
31 dn_cent=0.25*[0,-1,-1,-1;0,1,-1,-1;0,1,1,-1;0,-1,1,-1;0,-1,-1,1;0,1,-1,1;0,1,1,1;0,-1,1,1]';
32 order=[1,0,0;0,2,0;0,0,3;0,3,2;3,0,1;2,1,0];
33 B=dn_cent([order+1,order+5,order+9,order+13,order+17,order+21,order+25,order+29]);
34 else
35     if(nargin==3)
36         if(numel(lx)==3)
37             ly=lx(2);
38             lz=lx(3);
39             lx=lx(1);
40         else
41             ly=lx(1);
42             lz=lx(1);
43             lx=lx(1);
44         end
45     end
46     num_nodes=8;
47     J=[lx/2,ly/2,lz/2].*eye(3);
48     dn=cell(8,3);
49     n=[-1/2,1/2;1/2,1/2];    dn=[-1/2;1/2];
50     for(i=1:num_nodes)
51         xy=n(floor(mod(i-1,4)/2)+1,:)'*n(floor(mod(i,4)/2)+1,:);    %[y]'*[x]
52         dxy=n(floor(mod(i-1,4)/2)+1,:)'*dn(floor(mod(i,4)/2)+1,:);    %[y]'*[dx]
53         xdy=dn(floor(mod(i-1,4)/2)+1,:)'*n(floor(mod(i,4)/2)+1,:);    %[dy]'*[x]
54         for(c=1:size(xy,2))
55             if(c<=size(dxy,2))    %because partial wrt x will have 1 less column
56                 dn{i,1}=[dn{i,1},permute(dxy(:,c)*n(floor((i-1)/4)+1,:),[1,3,2])];
57             %[dxy(:,c)]*[z]
58         end
59         dn{i,2}=[dn{i,2},permute(xdy(:,c)*n(floor((i-1)/4)+1,:),[1,3,2])];    %[xdy(:,c)]*[z]

```

```

60         dN{i,3}=[dN{i,3},permute(xy(:,c)*dn(floor((i-1)/4)+1,:),[1,3,2])];    %[xy(:,c)]*[dz]
61     end
62 end
63
64     P_1D=[-1/3^0.5,1/3^0.5];    w=ones(1,8);
65     GPts=P_1D([1,1,1,1,2,2,2,2;1,1,2,2,1,1,2,2;1,2,1,2,1,2,1,2]);
66     Ng=poly3Deval(dN,GPts);
67     [num_nodes,D,G]=size(Ng);
68     Ke=zeros(num_nodes*D);
69     for(g=1:G)
70         de1N=J^(-1)*Ng(:, :, g)';
71         B=zeros(2*D, num_nodes*D);
72         for(n=1:num_nodes)
73             for(d=1:D)
74                 a=[1:d-1,d+1:3];
75                 B(d,n*D-D+d)=de1N(d,n);
76                 B(D+a(1),n*D-D+d)=de1N(a(2),n);
77                 B(D+a(2),n*D-D+d)=de1N(a(1),n);
78             end
79         end
80         Ke=Ke+B'*C*B*det(J)*w(g);
81     end
82 end
83
84
85
86 end

```

Published with MATLAB® R2018a

Appendix F: FEA Code

Variable Name	Size	Description
struc	(NSx)x(NSy)x(NSz)	Material distribution, 0 for void 1 for material
elements	(numelem)x8	Row for each element and a column for each of the element's node numbers
map	(numelem)x1	Index positions of each element in the LSF
KE	24x24	Elemental stiffness matrix
Po	1x3	Outward normal force each void element's nodes experience
fixeddofs	9x1	List of fixed degrees of freedom
oldstruc	(Nx)x(Ny)x(Nz)	Previous iteration's 'struc' matrix used to compare for changes in the FEA global matrices
oldK	(#dofs)x(#dofs)	Previous iteration's global stiffness matrix
oldF	(#dofs)x1	Previous iteration's global force vector
numnodes	scalar	Number of nodes
numelements	scalar	Number of elements
U	(#dofs)x1	Deflection values for each degree of freedom
dof	(numelem)x24	Degrees of freedom for each element of the mesh
fe	24x1	Elemental force vector for a void element
F	(#dofs)x1	Global force vector
K	(#dofs)x(#dofs)	Global stiffness matrix
eKE	24x24x(numelem)	Each element's stiffness matrix
ele	(r)x1	List of elements that changed since previous iteration
Ke_old	24x24	Previous iteration's elemental stiffness matrix
Ke	24x24	Current iteration's elemental stiffness matrix
Fe_old	24x1	Previous iteration's elemental force vector
Fe	24x1	Current iteration's elemental force vector
freedofs	(r)x1	List of non-partitioned degrees of freedom

FEA_3DP6.m

```

1  function [U,K,F] = FEA_3DP6(struc,elements,map,KE,Po,noF,fixeddofs,oldstruc,oldK,oldF)
2  %for irregular shapes
3
4  %Computes Finite element analysis for the structure where 1 means there is
5  %material and 0 corresponds to void
6  %Inputs:    -struc:material distribution representation (1=material & 0=void)
7  %           -elements:mapping of which nodes belong to each element and
8  %           their relative positionings
9  %           -KE:elemental k matrix
10 %           -Po:magnitude of pressure

```

```

11 % -e:used in computation of diriac delta function to determine
12 % pressure loading for the given LSF
13 % -fixeddofs:[degrees of freedom that are fixed]
14 % -oldstruc:The previous structure that the K matrix was
15 % calculated for so that the elements that don't change don't
16 % need to be recomputed in the global K-matrix
17 % -oldK:Previous global K matrix to serve as starting point for
18 % the this iteration
19 %Outputs: -U:dispacement vector result from FEA
20 % -K:current global K matrix to be used as starting point for the
21 % next iteration
22
23
24 %Initialize F,K and U Matrices-----
25 numnodes=max(max(elements));
26 numelements=size(elements,1);
27 U=zeros(3*numnodes,1);
28 dof=3*repelem(elements,1,3)-repmat([2,1,0],1,8);
29 fe=repmat(Po',8,1).*[1;1;1;-1;1;1;-1;-1;1;1;-1;-1;1;1;-1;-1;-1;-1;-1;-1];
30 %-----
31
32 %-----
33 if(nargin<7 || isempty(oldK)|| isempty(oldF))
34     %compute full K matrix
35     F=zeros(3*numnodes,1);
36     K=sparse(3*numnodes,3*numnodes);
37     eKE=permute(max(struc(map),0.0001),[3,2,1]).*KE;
38     for(e=1:numelements)
39         K(dof(e,:),dof(e,:))=K(dof(e,:),dof(e,:))+eKE(:,e);
40         if(struc(map(e))==1)
41             F(dof(e,:))=F(dof(e,:))+fe;
42         end
43     end
44 else
45     %only modify K and F where needed
46     K=oldK; F=oldF;
47     ele=find(struc(map)-oldstruc(map)); %elements that changed
48     for(i=1:numel(ele)) %0==no change, 1==added material, -1==removed material
49         Ke_old=max(oldstruc(map(ele(i))),0.0001)*KE;
50         Ke=max(struc(map(ele(i))),0.0001)*KE;
51         K(dof(ele(i,:),dof(ele(i,:),:))=K(dof(ele(i,:),dof(ele(i,:),:))-Ke_old+Ke;
52         Fe_old=oldstruc(map(ele(i)))*fe;
53         Fe=struc(map(ele(i)))*fe;
54         F(dof(ele(i,:),:))=F(dof(ele(i,:),:))-Fe_old+Fe;
55     end
56
57 end
58
59 %Solve System of Equations
60 F(noF)=0;
61 freedofs=setdiff(1:3*numnodes,fixeddofs);

```

```
62 U(freedofs,:)=K(freedofs,freedofs)\F(freedofs,:);  
63  
64 end
```

[Published with MATLAB® R2018a](#)

Appendix G: Update Code

Variable Name	Size	Description
lsf	(NLx)x(NLy)x(NLz)	Level-Set Function values
shapeSens	(NLx)x(NLy)x(NLz)	Shape sensitivity calculated from elemental strain energy densities and penalties
stepLength	scalar	Number of CFL time steps the evolution equation is solved every iteration
bearing	(r)x1	List of indexes of the LSF that are to remain constant and not change (regions outside the domain and the boundary elements)
Le	scalar	Element length, used in determining CFL condition
C	3x3x3	Matrix to perform convolution with for sensitivity smoothing
v	(NLx)x(NLy)x(NLz)	Velocities for the Hamilton-Jacobi equation
vFull	(Nx+2)x(Ny+2)x(Nz+2)	'v' with a border of zeros
dt	scalar	Time step, 0.1 of the CFL condition
dpx	(Nx+2)x(Ny+2)x(Nz+2)	Finite difference in the positive x direction
dmx	(Nx+2)x(Ny+2)x(Nz+2)	Finite difference in the negative x direction
dpy	(Nx+2)x(Ny+2)x(Nz+2)	Finite difference in the positive y direction
dmy	(Nx+2)x(Ny+2)x(Nz+2)	Finite difference in the negative y direction
dpz	(Nx+2)x(Ny+2)x(Nz+2)	Finite difference in the positive z direction
dmz	(Nx+2)x(Ny+2)x(Nz+2)	Finite difference in the negative z direction

Updatestep3

```

1  function [lsf] = updatestep3(lsf,shapeSens,stepLength,bearing,Le)
2  %updates the structure and the level-set function
3
4  %smooth sensitivities
5  C=reshape([0,1,0;1,2,1;0,1,0;1,2,1;2,3,2;1,2,1;0,1,0;1,2,1;0,1,0],3,3,3)/27;
6  shapeSens=convn(padarray(shapeSens,[1,1,1],'replicate'),C,'valid');
7
8  %Insure load bearing pixels remain solid
9  shapeSens(bearing)=0;
10
11  v=-shapeSens;
12  %add zeros to boarder of v
13  vFull=zeros(size(v)+2); vFull(2:end-1,2:end-1,2:end-1)=v;
14  lsf=padarray(lsf,[1,1,1],'replicate');
15
16  %determine timestep (based on CFL condition)

```



```

17 dt=Le*0.1/max(abs(v(:)));
18
19 for(i=1:(10*stepLength))
20     dpx=circshift(lsf,[-1,0,0])-lsf; %Find derivatives on the grid
21     dmx=lsf-circshift(lsf,[1,0,0]);
22     dpy=circshift(lsf,[0,-1,0])-lsf;
23     dmy=lsf-circshift(lsf,[0,1,0]);
24     dpz=circshift(lsf,[0,0,-1])-lsf;
25     dmz=lsf-circshift(lsf,[0,0,1]);
26     %Update LSF
27     lsf=lsf-
28     dt*min(vFull,0).*sqrt(min(dmx,0).^2+max(dpx,0).^2+min(dmy,0).^2+max(dpy,0).^2+min(dmz,0).^2+max(d
29     pz,0).^2) ...
30     -
31     dt*max(vFull,0).*sqrt(max(dmx,0).^2+min(dpx,0).^2+max(dmy,0).^2+min(dpy,0).^2+max(dmz,0).^2+min(d
32     pz,0).^2);
33 end
34
35 lsf=lsf(2:end-1,2:end-1,2:end-1);
36
37 end

```

[Published with MATLAB® R2018a](#)

Appendix H: Remesh Code

Variable Name	Size	Description
lsf	(NLx)x(NLy)x(NLz)	Level-Set Function values
LSFcoord	(numLSF)x3	Coordinates of LSF kernels
Esize	1x3	x, y, and z edge lengths of each element
file	String	STL file name
struc	(NSx)x(NSy)x(NSz)	Material distribution, 0 for void 1 for material
elements	(numelem)x8	Row for each element and a column for each of the element's node numbers
nodes	(numnodes)x3	[x,y,z] coordinates for each node
map	(numelem)x1	Index positions of each element in the LSF
boundary	(r)x1	List of elements that are on the boundary of the geometry
noF	1x(c)	Degrees of freedom on homogeneous boundary for FEA
sX, sY, & sZ	(NSx)x(NSy)x(NSz)	Meshgrid of coordinates of the elements' centroids
exterior	(r)x1	List of indices of 'struc' that are exterior to the design domain
faces	(#triangles)x3	Outward normal direction components of STL faces
vertices	3x3x(#triangles)	(each node of the triangle)x(x, y, z coordinate of the node)x(each STL triangle)
ranges	2x3	minimum STL node coordinates over the maximum
x_centroids	1x(c)	x-coordinates of the centroid for each cell
y_centroids	1x(c)	y-coordinates of the centroid for each cell
z_centroids	1x(c)	z-coordinates of the centroid for each cell
xyfaces	(r)x1	List of STL faces that have a z component
X	(r)x(c)	X values from a meshgrid of x and y centroids
Y	(r)x(c)	Y values from a meshgrid of x and y centroids
numxyzf	1x4	[number of cell elements in x, y, and z then the number of xy-faces]
cells	(r)x(c)x(p)	Logic representation of if a cell has material or not (-1=void, 1=solid)
p	3x2	x and y coordinates of the 3 nodes that make up the given triangle
intersects	(r)x(c)	Logic value of which x and y centroids are within the given triangle
A	scalar	Area of given triangle
up	scalar	1,2,3 index value for cross product to make shape function
down	scalar	1,2,3 index value for cross product to make shape function

Variable Name	Size	Description
Plane	2x2	Coefficient representation of shape function for triangle
I	(r)x1	Indices of x and y centroids that are within the given triangle
z_int	(r)x1	Z values of where the STL triangle intersects each x, y centroid
mult	1x(c)	Numbers to multiply along z to change 'cells'
Domain	2x3	1st row is the minimum coordinates in x,y, and z of the geometry and the second row is the maximum
Fullcells	(r)x(c)x(p)	'cells' matrix bordered by -1's
Nf_1458	(r)x(c)x(p)	True/False matrix the size of 'Fullcells' with a true value for cells without an element in the negative x-direction
Nf_2367	(r)x(c)x(p)	True/False matrix the size of 'Fullcells' with a true value for cells without an element in the positive x-direction
Nf_1256	(r)x(c)x(p)	True/False matrix the size of 'Fullcells' with a true value for cells without an element in the negative y-direction
Nf_3478	(r)x(c)x(p)	True/False matrix the size of 'Fullcells' with a true value for cells without an element in the positive y-direction
Nf_1234	(r)x(c)x(p)	True/False matrix the size of 'Fullcells' with a true value for cells without an element in the negative z-direction
Nf_5678	(r)x(c)x(p)	True/False matrix the size of 'Fullcells' with a true value for cells without an element in the positive z-direction
outer	1x(c)	Indices of cell that are not completely surrounded by elements
nelx	scalar	Number of elements in x
nely	scalar	Number of elements in y
nelz	scalar	Number of elements in z
Elements	(r)x(c)	List of element numbers
n1z	(r)x1	List of node one z-coordinates for each element
n1x	(r)x1	List of node one x-coordinates for each element
n1y	(r)x1	List of node one y-coordinates for each element
Relative	8x1	Relative node numberings from node 1 for each element
Nodes	(r)x3	List of all possible node coordinates
ind	(r)x1	Indices for repeated nodes
N	scalar	Node number counter
mapFull	(r)x(c)x(p)	Same as 'map' but for a matrix with a border
noFnodes	(r)x1	Nodes that are on the boundary

remesh.m

```

1 function [struc,elements,nodes,map,boundary,noF,sX,sY,sZ,exterior] =
2 remesh(1sf,LSFcoord,ESize,file)

```

```

3  %Remeshes material domain excluding void regions
4
5  [faces,vertices]=readSTL(file,'inches');
6
7  ranges=[min(min(vertices),[],3);max(max(vertices),[],3)];
8  x_centroids=ranges(1,1)+0.5*Esize(1):Esize(1):ranges(2,1);
9  y_centroids=ranges(1,2)+0.5*Esize(2):Esize(2):ranges(2,2);
10 z_centroids=ranges(1,3)+0.5*Esize(3):Esize(3):ranges(2,3);
11 xyfaces=find(faces(:,3)~=0);
12
13 [X,Y]=meshgrid(x_centroids,y_centroids);
14 numxyzf=[flipr(size(X)),numel(z_centroids),numel(xyfaces)];
15 cells=-1*ones(numxyzf([2,1,3]));
16 for(f=1:numxyzf(4))
17     p=vertices(:,1:2,xyfaces(f));
18     intersects=inpolygon(X,Y,p(:,1),p(:,2));
19     A=0.5*det([1;1;1],p);
20     Plane=zeros(2);
21     for(i=1:3)
22         up=rem(i,3)+1;
23         down=3-rem(4-i,3);
24         Plane=Plane+0.5*vertices(i,3,xyfaces(f))*[0,p(down,1)-p(up,1);p(up,2)-
25 p(down,2),p(up,1)*p(down,2)-p(down,1)*p(up,2)]/A;
26     end
27     I=find(intersects);
28     if(~isempty(I))
29         z_int=poly2Deval(Plane,[X(I),Y(I)]);
30         mult=-1*(z_centroids>=z_int)+(z_centroids<z_int);
31         for(i=1:numel(I))
32             [r,c]=ind2sub([numxyzf(2),numxyzf(3)],I(i));
33             cells(r,c,:)=cells(r,c,:).*permute(mult(i,:),[1,3,2]);
34         end
35     end
36 end
37 cells=permute(cells,[2,1,3]);
38 exterior=find(cells===-1);
39
40 Domain=ranges-ranges(1,:);
41 [sX,sY,sZ]=meshgrid(Esize(1)/2:Esize(1):Domain(2,1),...
42     Esize(2)/2:Esize(2):Domain(2,2),Esize(3)/2:Esize(3):Domain(2,3));
43 sX=permute(sX,[2,1,3]); sY=permute(sY,[2,1,3]); sZ=permute(sZ,[2,1,3]);
44 struc=griddata(LSFcoord(:,1),LSFcoord(:,2),LSFcoord(:,3),lsf,sX,sY,sZ)<=0;
45
46
47 Fullcells=padarray(cells,[1,1,1],-1);
48 Nf_1458=(Fullcells-circshift(Fullcells,[1,0,0]))==2;
49 Nf_2367=(Fullcells-circshift(Fullcells,[-1,0,0]))==2;
50 Nf_1256=(Fullcells-circshift(Fullcells,[0,1,0]))==2;
51 Nf_3478=(Fullcells-circshift(Fullcells,[0,-1,0]))==2;
52 Nf_1234=(Fullcells-circshift(Fullcells,[0,0,1]))==2;
53 Nf_5678=(Fullcells-circshift(Fullcells,[0,0,-1]))==2;

```

```

54
55
56 outer=(cells==1).*(convn(cells,ones(3,3,3),'same')<27);
57 struc(find(outer))==1;
58
59
60 nelx=numxyzf(1);    nely=numxyzf(2);    nelz=numxyzf(3);
61 Elements=1:nelx*nely*nelz;
62 nlz=floor((Elements-1)/(nelx*nely));
63 nlx=rem((Elements-(nelx*nely).*floor((Elements-1)/(nelx*nely))-1),nelx);
64 nly=floor((Elements-(nelx*nely).*floor((Elements-1)/(nelx*nely))-1)/nelx);
65 Relative=[0;1;nelx+2;nelx+1;...
66     (nelx+1)*(nely+1);(nelx+1)*(nely+1)+1;(nelx+1)*(nely+1)+nelx+2;(nelx+1)*(nely+1)+nelx+1];
67 Elements=(1+nlx+nly*(nelx+1)+nlz*(nelx+1)*(nely+1))+Relative';
68 [Nodes(:,1),Nodes(:,2),Nodes(:,3)]=ind2sub([nelx+1,nely+1,nelz+1],1:(nelx+1)*(nely+1)*(nelz+1));
69 Nodes=Esize.*(Nodes-[1,1,1]);
70
71
72 Elements=(cells(:)==1 & struc(:)==1).*Elements;    %find on elements
73 Elements((Elements(:,1)==0),:)=[];    %remove off elements
74
75 elements=zeros(size(Elements));
76 nodes=zeros(size(Nodes));
77 N=1;
78 while(sum(elements(:)==0)>0)
79     [c,r]=find(elements'==0,1);
80     ind=find(Elements==Elements(r,c));
81     elements(ind)=N;
82     nodes(N,:)=Nodes(Elements(r,c),:);
83     N=N+1;
84     if(~mod(N,5000))
85         fprintf('meshing node %d of %d \n',sum(elements(:)~=0),numel(Elements));
86     end
87 end
88 nodes(N:end,:)=[];
89
90 ind=round(nodes(elements(:,1),:)./Esize+1-Domain(1,:)./Esize);
91 map=sub2ind(numxyzf([1,2,3]),ind(:,1),ind(:,2),ind(:,3));    %map is a list for each element, which
92 struc index is used
93 mapFull=sub2ind(numxyzf([1,2,3])+2,ind(:,1)+1,ind(:,2)+1,ind(:,3)+1);
94
95 boundary=find(outer(map));
96 noFnodes=[elements(find(Nf_1458(mapFull)),[1,4,5,8]);...
97     elements(find(Nf_2367(mapFull)),[2,3,6,7]);...
98     elements(find(Nf_1256(mapFull)),[1,2,5,6]);...
99     elements(find(Nf_3478(mapFull)),[3,4,7,8]);...
100     elements(find(Nf_1234(mapFull)),[1,2,3,4]);...
101     elements(find(Nf_5678(mapFull)),[5,6,7,8])];
102 noF=reshape(3*unique(noFnodes(:))-[2;1;0],1,[]);
103

```

104
105

end

Published with MATLAB® R2018a

Appendix I: Polynomial Evaluation Code

Variable Name	Size	Description
coef	cell array	Each cell contains coefficients for a function
points	3x(c)	List of x,y,z coordinates to evaluate each function at
Req	scalar	Number of rows in coef
Ceq	scalar	Number of columns in coef
eval	(r)x(c)x(p)	Each function evaluated at each point
xp	scalar	Power of x to be multiplied to particular coefficient
yp	scalar	Power of y to be multiplied to particular coefficient
zp	scalar	Power of z to be multiplied to particular coefficient

poly3Deval.m

```

1  function [eval] = poly3Deval(coef,points)
2  if(iscell(coef))
3      [Req,Ceq]=size(coef);
4  else
5      Req=1;
6      Ceq=1;
7  end
8  if(nargin==1)
9      P=input('at what location would you like to evaluate? :');
10 else
11     P=points;
12 end
13 eval=zeros([Req,Ceq,size(points,2)]); %Initializes function value matrix
14 for(r=1:Req)
15     for(c=1:Ceq)
16         for(p=1:size(points,2))
17             xp=0;
18             for(i=size(coef{r,c},1):-1:1)
19                 yp=0;
20                 for(j=size(coef{r,c},2):-1:1)
21                     zp=0;
22                     for(k=size(coef{r,c},3):-1:1)
23                         eval(r,c,p)=eval(r,c,p)+coef{r,c}(i,j,k)*P(1,p)^xp*P(2,p)^yp*P(3,p)^zp;
24                         zp=zp+1;
25                     end
26                     yp=yp+1;
27                 end
28                 xp=xp+1;
29             end
30         end
31     end

```

```
32 end
33 end
```

[Published with MATLAB® R2018a](#)

Appendix J: Reading STL file Code

Variable Name	Size	Description
filename	Character	STL file name
units	Character	Units the STL file is in
vertices	3x3x(#triangles)	(each node of the triangle)x(x, y, z coordinate of the node)x(each STL triangle)
faces	(#triangles)x3	Outward normal direction components of STL faces
fid	scalar	File ID number in MATLAB
Title	Character	First line of the STL file
f	scalar	Face counter
line	Character	Current line being read
v	scalar	Vertex counter

readSTL.m

```

1  function [faces,vertices] = readSTL(filename,units)
2  %Reads an STL file found under the prescribed filename and then filters and
3  %outputs the face normals and vertices of each face
4
5
6  fid=fopen(filename,'r');
7
8  Title=fgetl(fid);
9  f=0;
10 while(feof(fid)==0)
11     line=fgetl(fid);
12     if(contains(line,'facet normal'))
13         f=f+1;
14         v=0;
15         faces(f,:)=str2num(line(17:end));
16     elseif(contains(line,'vertex'))
17         v=v+1;
18         vertices(v,:,f)=str2num(line(17:end));
19     end
20 end
21 fclose(fid);
22
23
24 if(units=='inches')
25     vertices=vertices/25.4;
26 end
27
28 end

```

Published with MATLAB® R2018a

Appendix K: Cross-Section Viewing Code

Variable Name	Size	Description
folder	String	Folder name that the iteration data is saved into
iteration	scalar	Which iteration to plot
Domain	2x3	1st row is the minimum coordinates in x,y, and z of the geometry and the second row is the maximum
W	2x3	Current window view
Esize	1x3	x, y, and z edge lengths of each element
Done	Logical	T/F for when to exit the loop
mat_files	(r)x1 cell array	List of MATLAB data files in 'folder'
volTot	Scalar	Total design domain volume
Yield	Scalar	Yield strength for the given material
VonoMises	(numelem)x1	VonMises stress value for each element
StrucSize	1x3	Number of elements of the structure in each direction
ind	(r)x1	Index positions for elements
e	Scalar	Counter through each element
stress	6x1	Stress state for the given element
CalcComp	Logical	T/F for if the elemental compliances need to be calculated
OldNodes	(numnodes)x3	Coordinates for undeformed nodes
Cent	(numelem)x3	Centroid coordinates for each element
use	(r)x1	List of elements to plot based on the current window
f	Handle	Figure Handle
Xrange	Handle	UI panel to control the x-value ranges of the view window
Xmax_down	Handle	UI button to decrease the max range of the view window in X
Xmax_up	Handle	UI button to increase the max range of the view window in X
Xmax_Text	Handle	Text number indicator for max x range
Xmin_down	Handle	UI button to decrease the min range of the view window in X
Xmin_up	Handle	UI button to increase the min range of the view window in X
Xmin_Text	Handle	Text number indicator for min x range
Yrange	Handle	UI panel to control the y-value ranges of the view window
Ymax_down	Handle	UI button to decrease the max range of the view window in Y
Ymax_up	Handle	UI button to increase the max range of the view window in Y

Variable Name	Size	Description
Ymax_Text	Handle	Text number indicator for max y range
Ymin_down	Handle	UI button to decrease the min range of the view window in Y
Ymin_up	Handle	UI button to increase the min range of the view window in Y
Ymin_Text	Handle	Text number indicator for min y range
Zrange	Handle	UI panel to control the z-value ranges of the view window
Zmax_down	Handle	UI button to decrease the max range of the view window in Z
Zmax_up	Handle	UI button to increase the max range of the view window in Z
Zmax_Text	Handle	Text number indicator for max z range
Zmin_down	Handle	UI button to decrease the min range of the view window in Z
Zmin_up	Handle	UI button to increase the min range of the view window in Z
Zmin_Text	Handle	Text number indicator for min z range
DeflectTB	Handle	Toggle button to view deflected structure
MagSlide	Handle	Slider to control deflection magnification factor
MagText	Handle	Text indicator for magnification factor
P	Handle	Button group for which
tb1	Handle	Button to view solid plot
tb2	Handle	Button to view transparent plot
tb3	Handle	Button to view stress plot
tb4	Handle	Button to view only void elements
S	Handle	Button to save the current view
D	Handle	Button to finish and exit the code
itr	Handle	Text indicator for which iteration is plotted
CompTot	Handle	Text indicator for the iteration's compliance
CompWin	Handle	Text indicator for the current window view's compliance
VolTot	Handle	Text indicator for the iteration's volume fraction
Volfrac	Handle	Text indicator for the current window view's volume fraction
Intfrac	Handle	Text indicator for the iteration's volume fraction excluding border
last	1x3 cell array	Last plots conditions
CompE	(numelem)x1	Compliance for each element
dof	(numelem)x24	Degree of freedoms for each element
p	Logical	T/F indicator if plotting needs to be done

Variable Name	Size	Description
num	Scalar	The number of already saved figures so a figure doesn't get saved over

CrossSectionPlot.m

```

1  clear all
2  close all
3  clc
4  addpath([pwd, '\IrregularShapeSubfunctions'])
5  %Plots Cross-Sections of Structures
6
7  folder='0.45finish23-May-2020 10,08,20'; %iIPV 30 v=0.45
8  iteration='end'; %iteration # or 'end' for last iteration
9
10
11
12  global Domain W Esize Done
13  if(ischar(iteration))
14      mat_files=dir([folder, '/*.mat']);
15      iteration=numel(mat_files)-1;
16  end
17  Done=0;
18  load([folder, '/Iteration0'])
19  Esize=max(nodes(elements(1,:),:))-min(nodes(elements(1,:),:));
20  volTot=prod(Esize)*size(elements,1);
21  load([folder, '/Iteration', num2str(iteration)])
22  Domain=[min(nodes);max(nodes)];
23  W=Domain;
24  Esize=max(nodes(elements(1,:),:))-min(nodes(elements(1,:),:));
25  [Ke,B,C]=stiff3D(29.5*10^6,0.29,Esize);
26  Yield=150*10^3; %psi
27  vonMises=zeros(size(elements,1),1);
28  strucSize=size(struc);
29
30  ind=find(struc(map));
31  for(e=1:sum(struc(map)))
32      stress=C*B*U(3*repelem(elements(ind(e),:),1,3)-repmat([2,1,0],1,8));
33      vonMises(ind(e))=sqrt(sum((stress(1:3)-stress([2,3,1])).^2)+6*sum(stress(4:6).^2))/sqrt(2);
34  end
35  CalcComp=~exist('compE','var');
36  oldNodes=nodes;
37  Cent=nodes(elements(:,1),:)+Esize/2;
38  use=find(Cent(:,1)>=W(1,1)&Cent(:,1)<=W(2,1)&...
39      Cent(:,2)>=W(1,2)&Cent(:,2)<=W(2,2)&...
40      Cent(:,3)>=W(1,3)&Cent(:,3)<=W(2,3));
41  f=figure('Units','normalized','color','w');

```

```

42 fig=plotstructure(elements(use,:),nodes,struc,map(use));
43 axis equal; axis tight; view([30,30]); drawnow;
44 xlabel('x'); ylabel('y'); zlabel('z');
45 lgd=legend('Solid');
46 lgd.Position=[0.85,0.85,0.1,0.1];
47
48 %X Limits Control Panel-----
49 Xrange=uipanel('Title','X Limits','Position',[0.01,0.775,0.18755,0.125]);
50 uicontrol(Xrange,'Style','text','String','Max:','Units','normalized','FontUnits','normalized',...
51 'Position',[0,0.6,0.185,0.225],'FontSize',0.9);
52 Xmax_down=uiicontrol(Xrange,'Style','pushbutton','Units','normalized','FontUnits','normalized',...
53 'Position',[0.2,0.6,0.25,0.225],'String','Down','FontSize',0.9,'Callback',@XmaxDPushed);
54 Xmax_up=uiicontrol(Xrange,'Style','pushbutton','Units','normalized','FontUnits','normalized',...
55 'Position',[0.5,0.6,0.25,0.225],'String','Up','FontSize',0.9,'Callback',@XmaxUPushed);
56 Xmax_Text=uiicontrol(Xrange,'Style','text','Units','normalized','FontUnits','normalized',...
57 'Position',[0.775,0.6,0.22,0.225],'FontSize',0.9);
58 uicontrol(Xrange,'Style','text','String','Min:','Units','normalized','FontUnits','normalized',...
59 'Position',[0,0.3,0.185,0.225],'FontSize',0.9);
60 Xmin_down=uiicontrol(Xrange,'Style','pushbutton','Units','normalized','FontUnits','normalized',...
61 'Position',[0.2,0.3,0.25,0.225],'String','Down','FontSize',0.9,'Callback',@XminDPushed);
62 Xmin_up=uiicontrol(Xrange,'Style','pushbutton','Units','normalized','FontUnits','normalized',...
63 'Position',[0.5,0.3,0.25,0.225],'String','Up','FontSize',0.9,'Callback',@XminUPushed);
64 Xmin_Text=uiicontrol(Xrange,'Style','text','Units','normalized','FontUnits','normalized',...
65 'Position',[0.775,0.3,0.22,0.225],'FontSize',0.9);
66 %-----
67
68 %Y Limits Control Panel-----
69 Yrange=uipanel('Title','Y Limits','Position',[0.01,0.625,0.18755,0.125]);
70 uicontrol(Yrange,'Style','text','String','Max:','Units','normalized','FontUnits','normalized',...
71 'Position',[0,0.6,0.185,0.225],'FontSize',0.9);
72 Ymax_down=uiicontrol(Yrange,'Style','pushbutton','Units','normalized','FontUnits','normalized',...
73 'Position',[0.2,0.6,0.25,0.225],'String','Down','FontSize',0.9,'Callback',@YmaxDPushed);
74 Ymax_up=uiicontrol(Yrange,'Style','pushbutton','Units','normalized','FontUnits','normalized',...
75 'Position',[0.5,0.6,0.25,0.225],'String','Up','FontSize',0.9,'Callback',@YmaxUPushed);
76 Ymax_Text=uiicontrol(Yrange,'Style','text','Units','normalized','FontUnits','normalized',...
77 'Position',[0.775,0.6,0.22,0.225],'FontSize',0.9);
78 uicontrol(Yrange,'Style','text','String','Min:','Units','normalized','FontUnits','normalized',...
79 'Position',[0,0.3,0.185,0.225],'FontSize',0.9);
80 Ymin_down=uiicontrol(Yrange,'Style','pushbutton','Units','normalized','FontUnits','normalized',...
81 'Position',[0.2,0.3,0.25,0.225],'String','Down','FontSize',0.9,'Callback',@YminDPushed);
82 Ymin_up=uiicontrol(Yrange,'Style','pushbutton','Units','normalized','FontUnits','normalized',...
83 'Position',[0.5,0.3,0.25,0.225],'String','Up','FontSize',0.9,'Callback',@YminUPushed);
84 Ymin_Text=uiicontrol(Yrange,'Style','text','Units','normalized','FontUnits','normalized',...
85 'Position',[0.775,0.3,0.22,0.225],'FontSize',0.9);
86 %-----
87
88 %Z Limits Control Panel-----
89 Zrange=uipanel('Title','Z Limits','Position',[0.01,0.475,0.18755,0.125]);
90 uicontrol(Zrange,'Style','text','String','Max:','Units','normalized','FontUnits','normalized',...
91 'Position',[0,0.6,0.185,0.225],'FontSize',0.9);
92 Zmax_down=uiicontrol(Zrange,'Style','pushbutton','Units','normalized','FontUnits','normalized',...

```

```

93     'Position',[0.2,0.6,0.25,0.225],'String','Down','FontSize',0.9,'Callback',@ZmaxDPushed);
94     Zmax_up=uicontrol(Zrange,'Style','pushbutton','Units','normalized','FontUnits','normalized',...
95     'Position',[0.5,0.6,0.25,0.225],'String','Up','FontSize',0.9,'Callback',@ZmaxUPushed);
96     Zmax_Text=uicontrol(Zrange,'Style','text','Units','normalized','FontUnits','normalized',...
97     'Position',[0.775,0.6,0.22,0.225],'FontSize',0.9);
98     uicontrol(Zrange,'Style','text','String','Min:','Units','normalized','FontUnits','normalized',...
99     'Position',[0,0.3,0.185,0.225],'FontSize',0.9);
100    Zmin_down=uicontrol(Zrange,'Style','pushbutton','Units','normalized','FontUnits','normalized',...
101    'Position',[0.2,0.3,0.25,0.225],'String','Down','FontSize',0.9,'Callback',@ZminDPushed);
102    Zmin_up=uicontrol(Zrange,'Style','pushbutton','Units','normalized','FontUnits','normalized',...
103    'Position',[0.5,0.3,0.25,0.225],'String','Up','FontSize',0.9,'Callback',@ZminUPushed);
104    Zmin_Text=uicontrol(Zrange,'Style','text','Units','normalized','FontUnits','normalized',...
105    'Position',[0.775,0.3,0.22,0.225],'FontSize',0.9);
106    %-----
107
108    %Deflection Controls-----
109    DeflectB=uicontrol(f,'Style','togglebutton','Units','normalized','Position',[0.01,0.42,0.1875,0.0
110    4],'String','Deflection','FontUnits','normalized','FontSize',0.75);
111    MagSlide=uicontrol(f,'Style','slider','Units','normalized','Position',[0.01,0.375,0.11,0.04],'Min
112    ',0,'Max',100,'Value',10,'SliderStep',[1/1000,0.01]);
113    MagText=uicontrol(f,'Style','text','Units','normalized','FontUnits','normalized',...
114    'Position',[0.12,0.375,0.08,0.04],'FontSize',0.75);
115    set(MagText,'String',sprintf('Mag:%3.1f',MagSlide.Value))
116    %-----
117    %Plot Type Controls-----
118    P=uibuttongroup(f,'Position',[0.01,0.135,0.1875,0.21875],'Units','normalized');%SelectionChange
119    dFcn,@Ptype
120    tb1=uicontrol(P,'Style','togglebutton','Units','normalized','Position',[0.05,0.76,0.9,0.2],'Strin
121    g','Solid','FontUnits','normalized','FontSize',0.75);
122    tb2=uicontrol(P,'Style','togglebutton','Units','normalized','Position',[0.05,0.52,0.9,0.2],'Strin
123    g','Transparent','FontUnits','normalized','FontSize',0.75);
124    tb3=uicontrol(P,'Style','togglebutton','Units','normalized','Position',[0.05,0.28,0.9,0.2],'Strin
125    g','Stress','FontUnits','normalized','FontSize',0.75);
126    tb4=uicontrol(P,'Style','togglebutton','Units','normalized','Position',[0.05,0.04,0.9,0.2],'Strin
127    g','Void Only','FontUnits','normalized','FontSize',0.75);
128    %-----
129
130    S=uicontrol(f,'Style','pushbutton','Units','normalized','FontUnits','normalized',...
131    'Position',[0.01,0.0775,0.1875,0.05],'String','Save','FontSize',0.9,'Callback',@SPushed);
132    D=uicontrol(f,'Style','pushbutton','Units','normalized','FontUnits','normalized',...
133    'Position',[0.01,0.015,0.1875,0.05],'String','Done','FontSize',0.9,'Callback',@DPushed);
134    uicontrol(f,'Style','text','String','Iteration:','Units','normalized','FontUnits','normalized',...
135    .
136    'Position',[0.025,0.94,0.15,0.05],'FontSize',0.9);
137    itr=uicontrol(f,'Style','edit','Units','normalized','FontUnits','normalized',...
138    'Position',[0.18,0.94,0.0575,0.05],'String',num2str(iteration),'FontSize',0.9);
139
140    %Objective and Constraint Values-----
141    CompTot=uicontrol(f,'Style','text','Units','normalized','FontUnits','normalized',...
142    'Position',[0.7375,0.15,0.25,0.025],'FontSize',0.9,'horizontalAlignment','right');
143    Compwin=uicontrol(f,'Style','text','Units','normalized','FontUnits','normalized',...

```

```

144     'Position',[0.7375,0.12,0.25,0.025],'FontSize',0.9,'horizontalAlignment','right');
145 volTot=uicontrol(f,'Style','text','Units','normalized','FontUnits','normalized',...
146     'Position',[0.7375,0.09,0.25,0.025],'FontSize',0.9,'horizontalAlignment','right');
147 volfrac=uicontrol(f,'Style','text','Units','normalized','FontUnits','normalized',...
148     'Position',[0.7375,0.06,0.25,0.025],'FontSize',0.9,'horizontalAlignment','right');
149 Intfrac=uicontrol(f,'Style','text','Units','normalized','FontUnits','normalized',...
150     'Position',[0.7375,0.03,0.25,0.025],'FontSize',0.9,'horizontalAlignment','right');
151 set(volTot,'String',sprintf('Total Volume Fraction:%1.3f',prod(Esize)*sum(struc(map))/volTot))
152 set(volfrac,'String',sprintf('Window Volume
153 Fraction:%1.3f',prod(Esize)*sum(struc(map(use)))/volTot))
154 set(Intfrac,'String',sprintf('Window Interior Volume
155 Fraction:%1.3f',prod(Esize)*sum(struc(map(setdiff(use,boundary))))/volTot))
156 set(CompTot,'String',sprintf('Total Compliance:%10.3f',-sum(CompE(:))))
157 set(Compwin,'String',sprintf('Window Compliance:%10.3f',-sum(CompE(use))))
158 %-----
159
160 last={'Solid',num2str(iteration),0,MagSlide.Value};
161 lastW=w;    oldNodes=nodes;    p=0;
162 set(Xmax_Text,'String',w(2,1))
163 set(Xmin_Text,'String',w(1,1))
164 set(Ymax_Text,'String',w(2,2))
165 set(Ymin_Text,'String',w(1,2))
166 set(Zmax_Text,'String',w(2,3))
167 set(Zmin_Text,'String',w(1,3))
168
169 while(Done==0)
170     pause(0.01)
171     set(MagText,'String',sprintf('Mag:%3.1f',MagSlide.Value))
172     if(~isequal(lastW,w))
173         lastW=w;    p=1;
174         set(Xmax_Text,'String',w(2,1))
175         set(Xmin_Text,'String',w(1,1))
176         set(Ymax_Text,'String',w(2,2))
177         set(Ymin_Text,'String',w(1,2))
178         set(Zmax_Text,'String',w(2,3))
179         set(Zmin_Text,'String',w(1,3))
180         %xlim(w(:,1));    ylim(w(:,2));    zlim(w(:,3));
181         %Cent=nodes(elements(:,1),:)+Esize/2;
182         use=find(Cent(:,1)>=w(1,1)&Cent(:,1)<=w(2,1)&...
183             Cent(:,2)>=w(1,2)&Cent(:,2)<=w(2,2)&...
184             Cent(:,3)>=w(1,3)&Cent(:,3)<=w(2,3));
185         set(CompTot,'String',sprintf('Total Compliance:%10.3f',-sum(CompE(:))))
186         set(Compwin,'String',sprintf('Window Compliance:%10.3f',-sum(CompE(use))))
187         set(volTot,'String',sprintf('Total Volume
188 Fraction:%1.3f',prod(Esize)*sum(struc(map))/volTot))
189         set(volfrac,'String',sprintf('Window Volume
190 Fraction:%1.3f',prod(Esize)*sum(struc(map(use)))/volTot))
191         set(Intfrac,'String',sprintf('Window Interior Volume
192 Fraction:%1.3f',prod(Esize)*sum(struc(map(setdiff(use,boundary))))/volTot))
193     end
194     if(~strcmp(itr.String,last{2}))

```

```

195     p=1;
196     iteration=max(1,min([str2num(itr.String),(numel(mat_files)-1])));
197     set(itr,'String',num2str(iteration));
198     load([folder,'/Iteration',num2str(iteration)])
199     OldNodes=nodes;
200     Esize=max(nodes(elements(1,:),:))-min(nodes(elements(1,:),:));
201     vonMises=zeros(size(elements,1),1);
202     ind=find(struc(map));
203     for(e=1:sum(struc(map)))
204         stress=C*B*U(3*repelem(elements(ind(e),:),1,3)-repmat([2,1,0],1,8));
205         vonMises(ind(e))=sqrt(sum((stress(1:3)-
206 stress([2,3,1])).^2)+6*sum(stress(4:6).^2))/sqrt(2);
207     end
208     if(CalcComp==1)
209         CompE=zeros(size(elements,1),1);
210         dof=3*repelem(elements,1,3)-repmat([2,1,0],1,8);
211         for(e=1:size(elements,1))
212             CompE(e)=-max(struc(map(e)),0.0001)*U(dof(e,:))'*ke*U(dof(e,:));
213         end
214     end
215     Cent=nodes(elements(:,1),:)+Esize/2;
216     use=find(Cent(:,1)>=w(1,1)&Cent(:,1)<=w(2,1)&...
217         Cent(:,2)>=w(1,2)&Cent(:,2)<=w(2,2)&...
218         Cent(:,3)>=w(1,3)&Cent(:,3)<=w(2,3));
219     set(CompTot,'String',sprintf('Total Compliance:%10.3f',-sum(CompE(:))))
220     set(CompWin,'String',sprintf('window Compliance:%10.3f',-sum(CompE(use))))
221     set(volTot,'String',sprintf('Total volume
222 Fraction:%1.3f',prod(Esize)*sum(struc(map))/volTot))
223     set(volfrac,'String',sprintf('window volume
224 Fraction:%1.3f',prod(Esize)*sum(struc(map(use)))/volTot))
225     set(Intfrac,'String',sprintf('window Interior volume
226 Fraction:%1.3f',prod(Esize)*sum(struc(map(setdiff(use,boundary)))))/volTot))
227     end
228     if(DeflecTB.Value~=last{3} || (MagSlide.Value~=last{4}&&DeflecTB.Value==1))
229         p=1;
230     end
231
232     if(p==1 || ~strcmp(P.SelectedObject.String,last{1}))
233         cla
234         colorbar('off'); legend('off');
235         nodes=OldNodes+DeflecTB.Value*MagSlide.Value*reshape(U,3,[]);
236         if(strcmp(P.SelectedObject.String,'Solid'))
237             plotstructure(elements(use,:),nodes,struc,map(use));
238             lgd=legend('Solid');
239             lgd.Position=[0.85,0.85,0.1,0.1];
240         elseif(strcmp(P.SelectedObject.String,'Transparent'))
241             plottrans(elements(use,:),nodes,struc,map(use),find(ismember(use,boundary)));
242         elseif(strcmp(P.SelectedObject.String,'Stress'))
243             plotstress(elements(use,:),nodes,struc,map(use),vonMises(use),Yield);
244         else
245             plotvoid(elements,nodes,struc);

```



```

246         hold on;
247         plotSTL('Rotated Irregular Pressure Vessel.STL');
248         lgd=legend('Void');
249         lgd.Position=[0.85,0.85,0.1,0.1];
250     end
251     last={P.SelectedObject.String,itr.String,DeflectB.Value,MagSlide.Value};
252     p=0;
253 end
254 end
255 disp('Done')
256
257 %X Max Functions-----
258 function XmaxDPushed(scr,event)
259     global Domain W Esize
260     w(2,1)=min(Domain(2,1),max(w(1,1)+Esize(1),w(2,1)-Esize(1)));
261 end
262
263 function XmaxUPushed(scr,event)
264     global Domain W Esize
265     w(2,1)=min(Domain(2,1),max(w(1,1)+Esize(1),w(2,1)+Esize(1)));
266 end
267 %-----
268
269 %X Min Functions-----
270 function XminDPushed(scr,event)
271     global Domain W Esize
272     w(1,1)=max(Domain(1,1),min(w(2,1)-Esize(1),w(1,1)-Esize(1)));
273 end
274
275 function XminUPushed(scr,event)
276     global Domain W Esize
277     w(1,1)=max(Domain(1,1),min(w(2,1)-Esize(1),w(1,1)+Esize(1)));
278 end
279 %-----
280
281 %Y Max Functions-----
282 function YmaxDPushed(scr,event)
283     global Domain W Esize
284     w(2,2)=min(Domain(2,2),max(w(1,2)+Esize(2),w(2,2)-Esize(2)));
285 end
286
287 function YmaxUPushed(scr,event)
288     global Domain W Esize
289     w(2,2)=min(Domain(2,2),max(w(1,2)+Esize(2),w(2,2)+Esize(2)));
290 end
291 %-----
292
293 %Y Min Functions-----
294 function YminDPushed(scr,event)
295     global Domain W Esize
296     w(1,2)=max(Domain(1,2),min(w(2,2)-Esize(2),w(1,2)-Esize(2)));

```

```

297     end
298
299     function YminUPushed(scr,event)
300         global Domain W Esize
301         w(1,2)=max(Domain(1,2),min(w(2,2)-Esize(2),w(1,2)+Esize(2)));
302     end
303
304     %-----
305 %Z Max Functions-----
306     function ZmaxDPushed(scr,event)
307         global Domain W Esize
308         w(2,3)=min(Domain(2,3),max(w(1,3)+Esize(3),w(2,3)-Esize(3)));
309     end
310
311     function ZmaxUPushed(scr,event)
312         global Domain W Esize
313         w(2,3)=min(Domain(2,3),max(w(1,3)+Esize(3),w(2,3)+Esize(3)));
314     end
315
316     %-----
317 %Z Min Functions-----
318     function ZminDPushed(scr,event)
319         global Domain W Esize
320         w(1,3)=max(Domain(1,3),min(w(2,3)-Esize(3),w(1,3)-Esize(3)));
321     end
322
323     function ZminUPushed(scr,event)
324         global Domain W Esize
325         w(1,3)=max(Domain(1,3),min(w(2,3)-Esize(3),w(1,3)+Esize(3)));
326     end
327
328     %-----
329
330 %Plot type callbacks-----SolidPushed
331     function SPushed(scr,event)
332         %save figure
333         num=numel(dir('*Cross*fig*'))+1;
334         savefig(gcf,[pwd,'\','Cross-Section',num2str(num)]);
335     end
336
337     function DPushed(scr,event)
338         global Done
339         Done=1;
340     end

```

[Published with MATLAB® R2018a](#)

Appendix L: Solid Structure Plotting Code

Variable Name	Size	Description
elements	(numelem)x8	Row for each element and a column for each of the element's node numbers
nodes	(numnodes)x3	Coordinates for each node
structure	(r)x1	Material distribution, 0 for void 1 for material of structure for given view
map	(numelem)x1	Index positions of each element in the LSF
fig	Handle	Axis handle
E	(r)x8	Solid elements of given view
s	(r)x4x6	Node number faces
p	Handle	Patch handle

plotstructure.m

```

1  function [fig] = plotstructure(elements,nodes,structure,map)
2
3  E=elements(find(structure(map)),:);
4
5  s(:,:,1) = E(:,[1,4,3,2]);
6  s(:,:,2) = E(:,[1,2,6,5]);
7  s(:,:,3) = E(:,[2,3,7,6]);
8  s(:,:,4) = E(:,[3,4,8,7]);
9  s(:,:,5) = E(:,[4,1,5,8]);
10 s(:,:,6) = E(:,[5,6,7,8]);
11
12 for(i=1:6)
13     p=patch('Vertices',nodes,'Faces',s(:,:,i));
14     set(p,'facecolor',[0.9290, 0.6940,
15 0.1250],'edgecolor','black','FaceLighting','gouraud','AmbientStrength',0.5);
16 end
17 camlight left; lighting phong;
18 fig=gca;
19
20 end

```

Published with MATLAB® R2018a

Appendix M: Transparent Border Plotting Code

Variable Name	Size	Description
elements	(numelem)x8	Row for each element, column for each of element's nodes
nodes	(numnodes)x3	Coordinates for each node
struc	(NSx)x(NSy)x(NSz)	Material distribution, 0 for void 1 for material
map	(numelem)x1	Index positions of each element in the LSF
boundary	(r)x1	List of elements that are on the boundary of the geometry
fig	Handle	Axis handle
Eouter	(r)x8	Solid elements of given view that are part of the border
Ecenter	(r)x8	Solid elements of given view that are not part of the border
s	(r)x4x6	Node number faces for interior elements
o	(r)x4x6	Node number faces for border elements
p	Handle	Patch handle

plottrans.m

```

1  function [fig] = plottrans(elements,nodes,struc,map,boundary)
2
3  Eouter=elements(boundary,:);
4  Ecenter=elements(setdiff(find(struc(map)),boundary),:);
5  s(:,:,1) = Ecenter(:, [1,4,3,2]);
6  s(:,:,2) = Ecenter(:, [1,2,6,5]);
7  s(:,:,3) = Ecenter(:, [2,3,7,6]);
8  s(:,:,4) = Ecenter(:, [3,4,8,7]);
9  s(:,:,5) = Ecenter(:, [4,1,5,8]);
10 s(:,:,6) = Ecenter(:, [5,6,7,8]);
11 o(:,:,1) = Eouter(:, [1,4,3,2]);
12 o(:,:,2) = Eouter(:, [1,2,6,5]);
13 o(:,:,3) = Eouter(:, [2,3,7,6]);
14 o(:,:,4) = Eouter(:, [3,4,8,7]);
15 o(:,:,5) = Eouter(:, [4,1,5,8]);
16 o(:,:,6) = Eouter(:, [5,6,7,8]);
17 for(i=1:6)
18     b=patch('Vertices',nodes,'Faces',o(:,:,i));
19     set(b,'facecolor',[0.8485,0.49959,0.17446],'FaceAlpha',0.1,'edgecolor','none');
20 end
21 for(i=1:6)
22     p=patch('Vertices',nodes,'Faces',s(:,:,i));
23     set(p,'facecolor',[0.60551,0.38649,0.69569],'edgecolor','black');
24 end
25 lgd=legend([b,p],'Boundary','Solid')
26 lgd.Position=[0.85,0.85,0.1,0.1];

```

```
27 fig=gca;  
28 end
```

[Published with MATLAB® R2018a](#)

Appendix N: Stress Plotting Code

Variable Name	Size	Description
elements	(numelem)x8	Row for each element and a column for each of the element's node numbers
nodes	(numnodes)x3	Coordinates for each node
structure	(r)x1	Material distribution, 0 for void 1 for material of structure for given view
map	(numelem)x1	Index positions of each element in the LSF
stress	(r)x1	VonMises stress state for each element
yield	Scalar	Yield strength for the given material
fig	Handle	Axis handle
E	(r)x8	Solid elements of given view
s	(r)x4x6	Node number faces
C	(r)x1	Color for each element
p	Handle	Patch handle
cb	Handle	Color bar handle

plotstructure.m

```

1  function [fig] = plotstress(elements,nodes,structure,map,stress,yield)
2
3  E=elements(find(structure(map)),:);
4  s(:, :, 1) = E(:, [1,4,3,2]);
5  s(:, :, 2) = E(:, [1,2,6,5]);
6  s(:, :, 3) = E(:, [2,3,7,6]);
7  s(:, :, 4) = E(:, [3,4,8,7]);
8  s(:, :, 5) = E(:, [4,1,5,8]);
9  s(:, :, 6) = E(:, [5,6,7,8]);
10
11 colormap(jet)
12 caxis([0,yield])
13 c=min(yield,stress(find(structure(map))));%(find(structure(map))/yield);
14 fprintf('Max VonMises Stress: %10.2f Average Stress: %10.2f Yield:
15 %10.2f\n',max(stress),mean(stress),yield);
16 for(i=1:6)
17     p=patch('Vertices',nodes,'Faces',s(:, :, i), 'FaceVertexCData',c, 'FaceColor', 'flat');
18     set(p, 'FaceLighting', 'gouraud', 'AmbientStrength', 0.5);
19 end
20 camlight left; lighting phong;
21 cb=colorbar('Position',[0.95,0.25,0.025,0.65], 'AxisLocation', 'in');
22 cb.Ticks=linspace(0,yield,6);
23 cb.TickLabels=strsplit(num2str(linspace(0,yield,6)), '\newline{yield}');

```

```
24 fig=gca;  
25 end
```

[Published with MATLAB® R2018a](#)

Appendix O: Void and STL Plotting Code

Variable Name	Size	Description
elements	(numelem)x8	Row for each element and a column for each of the element's node numbers
nodes	(numnodes)x3	Coordinates for each node
structure	(r)x1	Material distribution, 0 for void 1 for material of structure for given view
fig	Handle	Axis handle
faces	(#triangles)x3	Outward normal direction components of STL faces
vertices	3x3x(#triangles)	(each node of the triangle)x(x, y, z coordinate of the node)x(each STL triangle)
stl	Handle	Patch handle for STL plot
Esize	1x3	Size of elements in each direction
void	Logical	Opposite of structure
[r,c,p]	(numelem)x1	row, column, page index of void elements respectively
cent	(numvoid)x1	Centroid coordinates for void elements
Vnodes	(r)x3	Node coordinates of void elements
E	(r)x8	Solid elements of given view
s	(r)x4x6	Node number faces
p	Handle	Patch handle for void plot

plotvoid.m

```

1  function [fig] = plotvoid(elements,nodes,structure)
2
3  addpath([pwd, '\MakeMeshSubfunctions'])
4  [faces,vertices] = readSTL(file,'inches');
5
6  vertices=reshape(permute(vertices,[2,1,3]),3,[])';
7  vertices=vertices-min(vertices);
8
9  faces=reshape(1:size(vertices,1)/3,3,[])';
10 stl=patch('Vertices',vertices,'Faces',faces);
11 set(stl,'facecolor',[0.60551,0.38649,0.69569],'FaceAlpha',0.1,'edgecolor','black');
12
13
14 Esize=max(nodes(elements(1,:),:))-min(nodes(elements(1,:),:));
15 void=find(~structure);
16 [r,c,p]=ind2sub(size(structure),void);
17 cent=[r,c,p].*Esize-0.5*Esize;
18 Vnodes=permute(cent,[3,2,1])+0.5.*Esize.*[-1,-1,-1;1,-1,-1;1,1,-1;-1,1,-1;-1,-1,1;1,-
19 1,1;1,1,1;-1,1,1];

```



```

20 Vnodes=reshape(permute(Vnodes,[2,1,3]),3,[])';
21 E=reshape(1:size(Vnodes,1),8,[])';
22
23 s(:, :, 1) = E(:, [1,4,3,2]);
24 s(:, :, 2) = E(:, [1,2,6,5]);
25 s(:, :, 3) = E(:, [2,3,7,6]);
26 s(:, :, 4) = E(:, [3,4,8,7]);
27 s(:, :, 5) = E(:, [4,1,5,8]);
28 s(:, :, 6) = E(:, [5,6,7,8]);
29
30
31 for(i=1:6)
32     p=patch('Vertices',Vnodes,'Faces',s(:, :, i));
33     set(p,'facecolor','yellow','edgecolor','black');
34 end
35
36 fig=gca;
37
38
39
40 end

```

[Published with MATLAB® R2018a](#)