

12-2015

STRUCTURAL OPTIMIZATION USING PARAMETRIC PROGRAMMING METHOD

Krupakaran Ravichandraan
Clemson University, kravich@g.clemson.edu

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

 Part of the [Mechanical Engineering Commons](#)

Recommended Citation

Ravichandraan, Krupakaran, "STRUCTURAL OPTIMIZATION USING PARAMETRIC PROGRAMMING METHOD" (2015).
All Theses. 2264.
https://tigerprints.clemson.edu/all_theses/2264

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.

STRUCTURAL OPTIMIZATION USING PARAMETRIC
PROGRAMMING 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
Krupakaran Ravichandraan
December 2015

Accepted by:
Dr. Georges M. Fadel, Committee Chair
Dr. Margaret Wiecek
Dr. Lonny L. Thompson

ABSTRACT

The Parametric Programming method is investigated to consider its applicability to structural optimization problems. It is used to solve optimization problems that have design variables as implicit functions of some independent input parameter(s). It provides optimal solutions as a parametric function of the input parameter(s) for the entire parameter space of interest. It does not require the detailed discrete optimizations needed at a large number of parameter values as in traditional non-parametric optimization. Parametric programming is widely used in optimal controls, model predictive control, scheduling, process synthesis and material design under uncertainty due to the above mentioned benefits. Its benefits are however, still unexplored in the field of structural optimization. Parametric programming could for instance, be used to aid designers in identifying and optimizing for uncertain loading conditions in complex systems.

The first objective of this thesis is to identify a suitable multi-parametric programming algorithm among the many available ones in the literature to solve structural optimization problems. Once selected, the second goal is to implement the chosen algorithm and solve single parametric and multi-parametric sizing optimization problems, shape optimization problems, and use multi-parametric programming as a multi-objective optimization tool in structural optimization. In this regard, sizing optimization of truss structures and shape optimization of beams for load magnitude and load directions as varying parameters are solved for single and multi-parameter static and/or dynamic load cases. Parametric programming is also used to solve the multi-objective optimization of a honeycomb panel and the results are compared with those from non-parametric

optimization conducted using commercial optimization software. Accuracy of results, and computational time are considered. From these studies, inferences are drawn about the issues and benefits of using parametric programming in structural optimization.

DEDICATION

I would like to dedicate my work to the Almighty, my parents for their continuous support and encouragement in whatever I pursue, my brother, and those who always pray for my best.

ACKNOWLEDGMENTS

I take this opportunity to sincerely thank my advisor Dr. Georges M. Fadel for all his guidance, encouragement, and moral support. He has played a great role in shaping my research acumen, in improving my professional skills and in my improvement as a person. I thank him again for always being there for me and guiding me in the right direction.

I thank Dr. Margaret Wiecek for providing valuable inputs through the entire course of my research work. Her constant motivation and support has been a great driving force for my research. I also thank Dr. Lonny L. Thompson for giving me his valuable inputs and suggestions which have helped shape this research work.

Finally, I thank my two seniors, Dr. Meng Xu and Dr. Jonathon Leverenz. Whenever I needed help, they were always there for me in providing their inputs and feedback.

TABLE OF CONTENTS

	Page
TITLE PAGE	i
ABSTRACT.....	ii
DEDICATION	iv
ACKNOWLEDGMENTS	v
LIST OF TABLES	viii
LIST OF FIGURES	x
NOMENCLATURE	xv
CHAPTER	
1. INTRODUCTION	1
1.1 Motivation.....	1
1.2 Organization of the Thesis	4
1.3 Literature Review.....	5
2. CASE STUDY OF MULTI-PARAMETRIC PROGRAMMING ALGORITHMS ON A BENCHMARK STRUCTURAL OPTIMIZATION PROBLEM	21
2.1 Multi-parametric Toolbox 3.0 and Solution	23
2.2 Multi-parametric Quadratic/Outer Approximation Algorithm and Solution	30
2.3 Approximation Simplex Method Algorithm and Solution	35
2.3 Conclusion	47
3. APPLICATIONS OF PARAMETRIC PROGRAMMING METHOD.....	50
3.1 Sizing Optimization Using Parametric Programming Method	50
3.2 Parametric Programming as a Multi-objective Optimization Tool for Structural Optimization	88
3.3 Shape Optimization Using Parametric Programming Method	103

Table of Contents (Continued)

	Page
4. CONCLUSION AND FUTURE WORK	117
4.1 Conclusion	117
4.2 Issues Pertaining To Parametric Programming and ASM Algorithm	122
4.3 Future Work	123
APPENDICES	125
A: Syntax for Problem Formulation In MPT 3.0	127
B: MATLAB Code for Recursive Linear Approximation	128
C: MATLAB Code for mp-Q/OA Problem Formulation	132
D: MATLAB FEA Solver for Problem 3.1.1.1	152
E: MATLAB FEA Code for Problem 3.1.5.1	155
F: MATLAB Code for ASM Algorithm for Problem 3.1.1.1	161
G: MATLAB Code for Problem 3.1.2.1	172
H: MATLAB Code for Problem 3.1.3.1	182
I: MATLAB FEA Code for Problem 3.1.4.1	188
J: MATLAB ASM Code for Problem 3.1.5.1	195
K: MATLAB ASM Code for Problem 3.2.1.1	198
L: MATLAB Code for Problem 3.3.1	202
M: MATLAB FEA Code to Extract Optimized Beam Shape	212
REFERENCES	214

LIST OF TABLES

Table	Page
2.1 List of optimal function of objectives and decision variables for load range of 100 N to 1000 N for an error tolerance of 0.1	26
2.2 List of optimal value functions $x(p)$ corresponding to the associated simplexes (critical regions)	41
2.3 Comparison of results from ASM with search tool and without search tool	43
3.1 Table of optimal parametric function for x_2 and <i>mass</i> with corresponding parameter intervals (critical regions) for error tolerance of 0.01.....	57
3.2 Comparison of computational performance and accuracy of parametric optimization results via ASM algorithm with Non-parametric optimization	61
3.3 Table of optimal parametric function of x_1 and x_2 with corresponding parameter intervals (critical regions) for error tolerance of 0.105	67
3.4 List of critical regions with their associated vertices for error tolerance of 0.105	68
3.5 Performance of ASM algorithm with different error tolerance values for truss optimization with load directions as parameters	69
3.6 Comparison of performance of ASM algorithm with non-parametric optimization method for truss optimization with load magnitude and direction as parameters.....	72
3.7 List of optimal objective function for the optimization problem of ten bar truss structure	75
3.8 List of vertices that form each critical region for the optimization problem of ten-bar truss structure	77
3.9 Comparison of computational performance of ASM with non-parametric optimization method	78

List of Tables (Continued)

Table	Page
3.10 Table of optimal parametric function of unit cell thickness with corresponding parameter intervals (critical regions) for error tolerance of 12.5%	96
3.11 List of critical regions with their associated vertices for error tolerance of 12.5%	96
3.12 List of variable names used for objective and constraints in multi-parametric programming and ModeFRONTIER	98
3.13 List of five optimal Pareto designs in ModeFRONTIER for twenty five optimization calls	99
3.14 Comparison of relative performance of ASM with ModeFRONTIER	103
3.15 List of optimal nodal height of nodes one and two of the top surface of the cantilever beam	110

LIST OF FIGURES

Figure	Page
2.1 Four bar truss with two concentrated loads [4].....	22
2.2 Comparison of optimal value function $x(p)$ obtained from MPT 3.0 for single point approximation of g_1 with actual results [4].....	25
2.3 The Comparison of optimal value function $x(p)$ obtained from MPT 3.0 for recursive approximation of g_1 with actual results [4]	27
2.4 Flowchart for division of parameter space for recursive linear approximation	28
2.5 Comparison of non-linear g_1 with the linearly approximated g_1 at $p = 1000N$	29
2.6 Flow chart for the working procedure of mp-Q/OA algorithm	32
2.7 Segmentation of parameter space through mp-Q/OA algorithm	33
2.8 Comparison of optimal area function $x(p)$ obtained from MPQ/OA for an error tolerance of 0.1 with actual results [4]	35
2.9 Flow chart for the working procedure of ASM algorithm.....	37
2.10 Segmentation of parameter space through ASM algorithm.....	38
2.11 Comparison of optimal value function $x(p)$ obtained from ASM algorithm for an relative error tolerance of 2% with actual results [4]	40
2.12 Comparison of optimal objective function $f(p)$ obtained from ASM algorithm with non-parametric actual results for a relative error tolerance of 0.1%	40
2.13 Comparison of optimal value function $x(p)$ obtained from ASM-IP for relative error tolerance of 1% with actual [4] results.....	44

List of Figures (Continued)

Figure	Page
2.14 Comparison of optimal value function $x(p)$ obtained from ASM-SQP for relative error tolerance of 1% with non-parametric results for $4100N < p < 5000N$	45
2.15 Comparison of optimal value function $x(p)$ obtained from ASM-SQP for relative error tolerance of 1% with non-parametric results for $1000N < p < 10000N$	46
3.1 Four-bar truss problem from [1] with varying load direction as parameter.....	51
3.2 Contour plot of stress in the four-bar truss obtained using ANSYS R15 APDL	53
3.3 Comparison of vertical nodal deflection found using MATLAB FEA code with that from Ansys APDL.....	53
3.4 Comparison of element axial stress found using MATLAB FEA code with that from Ansys APDL	54
3.5 Plot of optimal decision variable (areas of cross-section) for an allowable error tolerance of 0.1 (10%)	56
3.6 Comparison of optimal areas of cross-section found using ASM algorithm (1% error) with that from traditional non-parametric optimization at thirty discrete values of load angle	56
3.7 Plot of polynomial curve of degree five obtained from non-parametric results using MATLAB curve fitting toolbox.....	58
3.8 Comparison of optimal areas of cross-section found using ASM algorithm (1% error) with that from traditional non-parametric optimization at twelve discrete values of load angle	59
3.9 Comparison of optimal areas of cross-section found using ASM algorithm (1% error) with that from traditional non-parametric optimization at twenty discrete values of load angle	60
3.10 Four-bar truss with two concentrated loads varying in direction	62

List of Figures (Continued)

Figure	Page
3.11 Optimal area of cross-section $x_1(t_1, t_2)$ found using ASM algorithm View 1	65
3.12 View 2 of Figure 3.11	65
3.13 Comparison of optimal areas $x_1(t_1, t_2)$ and $x_2(t_1, t_2)$ found using ASM algorithm	66
3.14 Comparison of optimal objective (mass) found using ASM algorithm with that from non-parametric optimization at 100 discrete points in the parameter space.....	66
3.15 Comparison of surface plot of optimal objective obtained from non-parametric optimization at various optimization calls for t_1 between 0.85 and 1	68
3.16 Four-bar truss with varying load magnitude and load direction	70
3.17 Comparison of optimal parametric objective found by ASM with results from nonparametric optimization at thirty six discrete points in the (t, p) parameter space (View 1)	71
3.18 View 2 of Figure 3.15	72
3.19 Ten-bar truss with two varying load magnitudes and two varying load directions as parameters	73
3.20 Four-bar truss structure with a dynamic load applied at a single node.....	79
3.21 Contour plot of Von-Mises stress for the four-bar truss found in Abaqus CAE 6.14	81
3.22 Plot of external transient load function defined over the time interval at 100 time steps and used in MATLAB FEA analysis	83
3.23 Plot of approximated external transient load function defined in Abaqus CAE 6.14 analysis	84

List of Figures (Continued)

Figure	Page
3.24 Comparison of vertical nodal deflection of the truss structure calculated at the point of application of load from MATLAB FEA code with that from Abaqus CAE 6.14	85
3.25 Plot of optimal area function $x_1(p)$ and $x_2(p)$ obtained from ASM algorithm for an approximation error tolerance of 0.10	85
3.26 Plot of optimal objective function $mass(p)$ obtained from ASM algorithm for an approximation error tolerance of 0.10	86
3.27 Comparison of optimal areas x_1 and x_2 obtained from parametric programming via ASM with that from non-parametric optimization at twenty five discrete points	86
3.28 Four-bar truss structure with two dynamic loads.....	87
3.29 Plot of optimal area function $x_1(p)$ and $x_2(p)$ obtained from ASM algorithm for an approximation error tolerance of 0.10	88
3.30 Regular hexagonal unit cell	91
3.31 Representation of a honeycomb panel with regular hexagonal unit cells	92
3.32 Pareto front of optimal thickness of hexagonal unit cell for an approximation relative error tolerance of 12.5% found using ASM (View 1)	94
3.33 Pareto front of optimal thickness of hexagonal unit cell for an approximation relative error tolerance of 12.5% found using ASM (View 2 showing 4 simplexes).....	94
3.34 Pareto front of optimal weight of the hexagonal unit cell for an approximation relative error tolerance of 12.5% found using ASM	95
3.35 Work flow of the honeycomb panel multi-objective optimization problem setup in ModeFRONTIER.....	97

List of Figures (Continued)

Figure	Page
3.36 Pareto front of optimal unit cell thickness found using MOGA-II in modeFRONTIER for twenty five optimization calls.....	100
3.37 Pareto front of optimal unit cell thickness found using MOGA-II in modeFRONTIER for sixty optimization calls	101
3.38 Cantilever beam geometry with discretization	104
3.39 Cantilever beam with tip load varying in direction	106
3.40 Comparison of maximum tip displacement of cantilever beam obtained from MATLAB FEA with that from Abaqus CAE 6.1.4	107
3.41 Plot of optimal height, h_i of nodes on the top surface of the beam for nodes eleven to fifteen	108
3.42 Plot of optimal height, h_i of nodes on the top surface of the beam for nodes twenty one to twenty five	108
3.43 Optimal shape of the cantilever beam for various load angles, t	111
3.44 Optimal shape of the cantilever beam for load angles in critical regions 1 and 4 (Zoomed view)	112
3.45 Optimal shape of the cantilever beam for load angle of 90°	113
3.46 Comparison of optimal shapes of the beam for symmetric and asymmetric method.....	113
3.47 Cantilever beam with mesh, load and boundary condition (Left). Optimized beam shape with contour plot of shape change magnitude (Right)	114
3.48 Comparison of the optimal beam shapes for $t = 1$ obtained through ASM with that from Optistruct	115

NOMENCLATURE

ρ	Density of the material
E	Young's modulus of the material
L	Length
x	Decision variable
p	parameter
$\sigma_{all,t}$	Maximum allowable tensile stress
$\sigma_{all,c}$	Maximum allowable compressive stress
δ_{all}	Maximum allowable nodal deflection
MPP	Multi-parametric Programming Problems
MPOA	Multi-parametric Optimization Approximation Algorithm
MPQA	Multi-parametric Quadratic Approximation Algorithm
MPQ/OA	Multi-parametric Quadratic/Outer Approximation Algorithm
MPT	Multi-parametric Toolbox
ASM	Approximation Simplex Method
IP	Interior Point
SQP	Sequential Quadratic Programming
TRR	Trust Region Reflective
CR	Critical Region

1 INTRODUCTION

1.1 Motivation

The parametric programming method (parametric optimization) is used to solve optimization problems where the design or decision variables of the optimization problem may be implicit functions of independent parameter(s). In such optimization problems where the optimal solution may continuously change with respect to independent parameters, it is both computationally expensive and time-consuming to use deterministic or traditional non-parametric optimization since optimization needs to be carried out at many discrete values of the parameter(s) space. On the other hand, parametric optimization provides the optimal solution of decision variable and objective as explicit functions of the parameter for the entire parameter(s) space and thus avoids the necessity to do a comprehensive non-parametric optimization.

For example, there are real-world optimization problems in the refinery production planning [15] where the optimal operating conditions of the refinery, such as the optimal flow rates of crude oils, vary with respect to the change in additional maximum allowable production of certain petroleum byproducts such as gasoline and kerosene. With varying time, the maximum allowable production of these byproducts also vary, which leads to a change in the optimal operating condition of the refinery. This problem can be solved as a parametric programming problem wherein the maximum allowable production of the byproducts can be considered as the parameters of the problem. The optimal solution obtained is a function defined over a range of the parameter values and thus does not require

optimization at every discrete value of the parameters. In this problem, production of both gasoline and kerosene affect the optimal solution indicating there are two parameters. Such parametric programming problems are called as Multi-parametric programming problems (Multi-parametric optimization).

Since the parametric optimization technique enables establishing the optimal objective and decision variables as explicit functions of independent parameters, mere function evaluations are required to obtain optimal solution at different values of the input parameters. This is believed to considerably reduce the computational expenses and computational time. Hence, this method has been widely used in optimal control problems, [2] where the optimal input parameters of control vary with time, and process engineering problems under uncertainty, [1, 3, and 14] where the optimal process structure varies with changes in the uncertain parameters like demand for products, price fluctuation, etc. Parametric programming has also been used in the utility plant synthesis, [1] where the optimal combination of using the high pressure boilers, low pressure boilers, high pressure and low pressure turbines is determined as a parametric function of varying parameters like the demand for medium pressure steam, low pressure steam, and electricity.

The potential of parametric programming method in the optimization of thermal systems and in structural optimization still remains unexplored. In structural optimization, the optimal design solution obtained for a given loading condition such as a given load magnitude, load direction, or a combination of both may not remain constant for varying loading conditions. The variation in the optimal solution with respect to the varying loading condition requires traditional non-parametric optimization at a large number of discrete

loading conditions to get the information about the optimal solution for the entire parameter space. This is computationally expensive and time consuming. As the number of independent parameters that affect the optimal solution increases and/or if the relationship between the independent parameters and optimal objective value or decision variable value is highly non-linear, the computational expenses in terms of optimization calls, FEA calls, and computational time may increase multifold.

In design optimization problems where the system is complex and is subjected to highly uncertain loading conditions, it may be difficult to predict the behavior of the system with respect to the loading condition. Thus, it may be highly improbable for a design engineer to identify and optimize the system for the worst loading condition through intuition. A numerical study of the system for a large number of loading conditions and optimization for the worst loading condition among the tested may sound feasible for a single- or two-parameter case. However, as the number of parameters increase, the numerical study and non-parametric optimization at a certain number of discrete loading conditions may not be enough to provide accurate information about the worst loading condition. A detailed analysis will result in enormous amount of computational time and computational expenses [5]. In such cases, the parametric programming method can be of great help. Since the optimal solution obtained through parametric programming is a function defined over the entire parameter space of interest, this method may help in the identification of the worst loading condition with lesser computational time and resources in design problems where it is difficult to identify the worst load through an iterative design

process. This helps to either avoid or reduce the overdesign scenario due to uncertainties in loading conditions and at the same time reduce computational time and expenses.

In this thesis, parametric programming is used to solve structural optimization problems. The motivation of this work is to explore and extend the benefits of parametric programming in structural optimization, identify the issues in parametric programming pertaining to structural optimization and suggest necessary solutions.

1.2 Organization of the Thesis

The first chapter of this thesis mainly deals with the motivation behind the research work in using parametric programming to solve structural optimization problems. It provides an overview of the contributions to the development of parametric programming algorithms and its evolution. Chapter one also introduces the audience to various applications in which the parametric programming has already been used. Chapter two deals with the identification of a suitable multi-parametric programming algorithm to solve structural optimization problems through a case study of a benchmark four-bar truss optimization problem. The relative performance of three state of the art multi-parametric programming algorithms are compared and one among the three is chosen to solve structural optimization problems.

In chapter three, the algorithm chosen as the most suitable for structural optimization is used to solve sizing optimization of a four-bar truss for load direction as a parameter, both load direction and load magnitudes as two parameters and compared with the results from non-parametric optimization. Then, a ten-bar truss weight minimization

problem with four parameters (two load magnitudes and two load directions) is solved and the comparison of computational expenses between parametric programming and non-parametric optimization is discussed. The four-bar truss weight minimization problem is also solved for dynamic loading condition. This is followed by an introduction to the method of using parametric programming to solve multi-objective optimization problems and then a honeycomb multi-objective optimization problem is solved. Finally, shape optimization of a cantilever beam is solved using parametric programming for concentrated tip load varying in direction as a parameter.

Chapter four discusses the inferences that are drawn from the results obtained in chapter three. The issues in the multi-parametric programming pertaining to structural optimization are discussed with suggestions for improvement. This is followed by an overview about the direction for future research and other potential applications of parametric programming method.

1.3 Literature Review

Parametric programming is an optimization method that provides the optimal value function (objective) and the optimization variable (decision variable) as explicit functions of parameters.

The general representation of a parametric optimization problem is as given below,

$$Z(t) = \min_{x,t} f(x,t) \quad (1.1)$$

$$S.t. \ g_i(x,t) \leq 0, i = 1,2, \dots m$$

$$h_j(x, t) = 0, j = 1, 2, \dots, n$$

$$x \in X \subseteq R^p$$

$$t \in \Theta \subseteq R^q$$

$$x_{lb} \leq x \leq x_{ub}$$

where, Z is the optimal objective parametric function, f is the objective function, X is a subset of R^p , Θ is a subset of R^q , g is the vector of inequality constraints and h is the vector of equality constraints. x_{lb} and x_{ub} are lower and upper bounds respectively for the decision variable, while m and n are the number of inequality constraints and equality constraints respectively. The optimal solution $Z_i(t)$ and $x_i(t)$ are the optimal value functions and decision variable functions defined over the regions (i) of the parameter space. Such segments in the parameter space will be herein called as critical intervals or critical regions.

Initial contributions in developing multiparametric linear programming were done by Gal and Nedoma [11]. They developed a method to find the optimal parametric solution space for multiparametric linear problems that either have linear constraints as a function of the parameter or coefficients of the linear objective as a function of the parameters. However, this method is applicable only if the parameter is in the right hand side of the constraint equation or the objective function. The general representations of the problem that could be solved using this method and the form of the optimal solution found by this method is given below.

$$\max_{x,t} z = c^T x \quad (1.2)$$

$$S. t \ Ax = b(t)$$

$$x \geq 0$$

where, $b(t) = b^* + Ft$, c^T is a coefficient vector and b^* is a constant vector. Or

problems of the form

$$\max_{x,t} z = c^T(t)x \quad (1.3)$$

$$S. t \ Ax = b$$

$$x \geq 0$$

where, $c^T(t) = c^* + Ft$, b and c^* are constant vectors. A and F are matrices of constant coefficients, t is the parameter vector.

Let CR^i be the region corresponding to every optimal basis X^i for the problem defined in Equation (1.3) $\forall t \in CR^i$. The method developed by Gal and Nedoma [11] provides a region, $R = \cup_i CR^i$ such that CR^i that forms the R space does not overlap for the defined multi-parametric linear programming problem. This is achieved by using an algorithm that determines a series of nodes that are connected to form CR^i and thus R .

Over the past three decades, a considerable amount of research work has been conducted in extending the ability of parametric programming algorithms to solve optimization problems with non-linear objective and non-linear constraints [6, 13, and 15] in addition to only linear constraints and linear objective. Further, work has been conducted in the development of algorithms to solve problems with integer decision variables and

optimization problems with a mixture of continuous and integer decision variables [12]. Recent research work has concentrated on improving the algorithms to decrease the computational expenses, computational time and increase the accuracy of the optimal parametric results when compared to the previously available algorithms [5, 7, and 12].

The developments in parametric optimization techniques and algorithms were mainly driven due to its potential applications in industrial engineering and operations research such as process synthesis under uncertainty [1 and 6], scheduling and planning under uncertainty [6, 14 and 24], material design under uncertainty [3] etc. Equal amount of research has been conducted in the areas pertaining to on-line optimization via off-line parametric optimization [15, 29], model predictive control [13, 16], robust model predictive control [19], etc.

Acevedo et al. [6] developed an algorithm to solve parametric mixed-integer nonlinear programming (pMINLP) problems applied to process synthesis problems under uncertainty. The general representation of parametric mixed-integer nonlinear programming problem is given below,

$$\begin{aligned}
 Z(t) &= \min_{y,x,t} d^T y + f(x,t) \\
 \text{S.t. } & Ey + g(x,t) \leq b + Ft \\
 &x \in X \subseteq R^p \\
 &t \in \Theta \subseteq R^q \\
 &y \in \{0,1\}^m \\
 &x_{lb} \leq x \leq x_{ub}
 \end{aligned} \tag{1.4}$$

where, E and F are constant matrices, b and d are constant vectors, y is a vector of 0-1 binary variables, x is a vector of continuous variables, f is continuously differentiable and convex scalar function, g is a constraint vector, and t is the parameter. x_{lb} and x_{ub} are lower and upper bounds respectively for the decision variable.

The algorithm finds the critical regions of the parameter, optimal integer configuration associated with the critical interval, and the corresponding optimal parametric objective function. This is achieved through an iterative process of solving parametric nonlinear programming (pNLP) subproblems at a number of vertices in the parameter space to obtain a linear optimal profile and then a parametric mixed integer linear programming (pMILP) master problem is solved to get another set of optimal integer solutions for the entire parameter space until no more better integer solution other than the previously found one is determined. A process synthesis problem involving mixing of two materials through four processes to get the final product has been solved as a profit maximization example problem. Here, demand for the final product is considered as the uncertain parameter and each process is considered as one integer variable. The optimal solution consists of three optimal integer configurations (combinations of processes) and associated optimal objective corresponding to the respective critical intervals of demand.

Later, Acevado et al. [1] developed an algorithm to solve multiparametric Mixed-Integer Linear Programming (mpMILP) process engineering problems under uncertainty. This algorithm is capable of handling more than one parameter unlike the single parametric Mixed-Integer Nonlinear Programming (pMINLP) algorithm [6] but is restricted to only linear constraints and objective. The method uses a branch and bound procedure to find the

optimal solution. They have solved three example problems and a power plant utility synthesis optimization problem is one among them. The objective of the problem is to minimize the cost function of the plant for 30 constraints, 8 integer variables and four uncertain demand parameters related to the plant equipment and process operating conditions.

The mpMILP algorithm [7] was used by Dua et al. [3] to solve a material design problem under uncertainty. In this article, a polymer design problem is solved with an objective to minimize the maximum deviation of polymer properties such as water absorption and glass transition temperature from the targets defined for the associated constraints. The uncertainty in the properties that define the water absorption and glass transition temperature of polymer are considered as the parameters while the molecular group to be chosen and the number of molecules required in each group is considered as the integer variable of the mpMILP problem. The optimal solution provides the optimal molecular structure and the optimal minimum deviation of the polymer properties corresponding to the critical regions of the parameter space.

Further, Dua et al. [12] developed a multiparametric outer approximation (mp-OA) algorithm for the solutions of multiparametric mixed-integer nonlinear programming (mpMINLP) problems. This algorithm can also be used for continuous variable problems and for objective and constraints that are convex in both ' x ' and ' t ' instead of the parameters ' t ' being present only in the right hand side of the constraints. Dominguez et al [2] have made a survey of the state of the art algorithms that are currently used to solve multiparametric nonlinear programming problems. In that survey, they have briefly

explained about the mp-OA algorithm pertaining to continuous variable problems instead of mixed-integer problems. A brief introduction to the mp-OA used for continuous variable problems is discussed below using eq. (1.1). An initial feasible point $[x^*, t^*]$ is obtained by considering parameter ‘ t ’ as a free variable and then solving Equation (1.1).

$$\hat{Z}(t) = \min_{x,t} f(x^*, t^*) + \nabla_{x,t} f(x^*, t^*)((x, t) - (x^*, t^*)) \quad (1.5)$$

$$\begin{aligned} S.t. \quad & g(x^*, t^*) + \nabla_{x,t} g(x^*, t^*)((x, t) - (x^*, t^*)) \leq 0, i \\ & = 1, 2, \dots m \end{aligned}$$

$$h(x^*, t^*) + \nabla_{x,t} h(x^*, t^*)((x, t) - (x^*, t^*)) = 0, j = 1, 2, \dots n$$

$$x \in X \subseteq R^p$$

$$t \in \Theta \subseteq R^q$$

where $\nabla_{x,t} f$ is the gradient of objective function, f .

The multiparametric linear programming problem (mp-LP) represented in Equation. (1.5) is obtained by first order Taylor’s series approximation of the objective and constraints at $[x^*, t^*]$ as a point of approximation. The solution to Equation (1.5) represents a lower bound to eq. (1.1). The difference or error between $Z(t) - \hat{Z}(t)$ at the vertices of the lower and upper bound of the parameter space is found. Another mp-LP approximation is carried out at the vertex where the error exceeds a certain limit ε . Through comparison of $\hat{Z}(t)$ obtained initially with that obtained later, a sub-partitioning of the critical region can be achieved [1]. Finally, solving for both Equation (1.1) and Equation (1.5) at the vertices of all of the critical regions generated through sub-partitioning and find the corresponding error $Z(t) - \hat{Z}(t)$. If the error is below ε for all of the critical regions then

the algorithm terminates, if not, the algorithm generates another mp-LP and carries out sub-partitioning for those critical regions where the error exceeds ε until convergence is met.

Later, Johansen [13] developed a multiparametric Quadratic Approximation algorithm (mpQA), which uses quadratic approximation of the objective and linear approximation of the constraints to develop the multiparametric Quadratic Problem (mp-QP) as an approximate problem for the original multiparametric Nonlinear Problem (mp-NLP). MPQA was initially developed to solve nonlinear model predictive control problems but it can also be extended to various multiparametric nonlinear programming problems. This algorithm was again revisited by Dominguez et al. [2] where they fixed the infeasibility issues in the parameter space with accumulated linearization of the nonlinear constraints and thus it is also called multiparametric Quadriac/Outer approximation algorithm (mp-Q/OA). A brief introduction to the working procedure of the algorithm is discussed below using mp-NLP problem given in Equation (1).

By considering parameter ' t ' as a free variable and solving the problem in Equation (1), initial feasible point $[x^*, t^*]$ is obtained. Then, a quadratic programming problem is developed using the quadratic approximation of the objective and linear approximation of the non-linear constraints.

$$\hat{Z}(t) = \min_{x,t} f(x^*, t^*) + \nabla_{x,t} f(x^*, t^*)((x, t) - (x^*, t^*)) \quad (1.6)$$

$$+ \frac{1}{2} ((x, t) - (x^*, t^*))^T \nabla_{x,t}^2 f(x^*, t^*) ((x, t) - (x^*, t^*))$$

$$S.t. \ g(x^*, t^*) + \nabla_{x,t} g(x^*, t^*)((x, t) - (x^*, t^*)) \leq 0, i = 1, 2, \dots m$$

$$h(x^*, t^*) + \nabla_{x,t} h(x^*, t^*)((x, t) - (x^*, t^*)) = 0, j = 1, 2, \dots, n$$

$$x \in X \subseteq R^p$$

$$t \in \Theta \subseteq R^q$$

where $\nabla_{x,t} f$ and $\nabla_{x,t}^2 f$ are the gradient and Hessian of objective function, f respectively.

Equation (1.1) is solved at each vertex ' t_v^i ' of all the critical regions generated. If some vertices of the critical regions are infeasible due to nonlinearity of the constraints, a feasible point as given in [2] is obtained with single linearization or through accumulation of all linearizations. For all the feasible vertices identified, the parametric solution is evaluated by solving problem (6). The error between $Z(t_v^i)$ and $\hat{Z}(t_v^i)$ at all vertices of each critical region is computed. If the error exceeds the set error tolerance ' ϵ ', then the center point of the critical region corresponding to the vertex (t_v^{i*}) must be found where error exceeds tolerance and reformulate problem (6) about ' t_v^{i*} ' and the same process must be continued as discussed from the beginning of this paragraph. Else, the algorithm terminates. The flow chart for the algorithm can be found in chapter 3, section 3.2.

In 2006, Bemporad et al. [5] developed an algorithm for approximating the optimal objective function and optimal decision variable functions as functions of parameters for multi-parametric convex nonlinear programming problems. They referred to this algorithm as Approximate Multi-parametric algorithm (AM). The procedure of the algorithm is discussed below with the problem given in Equation (1) as an example. In this method, the parameter space is initially defined by a single simplex for a single parameter problem, two simplexes for a two parameter problem, and so on. The optimal decision variable value x^i

and corresponding objective value Z^i is found at each and every vertex, ' t^i ' that forms the initial simplex ' S '. The approximate functions $\hat{x}(t)$ of the optimal decision variable and $\hat{Z}(t)$ of the optimal objective are obtained through the linear interpolation of the optimal decision variable and objective function values, x^i and Z^i respectively found at the vertices of the simplex.

$$\hat{x}(t) = XK^{-1} \begin{bmatrix} 1 \\ t \end{bmatrix} \quad (1.7)$$

$$\hat{Z}(t) = zK^{-1} \begin{bmatrix} 1 \\ t \end{bmatrix}$$

where,

$$X = [x^0, x^1, \dots, x^q]$$

$$K = \begin{bmatrix} 1 & 1 & \dots & 1 \\ t^0 & t^1 & \dots & t^q \end{bmatrix}$$

$$z = [z^0, z^1, \dots, z^q]$$

The maximum error of approximation δ^S over the simplex S is obtained by solving the optimization problem given below.

$$\delta^S = \max_{x,t} \hat{Z}(t) - Z(t) \quad (1.8)$$

$$S.t. \ g_i(x, t) \leq 0, i = 1, 2, \dots, m$$

$$h_j(x, t) = 0, j = 1, 2, \dots, n$$

$$K^{-1} \begin{bmatrix} 1 \\ t \end{bmatrix} \geq 0$$

$$x \in X \subseteq R^p$$

$$t \in \Theta \subseteq R^q$$

If $\delta^S < \epsilon$ (specified tolerance), then the algorithm terminates. If not, initial simplex S , is split at its center ' t^{mid} ' into two smaller simplexes in a single-parameter problem and three simplexes in a two-parameter problem and so on. The approximate function of optimal objective $\hat{Z}(t)$ and decision variable $\hat{x}(t)$ are obtained again for all of the newly found simplexes. The maximum error of approximation δ^S is found at the center of each newly formed simplex. The segmentation continues until δ^S for all the simplexes are well within the defined tolerance, ϵ . Since the algorithm uses simplexes for approximation within the parameter space, AM will be herein referred to as Approximation Simplex Multi-parametric (ASM) algorithm/method throughout this thesis. The flow chart for ASM can be found in Chapter 3, section 3.3

Dominguez et al. [2] conducted a literature review on the recent advances in multi-parametric nonlinear programming. In this work, they have compared the relative performance, computational time and computational resources of multi-parametric outer approximation (mp-OA) algorithm, multi-parametric quadratic/outer approximation (mp-Q/OA) algorithm, approximate multi-parametric (AM) algorithm, and geometric vertex search (GVS) algorithm with a numerical example.

Dominguez et al. have observed that the number of critical regions increases drastically for mp-OA as the degree of nonlinearity associated with the objective function and constraints increase. Even cubic and quadratic terms in the objective and/or constraint can increase the partitions in the parameter space to meet the given approximation error

tolerance. Moreover, mp-OA underestimates the optimization variable in all critical regions, which affects the approximation of optimal decision variables and the objective function. On the other hand, mp-Q/OA uses a second order approximation and thus requires less partitions or critical regions. However, the ability of mp-Q/OA to accurately partition the critical region based on active sets of the problem may decrease with increasing non-linearity in the constraints. Both, mp-OA and mp-Q/OA need to solve a certain number of multi-parametric linear problems (mp-LP) and multi-parametric quadratic problems (mp-QP) respectively in addition to the NLP's. The AM algorithm uses linear interpolation of optimal values found at the vertices and thus does need not to solve any mp-LP's or mp-QP's in addition to the NLP's. This reduces its computational time to a great extent. However, the approximations in AM are not as tight as those in mp-OA.

The other observations that the authors have made are that, among the algorithms discussed above, mp-OA solves the maximum number of non-linear problems (NLP's or optimization runs) to find the optimal parametric solution within given error tolerances, followed by AM which solves a meagre one fourth of NLP's that mp-OA solves and then the mp-QA which requires one eighth of that required by AM. However, GVS solves the largest number of NLP's which is twice as that of mp-OA. The behavior of computational time also follows that of the number of NLP's solved.

Generally model predictive control (MPC) based optimization problems are solved repetitively when the plant is on-line, at equal intervals of time to optimally control the dynamic response of the plant as a reaction to variation of the state variables with varying time. Pistikopoulos et al. [15] have used multi-parametric quadratic programming to solve

model predictive control (MPC) optimization problems with quadratic objective and linear constraints wherein the state variables of the control problem are considered as the parameters. The input variables are considered as the decision variables while the response or the output variable is considered as the objective of the multi-parametric optimization problem. The optimal solution obtained from multi-parametric quadratic programming is a parametric function defined over the entire time interval of interest. This way, repetitive on-line optimization at equal time intervals is replaced with a single off-line multi-parametric quadratic optimization followed by simple function evaluations of the optimal parametric function to obtain the optimal response at different time instances. This was observed to reduce the large number of repetitive non-parametric MPC based optimization at equal time intervals.

There has been research in the area of multi-objective optimization using multi-parametric programming. Papalexandri et al. [17] have used multi-parametric mixed-integer nonlinear programming to solve multi-objective optimization problems involving discrete decision variables. The general representation of a multi-objective optimization problem with continuous and discrete decision variables is as given below in Equation (1.9). They have reformulated the problem in Equation (1.9) into a multi-parametric programming problem by introducing the objective functions other than the first objective function ($f_1(x, y)$) as constraints of the multi-parametric programming problem and the targets for those objectives reformulated as constraints are considered as parameters of the problem as given in Equation (1.10).

$$\begin{aligned}
Z &= \min_{x,y} (f_1(x,y), f_2(x,y), \dots, f_n(x,y)) \\
\text{s.t } g_i(x,y) &\leq 0, i = 1, 2, \dots, q \\
x &\in X \subseteq R^p \\
y &\in Y \subseteq \{0,1\}^m \\
x_{lb} &< x < x_{ub}
\end{aligned} \tag{1.9}$$

where $f_n(x,y)$ is the n^{th} objective function, x is a vector of continuous decision variables, y is a vector of discrete decision variables, and $g_i(x,y)$ is a vector of constraints. x_{lb} and x_{ub} are the lower and upper bounds respectively for x .

$$\begin{aligned}
Z(t) &= \min_{x,y} f_1(x,y) \\
\text{s.t } f_j(x,y) &\leq t^j, j = 2, \dots, n \\
g_i(x,y) &\leq 0, i = 1, 2, \dots, q \\
x &\in X \subseteq R^p \\
y &\in Y \subseteq \{0,1\}^m \\
x_{lb} &< x < x_{ub} \\
t_{lb}^j &< t^j < t_{ub}^j
\end{aligned} \tag{1.10}$$

where, t^j represents the constraint value defined for the objective function $f_j(x)$. t_{lb}^j and t_{ub}^j are the lower and upper bounds for t^j .

Though research work has been conducted in using multi-parametric programming in multi-objective optimization, no considerable work has been done in the area of structural optimization.

The literature review helped to identify and understand the state of the art multi-parametric programming algorithms, their relative performances, benefits, and disadvantages. The literature review showed that there has been a lot of work done in the applications of parametric programming to optimal controls, model predictive control, process synthesis, and scheduling under uncertainty in operations research but there is a lack of research in the applications of parametric programming to structural optimization. There has been considerable research work on stochastic optimization applied to structural optimization. It is somewhat similar to parametric programming wherein the optimization is carried out for a small interval of the probability distribution of uncertain parameters [12]. However, the solution obtained is a single robust value obtained for the entire small interval of the parameter and it does not provide optimal parametric function defined over the entire parameter space unlike parametric programming.

It is found that there is no prior work done in realizing the potential benefits of parametric programming in structural optimization. The reduction in computational expenses and computational time that the parametric programming has to offer will benefit a wide variety of structural optimization problems for uncertain loading conditions in sizing optimization and shape optimization for static and dynamic loads. Similarly, there is no work in the use of multi-parametric programming in structural optimization. The literature review makes it evident that there is a great scope and need for research in structural optimization using parametric programming.

Firstly, a suitable multi-parametric programming algorithm must be identified to solve structural optimization problems that generally have non-linear constraints and

objective. Secondly, the chosen algorithm has to be used to solve sizing optimization, shape optimization and multi-objective optimization problems. Finally, inferences have to be drawn about the advantages and disadvantages of using parametric programming in structural optimization.

2 CASE STUDY OF MULTI-PARAMETRIC PROGRAMMING ALGORITHMS ON A BENCHMARK STRUCTURAL OPTIMIZATION PROBLEM

The relative performance, strengths and weaknesses of various multi-parametric programming algorithms in solving structural optimization problems that generally have non-linear constraints are determined by solving a benchmark four-bar truss optimization problem subjected to concentrated static loads [4]. The accuracy of the optimal results, the computational time and certain other factors such as the ability to connect to an FEA solver and the degree of non-linearity that the algorithm can handle are collectively considered in choosing the most suitable multi-parametric programming algorithm.

The objective of the problem in [4] is to minimize the mass of a four-bar truss with axial stress constraints on the truss members and vertical nodal deflection constraints as given in Equation (2.1) for constant static loads p and $2p$ as shown in Figure 2.1.

$$m = \min_{x_1, x_2} \rho l (3x_1 + \sqrt{3}x_2) \quad (2.1)$$

Deflection

$$s.t. \ g_1: \frac{6pl}{E} \left(\frac{3}{x_1} + \frac{\sqrt{3}}{x_2} \right) - \delta \leq 0$$

Tensile

$$g_2: -10^{-3} \frac{x_1 E}{p} + 5.73 \leq 0$$

Stress

$$g_3: -10^{-3} \frac{x_2 E}{p} + 7.17 \leq 0$$

Compressive

Stress

where $\delta = 3 \times 10^{-3}l$ is the limit for the vertical deflection at the point of application of load $2p$, E is the Elastic modulus of the material.

The original problem described in Equation (2.1) is modified to be able to solve it as a parametric programming problem. The load magnitude p is considered to be the varying parameter, the areas of the cross-sections of the truss members are considered as the decision variables. The rearranged constraints can be found in Equation (2.2).

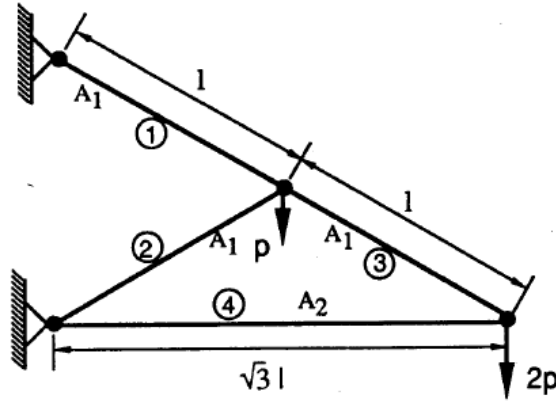


Figure 2.1: Four-bar truss with two concentrated loads [4]

Let $A_2 = x_1$, and $A_1 = x_2$ be the cross-sectional areas of the truss members as shown in Figure 2.1 and the decision variables of the problem. Let, $E = 200 \text{ MPa}$ be the Modulus of Elasticity of the material and the length, $l = 1 \text{ m}$.

$$\begin{aligned}
 \min_{x_1, x_2, p} \text{ mass} &= \rho l (3x_1 + \sqrt{3}x_2) \\
 \text{s.t. } g_1 &: \frac{-Ex_1x_2}{2000 p(3x_2 + \sqrt{3}x_1)} + 1 \leq 0
 \end{aligned} \tag{2.2}$$

$$g_2: \frac{-Ex_1}{5730p} + 1 \leq 0$$

$$g_3: \frac{-Ex_2}{7170p} + 1 \leq 0$$

$$100N < p < 1000N$$

$$10^{-5}m^2 < x_i < 10^{-1}m^2, i = 1,2$$

2.1 Multi-Parametric Toolbox 3.0 (MPT 3.0) & Solution

Multi-Parametric Toolbox 3.0 [20] is an open source toolbox that uses multi-parametric Linear Programming (mpLP) and multi-parametric Quadratic Objective Programming (mpQOP) algorithms to find optimal solutions of the multi-parametric programming problems. mpLP is only capable of solving problems with linear objective and constraints while mpQOP can solve problems with linear constraints and quadratic or linear objective. Moreover, MPT can solve problems if the parameter is present in the right hand side of the constraint equation. In other words, if a parameter is present in the objective function then it must be either in the quadratic form by itself or in a linear form if it is multiplied with a linear decision variable.

Thus, MPT 3.0 *cannot* handle non-linear constraints. Any non-linear constraint must be linearized before solving the problem using the toolbox. Since constraint g_1 is the only nonlinear constraint in Equation (2.2), it is linearized using the first order Multivariable Taylor's series approximation.

2.1.1 Problem Formulation

The nonlinear constraint g_1 is linearly approximated with the load value which is the average of the lower and upper bounds of the parameter p defined in Equation (2.2) and the optimal areas of cross-section x_1^* and x_2^* corresponding to p as the point of approximation (p^*, x_1^*, x_2^*) . The gradients of g_1 can be found in Equation (2.3).

$$\begin{aligned}\frac{\partial g_1}{\partial x_1} &= \frac{2000P(3x_2 + \sqrt{3}x_1)(-Ex_2) + 2000\sqrt{3}P(Ex_1x_2)}{[2000P(3x_2 + \sqrt{3}x_1)]^2} \\ \frac{\partial g_1}{\partial x_2} &= \frac{2000P(3x_2 + \sqrt{3}x_1)(-Ex_1) + 6000P(Ex_1x_2)}{[2000P(3x_2 + \sqrt{3}x_1)]^2} \\ \frac{\partial g_1}{\partial p} &= \frac{Ex_1x_2}{[2000(3x_2 + \sqrt{3}x_1)](p^*)^2}\end{aligned}\tag{2.3}$$

From Multivariable Taylor's series approximation, the first order approximation is as given below in Equation (2.4). This equation is rearranged to get the linearized constraint in the form $Cx_1 + Dx_2 \leq J + Kp$. The MPT 3.0 requires the user to input the constraints in the rearranged form.

$$\begin{aligned}\hat{g}_1(x_1, x_2, p) &= g_1(x_1^*, x_2^*, p^*) + \frac{\partial g_1}{\partial x_1}(x_1 - x_1^*) + \frac{\partial g_1}{\partial x_2}(x_2 - x_2^*) + \frac{\partial g_1}{\partial p}(p - p^*) \\ \hat{g}_1: \frac{\partial g_1}{\partial x_1}x_1 + \frac{\partial g_1}{\partial x_2}x_2 &\leq \left\{ -g_1 + \frac{\partial g_1}{\partial x_1}x_1^* + \frac{\partial g_1}{\partial x_2}x_2^* + \frac{\partial g_1}{\partial p}p^* \right\} \\ &\quad + \left\{ \frac{-x_1x_2E}{[2000(3x_2 + \sqrt{3}x_1)](p^*)^2} \right\}\end{aligned}\tag{2.4}$$

$$\hat{g}_1: Cx_1 + Dx_2 \leq J + Kp$$

where p^* is the point of approximation, x_1^* and x_2^* are the optimal decision variable values at p^* . $\hat{g}_1(x_1, x_2, p)$ is a linear approximation of the non-linear constraint $g_1(x_1, x_2, p)$.

In addition to the constraints g_2 , and g_3 in Equation (2.2), linearized constraint \hat{g}_1 in Equation (2.4) is used to solve the optimization problem through MPT 3.0. The syntax for the problem formulation to solve through MPT 3.0 is given in Appendix A.

2.1.2 Results and Discussion

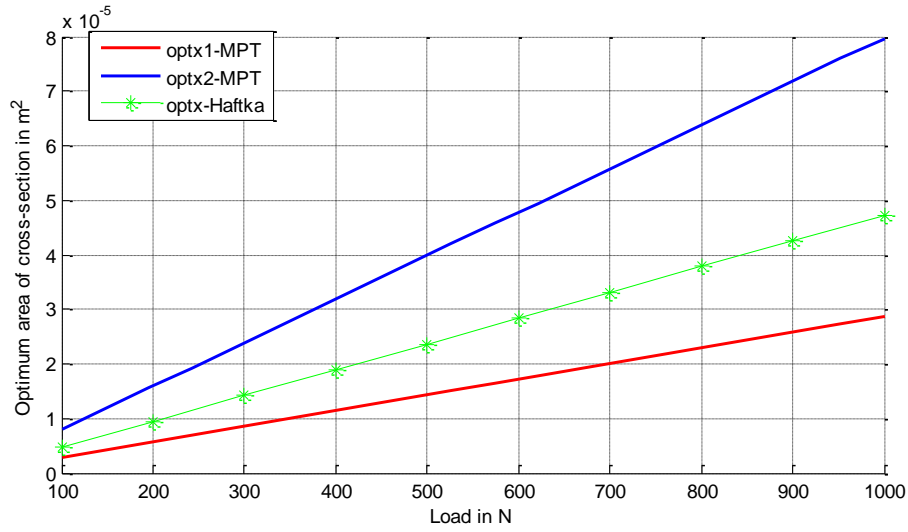


Figure 2.2: Comparison of optimal value function $x(p)$ obtained from MPT 3.0 for single point approximation of g_1 with actual results [4]

According to the results found in [4], for a given load magnitude p , the optimal values for both x_1 and x_2 must be the same. Figure 2.2 shows the comparison of optimal parametric results obtained by linear approximation of g_1 at $p = 450$ N as one single point

of approximation using MPT 3.0 with the results from [4] evaluated at 10 discrete load values from 100N to 1000N.

Table 2.1: List of optimal function of objectives and decision variables for load range of 100 N to 1000 N for an error tolerance of 0.1

Function	Optimal function
$f(p)$	$1.8 \times 10^{-3}p + 1 \times 10^{-7}$
$x_1(p)$	$2.9 \times 10^{-8}p + 6.1 \times 10^{-21}$
$x_2(p)$	$8 \times 10^{-8}p + 1.7 \times 10^{-8}$

In Figure 2.2, the green line with asterisk represents the actual results [4] obtained at discrete values of the load in the parameter space, while the red and blue lines represent optimal x_1 and x_2 determined by MPT 3.0. It is clearly evident that the parametric results calculated using MPT 3.0 deviate from the actual results and the approximation error is as high as 66.6% at load value of 1000N.

This is assumed to be due to the error induced by the approximation of constraint g_1 at one single parameter value in a large interval of the parameter range. Hence, a recursive linear approximation code is developed in MATLAB R2014 [56] to divide the parameter space ($100N < p < 1000N$) into smaller segments such that the error in approximation of g_1 is well within an acceptable error tolerance.

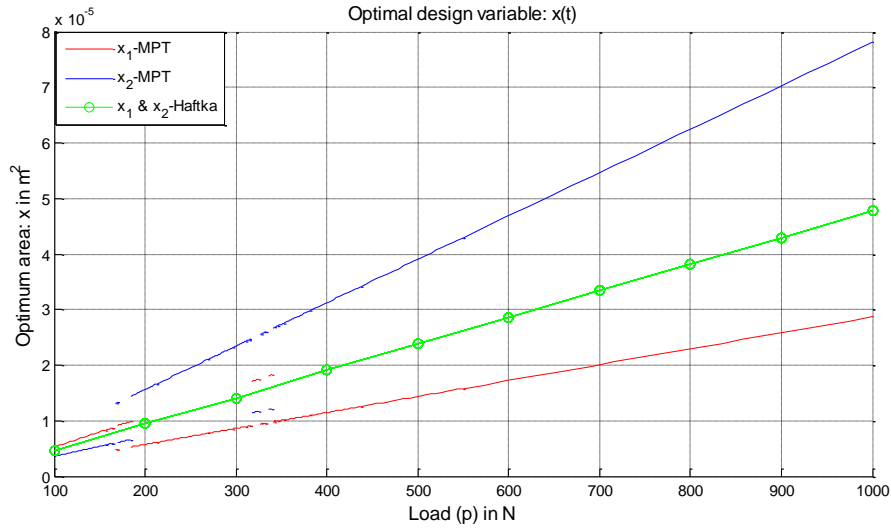


Figure 2.3: The comparison of optimal value function $x(p)$ obtained from MPT 3.0 for recursive approximation of g_1 with actual results [4]

Then, the parametric optimization is carried out in MPT 3.0 for each segment through linear approximation of the constraint g_1 at the midpoint of the corresponding segment. The MATLAB code for this method can be found in Appendix B. The flow chart for the parametric optimization in MPT 3.0 using the recursive linear approximation approach is given in Figure 2.4

In the flow chart below, p_{crib} and p_{crub} represent the lower and upper bound for each segment in the parameter space identified using the algorithm developed, for which the error in linear approximation of the non-linear constraint g_1 is well within the tolerance defined. ' i ' is the counter for the number of remaining linear approximations and ' j ' is the counter for the number of segments in the parameter space (pairs of parameter bounds) for which the MPT 3.0 parametric optimization must be carried out. ' tol ' is the error tolerance defined for the linear approximation of constraint g_1 and is given 0.05 (5%).

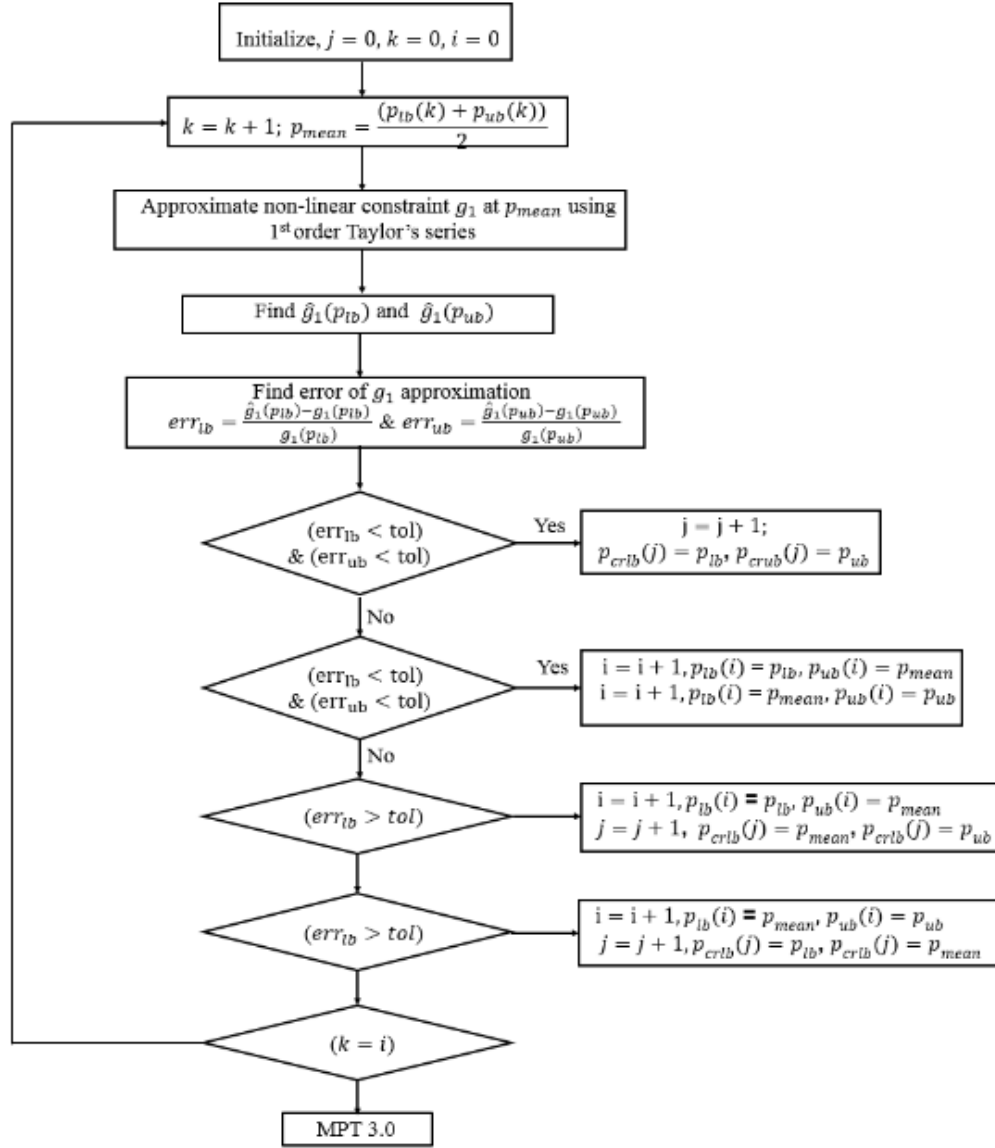


Figure 2.4: Flowchart for the division of parameter space for recursive linear approximation

This resulted in 21 segments within the load range of 100N to 1000N. The comparison of optimal decision variable obtained from MPT 3.0 after recursive linear approximation with the actual results [4] is given in Figure 2.3. It can be seen that the MPT

3.0 results for recursive linear approximation has improved in the load region between 100N and 200N when compared to the single point approximation. In addition the deviation of the parametric results from [4] is marginally lesser than that for single point approximation in the load range between 200N and 1000N. However, the decrement of the error in optimal parametric results is a meagre 4.1% for an increment of 21 MPT 3.0 optimization calls due to recursive linear approximation of g_1 .

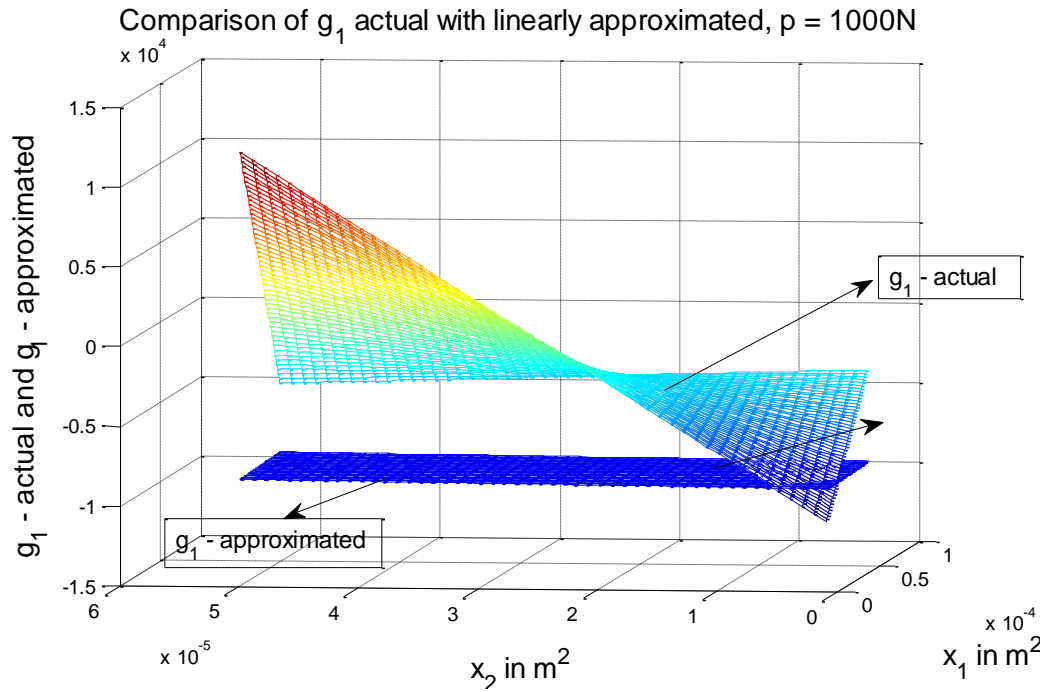


Figure 2.5: Comparison of nonlinear g_1 with the linearly approximated g_1 at $p = 1000N$

This is mainly attributed to the large error associated with the approximation of non-linear constraint g_1 . Even if the load range for which the approximation is carried out is smaller, the error between the approximated g_1 and the actual g_1 is large. This error persists irrespective of the load value at which the constraint g_1 is approximated. The comparison of actual g_1 with the approximated g_1 is shown in Figure 2.5.

2.2 Multi-parametric Quadratic/Outer Approximation Algorithm (MPQ/OA) &

Solution

The MPQ/OA is a state of the art multi-parametric programming algorithm developed by Dominguez et al. [7]. As discussed in the literature review, this algorithm uses the second order (quadratic) approximation of the objective function and first order approximation of the non-linear constraints to construct an approximated multi-parametric quadratic programming problem and then finds the optimal parametric function of the objective and decision variables. A MATLAB code for this algorithm was developed by Leverenz [21]. Since MPT 3.0 can solve optimization problems with quadratic objective and linear constraints, Leverenz coupled this MATLAB code to MPT 3.0 to find the optimal parametric solution.

The flow chart of general working procedure of MPQ/OA algorithm is given in Figure 2.6. In the figure, let $Z(t)$ be the optimization problem, $f(\mathbf{x}, t)$ be the objective, $g(\mathbf{x}, t)$ be the linear or nonlinear constraints, \mathbf{x} and t be the vector of decision variables, and parameter respectively. \mathbf{x}_{lb} and \mathbf{x}_{ub} be the vector of lower and upper bounds for the decision variables respectively while t_{lb} and t_{ub} are the lower and upper bounds for the parameter respectively.

$$\begin{aligned} Z(t) &= \min_{\mathbf{x}, t} f(\mathbf{x}, t) \\ S. t. \quad &g(\mathbf{x}, t) \leq 0 \\ &\mathbf{x}_{lb} \leq \mathbf{x} \leq \mathbf{x}_{ub} \end{aligned} \tag{2.5}$$

$$t_{lb} \leq t \leq t_{ub}$$

Let $\hat{Z}(t)$ be the approximated optimization problem, obtained from the second order approximation $\hat{f}(\mathbf{x}, t)$ of the objective function $f(\mathbf{x}, t)$ and first order approximation $\hat{g}(\mathbf{x}, t)$ of the nonlinear constraint $g(\mathbf{x}, t)$.

$$\hat{Z}(t) = \min_{\mathbf{x}, t} \hat{f}(\mathbf{x}, t) \quad (2.6)$$

$$S. t. \hat{g}(\mathbf{x}, t) \leq 0, i = 1, 2, \dots m$$

where,

$$\hat{f}(\mathbf{x}, t) = f(\mathbf{x}^*, t^*) + \nabla_{\mathbf{x}, t} f(\mathbf{x}^*, t^*)((\mathbf{x}, t) - (\mathbf{x}^*, t^*)) + \frac{1}{2}((\mathbf{x}, t)$$

$$- (\mathbf{x}^*, t^*)) \nabla_{\mathbf{x}, t}^2 f(\mathbf{x}^*, t^*)((\mathbf{x}, t) - (\mathbf{x}^*, t^*))$$

$$\hat{g}(\mathbf{x}, t) = g(\mathbf{x}^*, t^*) + \nabla_{\mathbf{x}, t} g(\mathbf{x}^*, t^*)((\mathbf{x}, t) - (\mathbf{x}^*, t^*))$$

In Equation (2.6), $\nabla_{\mathbf{x}, t} f(\mathbf{x}^*, t^*)$ and $\frac{1}{2} \nabla_{\mathbf{x}, t}^2 f(\mathbf{x}^*, t^*)$ are respectively the Jacobian (J_{mass}) and Hessian (H_{mass}) of the objective function $f(\mathbf{x}, t)$ evaluated at the point of approximation (\mathbf{x}^*, t^*) . Similarly, $\nabla_{\mathbf{x}, t} g(\mathbf{x}^*, t^*)$ is the Jacobian (J_g) for the non-linear constraint evaluated at the point of approximation (\mathbf{x}^*, t^*) .

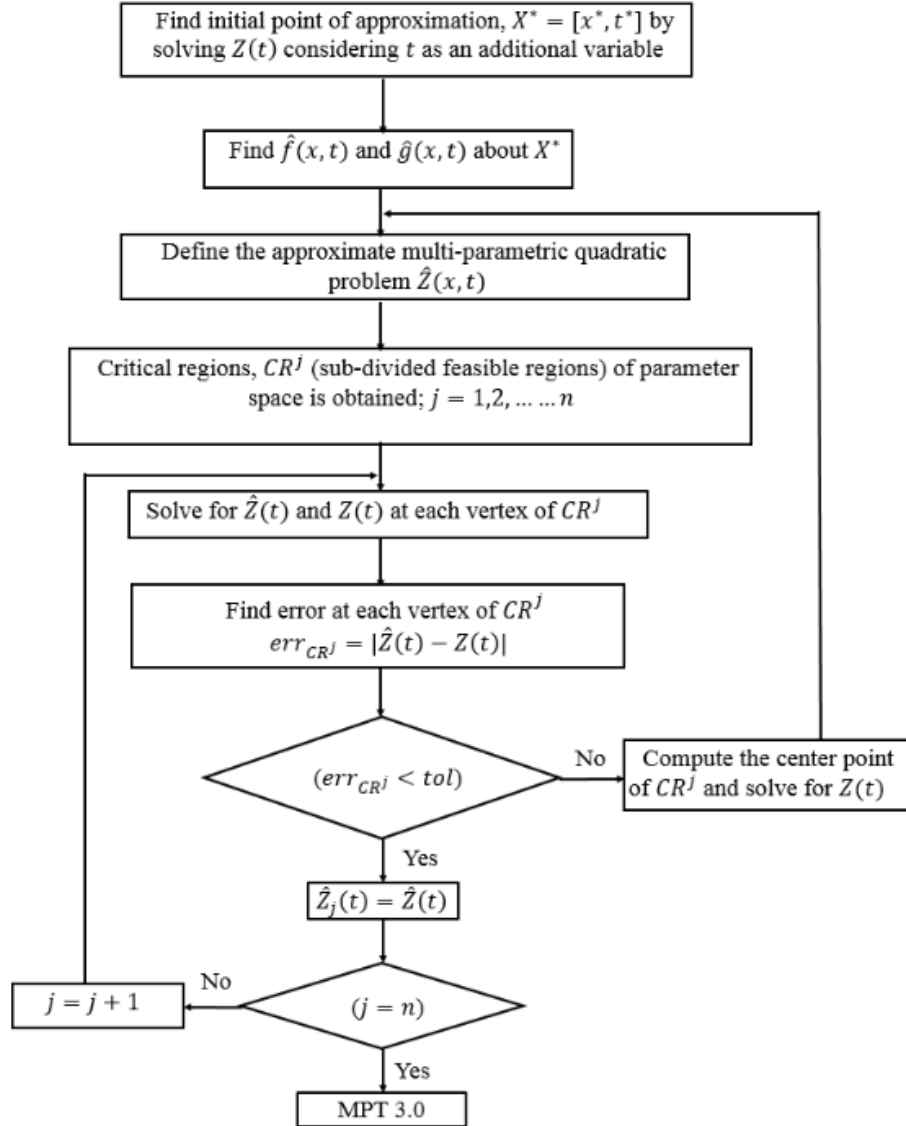


Figure 2.6: Flow chart for the working procedure of mp-Q/OA algorithm

The Jacobians and the Hessians must be determined by the user and passed on to MPQ/OA solver. For better understanding, a pictorial representation of the method of segmentation of the parameter space through MPQ/OA is given in Figure 2.7.

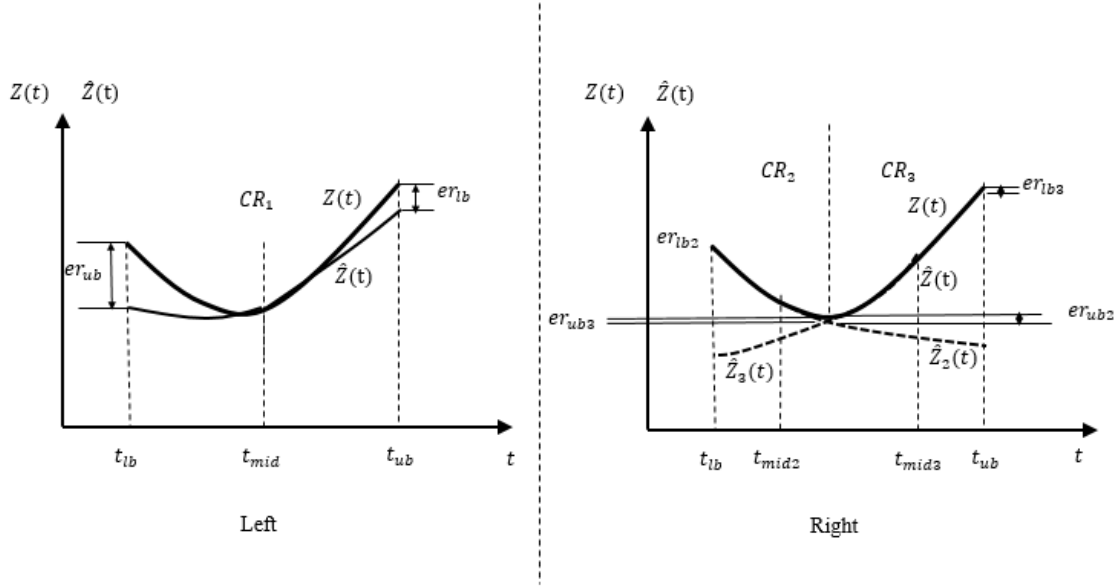


Figure 2.7: Segmentation of parameter space through mp-Q/OA algorithm

2.2.1 Problem Formulation

In the four-bar truss optimization problem (Equation (2.2)) under consideration, the objective is linear and thus the Hessian is zero. The constraint g_1 is the only non-linear constraint and the Jacobian is given in Equation (2.7) while the Jacobians for the linear constraints g_2 and g_3 are zero. The MATLAB code for MPQ/QA can be found in Appendix C.

$$J_{g1} = \begin{bmatrix} \frac{2000P(3x_2 + \sqrt{3}x_1)(-Ex_2) + 2000\sqrt{3}P(Ex_1x_2)}{[2000P(3x_2 + \sqrt{3}x_1)]^2} \\ \frac{2000P(3x_2 + \sqrt{3}x_1)(-Ex_1) + 6000P(Ex_1x_2)}{[2000P(3x_2 + \sqrt{3}x_1)]^2} \\ \frac{Ex_1x_2}{[2000(3x_2 + \sqrt{3}x_1)](p^*)^2} \end{bmatrix} \quad (2.7)$$

$$J_{g2} = 0$$

$$J_{g3} = 0$$

$$J_{mass} = 0$$

$$H_{mass} = 0$$

Let J_{g1} , J_{g2} , and J_{g3} be the Jacobian of constraints g_1 , g_2 and g_3 respectively. Let J_{mass} be the Jacobian and H_{mass} be the Hessian of the objective function given in Equation (2.2). This information is provided to the MPQ/OA algorithm for the construction of the approximated multi-parametric quadratic problem for the critical regions found in the parameter space as shown in Equation (2.6). The quadratic problem and corresponding critical regions are provided to the MPT 3.0 to find the optimal parametric solution as shown in the flowchart given in Figure 2.6.

2.2.2 Results and Discussion

For an error tolerance of 0.1 (10%), MPQ/OA found the optimal solution without any segmentation of the parameter space. In other words the approximated quadratic problem was well within the error tolerance for the entire parameter space. From Figure 2.8, it can be seen that the optimal parametric function for x_1 and x_2 obtained from MPQ/OA does not concur well with the actual results [4]. The maximum error for x_1 and x_2 within the parameter space was found to be 13.69% and -24.13% in comparison with the expected results. The error is believed to be due to its coupling with the Multi-Parametric Toolbox 3.0 because the linearized non-linear constraints from MPQ/OA are

passed on to MPT 3.0 to find the optimal parametric solution. When the error tolerance was reduced to $1e-2$, MPQ/OA did not converge and thus could not find an optimal solution. The reason for this behavior can be attributed to the error induced due to the linear approximation of the non-linear constraint function of several variables.

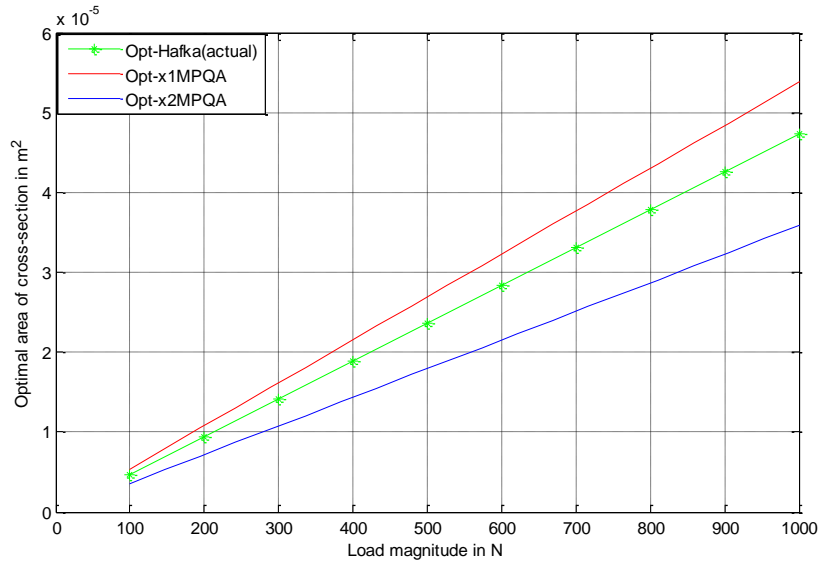


Figure 2.8: Comparison of optimal area function $x(p)$ obtained from MPQ/OA for an error tolerance of 0.1 with actual results [4]

2.3 Approximation Simplex Method (ASM) & Solution

The Approximation Simplex Method algorithm was developed by Bemporad and Filippi [5]. As mentioned in the literature review, ASM is a recursive approximation algorithm which provides optimal decision variables and objective as explicit functions of parameters over simplicial partitions of subset of feasible parameters [5]. This algorithm is implemented in MATLAB by Leverenz [21], who coupled this ASM MATLAB code with

MPT 3.0 to create the polyhedral simplexes (critical regions) based on the given parameter space.

In the flow chart given in Figure 2.9, $Z(t)$ is the objective function given in Equation (2.5), $\hat{x}(t)$ and $\hat{Z}(t)$ are the approximated optimal functions of the decision variables and the objective obtained through linear interpolation of the optimal decision variable values and objective values found at all the vertices of a given simplex. *err* is the maximum error in approximation of the objective function within the simplex for which the approximated optimal function is determined. For better understanding, the segmentation of the parameter space through the ASM algorithm and its working procedure is clearly shown in Figure 2.10.

$$\hat{x}(t) = XK^{-1} \begin{bmatrix} 1 \\ t \end{bmatrix} \quad (2.8)$$

$$\hat{Z}(t) = zK^{-1} \begin{bmatrix} 1 \\ t \end{bmatrix}$$

$$X = [x^0, x^1, \dots, x^q]$$

$$K = \begin{bmatrix} 1 & 1 & \dots & 1 \\ t^0 & t^1 & \dots & t^q \end{bmatrix}$$

$$z = [z^0, z^1, \dots, z^q]$$

$$err = \max_{x,t} \hat{Z}(t) - Z(t) \quad (2.9)$$

$$S. t. \ g(x, t) \leq 0$$

$$K^{-1} \begin{bmatrix} 1 \\ t \end{bmatrix} \geq 0$$

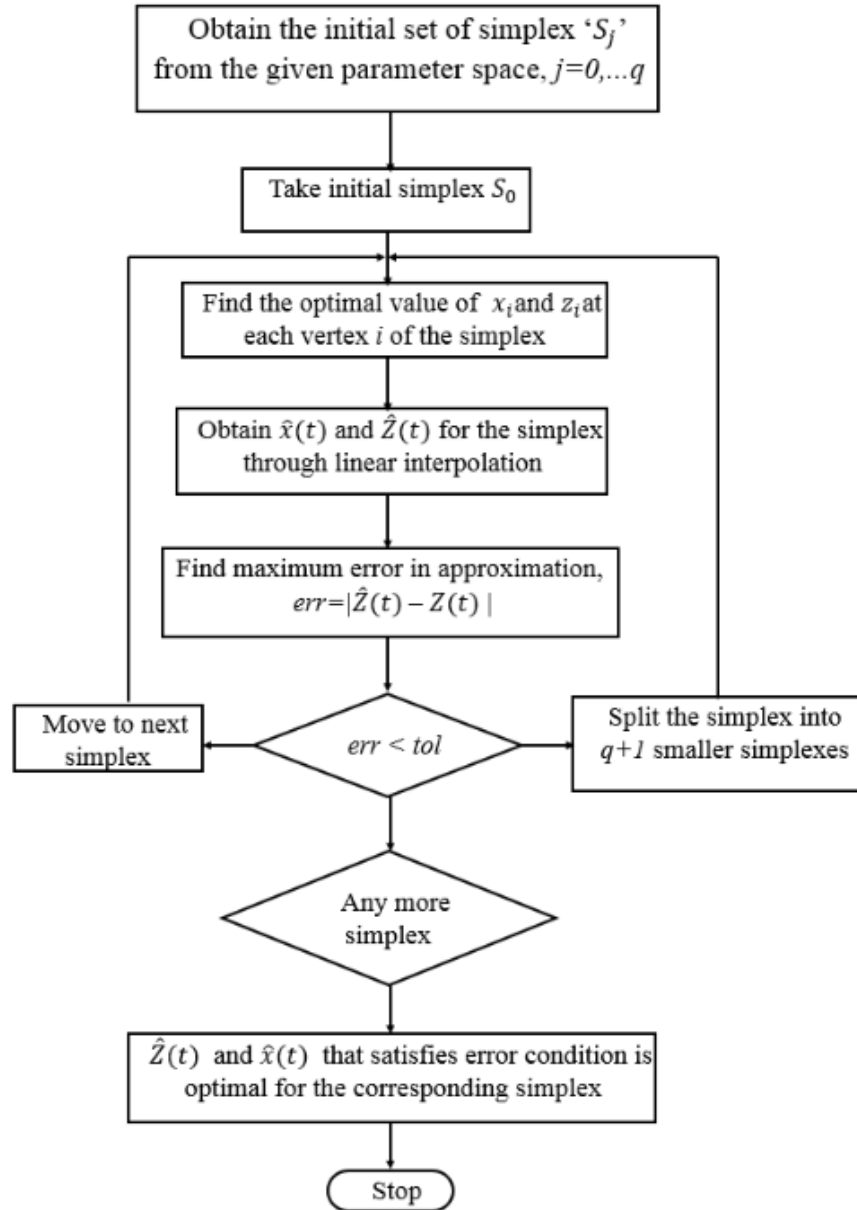


Figure 2.9: Flow chart for the working procedure of ASM algorithm

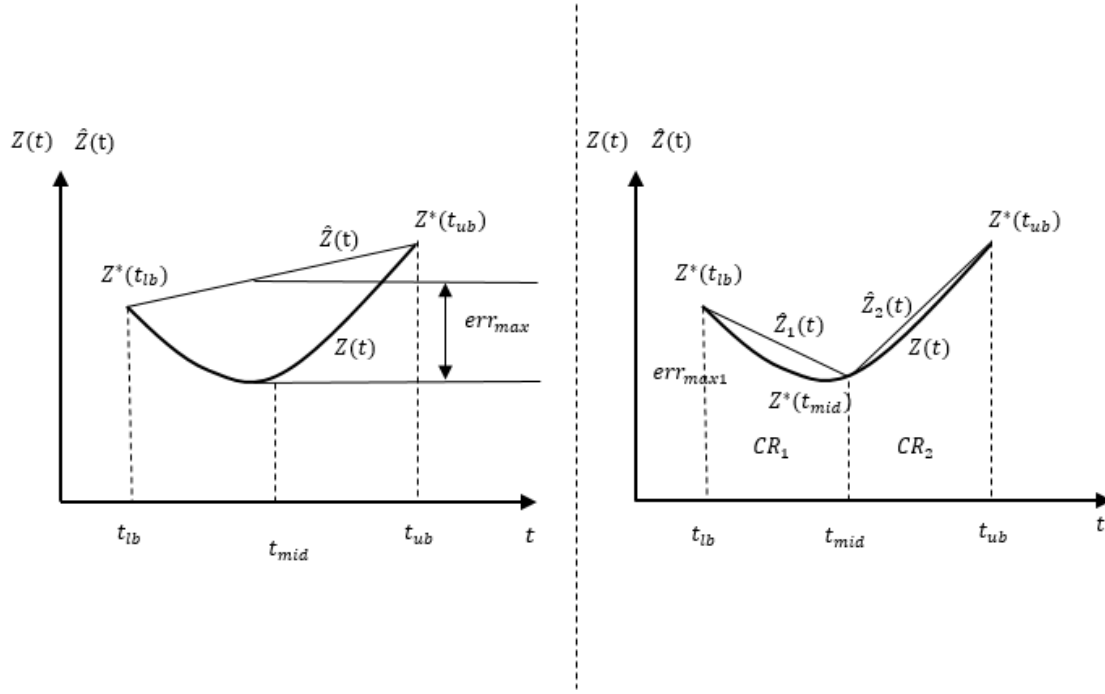


Figure 2.10: Segmentation of parameter space through ASM algorithm

The algorithm is slightly modified in the current implementation wherein the error in approximation evaluated at the center of the simplex (critical region) is used for comparison with the error tolerance instead of the method used in Equation (2.9).

This modification improves the results obtained using the ASM algorithm on the problems considered and reduces the number of segmentations. The modified absolute error function is $err = |Z(t^{mid}) - \hat{Z}(t^{mid})|$. Instead of using the absolute error value, the relative error value given by $err_{rel} = |(Z(t^{mid}) - \hat{Z}(t^{mid}))/Z(t^{mid})|$ is used in the current work to maintain consistency in the definition of error for all the problems irrespective of the order of objective value with which the algorithm deals with.

In addition to that, another termination criterion is added to the ASM algorithm in the MATLAB code implemented by Leverenz [21]. If the error in approximation is not below the set error tolerance, the segmentation of the parameter space continues until the distance between the end vertices and center of the newly formed simplex (critical region) is lesser than 10^{-4} times the value of the parameter(s). Thereafter the ASM algorithm terminates the segmentation process and no solution is returned.

2.3.1 Problem formulation

In the Approximation Simplex Method (ASM), the objective and constraints of the optimization problem can be directly input as analytical expressions or evaluated through a finite element analysis solver unlike the need to represent the equations in a particular format as in MPT 3.0 or the need to evaluate and input Jacobian and Hessians of the non-linear constraints and objective as in the MPQ/OA algorithm. The multi-parametric programming problem definition for the four bar truss represented in Equation (2.2) holds good for the ASM algorithm.

2.3.2 Results & Discussion

The modified ASM, found the optimal parametric solution using a meagre3 optimization calls and one simplex (critical region). In Figure 2.11, the solid blue and solid red lines represent the optimal parametric functions of areas of cross-section x_1 and x_2 calculated by ASM and the solid green line indicates the optimal areas of the cross-sections evaluated at discrete parameter values represented by green asterisks (actual results [4]).

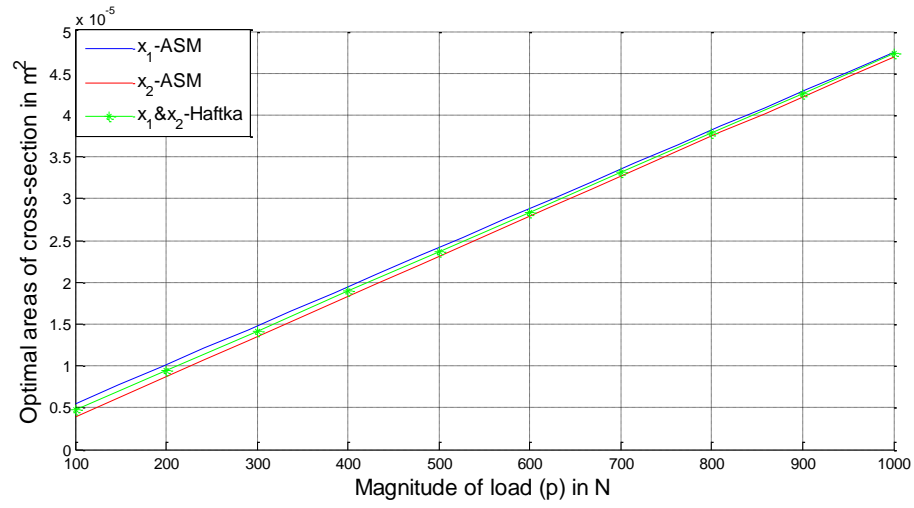


Figure 2.11: Comparison of optimal value function $x(p)$ obtained from ASM algorithm for an relative error tolerance of 2% with actual results [4]

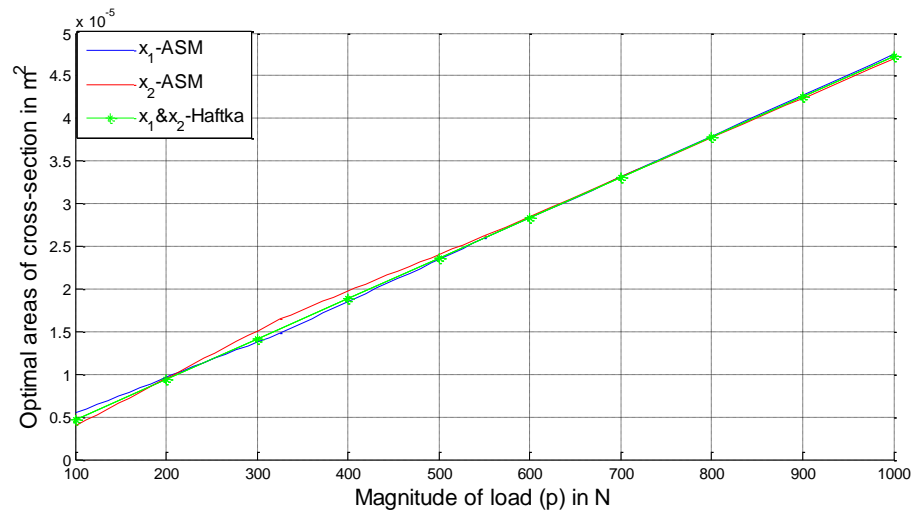


Figure 2.12: Comparison of optimal objective function $f(p)$ obtained from ASM algorithm with non-parametric actual results for a relative error tolerance of 0.1%

From Figure 2.11, it is clearly evident that the parametric results obtained from the ASM-SQP algorithm concur well with the actual [4] results and the accuracy of the results is significantly high when compared to the other algorithms previously used. As the error tolerance was decreased to 0.1%, the number of segmentations increased to 3 and the number of optimization calls increased to 9. The improvement in the results is predominant in the parameter region between 380 N and 1000 N as shown in Figure 2.12.

Table 2.2: List of optimal value functions $x(p)$ corresponding to the associated simplexes (critical regions)

Simplex/ Critical Region	$p_{low} \leq p \leq p_{up}$	$x_1(p)$	$x_2(p)$
1	$100 \leq p \leq 325$	$10^{-7}(0.4149p - 13.21)$	$10^{-7}(0.5575p - 16.39)$
2	$325 \leq p \leq 550$	$10^{-7}(0.4945p - 12.631)$	$10^{-7}(0.4314p + 24.595)$
3	$550 \leq p \leq 1000$	$10^{-7}(0.4794p - 4.372)$	$10^{-7}(0.4624p - 7.564)$

The optimal parametric function of the decision variables $x_1(p)$ and $x_2(p)$ for the 3 simplexes (critical regions) for the optimization of the truss structure with an error tolerance of 0.05% is given in Table 2.2. Evaluating the problem using these optimal

parametric functions provides the optimal values for the decision variables and the objective for every value of the parameter p in the parameter space.

In the original MATLAB code for the Approximation Simplex Method (ASM) developed by Leverenz, the optimizer is called to determine the optimal objective value and decision variable values at all the vertices for each and every simplex. This leads to unnecessary additional optimization calls in recalculation of the optimal values at vertices which are common to the simplex that is currently solved and simplexes that were previously solved. Similarly, as new simplexes are formed due to the segmentation, the optimizer is used to recalculate the optimal values at the vertices that were previously solved as part of the unsegmented simplex.

To eliminate the unnecessary optimization calls involved in finding the optimal values at vertices already solved, the algorithm is modified to store the optimal objective and optimal decision variable values corresponding to the vertices solved. When a new simplex (critical region) is solved, the storage area is first checked to determine if the vertex (parameter value) was already solved. If yes, the optimal objective and decision variable values are retrieved from the corresponding parameter value and used. If not, then the optimizer is called to determine the optimal values at that vertex and then stores the data for later use. This reduces the number of optimization calls by 50% to 60% based on the error tolerance chosen.

Table 2.3: Comparison of results from ASM with search tool and without search tool

	Original ASM without search tool	Modified ASM with search tool	Improvement due to search tool
Optimization calls	15	8	46.67%
FEA calls	1098	166	84.88%

However, this modification is incorporated only for single parametric optimization problems and is not implemented in multi-parametric optimization problems. The search process becomes cumbersome for optimization problems with more than two parameters and time for searching outweighs the time for additional optimization calls with the increase in the number of optimization calls.

The ASM algorithm uses the MATLAB fmincon nonlinear programming solver to find the optimal solution at the discrete values of the parameter. The fmincon solver incorporates four different algorithms namely, Sequential Quadratic Programming (SQP) algorithm, Interior-point (IP) algorithm, Trust-Region Reflective (TRR) and Active-Set (AS). In order to understand the effect of using different fmincon algorithms, the effect of size of parameter space, and the error tolerance used on the accuracy of the optimal results, the four bar truss optimization problem given in Equation (2.2) is solved with a combination of two of the fmincon algorithms and various approximation relative error tolerances for different sizes of the parameter space and different parameter values. Two of the four fmincon algorithms, TRR and Active-set are not used in the experimentation

because TRR requires gradients and active-set takes large steps and may affect the accuracy of results.

2.3.3 Experimentation on the effects of various fmincon algorithms, parameter size and parameter values on the accuracy of results

The results shown in Figure 2.11 and Figure 2.12 were obtained using fmincon-SQP algorithm. Following SQP, the interior-point algorithm was used for different values of error tolerance. The results are discussed below in detail.

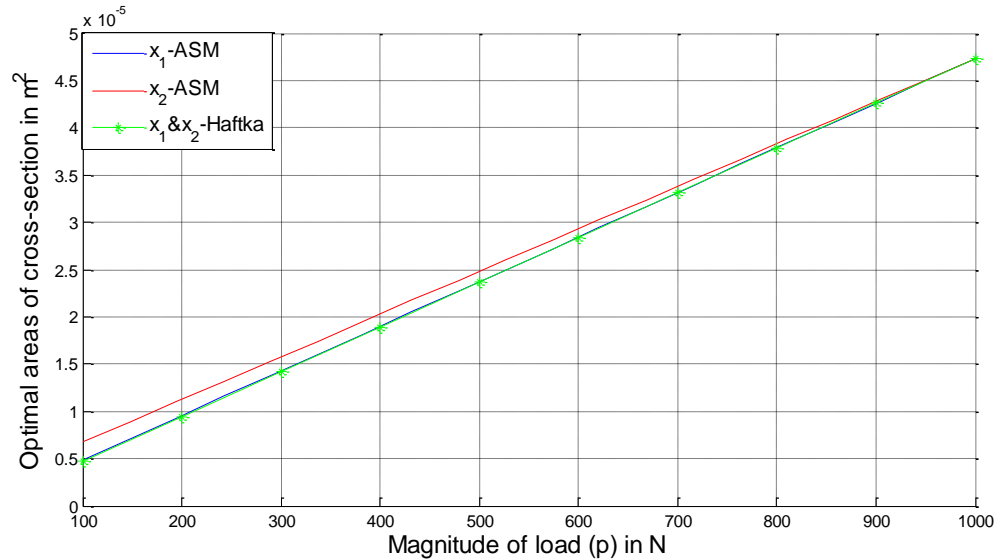


Figure 2.13: Comparison of optimal value function $x(p)$ obtained from ASM-IP for relative error tolerance of 1% with actual [4] results

For an error tolerance of 1%, the optimal value function $x_1(p)$ agrees well with the actual results from the literature [4]. However, the optimal value function $x_2(p)$ deviates from the actual by more and the deviation decreases as the load value increases to 1000 N where the error is absolute zero as shown in Figure 2.13. When absolute error tolerance is

increased to $1e-3$, the ASM goes into endless iterations of simplex segmentations and cannot find the optimal parametric solution.

The results using the SQP algorithm in Figure 2.11 show that the error in calculating the parametric results $x_1(p)$ and $x_2(p)$ remains the same, maintaining consistency with the actual behavior of optimal x_1 and x_2 . The optimal parametric $x_1(p)$ found using the interior-point algorithm concurs well with the actual results. However, the optimal parametric $x_2(p)$ found using the interior-point algorithm does not concur well with the actual results. Trust region reflective algorithm was unable to find parametric solutions within the tolerances set even for an error tolerance as high as 0.5. Thus, it is evident that the SQP performs better than the other available fmincon algorithm tested.

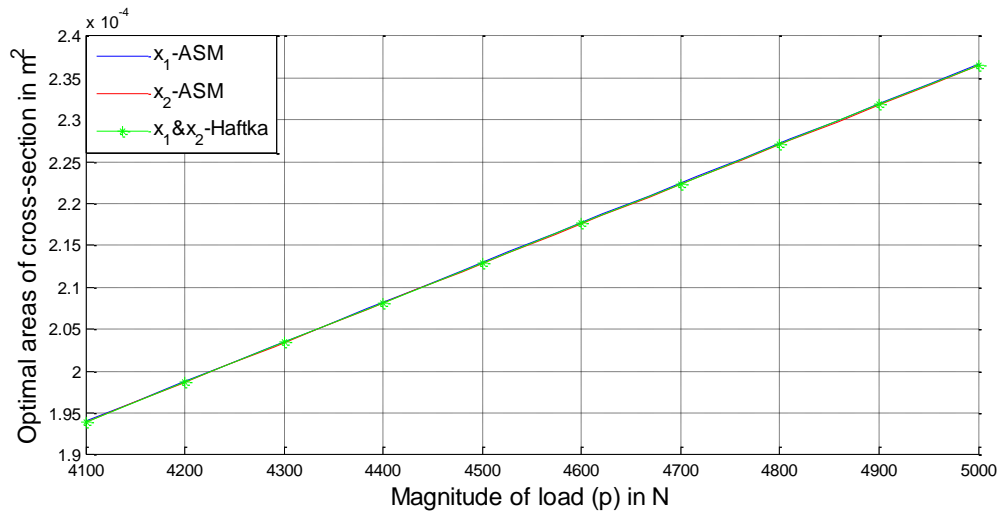


Figure 2.14: Comparison of optimal value function $x(p)$ obtained from ASM-SQP for relative error tolerance of 1% with non-parametric results for $4100N < p < 5000N$

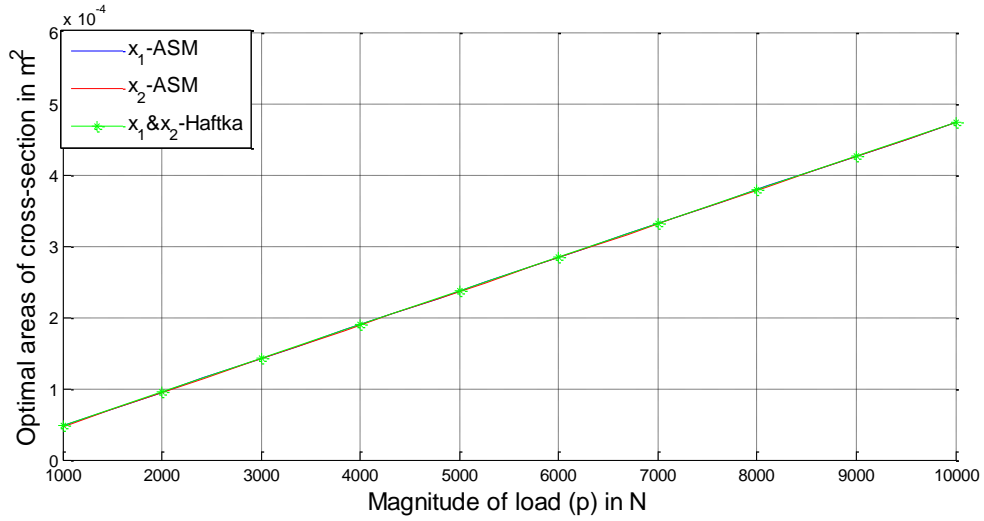


Figure 2.15: Comparison of optimal value function $x(p)$ obtained from ASM-SQP for relative error tolerance of 1% with non-parametric results for $1000N < p < 10000N$

Next, the sensitivity of the ASM to the magnitude of the parameter values is investigated. For the same size of the parameter space of $900N$, the lower bound was set as $4000N$ from the previous $100N$ and the upper bound was set as $5000N$ instead of $1000N$. The optimal parametric results obtained for an error tolerance of 1% using the fmincon-SQP algorithm agrees well with the non-parametric results evaluated at 10 equally distributed load values in the parameter space (actual results [4]) as shown in Figure 2.13. Similarly, the results for $3000N < p < 3900N$ concurs well with the non-parametric results. This shows that the ASM is sensitive to the magnitude of the parameter values which correspond to optimal decision variable values that are lesser than 10^{-5} . Scaling was done to offset this behavior and it was observed that the results improved slightly but still a negligible amount of error was present. Thus, it is necessary to scale the parameter

value to reduce the sensitivity of the ASM algorithm and get accurate optimal parametric solution.

The effect of size of the parameter space on the accuracy of the optimal solution was also determined. The parametric optimization was carried out for a parameter size (load range) of 9000 N using fmincon-SQP algorithm for an error tolerance of 1%. The optimal parametric results are in good agreement with the non-parametric actual results [4] as shown in Figure 2.14. The ASM algorithm is able to find optimal solution for parameter size as high as 20000 N and thereafter the algorithm does not converge and crashes. To solve for problems with parameter sizes greater than 20000, the parameter must be scaled to have a lesser value and then passed to the ASM algorithm. This avoids the non-convergence due to the large parameter size.

Setting the bounds for the decision variables is also an important criterion which decides whether the solution would be determined or not. If the range of bound is too large, the solver does not form the initial simplex and no approximation is made. For example, when the load range is greater than 20000 N , the ASM algorithm does not form the initial polyhedra. When the parameter range is reduced to 19000 N , the ASM algorithm converged and found optimal parametric solution with one critical region.

2.4 Conclusion

From the case study, the pros and cons of MPT 3.0, MPQ/OA algorithm and ASM algorithm were identified. It was observed that the major disadvantage of MPT 3.0 is that it can solve convex problems with only linear constraints, linear objectives or quadratic

objectives. Any non-linear constraint has to be linearized and the error in the optimal results when compared to the actual on the problem evaluated is excessive (about 60%). The linearization process becomes cumbersome as the number of non-linear constraints increases. In addition, both the equality and inequality constraints of the optimization problem should only be defined as shown in Appendix A. Above all, MPT 3.0 is like a black box. The type of MATLAB optimization solver that is used, the tolerance for the error, the approximation approach that is used within the MPT 3.0, all are unknown. In structural optimization, the optimizer is in general coupled to a Finite Element Analysis solver to solve complex problems which do not have simple analytical expressions for the constraints and objective. However, as each constraint equation must be input as an expression in MPT 3.0., it is not feasible to connect MPT 3.0 with an FEA solver. Due to the above mentioned reasons, MPT in its current form may not be an efficient tool to solve structural problems that need an FEA solver, and for structural optimization problems that are convex non-linear or non-convex.

Similarly in MPQ/OA, the constraints and objective have to be input as analytical expressions and thus a Finite Element Analysis solver cannot be coupled to this optimizer. It is possible to get the equations for constraint and objective for complex problems that do not have analytical expressions by generating the response surface for both the objective and constraints. However, it requires to solve a finite element analysis problem at various vertices in the parameter space which would lead to an increase in computational resources and time. This would defeat the purpose of using parametric optimization in place of traditional non-parametric optimization.

In contrast, the ASM algorithm can handle non-linear equations without any explicit linearization and the algorithm can be coupled to a finite element analysis solver. The maximum error in the parametric optimal results found using ASM algorithm on the test problem is a meagre 2% when compared to 62% and 24% for the results from MPT 3.0 and MPQ/OA algorithms respectively. Though, the ASM algorithm is sensitive to the lower values of the parameter and has a restriction in the maximum parameter size that can be solved, through scaling, these issues can be offset. Due to the above mentioned benefits of using ASM, the tool seems to be the ideal candidate to solve structural parametric programming problems. Thus, the Approximate Simplex Method algorithm with the aforementioned modifications in section 2.3 is used to solve various single parametric and multi-parametric sizing, shape and multi-objective structural optimization problems for static and/or dynamic loads. In sizing and shape optimization problems, the load direction and load magnitude are typically considered as varying parameters. Multi-objective optimization problems are solved considering the objective targets as parameters. The accuracy of optimal parametric results, the computational time, the number of optimization and FEA calls is compared with the traditional non-parametric results and discussed in detail in the following chapter.

3 APPLICATIONS OF PARAMETRIC PROGRAMMING

In this section, the Approximation Simplex Method (ASM) algorithm identified as the most suitable multi-parametric algorithm from the case study of the benchmark four-bar truss optimization is used to solve a variety of structural optimization problems. ASM is used to solve single parametric and multi-parametric sizing optimization problem of truss structures for static and or dynamic load cases. For these truss structure problems, load magnitude and load directions are chosen as parameters. The computational performance of ASM with respect to the non-parametric method and the applicability of multi-parametric programming to sizing optimization are inferred. Similarly, shape optimization of a cantilever beam is solved with varying load direction as a parameter. Finally, parametric programming is used as a multi-objective optimization tool to solve a multi-objective honeycomb optimization problem.

3.1 Sizing Optimization Using Parametric Programming Method

3.1.1 Single Objective Single Parametric Optimization of Truss Structure With Direction of Load As A Parameter

The four bar truss problem in [4] is modified wherein the load $2p$ is applied at an angle ' t ' varying from 0.1 (5.7°) to 1 (90°) instead of a single vertically downward load direction (90°). Here, the angle of the load $2p$ with respect to the horizontal axis in the negative direction is considered as a parameter, the magnitude of the load p is considered

as a constant with a value of 2500 N and the areas of cross-sections of truss members 1, 2, 3 and 4 are defined as the decision variables. Let x_1 be the cross-section area of truss member 1, and x_2 be the cross-sectional areas of truss members 2, 3, and 4 as shown in Figure 3.1. The objective of this optimization problem is to minimize the weight of the truss with constraints on the axial stress in the truss members and nodal deflection in the truss structure.

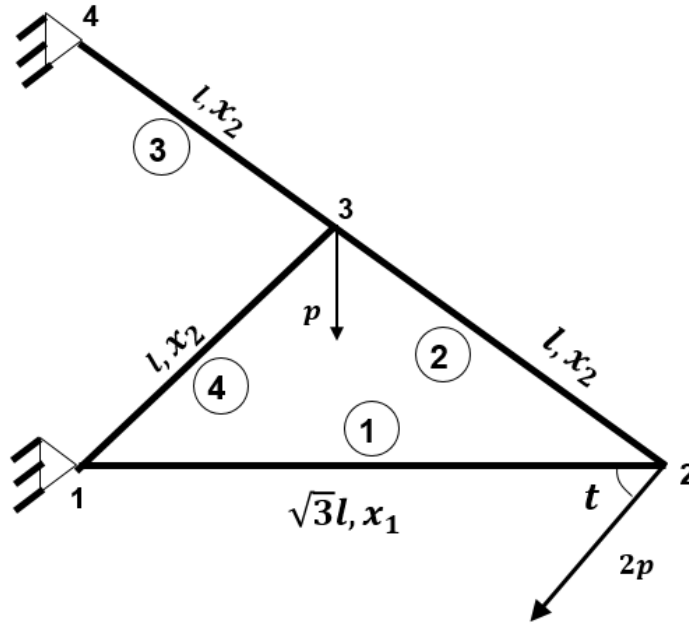


Figure 3.1: Four-bar truss problem from [1] with varying load direction as parameter

3.1.1.1 Problem Definition

Let $E = 200 \text{ MPa}$, be the Elastic Modulus of the material, $\rho = 8000 \frac{\text{kg}}{\text{m}^3}$ the density of the material, $l = 1\text{m}$ and $p = 1000\text{N}$.

$$\min_{x,t} mass = \rho l(3x_1 + \sqrt{3}x_2) \quad (3.1)$$

$$S.t.: g_1: \delta_{all} \leq 3 \times 10^{-3}l$$

$$g_2: \sigma_{all,t} \leq 8.74 \times 10^{-4}E$$

$$g_3: \sigma_{all,c} \leq 4.83 \times 10^{-4}E$$

$$1^{-7}m^2 < x_1 < 0.1 m^2$$

$$1^{-7} m^2 < x_2 < 0.1 m^2$$

$$0.1(5.7^0) < t(\theta) < 1(90^0)$$

where, x_1 and x_2 are cross-sectional areas (decision variables), P is the magnitude of the load applied, ' $mass$ ' is the total mass of the truss structure, θ in degrees is the angle made by load $2p$ with the horizontal axis (negative direction) and applied at the junction of members 3 and 4. When solving in MATLAB, trigonometric functions are represented in parametric form, wherein $t = \sin \theta$.

3.1.1.2 Results and Discussion

A Finite Element Analysis (FEA) MATLAB truss solver was developed to calculate the axial stress and nodal deflection in truss members. This FEA code is directly coupled to the ASM algorithm and the code can be found in Appendix D. The master ASM optimization solver for this problem is given in Appendix F. A validation was conducted to verify the accuracy of results obtained from the developed MATLAB FEA code through comparisons with results from a commercial FEA software package, ANSYS APDL [26].

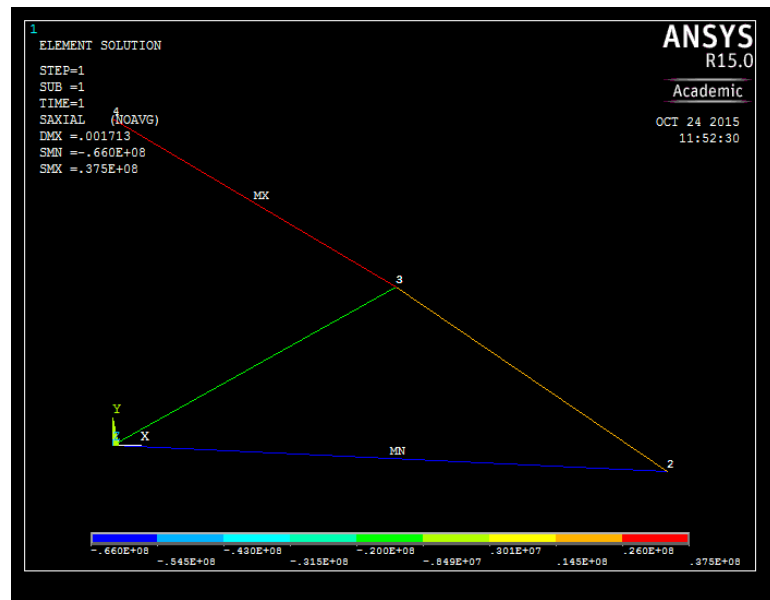


Figure 3.2: Contour plot of stress in the four-bar truss obtained using ANSYS R15.0 APDL

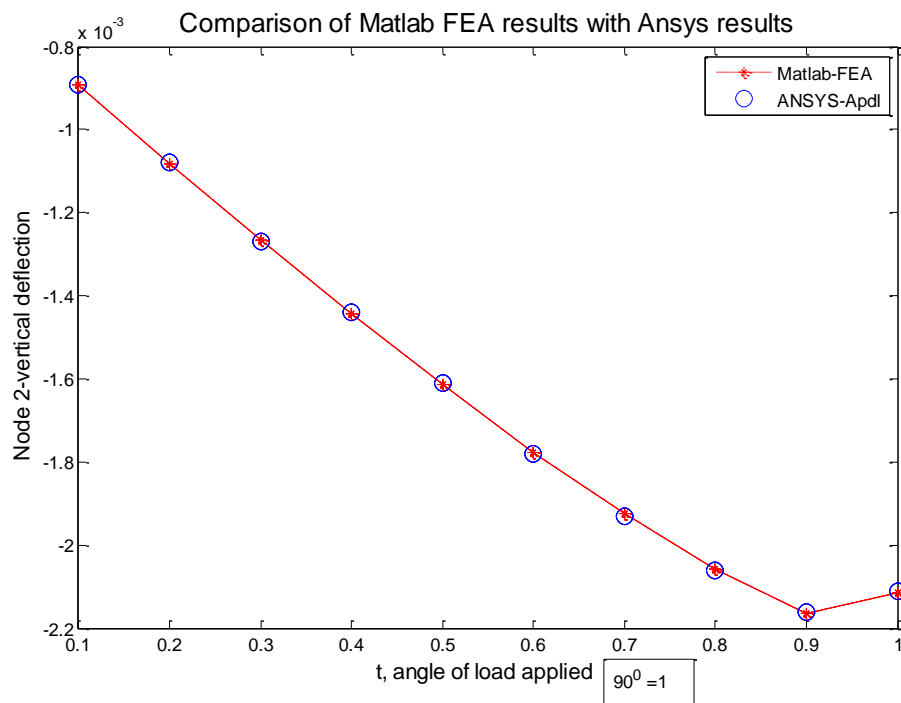


Figure 3.3: Comparison of vertical nodal deflection found using MATLAB FEA code with that from Ansys APDL

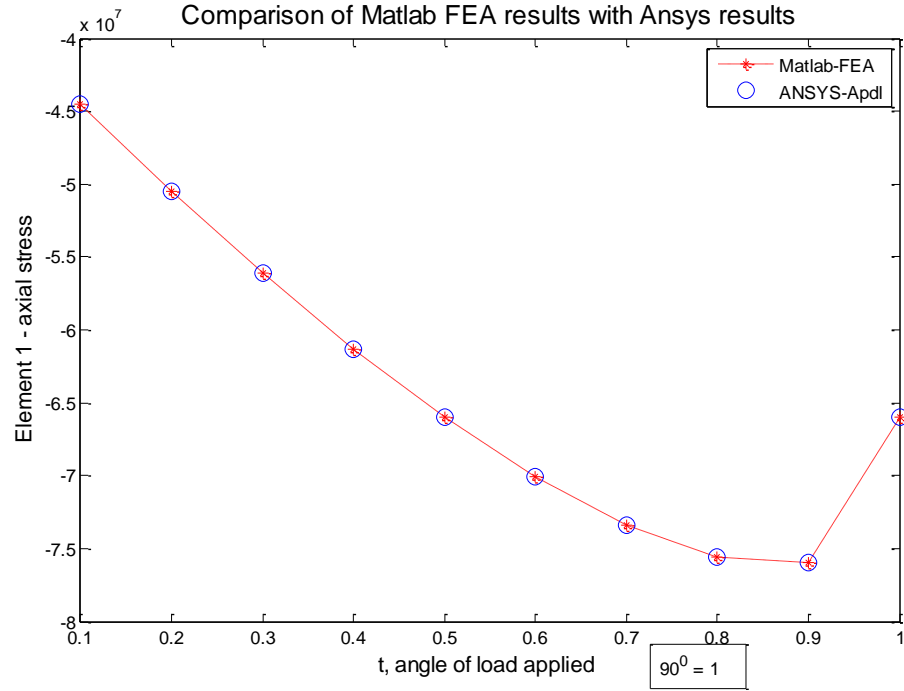


Figure 3.4: Comparison of element axial stress found using MATLAB FEA code with that from Ansys APDL

For a load magnitude ' p ' of 1000 N, the vertical deflection at node '2' of the truss structure found using the MATLAB FEA code agrees with the Ansys results for all the load angles/directions between 5.7° and 90° as shown in Figure 3.3. Similarly, axial stress in the horizontal member of the truss structure is also in agreement for all the load angles as shown in Figure 3.4.

The optimal parametric objective and decision variable values are obtained from coupling MATLAB FEA code with the ASM algorithm. The solid red and blue lines represent the optimal parametric results for the areas of cross-section x_1 and x_2 respectively while the dashed green line with circles and solid black line with asterisk represent that for

non-parametric results calculated at 10 equal interval discrete load angles between 5.7° and 90° . For an error tolerance of 0.1 (10% error), two segmentations or critical regions and six optimization calls are required to find the optimal parametric solution for the given parameter space. Each optimization call corresponds to optimization at a vertex of the parameter space. The parametric results are not in good agreement with the corresponding non-parametric results as can be seen in Figure 3.5. The allowable error tolerance was reduced to 0.01 (1% error) wherein the number of critical regions and optimization calls increased to six and eleven respectively. The number of (FEA) calls corresponding to eleven optimization calls is 633. The parametric results for the error tolerance of 0.01 concur with the non-parametric results obtained from discrete optimization at thirty equally distributed load angle values as shown in be seen in Figure 3.5.

It can be observed from the Figure 3.5 that the two optimal areas of cross-sections x_1 and x_2 increase with the increasing load angle and reached a peak of $6 \times 10^{-5} m^2$ and $3.78 \times 10^{-5} m^2$ respectively at a load angle of 71.80° ($t = 0.95$). Thereafter, the optimal areas of cross-section decreases with the increasing load. This shows that the worst loading condition corresponds to a load angle of 71.80° . In situations where the designer is unable to identify the worst loading condition and avoid over design scenario, parametric programming can be of useful in identifying and optimizing for worst loads through visual observation in a one parameter and two parameter cases.

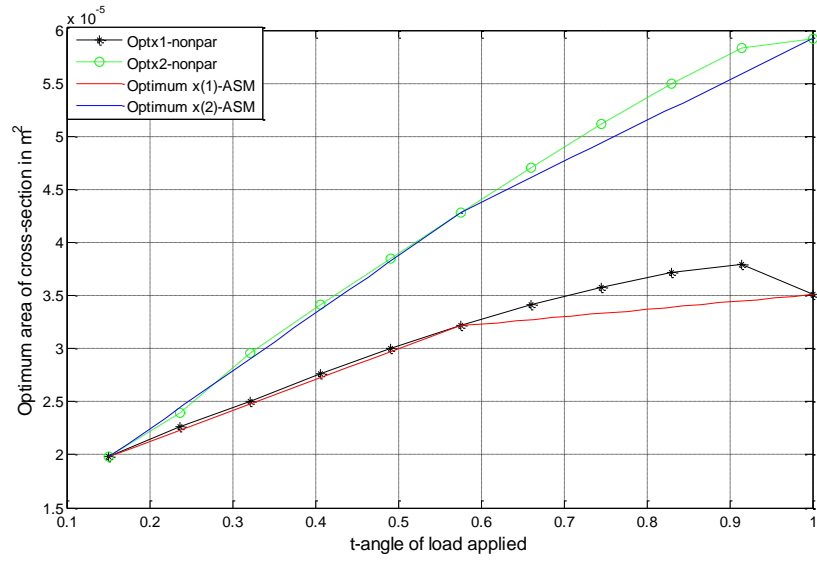


Figure 3.5: Plot of optimal decision variable (areas of cross-section) for an allowable error tolerance of 0.1 (10%)

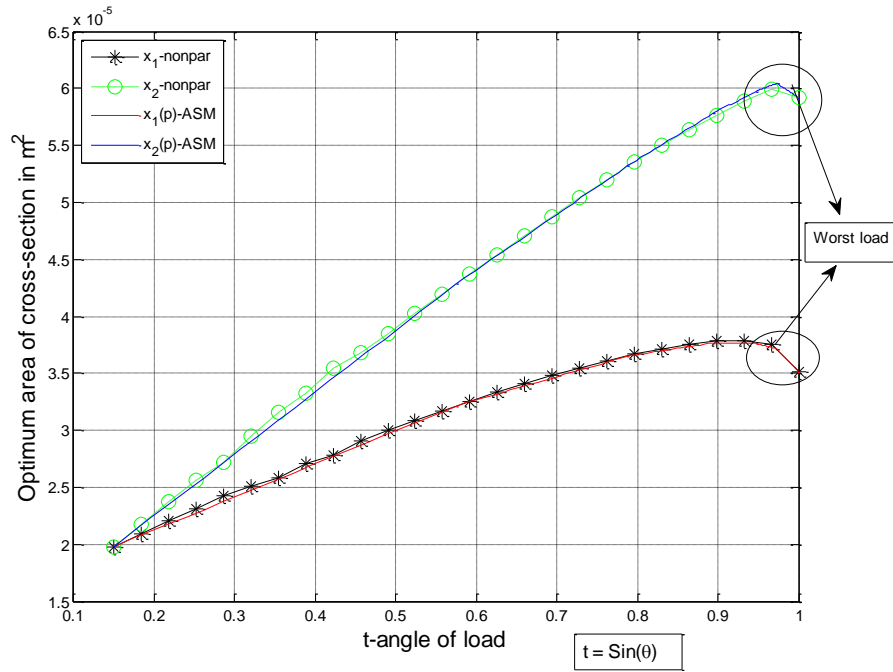


Figure 3.6: Comparison of optimal areas of cross-section found using ASM algorithm (1% error) with that from traditional non-parametric optimization at thirty discrete values of load angle

A list of optimal parametric functions of decision variable $x_2(t)$ and the optimal objective function $mass(t)$ generated by the ASM algorithm for an allowable error tolerance of 0.01 is given in Table 3.1.

Table 3.1: Table of optimal parametric function for x_2 and $mass$ with corresponding parameter intervals (critical regions) for error tolerance of 0.01

Critical Region	$x_2(t)$	$mass(t)$
$0.9734 < t < 1$	$(-7.75t + 11.27) \times 10^{-5}$	$(-7.267t + 11.27) \times 10^{-5}$
$0.9469 < t < 0.9734$	$(-1.743t + 5.412) \times 10^{-5}$	$(-1.744t + 5.412) \times 10^{-5}$
$0.8937 < t < 0.9469$	$(-0.105t + 3.861) \times 10^{-5}$	$(-0.105t + 3.860) \times 10^{-5}$
$0.7875 < t < 0.8937$	$(1.102t + 2.782) \times 10^{-5}$	$(1.102t + 2.783) \times 10^{-5}$
$0.5750 < t < 0.7875$	$(2.052t + 2.034) \times 10^{-5}$	$(2.052t + 2.034) \times 10^{-5}$
$0.150 < t < 0.5750$	$(2.902t + 1.545) \times 10^{-5}$	$(2.902t + 1.545) \times 10^{-5}$

The relative accuracy of the results obtained using parametric programming through ASM algorithm with the results from non-parametric optimization for the same computational expenses (FEA calls) is determined. For an error tolerance of 1%, ASM uses 633 FEA calls. For the same number of FEA calls, the number of discrete load angles (vertices in the parameter space) corresponds to twelve. Non-parametric optimization was conducted at twelve equally distributed load angles varying from 5.7° to 90° . Using the values of the optimal area of cross-section x_1 found at the twelve discrete load angles, a polynomial equation was obtained for optimal x_1 by fitting a polynomial curve of degree

5 as shown in Figure 3.7. The curve fitting was done using the MATLAB R2014 Curve Fitting Toolbox. The equation for the curve is given in Equation 3.2

$$x_{1,fit}(p) = 10^{-4}(-3.12t^5 + 7.8t^4 - 7.37t^3 + 3.139t^2 - 0.286 + 0.1921) \quad (3.2)$$

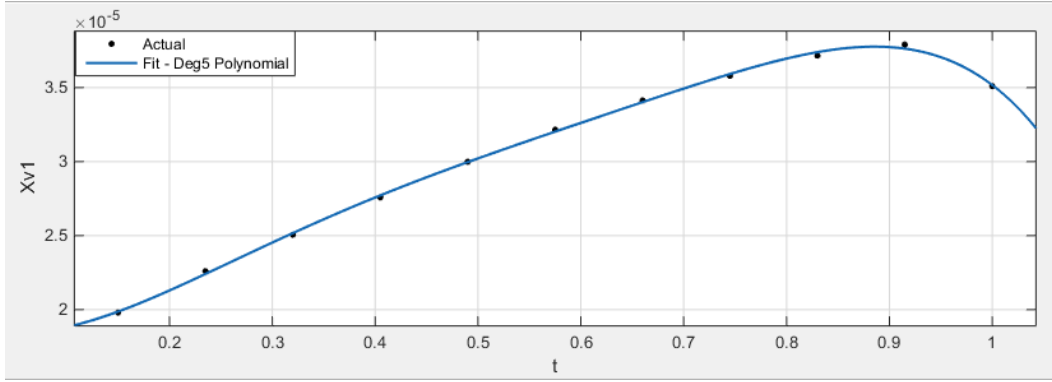


Figure 3.7: Plot of polynomial curve of degree five obtained from non-parametric results using MATLAB curve fitting toolbox

The accuracy of the polynomial curve obtained by curve fitting is determined by evaluating the $x_{1,fit}(p)$ function given in Equation (3.2) at 50 discrete load angles from 5.7° to 90° and compared with non-parametric optimization at the corresponding load angles. It was found that the maximum percentage error in the optimal areas calculated from evaluating $x_{1,fit}(p)$ with respect to the non-parametric results is 2.41% corresponding to a load angle of $t = 0.964$ (74.57°), a mean error of 0.561% and a Root Mean Square Error (RMSE) of 2.24×10^{-7} . The maximum error of 2.41% is greater than the maximum error of 1% corresponding to results from ASM algorithm for the same number of FEA calls. It can also be seen in Figure 3.8 indicated by dotted black circles that the twelve

equally distributed load angle values are not enough to accurately capture the non-linearity for ' t ' values beyond 0.9.

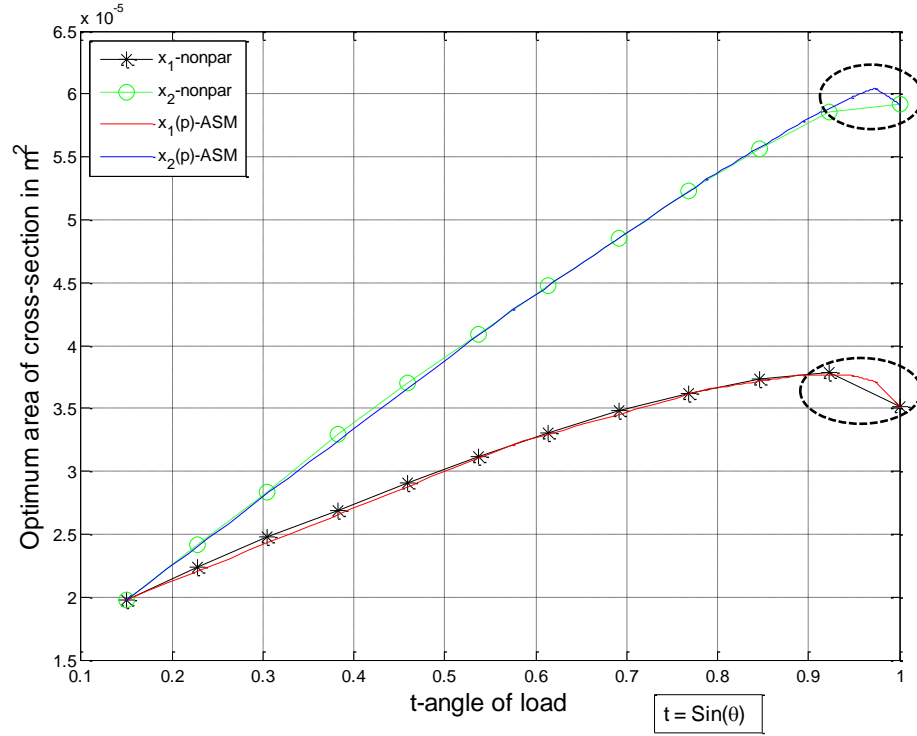


Figure 3.8: Comparison of optimal areas of cross-section found using ASM algorithm (1% error) with that from traditional non-parametric optimization at twelve discrete values of load angle

Hence, the number of discrete points was subsequently increased by two and the plot was compared with the parametric results. It can be seen from the plot in Figure 3.9 that the non-parametric results concurs with the parametric results for optimization at 20 equally distributed discrete load angles with 1252 corresponding FEA calls as shown in Figure 3.9.

An approximate polynomial equation of degree five was obtained from the optimal areas of the cross-section calculated from the non-parametric optimization at twenty discrete load angles. It was found that the maximum error corresponded to 2.06%, a mean error of about 0.58% with an RMSE of 5.07×10^{-7} . This shows that the maximum error in the polynomial function obtained from non-parametric optimization decreases with the increasing sample points.

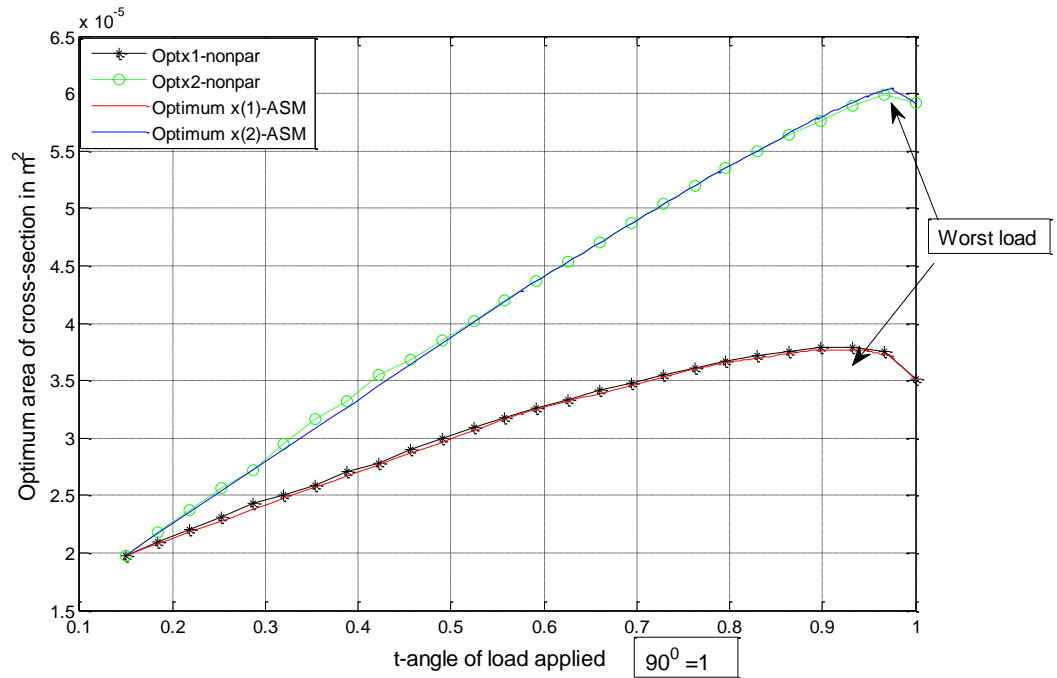


Figure 3.9: Comparison of optimal areas of cross-section found using ASM algorithm (1% error) with that from traditional non-parametric optimization at twenty discrete values of load angle

It is also evident that for the same number of FEA calls, the ASM algorithm provides optimal results with better accuracy than the non-parametric optimization. The comparison of optimization calls, FEA calls and computational time for parametric

programming using ASM algorithm and non-parametric optimization is given below in Table 3.2. For improved accuracy in non-parametric optimization, the number of optimization calls has to be increased which leads to increased computational expense. Note that the values of ' t ' at which the optimization is carried out to obtain the ASM approximation of the optimal parametric solution may not necessarily be the same as discrete ' t ' values at which non-parametric optimization is carried out.

Table 3.2: Comparison of computational performance and accuracy of parametric optimization results via ASM algorithm with Non-parametric optimization

	Relative maximum error (%)	Optimization calls	FEA calls	Computational time (s)
Parametric-ASM	1.0	11	633	3.119
Non-parametric	2.41	12	692	2.924
	2.06	20	1252	1.886

3.1.2 Single Objective Multi-Parametric Optimization of Truss Structure With Directions of Two Loads As Parameters

The objective of this problem is to minimize the mass of the truss structure subject to stress and deflection constraints with angles t_1 and t_2 subtended by loads p_1 and p_2 with the horizontal as the parameters of optimization. The point of application of loads p_1 and p_2 can be seen in Figure 3.10.

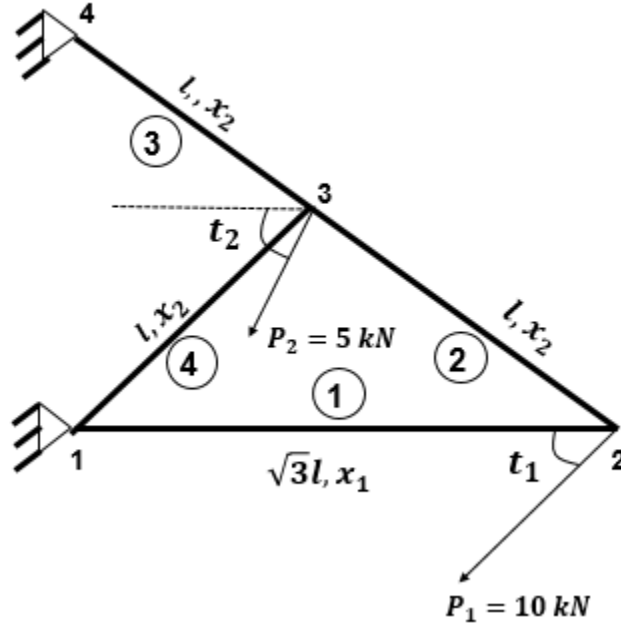


Figure 3.10: Four-bar truss with two concentrated loads varying in direction

Let $E = 200 \text{ MPa}$ be the Elastic modulus of the truss material, $\rho = 8000 \frac{\text{kg}}{\text{m}^3}$ be the density of the material. Let x_1 be the area of cross-section for truss member 1. Let x_2 be the area of cross-section for truss members 2, 3 and 4 as shown in Figure 3.10 and l is 1m . Let p_1 and p_2 be 10 kN and 5 kN respectively.

3.1.2.1 Problem Definition

$$\min_{x,t} \text{mass} = \rho l (3x_1 + \sqrt{3}x_2) \quad (3.3)$$

S.t.

$$\delta_{all} \leq 3 \times 10^{-3} l$$

$$\sigma_{all,t} \leq 8.74 \times 10^{-4} E$$

$$\sigma_{all,c} \leq 4.83 \times 10^{-4} E$$

$$1^{-6} m^2 < x_1 < 0.1 m^2$$

$$1^{-6} m^2 < x_2 < 0.1 m^2$$

$$0.1(5.7^0) < t_1(\theta_1) < 1(90^0)$$

$$0.1(5.7^0) < t_2(\theta_2) < 1(90^0)$$

where $t_1 = \sin \theta_1$ and $t_2 = \sin \theta_2$.

3.1.2.2 Results and Discussion

The MATLAB FEA code described in section 3.1.1 is used as the FEA solver with minor modification to incorporate load direction t_2 as a parameter in addition to t_1 . The master ASM optimization solver for this problem is given in Appendix G.

For an allowable approximation error of the objective function of 0.105 (10.5%), the optimal parametric results for the decision variables (areas of truss members) and the objective (mass of the truss structure) found using the Approximation Simplex Method (ASM) algorithm for loads $p_1 = 10kN$, $p_2 = 5kN$ is given below. The ASM algorithm generated ten segmentations (critical regions) to find optimal solution within the error tolerance. Figure 3.11 shows the plot of optimal decision variable $x_1(p)$ for the ten critical regions. For a better understanding, another view of Figure 3.11 is given in Figure 3.12.

The significant observation that is made with the help of the optimal parametric function plot of area of cross-section of the truss members is that the optimal area of the cross-section required for truss member 1 is greater than that required for the other truss

members for the load angle t_1 between 0.1 (5.7°) and 0.3 (17.46°). For load angles t_1 greater than 17.46° , the optimal area of cross-section required for truss members 2, 3 and 4 are larger than that required for the truss member 1 as shown in Figure 3.14. This shows that the plot of optimal parametric function of the decision variables can aid the designers to understand which of the truss members would be subjected to greater stresses and deflection for a given loading condition. The variation in the trend of the requirement of optimal areas for different truss members as the load direction varies can also be identified.

For the objective function approximation error tolerance of 0.105 (10.5%), ten critical regions were generated with a total of fifty six optimization calls and 2024 FEA calls. Non-parametric optimization is conducted for the same number of optimization calls corresponding to fifty six vertices in the (t_1, t_2) parameter space obtained by discretization of both t_1 and t_2 between 5.7° and 90° with equal intervals. The plot of optimal objective obtained from non-parametric optimization does not provide a smooth surface. To capture the results accurately for t_2 values beyond 0.8, the number of vertices was increased to 100 and 225 subsequently for experimentation. The plot with 100 optimization calls provides a smooth surface that captures the results better than the plot obtained from fifty six optimization calls and is comparable to the plot from 225 optimization calls as shown in Figure 3.15. Thus, a minimum of 100 optimization calls are required to get the optimal solution through non-parametric optimization. The comparison of parametric results from ASM with the non-parametric results is shown in Figure 3.14. The parametric functions of optimal decision variables for the four critical regions out of the eight critical regions are given in Table 3.3.

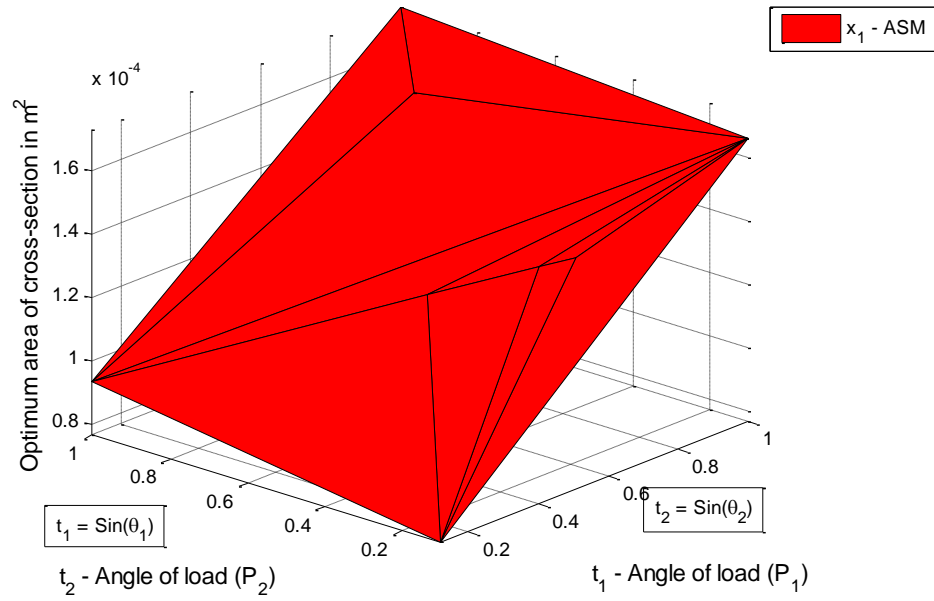


Figure 3.11: Optimal area of cross-section $x_1(t_1, t_2)$ found using ASM algorithm View 1

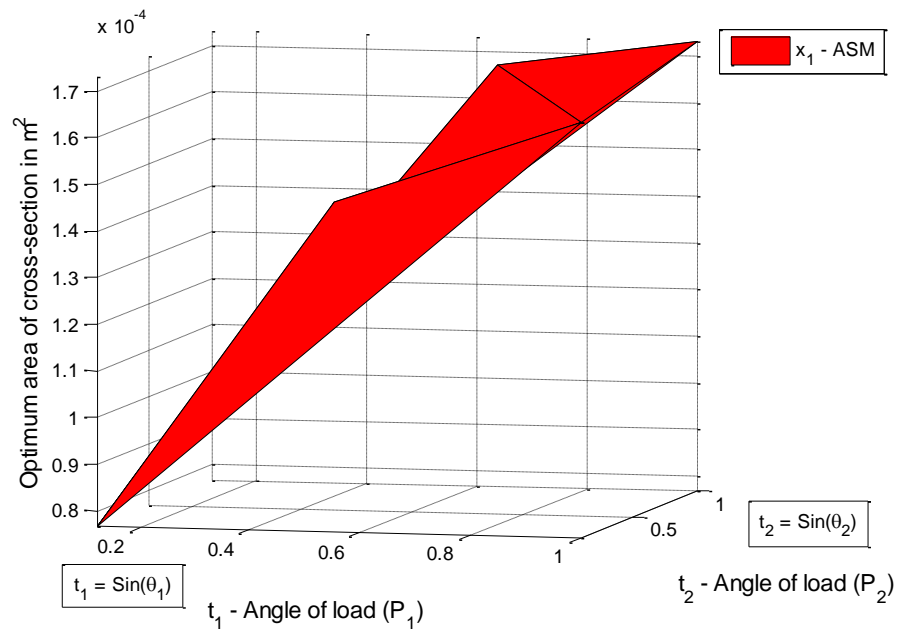


Figure 3.12: View 2 of Figure 3.11

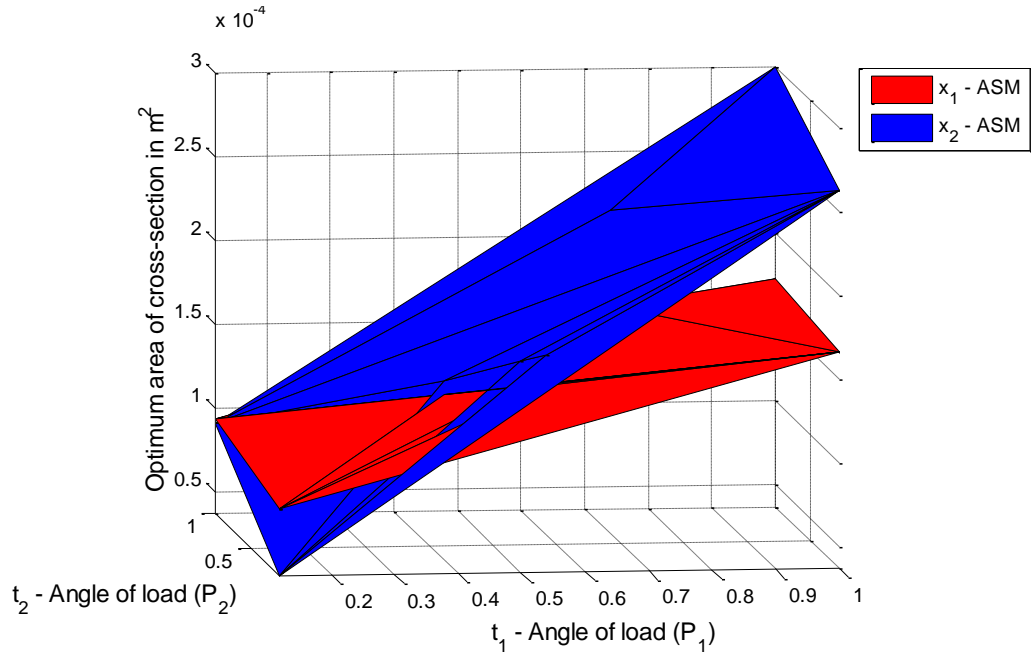


Figure 3.13: Comparison of optimal areas $x_1(t_1, t_2)$ and $x_2(t_1, t_2)$ found using ASM algorithm

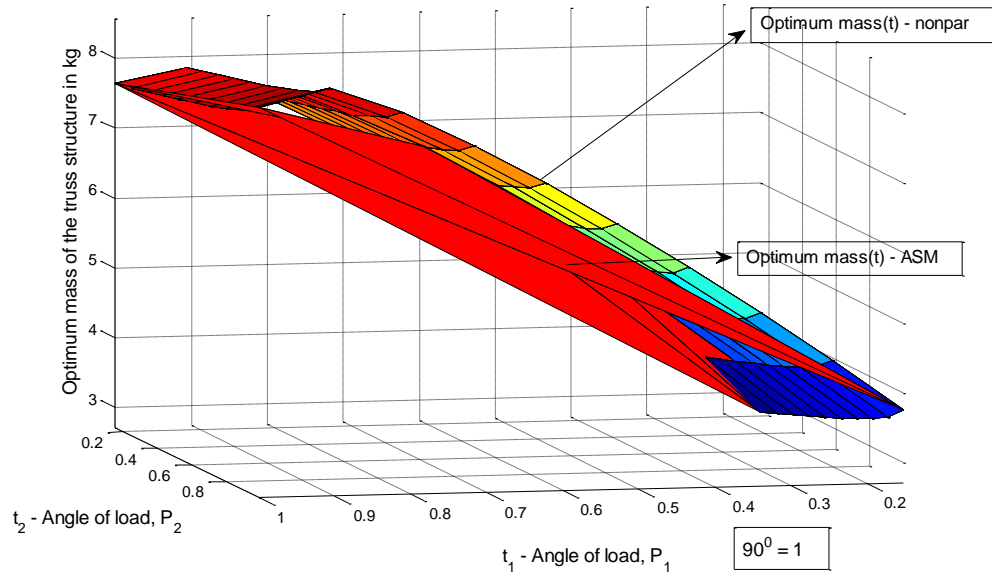


Figure 3.14: Comparison of optimal objective (mass) found using ASM algorithm with that from non-parametric optimization at 100 discrete points in the parameter space

In the Figure 3.14, the triangles in red represent the optimal objective values found using the ASM algorithm and the quadrilateral cells of different colors represent the optimal objective values found using the non-parametric method.

Table 3.3: Table of optimal parametric function of x_1 and x_2 with corresponding parameter intervals (critical regions) for error tolerance of 0.105

Critical Region	$x_1(t_1, t_2)$	$x_2(t_1, t_2)$
Simplex 1	$(-8.75t_1 - 8.32t_2 + 16.89) \times 10^{-5}$	$(23.6t_1 + 0.78t_2 + 5.64) \times 10^{-5}$
Simplex 2	$(17.91t_1 + 9.99t_2 - 2.81) \times 10^{-5}$	$(2.71t_1 + 7.69t_2 - 1.79) \times 10^{-5}$
Simplex 3	$(-0.404t_1 + 0.84t_2 - 16.89) \times 10^{-5}$	$(20.2t_1 + 4.24t_2 + 5.64) \times 10^{-5}$
Simplex 4	$(3.26t_1 - 4.65t_2 + 14.04) \times 10^{-5}$	$(22.3t_1 + 2.92t_2 + 3.70) \times 10^{-5}$

As the allowable error in approximation was lowered to 0.075 (7.5%), the number of critical regions increased to thirty with total optimization calls of 216. No feasible results were found for error tolerance below 0.075. The computing machine used for this study consists of an Intel Core i3-4030U CPU, 1.90 GHz, 4GB RAM with a 64-bit operating system.

Table 3.4: List of critical regions with their associated vertices for error tolerance of 0.105

Critical Region	Vertex 1 (t_1^1, t_2^1)	Vertex 2 (t_1^2, t_2^2)	Vertex 3 (t_1^3, t_2^3)
Simplex 1	(0.7167,0.7167)	(1,1)	(0.15,1)
Simplex 2	(1,0.15)	(0.7167,0.7167)	(0.15,1)
Simplex 3	(1,0.15)	(1,1)	(0.7167,0.7167)
Simplex 4	(0.4333,0.4333)	(1,0.15)	(0.15,1)

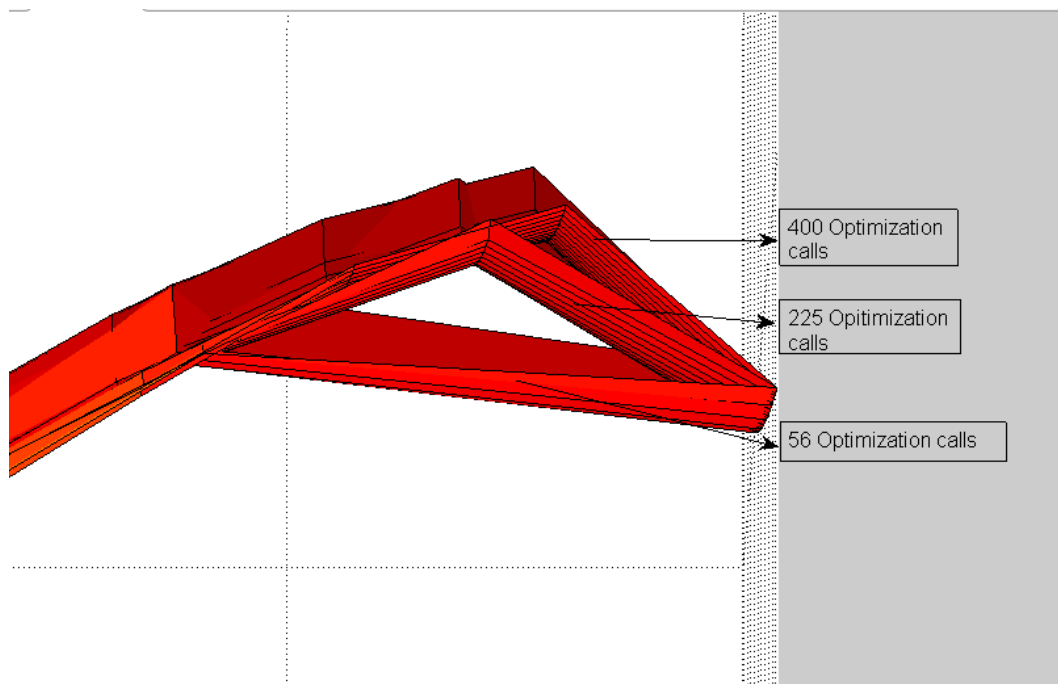


Figure 3.15: Comparison of surface plot of optimal objective obtained from non-parametric optimization at various optimization calls for t_1 between 0.85 and 1

Table 3.5: Performance of ASM algorithm with different error tolerance values for truss optimization with load directions as parameters

	Parametric ASM algorithm	
Error tolerance, ϵ	10.5%	7.5%
Optimization calls	56	216
FEA calls	2384	9955
Computation time (sec)	4.7031	139.66

3.1.3 Single Objective Multi-Parametric Optimization of Truss Structure With Both Load Direction and Load Magnitude As Parameters

Consider a four-bar truss with loads p and $2p$ applied at nodes 3 and 2 respectively as shown in Figure 3.16. The objective of this parametric optimization problem is to minimize the mass of the truss structure with axial stress constraints on the truss members represented by numbers enclosed within circles and vertical deflection constraints at the truss nodes 1, 2, 3 and 4 as shown in Figure 3.16. The load magnitude p and load direction t of load $2p$ acting at node 2 are the two parameters of the parametric optimization problem. They are allowed to vary between 5000N to 7000N and 5.7° to 90° respectively.

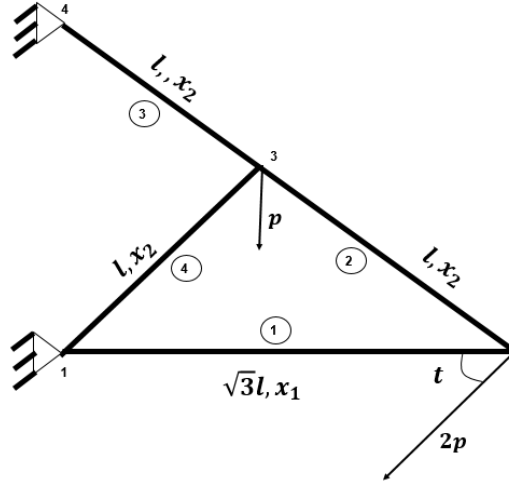


Figure 3.16: Four-bar truss with varying load magnitude and load direction

3.1.3.1 Problem Definition

Let $E = 200 \text{ MPa}$ be the Elastic modulus of the truss material, $\rho = 8000 \frac{\text{kg}}{\text{m}^3}$ be the density of the material. Let x_1 be the area of cross-section for truss member 1. Let x_2 be the area of cross-section for truss members 2, 3 and 4 as shown in Figure 3.15 and l is 1.

$$\min_{x,t} \text{mass} = \rho l (3x_1 + \sqrt{3}x_2) \quad (3.4)$$

$S. t$

$$\delta_{all} \leq 3 \times 10^{-3} l$$

$$\sigma_{all,t} \leq 8.74 \times 10^{-4} E$$

$$\sigma_{all,c} \leq 4.83 \times 10^{-4} E$$

$$1^{-6} \text{ m}^2 < x_i < 0.1 \text{ m}^2; i = 1,2$$

$$2000N < p < 4000N$$

$$0.1(5.7^\circ) < t(\theta) < 1(90^\circ)$$

3.1.3.2 where $t = \sin \theta$. Results and Discussion

The MATLAB FEA code developed in section 3.1.1 is used as the FEA solver and the master ASM parametric programming algorithm is given in Appendix H. For an allowable error ‘ ϵ ’ of 0.12 (12%) in the approximation of the objective function, parametric optimization using the ASM algorithm resulted in ten critical regions with a total of thirty two optimization calls.

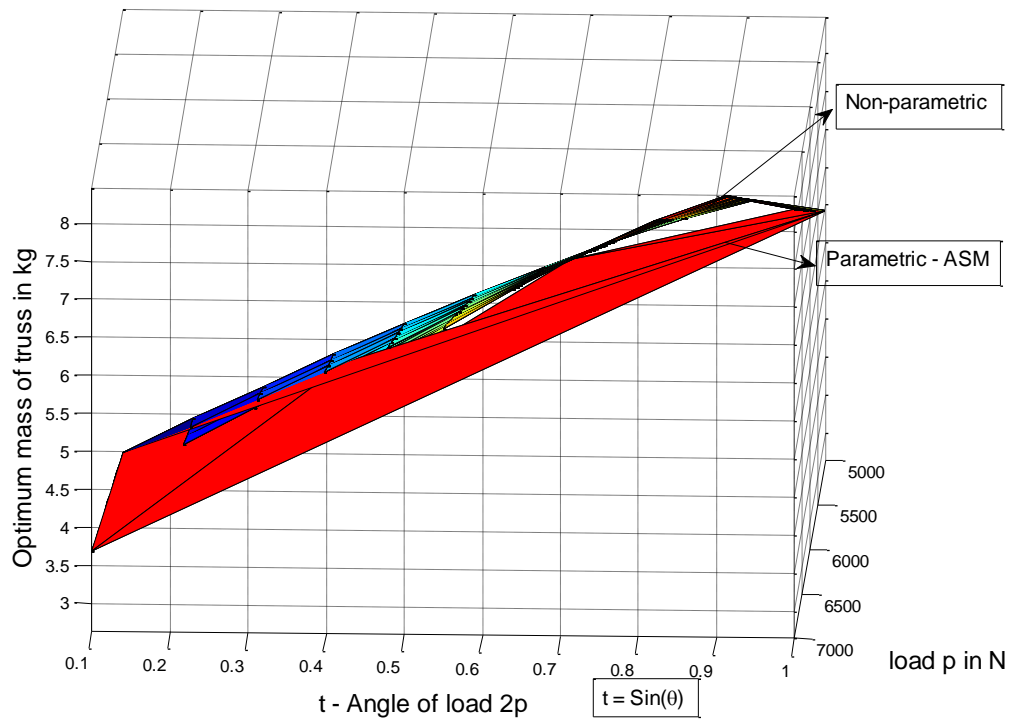


Figure 3.17: Comparison of optimal parametric objective found by ASM with results from nonparametric optimization at thirty six discrete points in the (t, p) parameter space (View 1)

Table 3.6: Comparison of performance of ASM algorithm with non-parametric optimization method for truss optimization with load magnitude and direction as parameters

	ASM algorithm	
Error tolerance (ϵ)	12%	10%
Optimization calls	32	404
FEA calls	264	3333
Computation time (s)	6.323	9.124

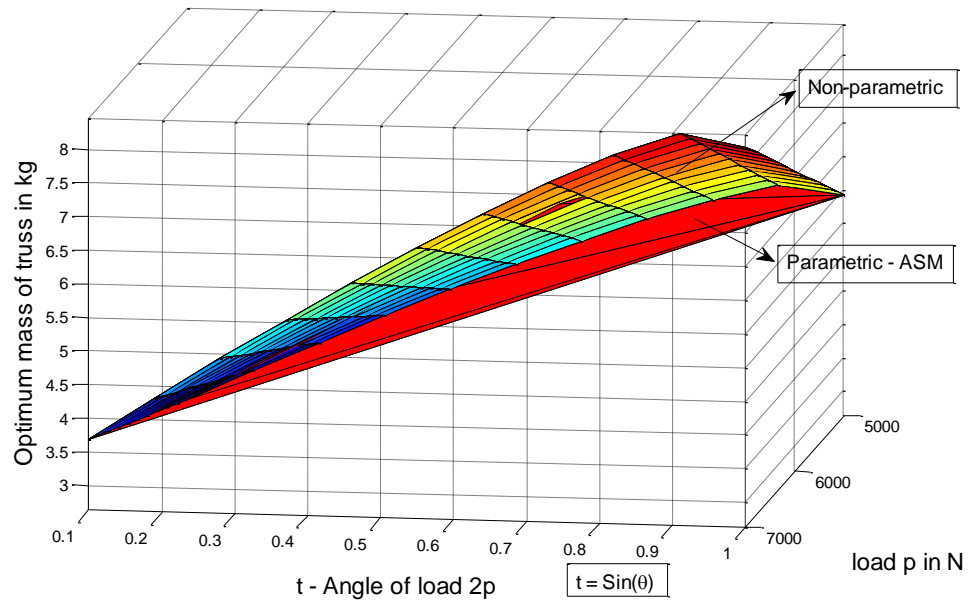


Figure 3.18: View 2 of Figure 3.15

The optimal parametric function of the objective function can be seen in figures 3.17 and 3.18 respectively. It can be observed that the behavior of the optimal objective is highly non-linear in the (t, p) parameter space defined for the problem.

3.1.4 Single Objective Multi-Parametric Optimization of 10 Bar Truss With 10 Decision Variables and 4 Parameters

The objective of this problem is to minimize the mass of a truss structure with ten bars subject to stress and deflection constraints. The angles t_1 and t_2 subtended by loads p_1 and p_2 with the horizontal, and load magnitudes p_1 and p_2 are considered as the parameters of optimization. The points of application of loads p_1 and p_2 can be seen in Figure 3.18.

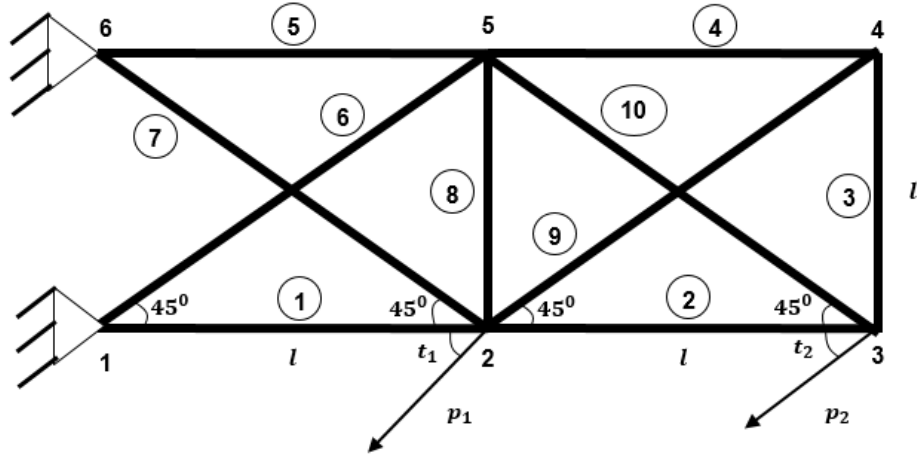


Figure 3.19: Ten-bar truss with two varying load magnitudes and two varying load directions as parameters

Let $E = 10^4 \text{ ksi}$ be the Elastic modulus of the truss material, , $\rho = 0.1 \frac{\text{lb}}{\text{in}^3}$ be the density of the material and l is equal to 360 *inches*. Let x_i be the areas of cross-section for the truss members ($i = 1, 2, \dots 10$) represented by numbers within the circles as shown in Figure 3.18.

The MATLAB FEA code given in Appendix D is modified to accommodate the ten truss members, two load magnitudes and two load directions and can be found in Appendix H. The master ASM parametric optimization solver for this problem can be found in Appendix I.

3.1.4.1 Problem Definition

$$\min_{x,t} mass = \rho \left(\sum_{i=1}^{10} x_i l_i \right) \quad (3.5)$$

$$\delta_{all} \leq 3 \times 10^{-3} l$$

$$\sigma_{all,t} \leq 8.74 \times 10^{-4} E$$

$$\sigma_{all,c} \leq 4.83 \times 10^{-4} E$$

$$0.1 \text{ in}^2 < x_i < 20 \text{ in}^2, \quad i = 1, 2, \dots 10$$

$$0.1(5.7^\circ) < t_1(\theta_1) < 1(90^\circ)$$

$$0.1(5.7^\circ) < t_2(\theta_2) < 1(90^\circ)$$

$$90 \text{ kips} < p_1 < 100 \text{ kips}$$

$$80 \text{ kips} < p_2 < 95 \text{ kips}$$

where $t_1 = \sin \theta_1$ and $t_2 = \sin \theta_2$

3.1.4.2 Results and Discussion

For an approximation error tolerance ϵ of 0.1 (10%), the ASM algorithm could not find optimal results due to convergence issues. As the error in approximation did not decrease below the set error tolerance, the segmentation of the parameter space continued until the distance between the end vertices and center of the newly formed critical region became lesser than $10^{-4}t_1$ and $10^{-4}t_2$. Thereafter the algorithm terminated the segmentation process and no solution was returned. This may be attributed to the higher degree of non-linearity that is associated with the constraints.

Table 3.7: List of optimal objective function for the optimization problem of ten-bar truss structure

Critical Region	Optimal objective function ($z(t_1, t_2, p_1, p_2)$)
1	$(8.905t_1 + 43.61t_2 + 0.0127p_1 + 0.00793p_2 - 149.77) \times 10^{-2}$
2	$(8.906t_1 + 43.61t_2 - 0.00135p_1 - 0.00141p_2 + 65.204) \times 10^{-2}$
3	$(9.696t_1 + 42.82t_2 + 0.0119p_1 + 0.0075p_2 - 138.86) \times 10^{-2}$
4	$(-5.868t_1 + 42.83t_2 - 0.00205p_1 + 0.00745p_2 + 2.845) \times 10^{-2}$
5	$(-5.868t_1 + 46.87t_2 - 0.00158p_1 + 0.00745p_2 - 33.941) \times 10^{-2}$

Thus, the error tolerance was increased to 0.15 (15%). The ASM algorithm converged and an optimal solution was obtained. The parameter space was divided into

twenty three segments with a total of 138 optimization calls. A list of parametric optimal objective (mass of truss structure) functions for the first 5 critical regions and geometric functions for corresponding critical regions is given below in Table 3.7 and Table 3.8 respectively. Since there are four parameters the number of vertices for each critical region is five.

Non-parametric optimization was conducted at discrete parameters values. Each individual parameter range was discretized into 10 equally spaced vertices and optimization was conducted at each vertex to determine the complete behavior of the optimal solution with respect to the parameter. Since there are four parameters, non-parametric optimization was conducted at a total of 10^4 vertices in the entire parameter space. The computational time and number of finite element analysis (FEA) calls corresponding to the 10^4 Optimization calls is 3.28 hours and 2.466 million FEA calls. A non-parametric optimization study at five vertices per parameter still leads to a considerably large value of 3125 total optimization calls with a computational time of 1.36 hours and 0.913 million FEA calls. The comparison of computational time, optimization calls and FEA calls between the parametric ASM method and non-parametric optimization is given in Table 3.9.

Table 3.8: List of vertices that form each critical region for the optimization problem of ten-bar truss structure

Critical Region	Vertex 1 (t_1, t_2, p_1, p_2)	Vertex 2 (t_1, t_2, p_1, p_2)	Vertex 3 (t_1, t_2, p_1, p_2)	Vertex 4 (t_1, t_2, p_1, p_2)	Vertex 5 (t_1, t_2, p_1, p_2)
1	(0.1,0.1,1000 00,80000)	(0.1,1,90000, 80000)	(0.1,0.1,9000 0,95000)	(1,1,90000,9 5000)	(0.1,1,90000 ,95000)
2	(0.1,0.1,1000 00,80000)	(1,0.1,10000 0,95000)	(0.1,0.1,9000 0,95000)	(1,1,90000,9 5000)	(0.1,1,90000 ,95000)
3	(0.1,0.1,1000 00,80000)	(1,0.1,90000, 80000)	(0.1,1,90000, 80000)	(0.1,0.1,900 00,95000)	(1,1,90000,9 5000)
4	(0.1,0.1,1000 00,80000)	(1,0.1,90000, 80000)	(1,0.1,10000 0,95000)	(1,0.1,10000 0,80000)	(1,1,90000,9 5000)
5	(0.1,0.1,1000 00,80000)	(1,1,100000, 80000)	(1,0.1,10000 0,95000)	(1,0.1,10000 0,80000)	(1,1,90000,9 5000)

It is difficult to represent the optimal results in a 5-Dimensional polyhedral in a plot for both parametric and non-parametric optimization. Thus, the parametric results are compared with non-parametric results at a series of 23 vertices randomly chosen in the parameter space with one vertex from each critical region. In this regard, the parametric optimal objective function is evaluated at all the 23 vertices and compared with optimal objective value obtained through non-parametric optimization at the corresponding vertices. The maximum error is found to be 13.6% with an average error of 5.98% and a

median of 4.79%. This shows that the error in parametric results is well within the acceptable error when compared to the non-parametric results.

Table 3.9: Comparison of computational performance of ASM with non-parametric optimization method

	Optimization Calls	FEA Calls	CPU time
Parametric – ASM	128	61659	28 mins
Non-parametric	10^4	2.1 million	3h 11 mins

3.1.5 Sizing Optimization of A Four-Bar Truss Structure For Dynamic Impulse

Loading With Load Direction As A Parameter

The objective of this optimization problem is to minimize the mass of a four-bar truss structure for a dynamic transient load varying in direction. The optimization problem is subject to constraints on axial stress in truss members, vertical nodal deflection, and the allowable range of first natural frequency of the truss structure. Figure 3.20 shows the truss structure with truss members represented by numbers within circles and the four truss nodes (junctions) are represented by the numbers.

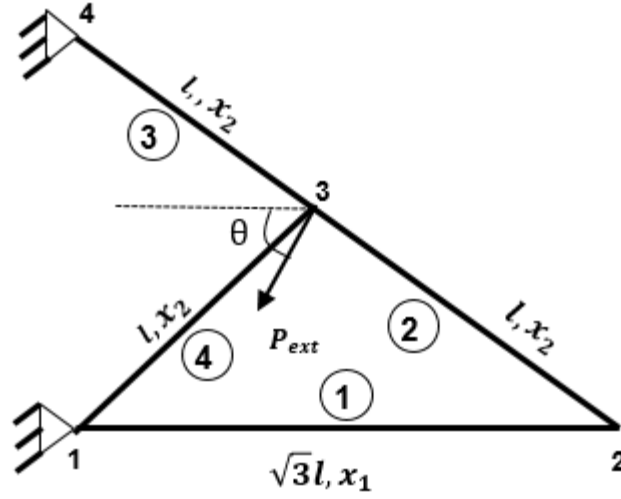


Figure 3.20: Four-bar truss structure with a dynamic load applied at a single node

Let x_1 be the area of cross-section for truss member 1. Let x_2 be the area of cross-section for truss members 2, 3 and 4. Let $E = 200 \text{ MPa}$ be the Elastic modulus of the truss material, $\rho = 8000 \frac{\text{kg}}{\text{m}^3}$ be the density of the material, and $l = 1\text{m}$. Let P_{ext} be the transient impulse load applied at node 3 of the truss structure and is defined in Equation 3.6. Let θ be the angle representing the varying direction of load P_{ext} .

$$P_{ext} = 10^{10} t e^{-500t}, \quad 0s < t < 0.0125s \quad (3.6)$$

where, t is the time in seconds.

3.1.5.1 Problem Definition

$$\begin{aligned}
 \min_{x,t} mass &= \rho l(3x_1 + \sqrt{3}x_2) \\
 \text{S.t.} \quad \delta_{all} &\leq 3 \times 10^{-4} l \\
 \sigma_{all,t} &\leq 8.74 \times 10^{-4} E \\
 \sigma_{all,c} &\leq 4.83 \times 10^{-4} E \\
 600 \frac{rad}{s} &< \omega < 1200 \frac{rad}{s} \\
 1^{-6} m^2 &< x_1 < 0.5 m^2 \\
 1^{-6} m^2 &< x_2 < 0.5 m^2 \\
 0.1(5.7^0) &< \theta (\phi) < 1(90^0)
 \end{aligned} \tag{3.7}$$

where $\theta = \sin \phi$

3.1.5.2 Results and Discussion

A MATLAB Finite Element Analysis (FEA) code is developed to solve for transient impulse loads on truss structures using the Modal superposition method. In the finite element analysis, each truss member is considered as a truss element. Each element has two nodes and each node has two degrees of freedom. One degree of freedom representing horizontal deflection and the other vertical deflection. Since nodes 1 and 4 are fixed in the truss structure as shown in Figure 3.21, there is a total of 4 free degrees of freedom and thus four mode shapes for the truss structure

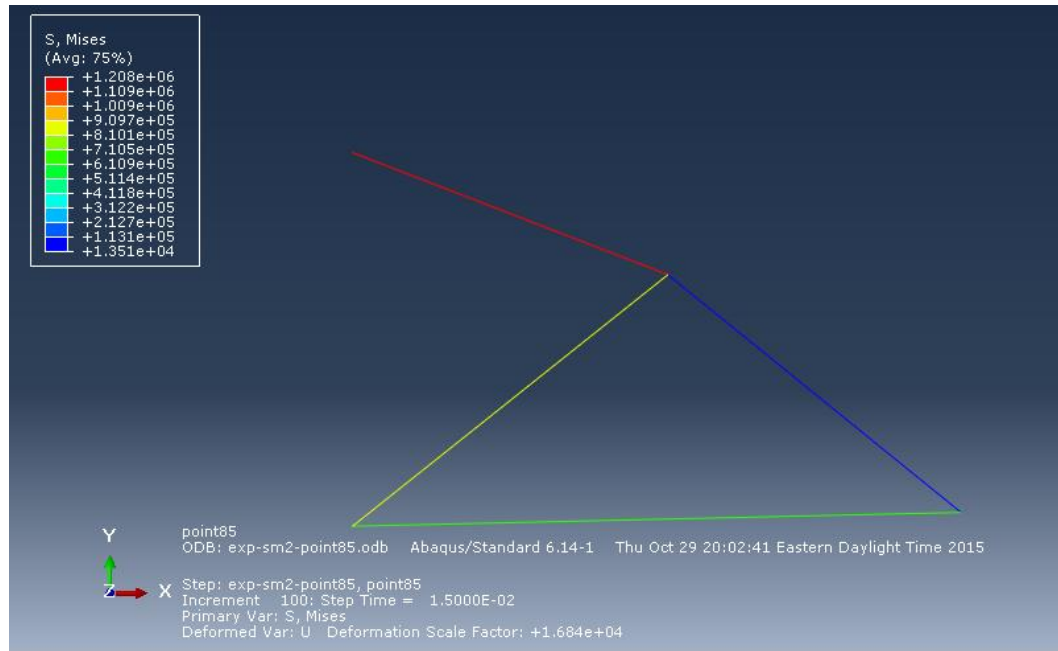


Figure 3.21: Contour plot of Von-Mises stress for the four bar truss found in Abaqus CAE 6.14

All the four mode shapes are utilized to determine the nodal deflection and element stress. The dynamic analysis is solved for a total of 100 time steps to accurately capture the behavior of stress and deflection for the given time interval. The maximum axial stress in the truss members and maximum vertical nodal deflection are then determined from the results obtained at 100 time steps.

The MATLAB FEA code for the dynamic analysis of truss can be found in Appendix E and the master ASM parametric programming algorithm for this problem can be found in Appendix J.

The accuracy of the dynamic results obtained from MATLAB FEA code are verified with the results from Abaqus CAE 6.14, a commercial FEA package. For the

verification FEA is carried out for the dynamic load given in Equation (3.6) at eight discrete load angles between 5.7^0 and 90^0 . The vertical nodal deflection at node 3 shown in Figure 3.20 is used for this comparison. The results from the MATLAB FEA code are in agreement with the Abaqus 6.14 results as shown in Figure 3.24 with a maximum deviation of 25% corresponding to a load angle of 30 degrees. In Abaqus CAE 6.14 [28], the amplitude of the transient external load was represented as a smooth curve with six data points as input. This consisted of six amplitude values corresponding to six respective time instances. The plot of the amplitude curve is shown in the Figure 3.23. The representation of the load amplitude in Abaqus was not as accurate as the actual load amplitude function shown in the Figure 3.22. The difference in the results between Abaqus and MATLAB FEA is believed to be due to the approximate definition of the load amplitude function in Abaqus.

Since the results from MATLAB FEA code concur with the Abaqus, the same MATLAB code is used for the parametric optimization of the four bar truss structure. The objective function approximation error tolerance ϵ for the ASM algorithm was set as 0.01 (1%). For the set of constraints and the objective function defined in Equation 3.7, the ASM algorithm generated a total of 16 segmentations (critical regions).

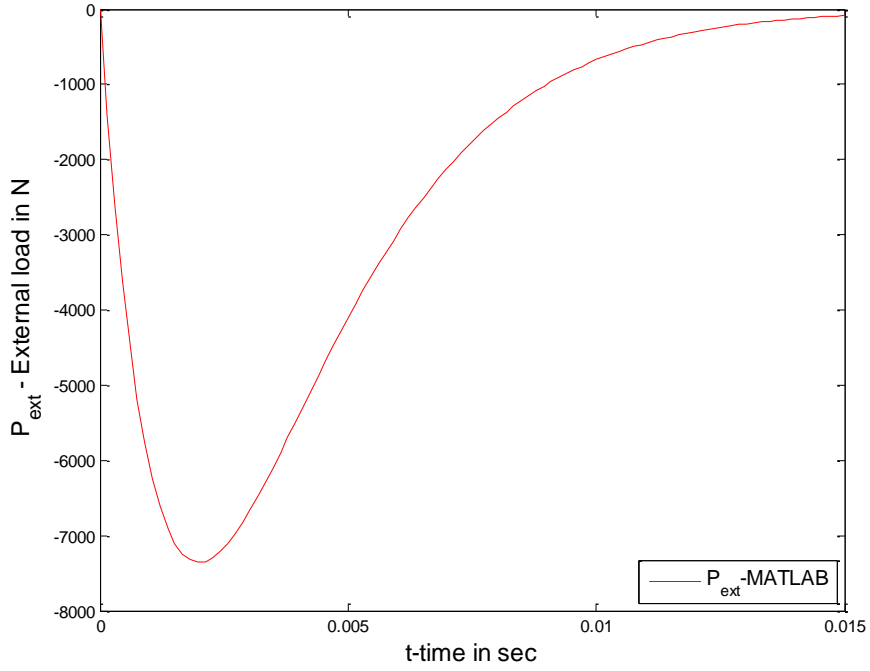


Figure 3.22: Plot of external transient load function defined over the time interval at 100 time steps and used in MATLAB FEA analysis

The plot of optimal function for the areas of cross-section x_1 and x_2 of the truss members is shown in Figure 3.25. It can be seen from Figure 3.25 that the optimal area of the cross-section for truss member 1 is consistently higher than that of the truss members 2, 3 and 4 for all the load angles between 5.7° and 90° .

The optimal areas x_1 and x_2 are $3.6 \times 10^{-4} m^2$ and $1.85 \times 10^{-4} m^2$ respectively for a load angle of $\varphi = 0.15$ corresponding to 5.7° . Thereafter, both the optimal areas decrease with increase in the load angle and reaches a minimum of $2 \times 10^{-4} m^2$ and $1 \times 10^{-4} m^2$ respectively for a load angle of $\varphi = 0.31$ corresponding to 18.06° .

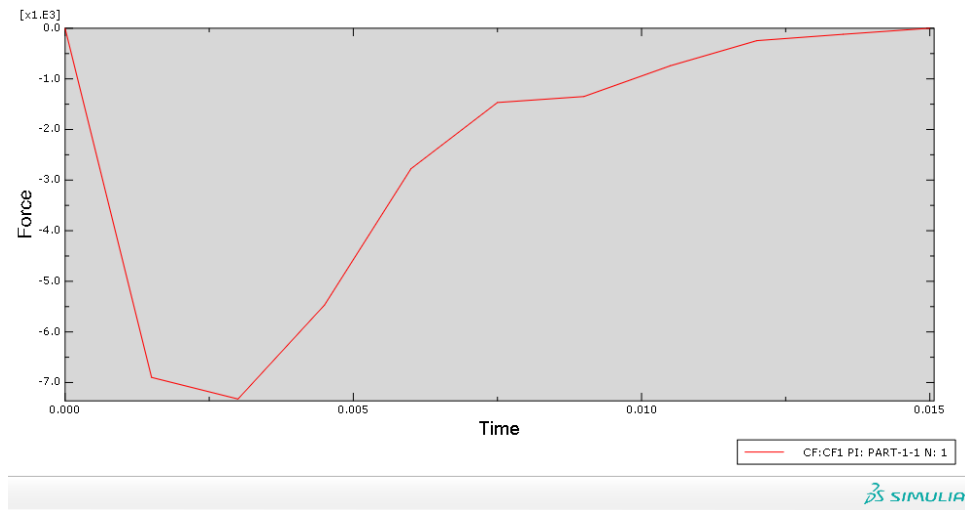


Figure 3.23: Plot of approximated external transient load function defined in Abaqus CAE 6.14 analysis

From $\varphi = 0.31$ to $\varphi = 0.80$ both optimal areas increase linearly and beyond that, the optimal areas increase exponentially. x_1 and x_2 reach a maximum of $10 \times 10^{-4} \text{ m}^2$ and $5 \times 10^{-4} \text{ m}^2$ for a load angle of $\varphi = 1.0$ corresponding to 90° . This shows that the worst loading condition for the given stress, deflection and first frequency constraints correspond to a load angle of 90° . The behavior of the optimal areas are verified with the plot of optimal function of the mass of the truss structure (objective) shown in Figure 3.26.

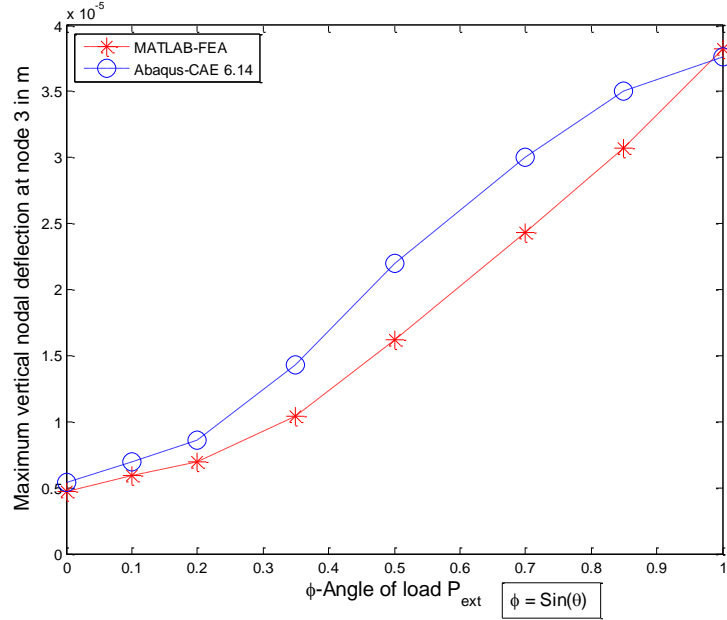


Figure 3.24: Comparison of vertical nodal deflection of the truss structure calculated at the point of application of load from MATLAB FEA code with that from Abaqus CAE 6.14

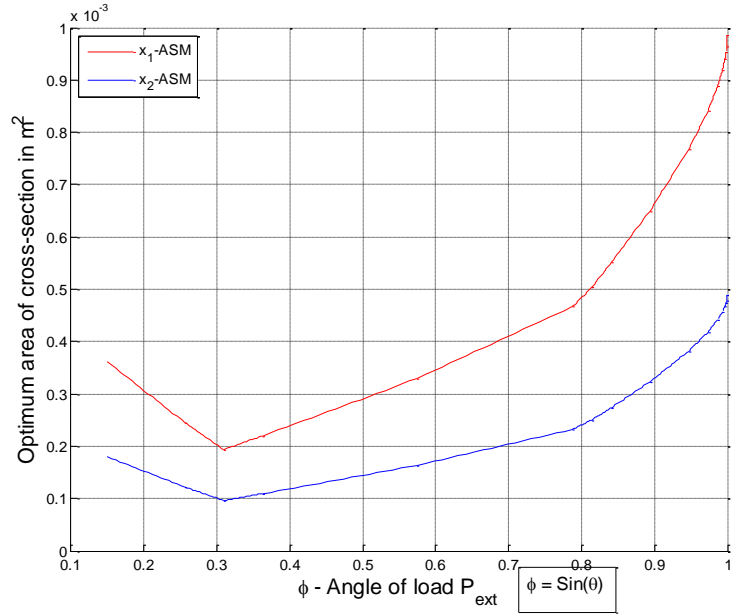


Figure 3.25: Plot of optimal area function $x_1(p)$ and $x_2(p)$ obtained from ASM algorithm for an approximation error tolerance of 0.010

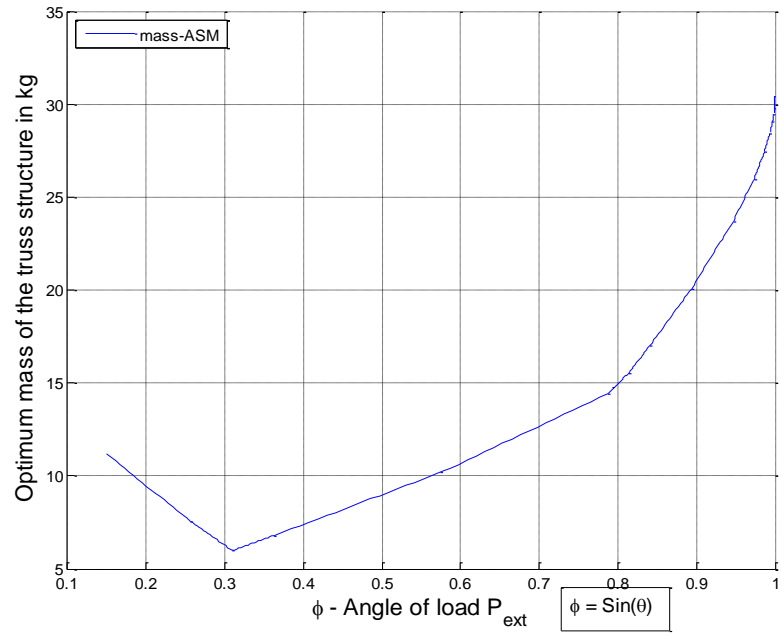


Figure 3.26: Plot of optimal objective function $mass(p)$ obtained from ASM algorithm for an approximation error tolerance of 0.010

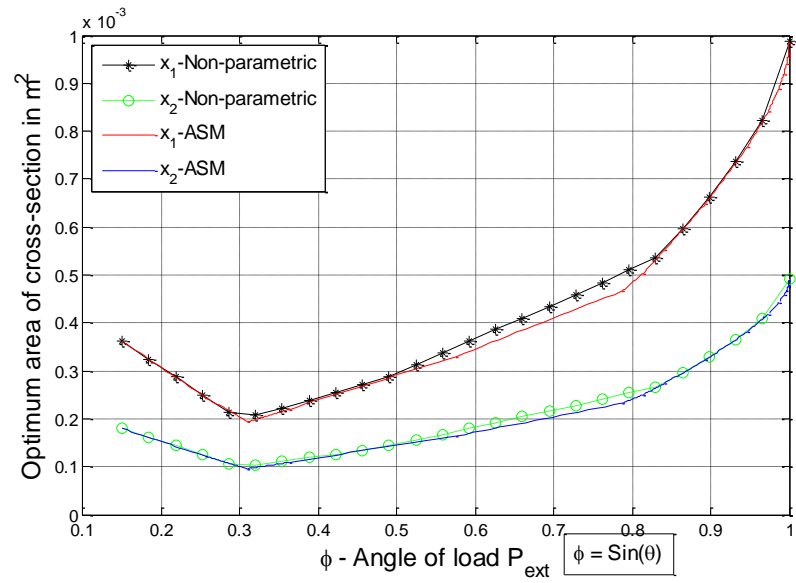


Figure 3.27: Comparison of optimal areas x_1 and x_2 obtained from parametric programming via ASM with that from non-parametric optimization at twenty five discrete points

The parametric results found using the ASM algorithm are verified with the results from the non-parametric optimization conducted at 25 equally distributed load angle values from $\varphi = 0.15$ to $\varphi = 1.0$. The plot of comparison can be found in Figure 3.27 and it is evident that the parametric results are in good agreement with the non-parametric results.

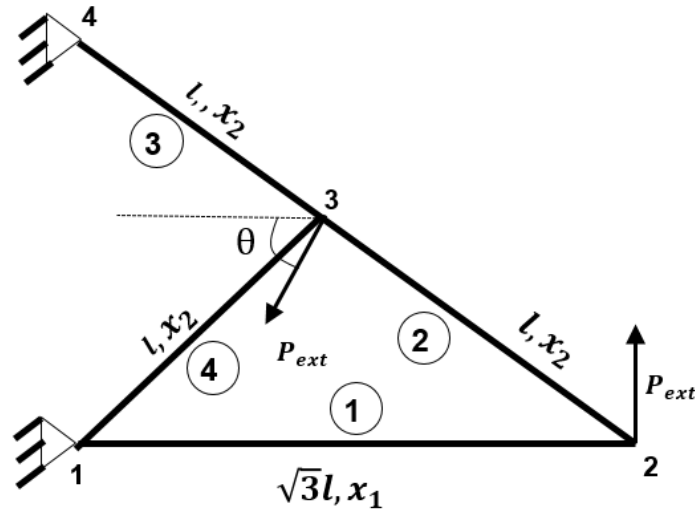


Figure 3.28: Four-bar truss structure with two dynamic loads

As a variation to the above loading condition, another vertically upward external dynamic load P_{ext} is applied at node two in addition to the external dynamic load P_{ext} varying in direction at node 3 as shown in Figure 3.28. The parametric optimization was conducted for θ varying between 5.7° and 90° . The load angle corresponding to the worst loading condition changes from 90° to 5.7° due to the addition of the external vertically upward dynamic load at node 2. It can be seen from Figure 3.29 that the parametric optimal areas of cross-sections x_1 and x_2 are at a maximum for the load angle of 5.7° and thereafter

decrease with increasing load and are at a minimum for the load angle of 90° . This shows that the position of load highly affects the load direction corresponding to the worst loading condition.

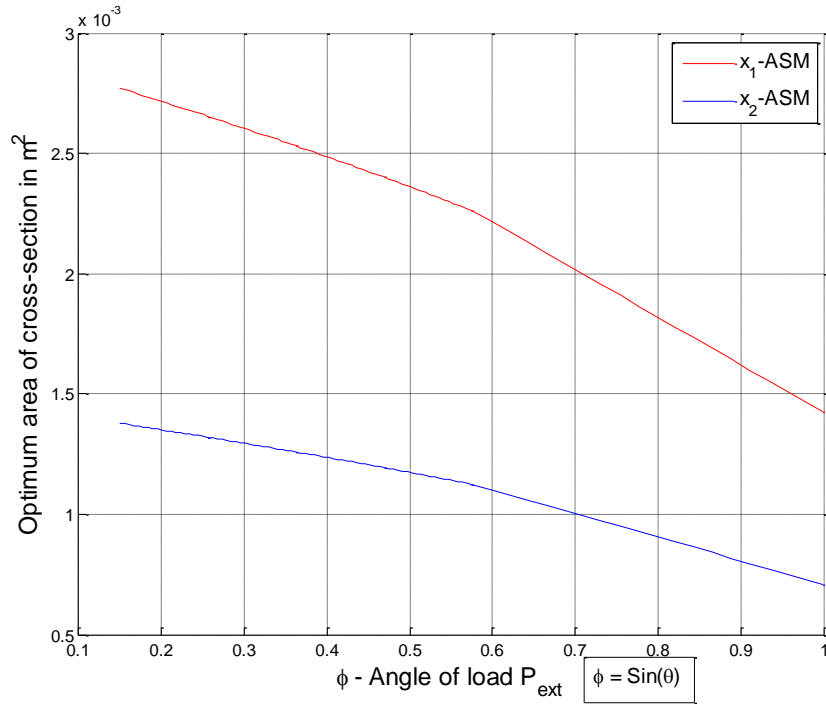


Figure 3.29: Plot of optimal area function $x_1(p)$ and $x_2(p)$ obtained from ASM algorithm for an approximation error tolerance of 0.10 (10%)

3.2 Parametric Programming As a Multi-objective Optimization Tool For Structural Optimization

One of the methods of solving the multi-objective optimization problems is by scalarizing the multi-objective optimization problem. In this method, the multi-objective optimization problem is solved as a single objective optimization problem such that the

optimal solution for the single objective problem is one of the Pareto optimal solutions for the multi-objective problem. Scalarizing a multi-objective problem can be done by many methods and the epsilon constraint (ϵ -constraint) method [30] is one such method. The interpretation of this method found in [23] is used for this research. In this method, except for the least important objective which does not have any targets to meet, all other conflicting objectives that are relatively important and have certain targets to achieve are represented as constraints of the optimization problem allowed to vary only within certain values.

The optimization problem given in Equation 3.8 is a multi-objective optimization problem with n conflicting objectives such that the vector of optimal Pareto solutions satisfy the constraint vector $g(x)$. This multi-objective optimization problem can be represented as a single objective ϵ -constraint problem [23] with m additional constraints corresponding to the m higher ranked objectives defined as constraints as given in Equation 3.9.

$$F = \min_x (f_1(x), f_2(x), \dots, f_n(x)) \quad (3.8)$$

$$\text{s.t } g_i(x) \leq 0, i = 1, 2, \dots, q$$

$$x \in X \subseteq R^p$$

$$x_{lb} < x < x_{ub}$$

where, n is the number of conflicting objectives, x is the set of decision variables, x_{lb} and x_{ub} are the lower and upper bounds for the decision variable, p represents the size of the decision variable vector, and q represents the number of constraints.

$$F = \min_x f_n(x) \quad (3.9)$$

$$\text{S.t } f_j(x) \leq \epsilon_j, j = 1, 2, \dots, m, j \neq n$$

$$g_i(x) \leq 0, i = 1, 2, \dots, q$$

$$x \in X \subseteq R^p$$

$$x_{lb} < x < x_{ub}$$

where, ϵ_j represents the constraint defined for the objective function $f_j(x)$.

The ϵ -constraint problem given in Equation 3.9 can be solved as a multi-parametric programming problem. The ϵ_j constraint values corresponding to objectives f_j can be represented as the parameters of the multi-parametric optimization problem as given in Equation 3.10.

$$F(\epsilon_1, \dots, \epsilon_{n-1}) = \min_x f_n(x) \quad (3.10)$$

$$\text{S.t } f_j(x) \leq \epsilon_j, j = 1, 2, \dots, n-1$$

$$g_i(x) \leq 0, i = 1, 2, \dots, q$$

$$x \in X \subseteq R^p$$

$$\epsilon_{lb} < \epsilon_j < \epsilon_{ub}$$

where, j is the number of parameters, ϵ_{lb} and ϵ_{ub} represent the lower and upper bounds for the constraint parameters ϵ_j .

3.2.1 Multi-Objective Optimization of Honeycomb Panel For A Given Value of Cell Angle And Corresponding Cell Height

A honeycomb design problem from [18] is chosen to define a multi-objective optimization problem. The conflicting objectives of this optimization problem are minimization of the weight of the honeycomb panel of a given length and height, maximization of the effective shear flexure modulus and the maximization of the shear strain rate of the honeycomb panel. This multi-objective optimization problem is solved as an ϵ -constraint problem through the multi-parametric programming method. Maximizing the effective shear flexure modulus and maximum effective shear strain are considered as the higher ranked objectives and thus are defined as ϵ -constraints while minimizing the weight is considered as the objective of the multi-parametric programming problem. The parameters in this multi-objective multiparametric programming problem are not design parameters like loading condition in section 3.1 instead they are the auxiliary parameters.

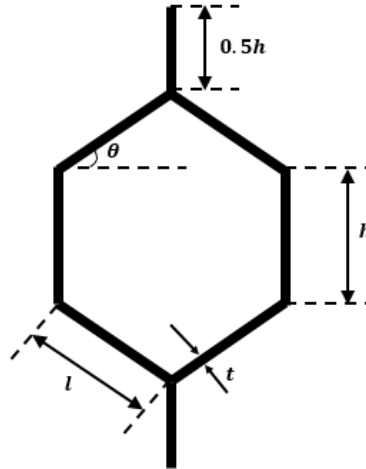


Figure 3.30: Regular hexagonal unit cell

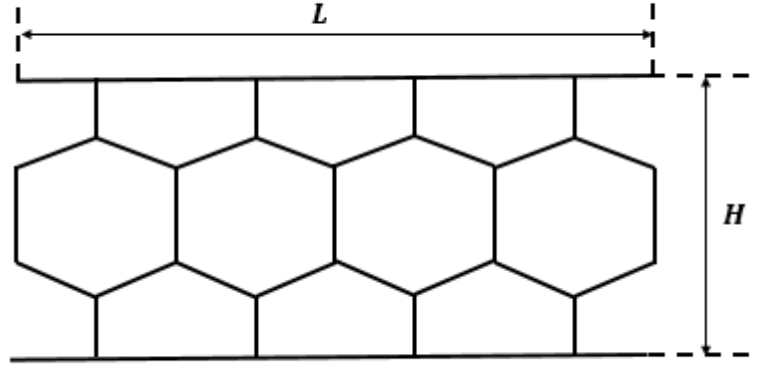


Figure 3.31: Representation of a honeycomb panel with regular hexagonal unit cells

3.2.1.1 Problem Definition

Consider a honeycomb panel with regular hexagonal unit cells of length $L = 65mm$ and height, $H = 12.7mm$. Let $N_x = 5$ be the number of unit cells along the length of the panel, $N_y = 2$ be the number of unit cells along the height of panel. Let $\theta = 30^\circ$ be the angle of the unit cell with a corresponding cell height of $h = 2.117mm$ and cell length of $l = 2.117mm$. Let $\rho = 8 \times 10^{-6} \frac{kg}{mm^3}$ be the density of the honeycomb material, and $E = 210 \times 10^3 MPa$ be the Elastic modulus of the material.

$$\min_{t, p_1, p_2} mass = \rho * N_x * N_y * [6(h * t) + 2(\frac{h}{2} * t)] \quad (3.11)$$

S.t.

$$g_1: \frac{G_{12}^*}{p_1} - 1 \leq 0$$

$$g_2: \frac{v_{12}^*}{p_2} - 1 \leq 0$$

$$4.7 \text{ MPa} < p_1 < 7.7 \text{ MPa}$$

$$0.01 < p_2 < 0.03$$

$$\text{where, } G_{12}^* = E \left(\frac{t}{l} \right)^3 * \left(\frac{\frac{h}{t} + \sin \theta}{\left(\frac{h}{l} \right)^2 \left(1 + \frac{2h}{l} \right) \cos \theta} \right)$$

$$v_{12}^* = \frac{1}{4} \left(\frac{\sigma^{yield}}{G_{12}^*} \right) * \left(\frac{t}{l} \right)^2 * \left(\frac{1}{\frac{h}{l} * \cos \theta} \right)$$

where, G_{12}^* is the effective shear modulus, v_{12}^* is the maximum effective shear strain, and $\sigma^{yield}=200 \text{ MPa}$ be the yield strength of the honeycomb material. p_1 and p_2 are the constraint values defined as parameters for G_{12}^* and v_{12}^* respectively. The lower and the upper bounds for the target of G_{12}^* are 4.7 MPa and 7.7 MPa respectively while those for the target of v_{12}^* are 1% and 3% respectively.

3.2.1.2 Multi-parametric Optimization and Results

The master ASM MATLAB code for the multi-parametric multi-objective optimization problem can be found in Appendix K. In the ASM algorithm, the allowable approximation relative error tolerance for the objective function was set as 0.125 (12.5%). For this error tolerance, the ASM algorithm segmented the parameter space into four simplexes (critical regions) with a corresponding number of optimization calls of twenty.

From Figure 3.32 and Figure 3.33 it can be seen that the Pareto optimal solution for the thickness of hexagonal unit cell varies between 0.030mm to 0.085mm in the parameter space defined as target values for G_{12}^* and v_{12}^* given in Equation 3.11.

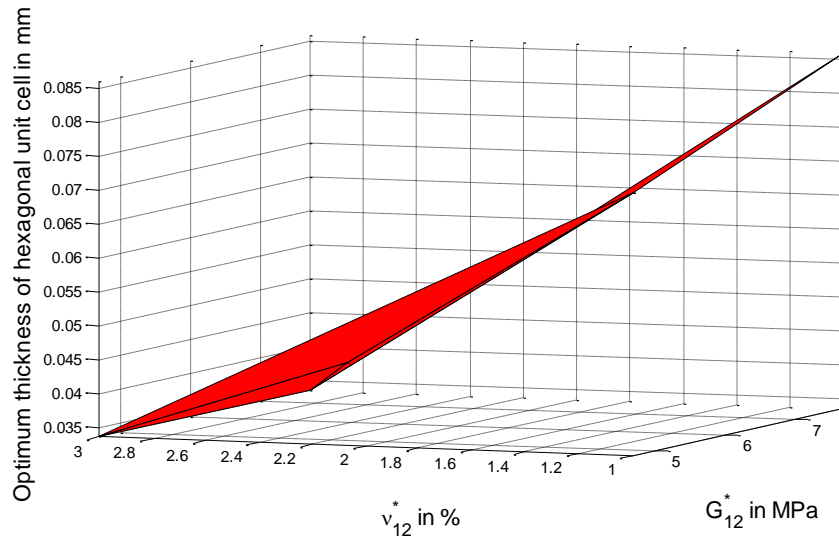


Figure 3.32: Pareto front of optimal thickness of hexagonal unit cell for an approximation relative error tolerance of 12.5% found using ASM (View 1)

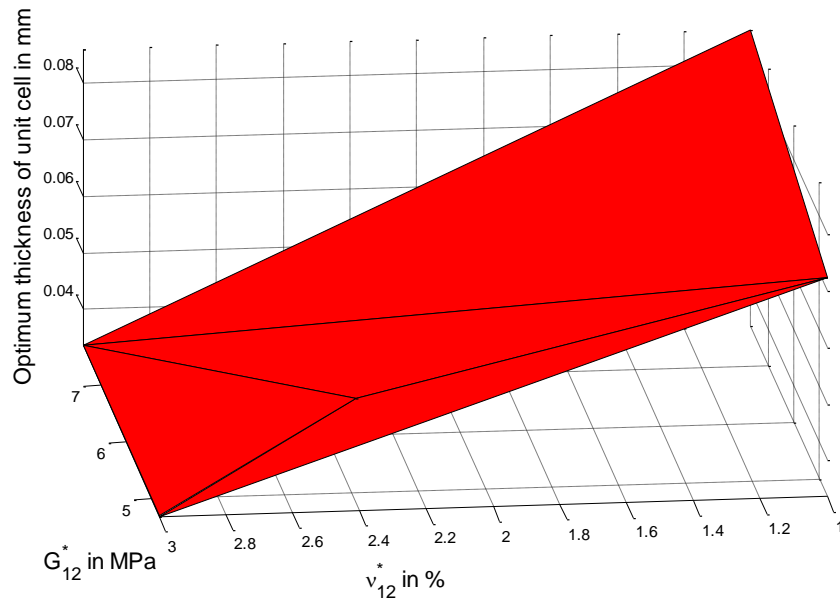


Figure 3.33: Pareto front of optimal thickness of hexagonal unit cell for an approximation relative error tolerance of 12.5% found using ASM (View 2 showing four simplexes)

It can be observed that the optimal thickness is a maximum with a value of 0.085mm for the maximum value of effective shear modulus of 7.7 MPa and minimum value of maximum effective shear strain of 1%. In contrast, for a maximum value of 3% for the maximum effective shear strain and a minimum value of 4.7 MPa for the effective shear modulus, the optimal thickness is a minimum with a value of 0.035mm.

The optimal weight of the honeycomb panel varies between $0.7 \times 10^{-4}kg$ and $1.6 \times 10^{-4}kg$ for the target parameter space defined for G_{12}^* and ν_{12}^* . The list of optimal unit cell thicknesses found through the ASM algorithm for the four critical regions and the information about the corresponding critical regions can be found in Tables 3.10 and 3.11 respectively.

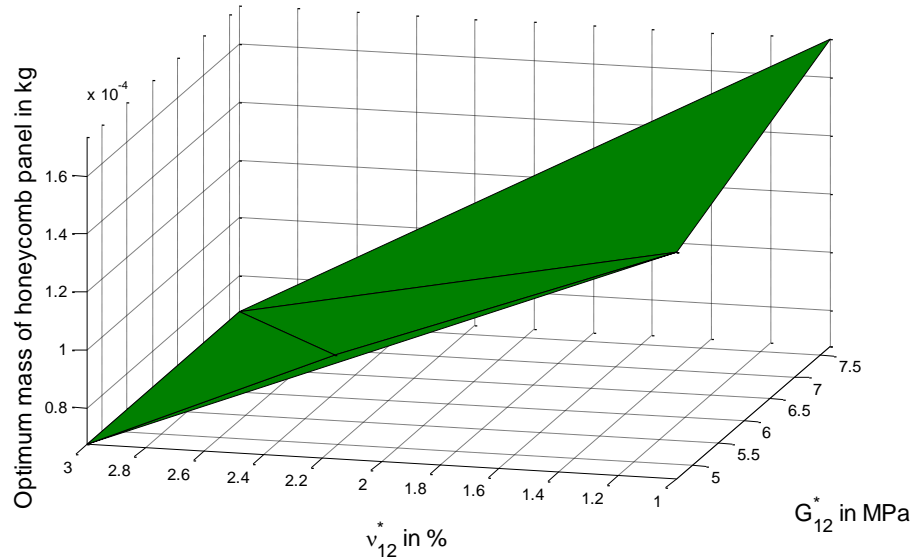


Figure 3.34: Pareto front of optimal weight of the hexagonal unit cell for an approximation relative error tolerance of 12.5% found using ASM

Table 3.10: Table of optimal parametric function of unit cell thickness with corresponding parameter intervals (critical regions) for error tolerance of 12.5%

Critical Region	Optimal unit cell thickness $h(p_1, p_2)$ in mm
Simplex 1	$(0.46p_1 - 2.62p_2 + 7.71) \times 10^{-2}$
Simplex 2	$(0.33p_1 - 2.43p_2 + 8.11) \times 10^{-2}$
Simplex 3	$(0.0p_1 - 1.44p_2 + 7.68) \times 10^{-2}$
Simplex 4	$(-0.33t_1 - 1.94t_2 + 10.73) \times 10^{-2}$

Table 3.11: List of critical regions with their associated vertices for error tolerance of 12.5%

Critical Region	Vertex 1 (p_1^1, p_2^1)	Vertex 2 (p_1^2, p_2^2)	Vertex 3 (p_1^3, p_2^3)
Simplex 1	(7.7,0.03)	(4.7,0.01)	(7.7,0.01)
Simplex 2	(5.7,0.0233)	(4.7,0.01)	(7.7,0.03)
Simplex 3	(4.7,0.03)	(5.7,0.0233)	(7.7,0.03)
Simplex 4	(4.7,0.03)	(4.7,0.01)	(5.7,0.0233)

3.2.1.3 Non-parametric Optimization and Results

For comparison, the honeycomb panel multi-objective optimization problem given in Equation 3.11 is solved using the traditional non-parametric optimization through the commercial optimization software ModeFRONTIER [27].

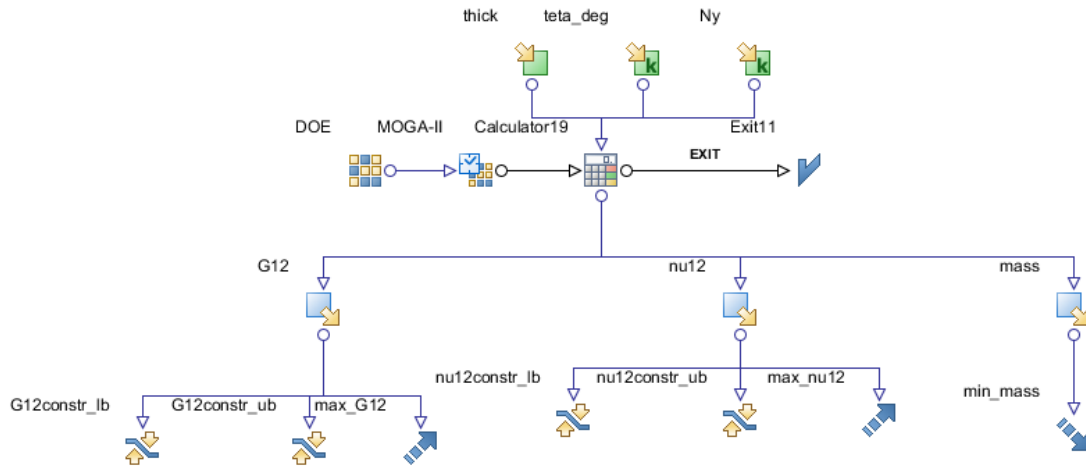


Figure 3.35: Work flow of the honeycomb panel multi-objective optimization problem setup in ModeFRONTIER

The Multi-Objective Genetic Algorithm-II (MOGA-II) optimization solver available in ModeFRONTIER is used for this purpose. MOGA-II is a fast Pareto convergence solver which uses elitism for the multi-objective search and enforces the defined constraints through objective function penalization. The variable names used in ModeFRONTIER are different from those used in the multi-parametric programming due to the limitations in the use of Greek letters and subscripts in ModeFRONTIER. For better

understanding, a list of variable names used for the objectives, constraints and constants in both multi-parametric programming and ModeFRONTIER are given below in Table 3.12.

Table 3.12: List of variable names used for objective and constraints in multi-parametric programming and ModeFRONTIER

Variables	Multi-parametric Programming	modeFRONTIER
Mass of honeycomb panel	$mass$	mass
Effective shear modulus	G_{12}^*	G12
Maximum effective shear strain	ν_{12}^*	nu12
Cell thickness	t	thick
Cell angle	θ	teta_deg
Number of unit cells along the height of panel	N_y	Ny

In this optimization method, thickness is defined as an input variable allowed to vary between 0.01mm to 0.12mm, the number of unit cells along the height of the panel and cell angle are defined as constants with values 2 and 30^0 respectively. Mass is defined as an objective to be minimized, G12 and nu12 are defined as objectives to be maximized but are also constrained between 4.7 MPa to 7.7 MPa and 0.01 to 0.03 respectively as defined in the problem definition and Equation. 3.11. All other constants are maintained the same as in multi-parametric programming. MOGA-II searches within the allowed

thickness (thick) space to find the Pareto Optimal solutions for the unit cell thickness such that the constraints and objective goals are satisfied. Such thickness values are called as feasible designs.

Table 3.13: List of 5 optimal Pareto designs in ModeFRONTIER for twenty five optimization calls

Optimum thickness ‘thick (mm)’	Effective shear modulus ‘G12 (MPa)’	Maximum effective shear strain ‘nu12 (MPa)’	Mass of the honeycomb panel Mass (kg)
0.071608	4.6922	0.01398	1.4432×10^{-4}
0.079452	6.4092	0.01288	1.6013×10^{-4}
0.079442	6.4069	0.01269	1.6011×10^{-4}
0.075888	5.5848	0.013284	1.5294×10^{-4}
0.073307	5.0342	0.013752	1.4774×10^{-4}

To maintain the same optimization calls as in Multi-parametric programming for comparison, the total number of optimization calls was set as twenty five. The number of initial user defined designs is set as five through “Random” sampling and the “Number of Generations” in MOGA-II is set to five to a get a total of twenty five optimization calls. The probabilities of directional cross-over, mutation and selection are defined as 0.5, 0.3 and 0.5 respectively. Out of the twenty five designs corresponding to twenty five optimization calls, only eight designs (thickness values) are feasible designs satisfying the

constraints and objective while the remaining seventeen designs are unfeasible. The list of optimal solutions (designs) found in the Pareto front is given in Table. 3.13. The Pareto front of the optimal thickness values can be seen in Figure 3.36 wherein the feasible designs are indicated with grey diamonds and the unfeasible designs are indicated with orange diamonds.

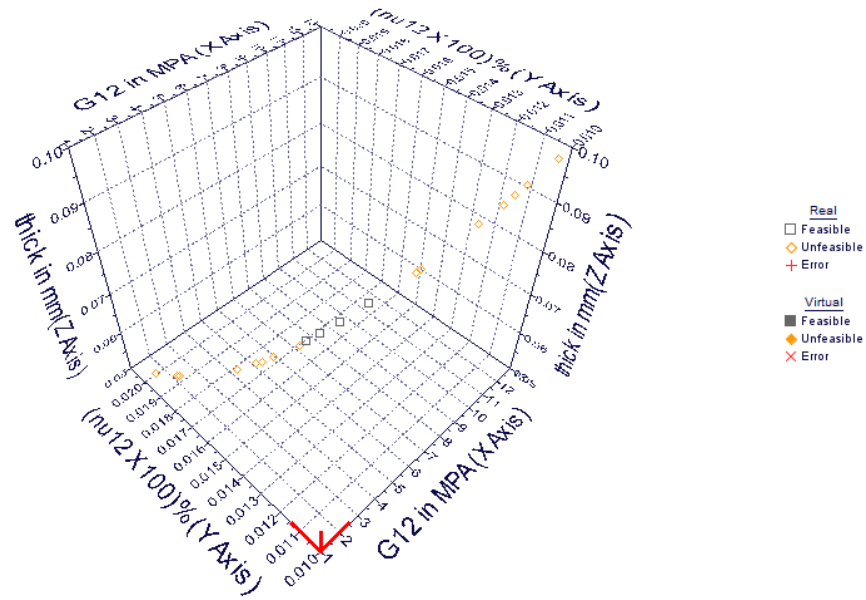


Figure 3.36: Pareto front of optimal unit cell thickness found using MOGA-II in modeFRONTIER for twenty five optimization calls

The eight feasible Pareto optimal designs are not enough to provide detailed information about the optimal solutions for the entire parameter space of G12 and nu12.

To obtain Pareto optimal solutions for a wide range of (G12, nu12) values within the defined parameter space, the number of optimization calls was increased to sixty. This resulted in thirty nine feasible designs. Since thirty nine design points is not a sufficiently

large set of optimal Pareto solutions, the number of optimization calls was again increased to 300 which lead to a total of 205 feasible designs and 95 unfeasible designs with a total computational time of 23.644s. This is a relatively large population of Pareto optimal designs corresponding to a wide range of (G12, nu12) values. The Pareto front of the optimal thickness for 300 optimization calls with 205 feasible designs is given in Figure 3.37.

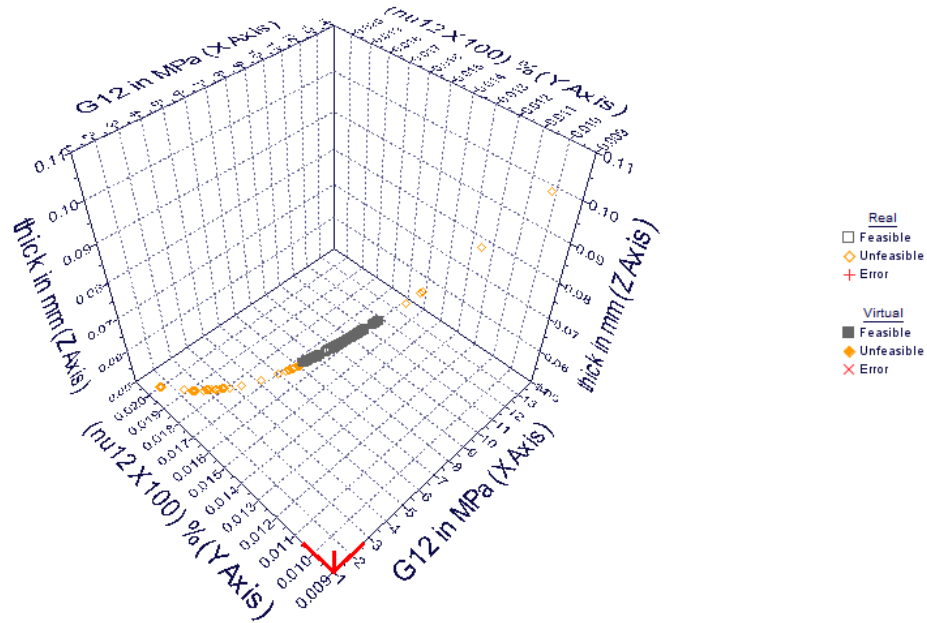


Figure 3.37: Pareto front of optimal unit cell thickness found using MOGA-II in modeFRONTIER for 60 optimization calls

3.2.1.4 Comparison of Multi-parametric Results with Non-parametric results

The accuracy of the parametric Pareto optimal results is compared with the non-parametric Pareto optimal results at eighteen (G12, nu12) feasible designs randomly chosen from the total of 205 feasible non-parametric designs.

The parametric optimal results corresponding to an error tolerance of 12.5% are used for this comparison. The critical regions corresponding to the eighteen (G12, nu12) values are identified and the parametric optimal thicknesses for the respective (G12, nu12) values are evaluated from the parametric functions given in Table 3.10. The comparison showed that the maximum error is 12.446%, while the average and median errors are 9.26% and 9.811% respectively. This shows that the error in ASM results is within the acceptable error tolerance and concurs well with the non-parametric results from ModeFRONTIER.

For an approximation error tolerance of 12.5%, the ASM required a meagre number of 22 optimization calls with four segmentations to provide the parametric optimal Pareto front for the entire parameter space of G_{12}^* and ν_{12}^* . For the same number of optimization calls, modeFRONTIER could find a meagre 8 feasible optimal Pareto designs in the entire parameter space. The number of feasible designs increased to 39 for 60 optimization calls and finally to 205 for 300 optimization calls. Though 205 feasible designs is a reasonably large set of Pareto solutions for the given parameter space of G_{12}^* and ν_{12}^* , it does not provide the Optimal Pareto information for the entire parameter space unlike the multi-parametric ASM algorithm. Moreover, the number of optimization calls required in the non-parametric optimization through ModeFRONTIER is approximately thirteen times larger than the number of optimization calls required for the ASM algorithm. The comparison of number of optimization calls, computational time and number of feasible designs is given in Table 3.14.

Table 3.14: Comparison of relative performance of ASM with ModeFRONTIER

	Parametric -ASM	Non-parametric ModeFRONTIER		
Optimization calls	22	25	60	300
Computational time (sec)	5.793	2.283	6.087	23.644
Feasible designs	NA	8	39	205

3.3 Shape Optimization Using Parametric Programming Method

3.3.1 Shape Optimization of a 2-Dimensional Cantilever Beam With Load Direction As a Parameter

The parametric programming method is used to perform shape optimization of a cantilever beam. A load of varying direction is applied at the tip of a cantilever beam of constant length and thickness. The objective of this shape optimization problem is to minimize the weight of the cantilever beam by varying the height of the beam along its length subject to constraints on maximum allowable stress in the cantilever beam and maximum allowable vertical deflection of the beam. This is achieved with the following methodology.

3.3.1.1 Methodology for Using Parametric Programming

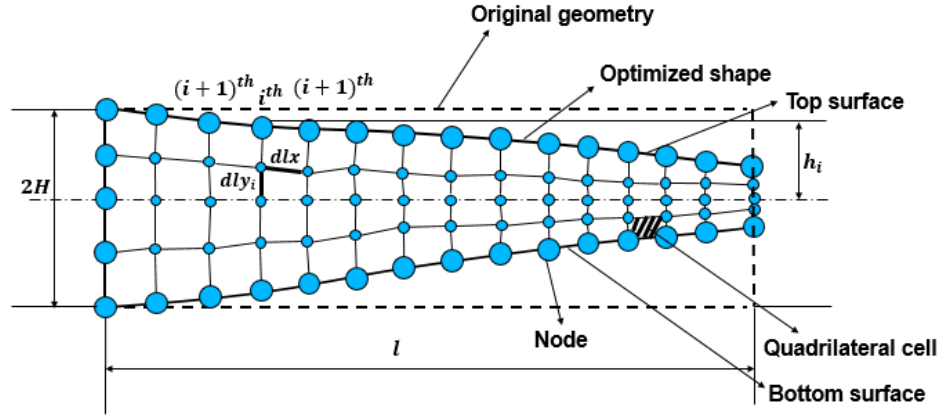


Figure 3.38: Cantilever beam geometry with discretization

Let l be the length of the cantilever beam, th be the thickness of the beam, and $2H$ be the original uniform height of the cantilever beam before shape optimization. A FEA code is developed to generate the optimized shape of the cantilever beam found using ASM. The beam geometry obtained is discretized using 2-dimensional quadrilateral cells and solved for concentrated tip load varying in direction to compute the Von-Mises stresses and nodal deflections of the beam. The MATLAB FEA code for the cantilever beam shape optimization can be found in Appendix L. The process of discretization is discussed below in detail.

Let ne_x be the number of quadrilateral cells along the length of the beam, ne_y (number of layers) be the number of cells along the height of the beam, $ne = ne_x \times ne_y$ be the total number of quadrilateral cells within the beam geometry. As the name suggests, each quadrilateral cell has four nodes. Let $n_x = ne_x + 1$ be the number of nodes along the

length of the beam on the surface of each layer of quadrilateral cells as shown in Figure 3.38.

The height h_i of the nodes on the top surface from the axis of symmetry of the 2-D cantilever beam shown in Figure 3.38 are considered as the decision variables of the parametric programming problem which means there are n_x decision variables in the optimization problem. Since the maximum height is $2H$ and since the top and bottom surface cannot coincide with the axis of symmetry of the beam, the lower bound of h_i must be always greater than $(0.05 * H)$ and the upper bound can be equal to or lesser than H . In the first implementation of the shape optimization process, the height of the nodes on the bottom surface of the beam are constrained to be symmetrically opposite to the respective nodes on the top surface. It is possible to allow the bottom surface of the cantilever beam to change in shape independent of the top surface of the beam by considering the height of the nodes on the bottom surface to be decision variables. This method is implemented following the symmetric shape optimization.

Based on the h_i passed on by the ASM algorithm to the cantilever beam FEA solver, the outer symmetrical geometry of the cantilever beam is defined. Let $dly_i = \frac{2h_i}{ne_y}$ be the distance between the intermittent nodes in a column below the i^{th} node on the top surface of the cantilever beam as shown in Figure 3.38.

3.3.1.2 Problem Definition

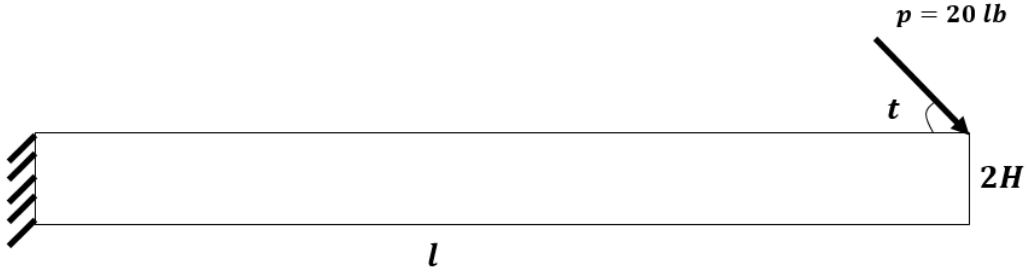


Figure 3.39: Cantilever beam with tip load varying in direction

Consider a cantilever beam of length $l = 5 \text{ in}$, thickness $th = 0.1 \text{ in}$ and initial uniform height of the cantilever beam before shape optimization is $2H = 1 \text{ in}$. Let $E = 29 \times 10^6 \text{ p.s.i}$ be the Elastic Modulus, $\rho = 0.26 \frac{\text{lb}}{\text{in}^3}$ be the density $\nu = 0.3$ be the Poisson's ratio of the cantilever beam material. Let $p = 20 \text{ lb}$ be the magnitude of the concentrated load applied at the tip of the beam.

$$\min_{x,t} \text{mass} = \rho * th * \left(\sum_{j=1}^{ne} A_j \right) \quad (3.12)$$

$$\begin{aligned} \text{S.t} \quad & \delta_{all} \leq 3 \times 10^{-3} l \\ & \sigma_{all,t} \leq 8.74 \times 10^{-4} E \\ & \sigma_{all,c} \leq 4.83 \times 10^{-4} E \\ & 0.1 \text{ in} < h_i < 0.4999 \text{ in}, i = 1, 2, \dots, n_x \\ & 0.15(8.63^\circ) < t(\theta) < 1(90^\circ) \end{aligned}$$

where $t = \sin \theta$, A_j is the area of each quadrilateral cell calculated through the FEA solver.

Let $n_x = 41$ be the number of nodes along the surface of the 2-D beam.

3.3.1.3 Parametric Results and Discussion

The analysis results from the MATLAB FEA code for shape optimization given in Appendix L is validated with the results from Abaqus CAE 6.1.4. This validation study is conducted on the original rectangular geometry of the cantilever beam given in section 3.3.1.2 for concentrated tip load of $p = 20 \text{ lb}$ for various load angles between 8.63° and 90° . The magnitude of maximum tip displacement of the cantilever beam obtained from MATLAB FEA code concurs with that from Abaqus CAE 6.1.4 at all the discrete load angles with a negligible error as shown in Figure 3.42.

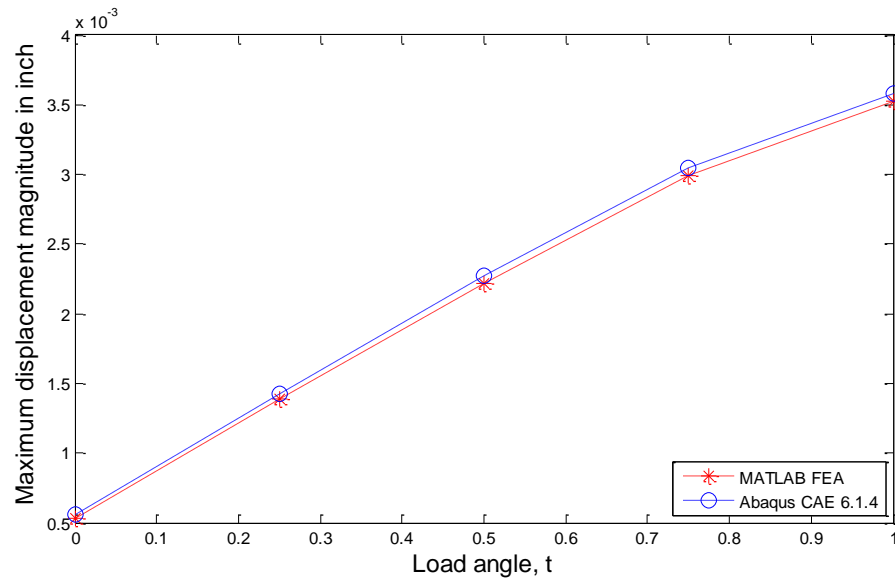


Figure 3.40: Comparison of maximum tip displacement of cantilever beam obtained from MATLAB FEA with that from Abaqus CAE 6.1.4

A constant mesh size that corresponds to ne_x of forty and ne_y of four is used for this shape optimization. For an objective function approximation error tolerance of 0.01 (1%), the Approximation Simplex Method (ASM) algorithm found the optimal shape (optimal height of the nodes on the top surface of the beam) with four critical regions, twenty one optimization calls and 11690 corresponding FEA calls. The optimal parametric function obtained for each of the forty one decision variables provides the optimal vertical position of the respective nodes on the top surface of the beam with respect to the datum over the entire interval of load direction varying from 8.63^0 to 90^0 . The optimal decision variable function for the first two of the forty one nodes is given in Table 3.15. To get the optimal shape of the beam, a MATLAB code (Appendix M) is developed to automatically extract the vertical nodal position of all the nodes on the top surface of the beam for a given value of the load angle through function evaluation and then plot both the top surface and symmetrical bottom surface of the optimized shape of the cantilever beam.

The nodes on the top surface are numbered one to forty one from the fixed end to the tip of the cantilever beam. The plot of optimal vertical position of the nodes numbered eleven to fifteen over the entire interval of load angles is given in Figure 3.41. Similarly, Figure 3.42 shows the optimal vertical position of the nodes numbered twenty one to twenty five. From the figures, 3.41 and 3.42, we can understand the behavior of optimal vertical position of individual nodes of the cantilever beam from the axis of symmetry for varying load direction. However, this does not provide any information about the optimal shape.

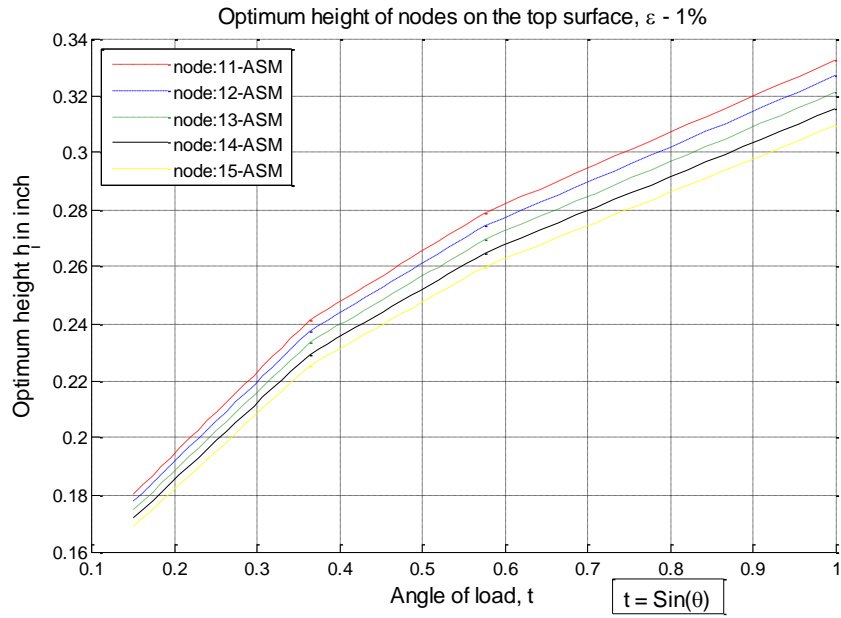


Figure 3.41: Plot of optimal height, h_i of nodes on the top surface of the beam for nodes eleven to fifteen

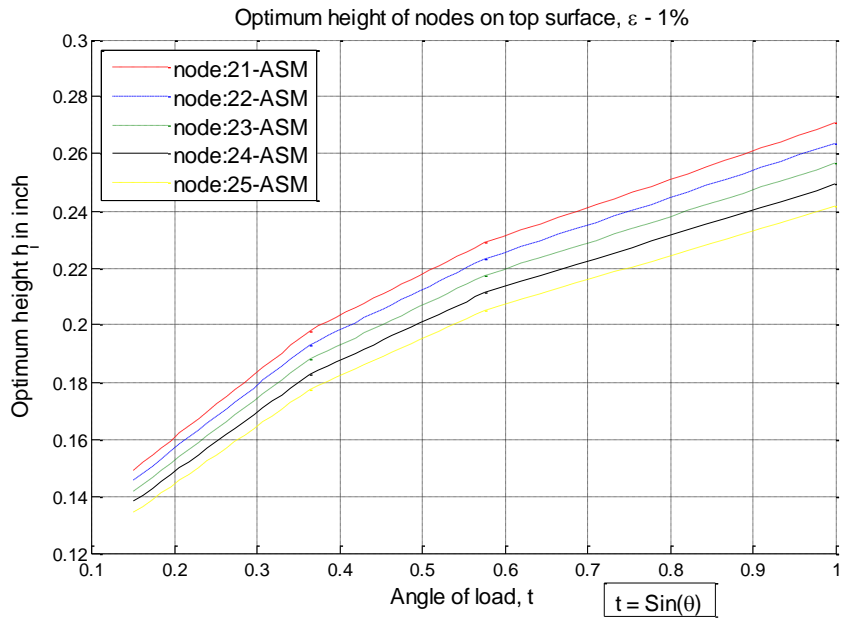


Figure 3.42: Plot of optimal height, h_i of nodes on the top surface of the beam for nodes twenty one to twenty five

Using the MATLAB FEA code given in Appendix M, the optimal shape of the beam for a load angle corresponding to the midpoint of every critical region is obtained as shown in Figure 3.43. In Figure 3.43, the red solid line connected with the asterisk symbol represents the optimal beam shape obtained for a load angle of $t = 0.89375(63.34^\circ)$ corresponding to the critical region one (refer table 3.15). Similarly, the green solid line with the plus symbol represents the optimal beam shape for load angle $t = 0.68125(42.94^\circ)$ and so on as given in Figure 3.46. In Figure 3.44, the optimal shapes of the cantilever beam for load angles 63.35° and 14.85° are shown for better understanding of the variation in the shapes with varying load direction.

Table 3.15: List of optimal nodal height of nodes one and two on the top surface of the cantilever beam

No.	Critical Region	Optimal height of node 1 $x_1(t)$ in inch	Optimal height of node 2 $x_2(t)$ in inch
1	$0.7875 < t < 1$	$0.1340t + 0.7433$	$0.1385t + 0.7511$
2	$0.5750 < t < 0.7875$	$0.1184t + 0.7556$	$0.1768t + 0.7209$
3	$0.3625 < t < 0.5750$	$0.1798t + 0.7203$	$0.2365t + 0.6866$
4	$0.15 < t < 0.3625$	$-0.1457t + 0.8383$	$0.3462t + 0.6468$

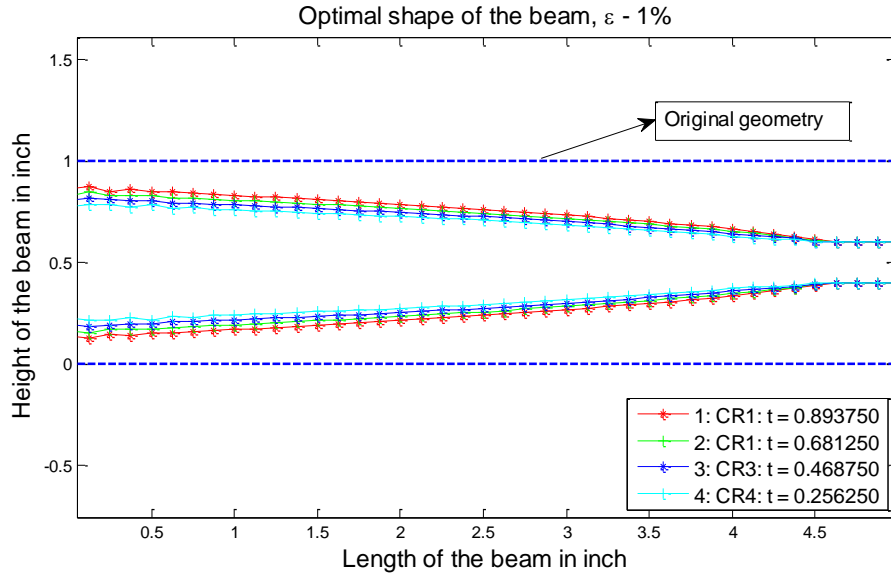


Figure 3.43: Optimal shape of the cantilever beam for various load angles, t

The height of the first node near the fixed end on the top surface of the beam for a load angle of 63.35° is $0.8631in$ and that for a load angle of 14.85° is $0.7777in$. The difference between the heights of the first node for the two different load angles is $0.0861in$. However, the difference in height of a node at a beam length of $3.66in$ from the fixed end of the beam for both the load angles is $0.041in$. The difference in height decreases along the length of the beam starting from the fixed end. This can be attributed to the larger stress values near the fixed end requiring more material and thus requiring more node height near the fixed region than away from the fixed region.

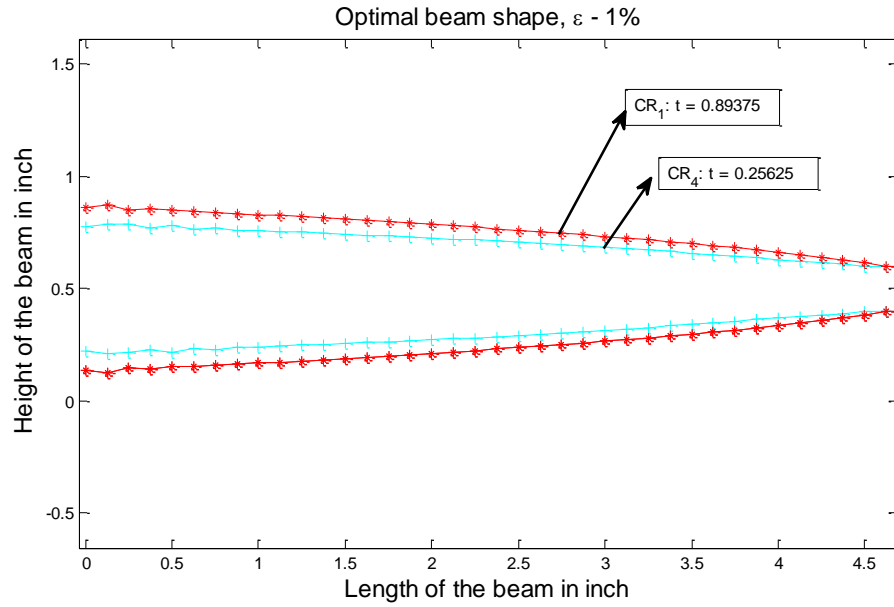


Figure 3.44: Optimal shape of the cantilever beam for load angles in critical regions one and four (Zoomed view)

As a second method, shapes of both the top and bottom surfaces of the beam are allowed to vary independently. Here, height of the nodes on both the top and bottom surface of the beam are considered as decision variables. For a mesh size of 40 X 4, there are eighty two decision variables instead of forty one unlike the symmetric shape optimization method discussed previously. The height of the all the nodes is measured from the bottom surface of the original beam shape with rectangular geometry.

For an objective function approximation error tolerance of 10%, one critical region is obtained. The optimal beam shape for a load angle of 90^0 is given in Figure 3.45. The comparison of optimal beam shapes obtained from the symmetric shape optimization with asymmetric shape optimization method for a load angle of 90^0 and an error tolerance of 10% is given in Figure 3.46.

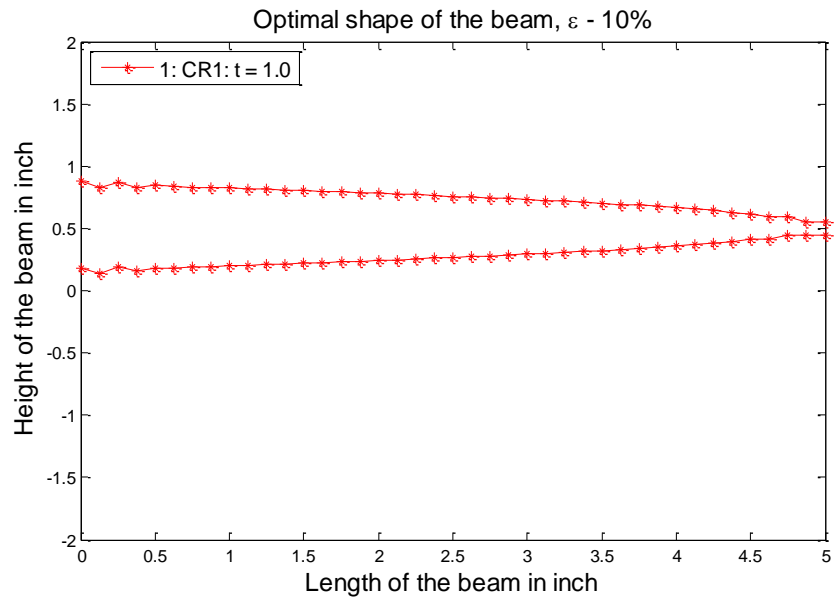


Figure 3.45: Optimal shape of the cantilever beam for load angle of 90°

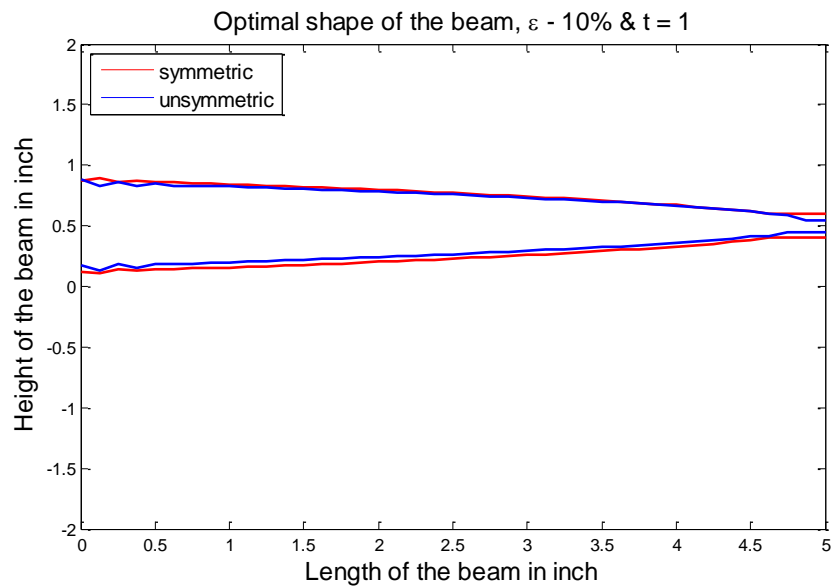


Figure 3.46: Comparison of optimal shapes of the beam for symmetric and asymmetric method

By allowing the shape of bottom surface to vary independently, ASM has removed more material near the bottom surface than the top surface as shown in Figure 3.46. This is because the bottom surface is subjected to compressive stresses and yield strength for compression is higher than that for tension.

3.3.1.4 Non-Parametric Results and Comparison

To do a comparison with the non-parametric optimization method, the shape optimization of the 2-dimensional cantilever beam problem defined in section 3.3.1.2 is solved using the commercially available Optistruct solver in HyperWorks V13.0 (student version). The mesh size of 40 X 4 used for parametric optimization is maintained the same for non-parametric optimization using HyperWorks Optistruct.

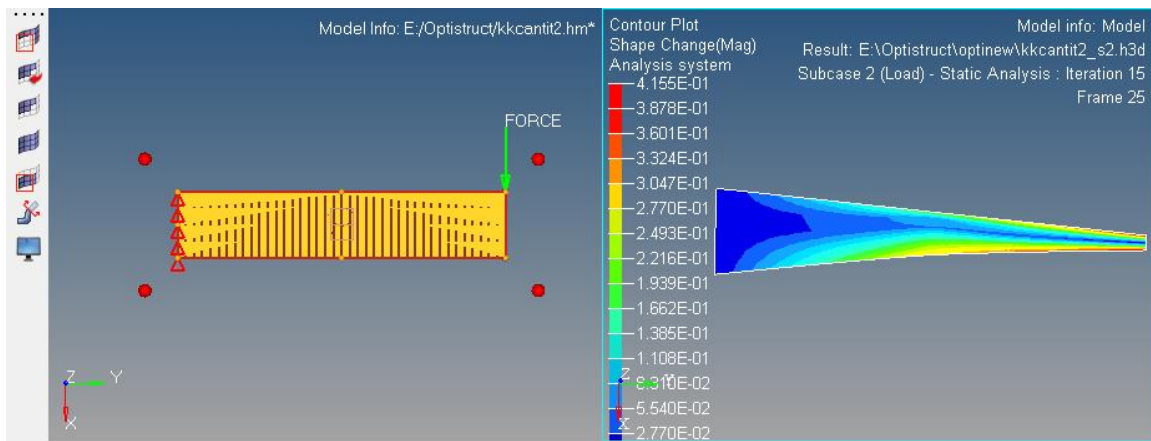


Figure 3.47: Cantilever beam with mesh, load and boundary condition (Left). Optimized beam shape with contour plot of shape change magnitude (Right)

The optimization is carried out for a single value of load direction corresponding to $t = 1.0$ (90°). The shape optimization procedure in HyperWorks is discussed below in brief.

The 2-dimensional geometry of the cantilever beam was modeled and the geometry was discretized with the above mentioned mesh details in the Hypermesh solver of HyperWorks V13.0. A vertically downward ($t = 1$) load of magnitude $20\ lb$ is applied at the tip of the beam and the left edge of the beam is given fixed boundary condition as shown in Figure 3.47 (Left). Hypermorph option is used to create morph handles, and shapes of the top and bottom surface of the beam are allowed to vary independently of one another because Hypermesh does not provide an option to maintain symmetry between the top and bottom surfaces. Responses are created to define objective, deflection constraint and stress constraints given in Equation 3.12. The optimized shape is given in Figure 3.47 (Right).

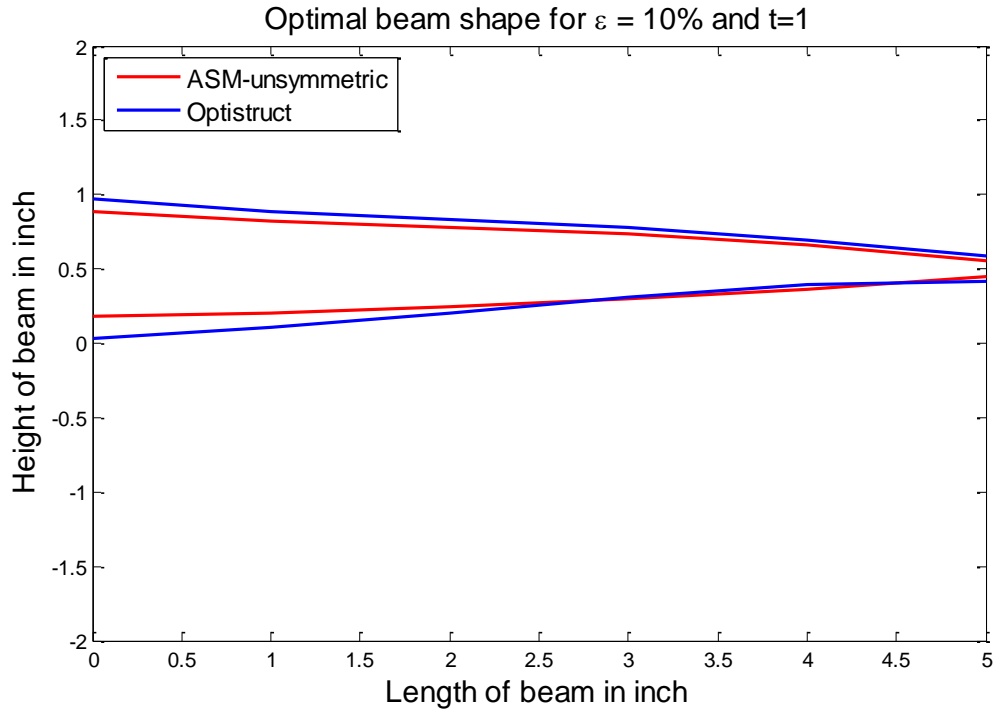


Figure 3.48: Comparison of the optimal beam shapes for $t = 1$ obtained through ASM asymmetric shape optimization with that from Optistruct

The comparison of the optimal shape of the cantilever beam obtained through ASM asymmetric shape optimization with that from Optistruct is shown in Figure 3.48. From the figure it can be observed that the nature of both the optimal shapes are in agreement with one another. The net optimal weight of the beam calculated by ASM is $0.083lb$ and that from Optistruct is $0.0913lb$.

Non-parametric shape optimization for each discrete load angle value in Hypermesh Optistruct is a tedious task. As the number of parameters in the optimization problem increases, it may not be possible to identify the worst load for complex geometries and uncertain loading conditions through non-parametric optimization at discrete parameter values in Optistruct. A detailed non-parametric optimization at a large number of parameter values spanning the entire parameter space will be computationally expensive.

4 CONCLUSION AND FUTURE WORK

4.1 Conclusion

In this thesis, the parametric programming method is used for structural optimization. No prior work of using parametric programming in structural optimization has been found in the literature and it is believed to be implemented here for the first time. A detailed case study is conducted wherein a benchmark truss optimization problem from [4] is solved to identify a suitable multi-parametric programming algorithm from among the various algorithms available in the literature to solve structural optimization problems. The Approximation Simplex Method (ASM) algorithm is found to be the most suitable multi-parametric programming algorithm. Sizing optimization of a four-bar truss for varying load direction as a single parameter is solved using the ASM algorithm for both static and dynamic load cases. The same four-bar truss problem is modified to solve a multi-parametric programming problem with both varying load magnitude and varying load direction as the two parameters of the problem. To test the ability of the ASM in handling more than two parameters, more than two design variables and the corresponding accuracy, a ten-bar truss optimization problem with two varying load magnitudes and two corresponding varying load directions is solved as a four parameter problem with ten design or decision variables. Multi-objective optimization of a honeycomb panel with three objectives namely, minimization of the mass of the honeycomb panel, maximization of the effective shear modulus and maximum effective shear strain is solved using the multi-parametric ASM algorithm by considering the objectives that are more important and have certain targets to meet, as ϵ -constraints. The least important of all is considered as the main

objective. A cantilever beam shape optimization problem is solved as a parametric programming problem with the tip load varying in direction as a parameter. The optimal shapes are obtained as functions of varying load direction for the entire parameter space.

The optimal parametric results are compared with the results from traditional non-parametric optimization with respect to the accuracy of the results, the computational expenses like number of optimization calls, number of FEA calls, and the computational time. The benefits and issues pertaining to multi-parametric programming in structural optimization is addressed. The shortcomings of the ASM algorithm are also identified and suggestions for improvement are provided.

The Multi-Parametric Toolbox 3.0 [20], multi-parametric Outer/Quadratic Approximation (mp-O/QA) algorithm and Approximation Simplex Method (ASM) algorithm are used for the case study of the benchmark four-bar truss optimization problem. It is found that the maximum error in the optimal parametric results obtained from MPT 3.0 is about 60% and that obtained from mp-Q/OA algorithm is about 24% with respect to the actual results given in [4]. The error in MPT 3.0 is mainly attributed to the incapability of MPT 3.0 to handle non-linear constraints. In contrast, mp-Q/OA can handle non-linear constraints however the linearized non-linear constraint from mp-Q/OA is in turn passed onto MPT 3.0 to find the parametric results, and thus is the error. Moreover, both MPT 3.0 and mp-Q/OA can solve problems only with constraints and objective that have analytical expressions. On the other hand, the maximum error in parametric results obtained from ASM is just 1%. ASM can handle non-linear constraints either as analytical expressions or can be coupled to a Finite Element Analysis (FEA) solver. This makes ASM to be the most

preferable multi-parametric programming algorithm for structural optimization as most of the structural problems may not have simple analytical expressions but deal with non-linear constraints and objective.

The results from the sizing weight minimization problem of the four-bar truss with varying load direction as the single parameter showed that the ASM algorithm required twelve optimization calls and just 633 FEA calls to find the optimal parametric results for an allowable error tolerance of 1% when compared to about twenty optimization calls and a corresponding 1250 FEA calls in the non-parametric optimization for a maximum error of 2%. The visual observation of the plot of optimal parametric decision variables clearly aids the designer to identify the load direction that corresponds to the worst loading condition.

When designing and optimizing a complex structure, a designer may intuitively presume certain loading conditions to be the worst and do finite element analysis for those loading conditions to identify the worst load among those loading conditions. Then optimization is conducted for the identified worst loading condition to avoid overdesign. There is a high possibility that the designer may actually fail to identify and optimize for the worst load case due to the uncertainty in the behavior of a highly complex structure. For instance, in the four-bar truss problem with uncertainty in varying load direction, through parametric programming it is identified that the worst load corresponds to a load angle of 71.8° . Identifying this with intuition is highly improbable. A detailed numerical analysis and optimization at a large number of discrete values of the parameter is computationally expensive as shown in the four-bar truss problem. The computational

expenses to do a detailed non-parametric study will increase multifold as the dimension of the parameter space increases. Thus, parametric programming can be used to identify and optimize for worst loading condition with least amount of computational time and resources in addition to providing the optimal function for the entire parameter space.

Similarly, in the four-bar truss optimization problem with two load directions as parameters, the surface plot of optimal parametric results clearly showed the area of cross-section of truss member one is larger than that for the other truss members for load angle between 5.7° and 17.46° . Thereafter, the trend reverses wherein the area of cross-section of truss member one is lesser than that for the other truss members. The optimal parametric results stands good in comparison with the non-parametric results.

The results for the ten-bar truss optimization problem with four parameters shows the computational benefits of using parametric programming more predominantly. To get a detailed information about the optimal solution for the entire four dimensional parameter space, with a minimum of ten parameter values for each parameter, non-parametric optimization required 2.1 million FEA calls in comparison with just 61659 FEA calls for parametric optimization using ASM algorithm. The computational time is 6.66 times larger for non-parametric optimization when compared to the parametric optimization for an error tolerance of 15%. This shows the enormous savings in terms of FEA calls and computational time by using parametric programming for optimization problems with higher dimensional parameter space.

The parametric programming of a four-bar truss for dynamic loading condition with load direction as a single parameter shows how an additional external load can change the worst loading condition from 5.7° to 90° . This is a clear indication that in complex systems when loading condition is uncertain, the worst load is highly unpredictable and parametric programming can come in handy to identify the worst load through lesser computational resources. The other observation made is that the optimal areas of truss members does not increase continuously with the increasing load angle. The optimal areas of cross-section of truss members decreased linearly at a larger rate with increasing load angle up to 18.66° and thereafter increased linearly with a slower increment rate up to a load angle of 53.13° and thereafter increased exponentially. This shows that the parametric programming can be helpful in easily understanding the behavior of optimal areas over a range of uncertain loading conditions.

The other important application of parametric programming is using it as a multi-objective optimization tool. The results of using parametric programming in the multi-objective optimization of a honeycomb panel revealed that the parametric programming can provide a larger pool of optimal Pareto designs spread across the entire parameter space of interest when compared to the number of Pareto designs obtained from non-parametric optimization through ModeFRONTIER. To get a larger set of optimal Pareto designs through non-parametric optimization, the number of optimization calls has to be increased resulting in higher computational time and resources. There is an error in the optimal parametric Pareto solutions obtained through ASM algorithm but since the maximum error is already known, the solution can be rescaled to compensate for the error.

Finally, the parametric programming is also used to solve a cantilever beam shape optimization problem wherein the varying load direction of the concentrated tip load is considered as a parameter of the problem. This helped in understanding the sensitivity of the optimal shape of the cantilever beam for varying load direction. In the cantilever beam problem it is obvious that the worst loading condition corresponds to the load in the vertical direction but for shape optimization problems with complex geometries and highly uncertain loading conditions, parametric programming may aid designers to explore the sensitivity of the optimal shapes for varying loading condition.

4.2 Issues Pertaining To Parametric Programming and ASM Algorithm

The Approximate Simplex Method (ASM) algorithm can handle non-linear constraints but as the degree of the non-linearity of constraints and objective increase, the number of segmentations, optimization calls and the corresponding finite element analysis (FEA) calls may increase. In certain cases where the non-linearity is very high, the ASM algorithm goes into an endless loop of segmentations until the error of approximation is lesser than the set tolerance or until the size of the newly formed critical region is lesser than the minimum allowable size defined by the user, whichever is earlier. In the current implementation of the ASM algorithm, if the set limit of the minimum allowable size for critical region is reached, the algorithm will terminate without yielding any parametric results. An improvement would be to provide parametric results for the region where the approximation error is well within the tolerance and provide information about that region of the parameter space where the error is not decreasing below the set tolerance.

The visual identification of worst load from the plot of optimal parametric objective and/or decision variable in the parameter space becomes difficult for the parametric programming problems with more than two parameters. In such cases, function evaluation of optimal objective at a series of design points in the parameter space can help identify the worst loading condition. It may seem an extra effort but the benefits gained by avoiding a large number of optimization calls and finite element analysis calls in non-parametric optimization may outweigh the cost of function evaluations.

The ASM algorithm was developed to solve convex optimization problems but certain structural optimization problems may be non-convex. Though the ASM algorithm may provide optimal parametric solutions to non-convex problems, the accuracy of the results may not be guaranteed and the solution may not be global.

4.3 Future Work

In the current work, the ASM algorithm implemented in MATLAB by Leverenz [21] is usually coupled to FEA solvers developed in-house in MATLAB R2014 to solve structural optimization problems. Writing MATLAB FEA codes for highly complex geometries and complex problems is a tedious task and it also needs validation from a commercial FEA software. This currently restricts the type of structural optimization problems that can be solved using ASM algorithm and thus restricts the scope of applications of ASM and multi-parametric programming. Since commercial FEA software is reliable and can solve linear problems, non-linear, static and dynamic problems, it is

recommended to couple a commercial FEA package like Abaqus CAE 6.1.4 to the ASM algorithm. Using python scripting language, ASM in MATLAB can be coupled to Abaqus CAE 6.1.4 which would open up the possibility of using parametric programming to solve a wide variety of problems and tap the benefits that it has to offer.

The multi-parametric programming method can be used to solve topology optimization problems similar to shape optimization problems. For example the multi-parametric ASM algorithm can be used to solve topology optimization of a 2-dimensional beam for varying loading direction as a parameter. The number of holes that have to be made in the beam geometry, relative position of each hole and the size of the holes can be decision variables while the constraints can be on the geometry such as allowable minimum center distance between the holes, in addition to constraints on stress and deflection. This may help to obtain an optimal topology that may be applicable to wide range of uncertain loading conditions instead of just optimizing for an assumed single worst loading condition. The same methodology can be extended to a variety of complex geometries with highly uncertain loading conditions.

The other possible future application of multi-parametric programming method is in solving multidisciplinary structural optimization problems. Initial work has already been conducted by Leverenz [21] in his thesis dissertation. He solved a mathematical example problem in multi-disciplinary optimization using multi-parametric Outer/Quadratic Approximation (mp-O/QA) algorithm and showed how multi-parametric programming can help reduce the computational expenses when compared to multi-disciplinary optimization using traditional non-parametric optimization. The same methodology can be extended to

solve multidisciplinary optimization problems in engineering such as in automotive industry and aerospace industry.

Generally, the cost of a multidisciplinary design optimization in the automobile and aerospace industries is very high due to the enormous computational expenses and computational time that is involved in doing routine FEA and optimization for various subsystems and individual components. For example multidisciplinary design optimization of a car body for crash worthiness and Noise, Vibration and Harshness (NVH) may take up to 300 to 400 hours even with multiple processors [22]. There is a high possibility that the computational expenses can be reduced using the parametric programming method. Even a reduction of a few hours of the computational time will lead to a reduced lead time and cost of analysis.

Thus, parametric programming seems to have a huge potential in structural optimization and it is believed that parametric programming will benefit the industry by reducing computational expenses involved in routine analysis and optimization, and also the computational time.

APPENDICES

Appendix A

Syntax For Problem Formulation In MPT 3.0

```
x = sdpvar (2, 1);    % decision variable vector
p = sdpvar (1, 1);    % parameter vector
A = [C, D; -E , 0; 0, -E ];    % constant matrix of size (m x n) where m=3 is number of
constraints and n=2 is the number of decision variables
b = [J; 0; 0];        % constant vector of size (m x 1)
F = [K; -5730; -7170]; % constant vector of size (m x t ) where t=1 is the number of
parameter(s)
M = [Ax <= b+Fp, 0 <= x , 100 <= p <= 1000 ] ;    % constraint definition, bounds
definition for decision variable vector and parameter vector
obj =  $\rho L(3x(1) + \sqrt{3}(x(2)))$ ;    % objective function
ops = sdpsettings;    %
ops.verbose = 0 ;    %
[sol , ~ , ~ , optObjective, optvar] = solvemp(F, obj, ops, p );
Plot(optvar);
Plot(optObjective);
```

Appendix B

MATLAB Code For Recursive Linear Approximation

```
clear all
clc
global E Xl Xf
E = 200e9;           % Youngs modulus
Xl = [1e-7;1e-7];    % Lower bound for decision variables
Xf = [0.5;0.5];       % Upper bound for decision variables
pl(1) = 100;          % Lower bound for parameter (Load magn in N)
pu(1) = 1000;         % Upper bound for parameter (Load magn in N)
loadnp = 100:100:1000; % size of 10;
i = 1;                % counter for no. of segmentations remaining
j = 0;                % counter for no. of critical regions found
count = 0;            % total no. of segmentations
ep = 0.05;            % termination criteria
xx = cell(0);         % Cell to store x(1) and x(2) equations
ff = cell(0);         % Cell to store f equation
CR = [];              % For set of all critical regions

% Process to find the critical intervals which satisfy termination criteria(critical regions)
% For linearization of the g1 constraint for different segment

plow(1) = 1; % to supply for initial break statement
pup(1) = 3; % to supply for initial break statement
for count=1:100
    plb = pl(count);
    pub = pu(count);
    pm = (plb + pub)/2;
    [a1m,a2m] = fmin(pm);
    % To calculate the error

    % Case 1 (in the lower bound region)
    pvar = plb;
    [a1var,a2var] = fmin(pvar);
    g1_val = gval(a1m,a2m,pm) % g(a) i.e, function val at point of approx
    gvar_val = gval(a1var,a2var,pvar) % g(x) value of non-linear g1 (actual func)
    at pvar
    bb1 = -a1m*a2m*E;
    bb2 = 2000*pm*(3*a2m+sqrt(3)*a1m);
    c_den = bb2^2;
```



```

    g1cap_val = g1_val + (((bb2)*(-a2m*E) - bb1*(2000*sqrt(3)*pm))/c_den)*(a1var-
a1m)...
        + ((bb2*(-a1m*E) - bb1*(2000*3*pm))/c_den)*(a2var - a2m)...
        + (-bb1*(2000*(3*a2m+sqrt(3)*a1m))/c_den)*(pvar-pm)
    errl = (g1cap_val - gvar_val)/gvar_val      % Calc of error
    errlow(count) = errl;

    %Case 2 (at the upper bound region)
    pvar = pub;
    [a1var,a2var] = fmin(pvar)
    gvar_val = gval(a1var,a2var,pvar)
    g1cap_val = g1_val + (((bb2)*(-a2m*E) - bb1*(2000*sqrt(3)*pm))/c_den)*(a1var-
a1m)...
        + ((bb2*(-a1m*E) - bb1*(2000*3*pm))/c_den)*(a2var - a2m)...
        + (-bb1*(2000*(3*a2m+sqrt(3)*a1m))/c_den)*(pvar-pm)
    erru = (g1cap_val - gvar_val)/gvar_val      % Calc of error;
    errup(count) = erru;

    if (abs(errl)<ep && abs(erru)<ep)
%       i = i-1
        j = j+1;
        plow(j) = plb;
        pup(j) = pub;
    elseif (abs(errl)>ep && abs(erru)>ep)
        i=i+1;
        pl(i) = plb;
        pu(i) = pm;
        i=i+1;
        pl(i) = pm;
        pu(i) = pub;
    elseif abs(errl)>ep
        %i is unchanged since one region needs segmentation and one doesnt
        i = i+1;
        pl(i) = plb;
        pu(i) = pm;
        j=j+1;
        plow(j) = pm;
        pup(j) = pub;
    elseif abs(erru)>ep
        %i is unchanged
        i=i+1;
        pl(i) = pm;
        pu(i) = pub;
        j=j+1;

```

```

        plow(j) = plb;
        pup(j) = pm;
    end

    if (j>1 && (abs(plow(j)-pup(j))) < 1*1)      % Termination criteria if interval < 1N
        break
    end
    if count == i;          %if count == i, then all segmentations are completed
        break
    end
end

nseg = j;
% To calculate initial feasible point (mid point) area and p
z = nseg;      %counter for MPT runs
ninfea = 0;    %counter for infeasible segments bumped by MPT
pos_infea = []; %array with index of infeasible segment
i = 1;
tnseg = nseg;
for count = 1:nseg
    pmid = (plow(i)+pup(i))/2;
    [a1m,a2m] = fmin(pmid);
    den_root = (2000*pmid*(3*a2m + sqrt(3)*a1m));
    bb1 = -a2m*E*(den_root) + 2000*sqrt(3)*pmid*(E*a1m*a2m);
    bb2 = -a1m*E*(den_root) + 2000*3*pmid*(E*a1m*a2m);
    bb3 = (E*a1m*a2m)/(den_root*pmid);

    A1cof1 = bb1/den_root^2;      % Co-efficient for part of g1 in matrix A in mp
    A2cof2 = bb2/den_root^2;
    b1 = -gval(a1m,a2m,pmid) + A1cof1*a1m + A2cof2*a2m + bb3*pmid;

    teta = sdpvar(1,1);
    x = sdpvar(2,1);

    A = [A1cof1,A2cof2; -200e9,0; 0,-200e9 ];      %A as in Ax<=b+Fteta % Mention as
-200e9 instead of E
    b = [b1; 0; 0];
    F = [-bb3; -5730; -7170];

    M = [A*x<=b+F*teta, x>0, plow(i) <= teta <= pup(i)]; % Feasible set containing linear
constraints for parameters and dec variables
    obj = 8000*1*(3*x(1) + sqrt(3)*x(2)); % Objective function
    %ops = sdpsettings;
    %ops.verbose = 0;

```

```

[Sol,diagnostic,~, optObjective, optVar] = solvemp(M, obj,[],teta,x);
% To overcome the problem of infeasibility
if isempty(Sol{1})
    %replace infeasible domain by a feasible domain
    tnseg = tnseg+1;
    ninfea = ninfea + 1; %counter for infeasible regions
    plowinfea(ninfea) = plow(i); %store index of infeasible segment for skipping
during plotting
    pupinfea(ninfea) = pup(i);
    plow(i) = 1.1*plow(i)
    pup(i) = 0.9*pup(i)
    i = i+1;
else
    i = i+1; %to solve for next segment
    z = z-1; %decreasing the count of segments to be solved
    [CRa, xa, fa] = processSolution(Sol);
    xx = [xx; xa];
    ff = [ff; fa];
    CR = [CR; CRa];
end
if z==0
    break
end
end

%length of regions to be cheked for plotting
CRsize = length(CR);
%Calculate nonparametric values
for i = 1:length(loadnp)
    [ar1(i),ar2(i)] = fmin(loadnp(i));
    obj(i) = 8000*1*(3*ar1(i) +sqrt(3)*ar2(i));
end
% plotting x1 and x2 variables in different figures
figure(1)
for i=1:CRsize
    plotParamSolution(xx(i),CR(i),1,[],'red')
    plotParamSolution(xx(i),CR(i),2,[],'blue')
    plot(loadnp,ar1,'-go');
    legend('x_{1} - MPT','x_{2} - MPT','x_{1}&x_{2}-Haftka');
    hold on
    title('Optimal design variable: x(t)');
    xlabel('Applied load (p) in N'); ylabel('Optimum area: x in m^2');end

```

Appendix C

MATLAB Code for mp-Q/OA Problem Formulation

Master code

```
%Definition
plow = 100;      %Lower bound of parameter in N
pup = 1000;      %Upper bound of parameter in N

%optimization problem structure
problem = struct;

dens = 8000;      %Density of material in kg/m^3
L = 1;           %Length in m
%nonquadratic objective function handle
problem.objective = @(x,p)(dens*L*(3*x(1)+sqrt(3)*x(2)));

%quadratic objective function handle
problem.quadObjective = @(x,p)(0);

%only nonquadratic functions need to have a jacobian and hessian computed
%for them at a point:
%nonquadratic objective function Jacobian evaluated at a point
problem.objJacob = @(x,p)(dens*L*[3,sqrt(3),0]);

%nonquadratic objective function Hessian evaluated at a point
problem.objHess = @(x,p)([0,0,0;0,0,0;0,0,0]);

%linear constraint matrices (inequality and equality)
%Form: A*x <= b+E*p
problem.A = [-200e9,0;0,-200e9];
problem.b = [0;0];
problem.E = [-5730;-7170];
problem.Aeq = [];
problem.Eeq = [];
problem.beq = [];

%nonlinear constraint function handle
problem.constraints = @nonlinConstraints; % must return two outputs . Hence mention
in function format
```

```

%nonlinear constraint Jacobian evaluated at a point
% dgx1n = 2000*p*(3*x(2)+sqrt(3)*x(1))*(-x(2)*200e9) +
2000*sqrt(3)*p*x(1)*x(1)*x(2)*200e9;
% dgx2n = (2000*p*(3*x(2)+sqrt(3)*x(1))*(-x(1)*200e9) +
6000*p*x(1)*x(1)*x(2)*200e9);
% dgp = (x(1)*x(2)*200e9)/(2000*(3*x(2)+sqrt(3)*x(1))*p^2);
% dgxd = ((2000*p*(3*x(2)+sqrt(3)*x(1)))^2)
problem.nonlinJacob = @(x,p)([-x(2)*200e9 + 2000*p*sqrt(3), -x(1)*200e9 + 6000*p,
2000*(3*x(2)+sqrt(3)*x(1))]);

%lower and upper bounds
problem.lb = 1e-9*ones(2,1);
problem.ub = 1*ones(2,1);

tol = 1e-1;
pSpace = Polyhedron([plow;pup]);
%varargin = [];

[xstar,fstar,CR,metrics]=mpqaMain(tol,pSpace,problem);
CRsize = length(CR);

%Obtain actual(Haftka) results
i=1;
l_bound = plow;
u_bound = pup;
inc = 100;
for P= l_bound:inc:u_bound
    X_haftka(i,1)=9.464*P/(10^6*200);
    i=i+1;
end

% Plot parametric & Haftka results
plot(l_bound:inc:u_bound,X_haftka(:,1),'-g*','linewidth',1);
for i=1:CRsize
    figure(1)
    plotParamSolution(xstar(i),CR(i),1,[],'red')
    hold on
    plotParamSolution(xstar(i),CR(i),2,[],'blue')
    hold on
end
xlabel('Load magnitude in N');
ylabel('Optimal area of cross-section in m^2');
legend('Opt-Hafka(actual)','Opt-x{ 1}','Opt-x{ 2}');

```

Sub-Function : mpqaMain

```

function [xstar,fstar,CR,metrics] = mpqaMain(tol,pSpace,problemData,varargin)
%This function solves a general multi-parametric convex optimization
%problem (mp-nlp) using the general parametric algorithm of
%Pistokopoulos et al (2010,2012)
%-----
%INPUT      TYPE      DESCRIPTION
%tol:       scalar     error tolerance
%pSpace:    polyhedron  parameter space
%problemData: struct    mp-NLP information
%-----
% VARIABLE INPUT
% 1:        vector     optional initial point of approximation
%              variables AND parameters
% 2:        scalar     optional for graphing parameter space
%              during algorithm
%-----
%OUTPUT
%xstar:     cell        optimal decision variable functions
%fstar:     cell        optimal objective function
%CR:        polyhedron vector  partition of parameter space
%metrics:   struct      solution information
%-----
%METRICS FIELDS:
%error:     sum of largest single point errors for each critical region
%intError:  approximate error integrated over entire parameter space
%nlp:       total number of NLP problems solved
%mpqps:     total number of mp-quadratic problems solved
%tol:       maximum error tolerance used
%-----

%optional initial solution
if isempty(varargin) || isempty(varargin{1})
    x0 = problemData.lb;
    p0 = pSpace.chebyCenter.x;
    u0 = [x0;p0];
else
    u0 = varargin{1};
    if isrow(u0)
        u0 = u0';
    end
end
end

```

```

%optional graphing of parameter space
if length(varargin) <= 1 || pSpace.Dim > 3
    graphyes = 0;
else
    pSpace.plot('linewidth',1)
    drawnow
    pause(0)
    hold on
    graphyes = 1;
end
%=====
=====
%Solve the NLP treating the parameter as a decision variable
%u = [x;p]
n = length(x0);
u = relaxedProblem(n,u0,pSpace,problemData);
xHat = u(1:n);
pHat = u(n+1:end);
%=====
=====

%=====
=====
%Construct a quadratic approximation of the problem and solve over the
%parameter space
A = problemData.A;
b = problemData.b;
E = problemData.E;
[xstar,fstar,CR,metrics] =
approximateProblem(xHat,pHat,A,b,E,tol,pSpace,problemData,graphyes);
metrics.nlps = metrics.nlps + 1;
metrics.tol = tol;
%=====
=====

end

function u = relaxedProblem(n,u0,pSpace,data)
%Solves the mp-nlp treating parameters as variables

%INPUT
%n:      number of decision variables
%u0:     initial solution
%pSpace: parameter space

```

```

%data:    optimization problem data (see mpqaMain for details)

%OUTPUT
%u:       optimal solution note that u = [x;p]
pA = pSpace.A;
pb = pSpace.b;
m = length(pb);

if isempty(data.A)
    A = [zeros(m,n),pA];
    b = pb;
else
    A = [data.A, -data.E;
         zeros(m,n),pA];
    b = [data.b;pb];
end
Aeq = [data.Aeq, -data.Eeq];
beq = data.beq;
lb = [data.lb;min(pSpace.V)'];
ub = [data.ub;max(pSpace.V)'];
obj = @(x)nlpObjective(x,n,data.objective,data.quadObjective);
if isempty(data.constraints)
    const = [];
else
    const = @(x)nlpConstraints(x,n,data.constraints);
end
options = optimoptions('fmincon','Algorithm','interior-point','Display','Off');
u = fmincon(obj,u0,A,b,Aeq,beq,lb,ub,const,options);
options = optimoptions('fmincon','Algorithm','sqp','Display','Off');
u = fmincon(obj,u,A,b,Aeq,beq,lb,ub,const,options);
end

function f = nlpObjective(u,n,func,quadfunc)
%objective function for mp-nlp with parameters treated as variables
x = u(1:n);
p = u(n+1:end);
f = func(x,p)+quadfunc(x,p);
end

function [c,ceq] = nlpConstraints(u,n,func)
%Nonlinear constraints for mp-nlp with parameters treated as variables
x = u(1:n);
p = u(n+1:end);
[c,ceq] = func(x,p);

```


End

Sub-Function : approximateProblem

```
function [xOpt,fOpt,CROpt,metrics] = ...
    approximateProblem(xHat,pHat,A0,b0,E0,tol,pSpace,data,graphyes)
% This function creates and solves a quadratic approximation of the mp-nlp
%(quadratic objective, linear constraints) and checks error at vertices of
%each critical region. If error is too large the space is partitioned and
%a new approximation is made and solved.
%-----
%INPUT          TYPE          DESCRIPTION
%xHat:          column vector  variable point of approximation
%pHat:          column vector  parameter point of approximation
%fhat:          scalar         f(xhat,phat)
%A0:            matrix         coefficient matrix for linearized
%               constraint set
%b0:            column vector  RHS for linearized constraint set
%E0:            matrix         parameter matrix for linearized
%               constraint set
%tol:           scalar         error tolerance
%pSpace:        polyhedron     parameter space
%data:          struct         mp-NLP information
%-----
%OUTPUT
%xOpt:          cell           optimal decision variables
%fOpt:          cell           optimal objective function
%CRopt:         polyhedron vector partition of parameter space
%metrics:       struct         solution information
%-----
n = length(xHat);
CROpt = [];
xOpt = cell(0);
fOpt = cell(0);

%minimum allowable region radius
%rtol = tol/10;
%=====
%Evaluation Metrics
%=====
metrics = struct;
metrics.error = 0;
metrics.intError = 0;
```

```

metrics.nlps = 0;
metrics.mpqps = 1;
%=====

%=====
%Nonquadratic Objective Value, Jacobian, and Hessian
%at (xhat,phat)
%=====
H = data.objHess(xHat,pHat);
J = data.objJacob(xHat,pHat);
fhat = data.objective(xHat,pHat);
%=====

%=====
%Linear approximation of nonlinear constraints
%=====
if isempty(data.constraints)
    A = A0;
    b = b0;
    E = E0;
else
    [An, bn, En] = constraintApproximation(xHat,pHat,data);
    C = unique([A0, b0, E0;An, bn, En],'rows');
    A = C(:,1:n);
    b = C(:,n+1);
    E = C(:,n+2:end);
end

%=====

%=====
%mp-QP solution to approximate problem
%=====
[CR,x,f] =
mpqaProblem(xHat,pHat,fhat,H,J,A,b,E,pSpace,data.quadObjective,data.lb,data.ub);
%=====

%visually show the split of the simplex if desired
if graphyes
    CR.plot('linewidth',1,'ColorOrder','random')
    drawnow
    pause(0)
end

```

```

CRnumber = length(CR);

%Evaluate critical regions to determine if approximate solution is within
%specified error tolerance
for j = 1:CRnumber

    %Obtain all vertices of critical region
    vSet = CR(j).V;
    vNumber = size(vSet,1);

    %=====
    %Check feasibility at vertices (only checks inequality constraints)
    %Add additional linearizations if necessary

    %=====
    disp('!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!')
    disp(['Critical Region ' num2str(j) ' of ' num2str(CRnumber) ' is checked'])
    disp(['for approximation at t = ' num2str(pHat) ' and x = ' num2str(xHat)])
    %disp(['Region has Volume = ' num2str(CR(j).volume)])
    disp('!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!')
    disp(' ')

    if ~isempty(data.constraints)
        [An, bn, En] = feasibilityCheck(vSet,x{j},CR(j).A,CR(j).b,tol,data);
        if ~isempty(An)
            infeasible = 1;
            metrics.nlps = metrics.nlps + length(bn);
        else
            infeasible = 0;
        end
    else
        infeasible = 0;
        An = []; bn = []; En = [];
    end

    %=====
    %Check optimal f accuracy at vertices

```

```

%=====
=====
    [maxError,totalError,avgError] = errorCheck(vSet,vNumber,x{j},f(j,:),data);
    metrics.nlps = metrics.nlps + vNumber;

%=====
=====

%=====
=====
    % Make new approximation if not good enough

%=====
=====
    if ((maxError > tol) || (infeasible == 1)) %&& (CR(j).volume > rtol)

%=====
=====
        % new approximation at center if pHat is not the center
        % or if additional linearizations added to approximation

%=====
=====
        if (pHat ~= CR(j).chebyCenter.x) %norm(pHat-CR(j).chebyCenter.x) > 1e-5
%            disp(['New approximation of region made at center: t = '
num2str(CR(j).chebyCenter.x)])
%            disp(' ')

            [xNew,fNew,CRnew,newMet] =
newApprox(CR(j),[A;An],[b;bn],[E;En],tol,data,graphyes);
            %update evaluation metrics
            metrics.error = metrics.error + newMet.error;
            metrics.intError = metrics.intError + newMet.intError;
            metrics.nlps = metrics.nlps + newMet.nlps;
            metrics.mpqps = metrics.mpqps + newMet.mpqps;
            %visually show the simplex is optimal if desired
            if graphyes
                CRnew.plot('linewidth',1,'color','white')
                drawnow
                pause(0)
            end
            %Update solution to combine new and old CR info

```

```

        CROpt = [CROpt;CRnew];
        xOpt = [xOpt; xNew];
        fOpt = [fOpt; fNew];

%=====
=====
        %o/w split region and solve for each subregion

%=====
=====
        else %if (pHat == CR(j).chebyCenter.x)
%            disp('Partition region at center')
%            disp(' ')

            splitCR = splitRegion(CR(j));

            %visually show the split if desired
            if graphyes
                splitCR.plot('linewidth',1,'ColorOrder','random')
                drawnow
                pause(0)
            end

            for ij = 1:length(splitCR)

                [xNew,fNew,CRnew,r1met] =
newApprox(splitCR(ij),[A;An],[b;bn],[E;En],tol,data,graphyes);

                %update evaluation metrics
                metrics.error = metrics.error + r1met.error;
                metrics.intError = metrics.intError + r1met.intError;
                metrics.nlps = metrics.nlps + r1met.nlps;
                metrics.mpqps = metrics.mpqps + r1met.mpqps;

                %Update solution to combine new and old CR info
                CROpt = [CROpt; CRnew];
                xOpt = [xOpt; xNew];
                fOpt = [fOpt; fNew];
            end
        end

%=====
=====

```

```

else %Keep current CR results
    metrics.error = metrics.error + totalError;
    %metrics.intError = metrics.intError + avgError*CR(j).volume;
    %visually show the simplex is optimal if desired
    if graphyes
        CR(j).plot('linewidth',1,'color','white')
        drawnow
        pause(0)
    end
    CROpt = [CROpt; CR(j)];
    xOpt = [xOpt; x(j)];
    fOpt = [fOpt; f(j,:)];
end

%=====
=====
end
end

function [An, bn, En] = feasibilityCheck(v,x,polyA,polyb,tol,data)
%checks the value of nonlinear constraints at the approximate solution x
%evaluated at parameter vertices v
xn = size(x,1);
[vn,vm] = size(v);
constraintViolations = cell(vn,1);
totalViolatedConstraints = 0;
maxViolation = -inf;
%determine constraint value at each vertex of critical region
%identify all violated constraints, if any
for i = 1:vn
    p = v(i,:);
    [g,~] = data.constraints(x*[p';1],p);
    constraintViolations{i} = find(g>tol);
    maxViolation = max([maxViolation;g]);
    totalViolatedConstraints = totalViolatedConstraints + length(constraintViolations{i});
end
%initialize new constraint linearizations:  $A_n * x \leq b_n + E_n * p$ 
%if max constraint value is too large, determine closest feasible parameter
if totalViolatedConstraints > 0
    An = zeros(totalViolatedConstraints,xn);
    bn = zeros(totalViolatedConstraints,1);
    En = zeros(totalViolatedConstraints,vm);
    pointer = 1;
    for j = 1:vn

```

```

    if ~isempty(constraintViolations{j})
        addedC = length(constraintViolations{j});
        pInfeas = v(j,:);
        xInfeas = x*[pInfeas';1];
        [xFeas,pFeas] = makeFeasible(xInfeas,pInfeas',v,polyA,polyb,data);
        [Aj, bj, Ej] = constraintApproximation(xFeas,pFeas,data);
        % disp(['New linear approximations added for nonlinear constraints '
        num2str(constraintViolations{j})])
        % disp(['at t = ' num2str(pFeas) ' and x = ' num2str(xFeas)])
        % disp(' ')
        An(pointer:pointer+addedC-1,:) = Aj(constraintViolations{j},:);
        bn(pointer:pointer+addedC-1) = bj(constraintViolations{j});
        En(pointer:pointer+addedC-1,:) = Ej(constraintViolations{j},:);
        pointer = pointer+addedC;
    end
end
else
    An = [];
    bn = [];
    En = [];
end
end
end

```

```

function [maxfError,totalError,avgError] = errorCheck(v,vNum,x,f,data)
% determines the objective function value error for the quadratic
% approximation solution at each vertex in a critical region
objSoln = zeros(vNum,1);
approxf = zeros(vNum,1);
for i = 1:vNum
    p = v(i,:);
    x0 = x*[p';1];
    [~, objSoln(i)] = optimizationProblem(x0,p',data);
    approxf(i) = p*f{1}*p' + f{2}*p' + f{3};
end
errorf = abs(objSoln - approxf);
maxfError = max(errorf);
totalError = sum(errorf);
avgError = mean(errorf);
end

```

```

function [xs,fs,CRs,metric1] = newApprox(CR,A,b,E,tol,data,graphyes)
% Construct a new quadratic approximation of the problem and solve over the
% smaller critical region
pNew = CR.chebyCenter.x;

```

```

xNew = optimizationProblem(data.lb,pNew,data);
[xs,fs,CRs,metric1] = approximateProblem(xNew,pNew,A,b,E,tol,CR,data,graphyes);
metric1.nlps = metric1.nlps+1;

```

```

end

```

Sub-Function: constraintApproximation

```

function [A, b, E] = constraintApproximation(x,p,data)
%Creates a set of linear approximations to the nonlinear constraints in an
%mp-nlp at a specific point x and parameter value p.
%linear approximation:  $g(u^*) + \text{gradient}(g(u^*))' * [u - u^*] \leq 0$ 
%for  $u = [x;p]$ 
%Output gives  $Ax \leq b + E * p$ 
%-----
%INPUT   TYPE           DESCRIPTION
%x:      column vector   variable point of approximation
%p:      column vector   parameter point of approximation
%data:   struct          mp-NLP information
%-----
%OUTPUT
%A:      matrix          coefficient matrix for linearized constraint set
%b:      column vector   RHS for linearized constraint set
%E:      matrix          parameter matrix for linearized constraint set
%-----
rounding = 10000;

if isempty(data.nonlinJacob)
    b = [];
    A = [];
    E = [];
else
    %nonlinear Jacobian evaluated at (x,p)
    J = data.nonlinJacob(x,p);
    J = round(J*rounding)/rounding;
    %Evaluate nonlinear constraints at [x;p]
    %currently only nonlinear inequalities are considered
    [b0,~] = data.constraints(x,p); %  $g(u^*)$ 
    b1 = J*[x;p]; %  $\text{gradient}(g(u^*))' * (u^*)$ 
    %constant:  $b = g(u^*) - \text{gradient}(g)' * (u^*)$ 
    b = (b0 - b1);
    b = round(b*rounding)/rounding;

```



```

%Ax + E*p + b <= 0
%J = [A E]
%coefficients for variables
n = length(x);
A = J(:,1:n);
%coefficients for parameters
E = J(:,n+1:end);
%move b and E to other side of inequality
b = -b;
E = -E;
end
end

```

Sub-Function: mpqaProblem

```

function [CR,xstar,fstar] = mpqaProblem(xs,ps,f,H,J,A,b,E,space,quadObj,lb,ub)
%Solves the multiparametric quadratic approximation (mp-qa) problem
%for approximation made at (xs,ps)
%If a solution is not returned for some reason, the quadratic terms are
%dropped and a linear approximation is solved instead
%-----
%INPUT      TYPE      DESCRIPTION
%xs:        column vector  variable point of approximation
%ps:        column vector  parameter point of approximation
%f:         scalar         f(xhat,phat)
%H:         matrix         objective Hessian evaluated at [xs,ps]
%J:         row vector     objective Jacobian evaluated at [xs,ps]
%A:         matrix         constraint coefficients for variables
%b:         column vector   RHS for linear constraints
%E:         matrix         constraint coefficients for parameters
%space:     polyhedron     parameter space
%lb:        row vector     lower bounds for variables
%ub:        row vector     upper bounds for variables
%-----
%OUTPUT
%CR:        polyhedron vector  partition of parameter space
%xstar:     cell            optimal decision variables
%fstar:     cell            optimal objective function
%-----

n = length(xs);
m = length(ps);

```

```

%express parameter space as inequalities
PCb = space.b;
PCA = space.A;

%create decision variables and parameters
z = sdpvar(n,1);
t = sdpvar(m,1);

%specify feasible region for variables and parameters
if isempty(A)
    F = [lb <= z <= ub, PCA*t <= PCb];
else
    F = [A*z <= b+E*t, lb <= z <= ub, PCA*t <= PCb];
end

%specify objective function
obj = .5*([z;t]-[xs;ps])*H*([z;t]-[xs;ps]) + J*([z;t]-[xs;ps]) + f + quadObj(z,t);

%set options for YALMIP
ops = sdpsettings;
ops.verbose = 0;

sol = solvemp(F,obj,ops,t);

%if no solution is returned check if the space is infeasible and then solve
%using a linear approximation of objective
if isempty(sol) || isempty(sol{1})
    P = Polyhedron(F);
    if P.isEmptySet()
        disp('No solution returned by mpt: Infeasible space.')
    else
        disp('No solution returned by mpt. Space is Feasible.')
    end
    pause
end

%get required elements of solution and put them into proper format
[CR,xstar,fstar] = processSolution(sol);
End

```

Sub-Function: splitRegion

```

function P = splitRegion(CR,varargin)
%partitions the simplex CR by connecting each facet of CR with a

```

```

%point in CR
%uses specified point if given, otherwise uses the Chebychev Center
%-----
%INPUT          TYPE          DESCRIPTION
%CR:            polyhedron     parameter space
%varargin:      row vector     point to use to split region
%-----
%OUTPUT          TYPE          DESCRIPTION
%P:            polyhedron array parameter space partition
%-----
if isempty(varargin)
    center = mean(CR.V)';
    %center = CR.chebyCenter.x;
else
    center = varargin{ 1 };
end
CR.minHRep;
F = CR.getFacet();
m = length(F);
P(m) = Polyhedron([]);
for i = 1:m
    FV = F(i).V;
    Ptemp = Polyhedron([FV;center]');
    Ptemp.minVRep;
    if Ptemp.isFullDim
        P(i) = Polyhedron([FV;center]');
    end
    clear Ptemp
end
P(P.isEmptySet)=[];
if isempty(P)
    CR.V
    disp(num2str(center))
    %pause
end
end

```

Sub-Function: plotParamSolution

```

function U = plotParamSolution(x,CR,varargin)
%graphs the piecewise function x with critical region CR
%-----
%INPUT          TYPE          DESCRIPTION

```

```

%x:          cell          function
%CR:         polyhedron array    function space partition
%-----
%VARARGIN Inputs
% 1: plotting properties: line width and color
% 2: subset of variables to plot
%-----
n = size(x,2);
n1 = length(CR);
n2 = length(varargin);
switch n2
    case 0
        varSubset = 1;
        linewidth = 1;
        color = 'black';
    case 1 %subset indicator
        if isempty(varargin{1})
            varSubset = 1; %plot all
        else
            varSubset = 0; %plot subset
            plotIndex = varargin{1};
        end
        linewidth = 1;
        color = 'black';
    case 2 %linewidth indicator
        if isempty(varargin{1})
            varSubset = 1; %plot all
        else
            varSubset = 0; %plot subset
            plotIndex = varargin{1};
        end
        if isempty(varargin{2})
            linewidth = 1;
        else
            linewidth = varargin{2};
        end
        color = 'black';
    case 3 %color indicator
        if isempty(varargin{1})
            varSubset = 1; %plot all
        else
            varSubset = 0; %plot subset
            plotIndex = varargin{1};
        end
end

```

```

    if isempty(varargin{2})
        linewidth = 1;
    else
        linewidth = varargin{2};
    end
    if isempty(varargin{3})
        color = 'black';
    else
        color = varargin{3};
    end
end
if varSubset
    if n == 1 %linear function
        [m1,m2] = size(x{1});
        for i=1:n1
            for j=1:m1
                fij = AffFunction(x{i}(j,1:m2-1),x{i}(j,m2));
                CR(i).addFunction(fij,['x' num2str(j)]);
            end
        end
        U = PolyUnion(CR);
        hold on
        for j=1:m1
            U.fplot(['x' num2str(j)],'linewidth',linewidth,'color',color)
        end
    else
        for i=1:n1
            fij = QuadFunction(x{i,1},x{i,2},x{i,3});
            CR(i).addFunction(fij,'f');
        end
        U = PolyUnion(CR);
        U.fplot('f','linewidth',linewidth,'color',color)
    end
else
    if n == 1 %linear function
        m1 = length(plotIndex);
        m2 = size(x{1},2);
        for i=1:n1
            for j=1:m1
                fij = AffFunction(x{i}(plotIndex(j),1:m2-1),x{i}(plotIndex(j),m2));
                CR(i).addFunction(fij,['x' num2str(plotIndex(j))]);
            end
        end
        U = PolyUnion(CR);
    end
end

```

```

hold on
for j=1:m1
    U.fplot(['x' num2str(plotIndex(j))'],'linewidth',linewidth,'color',color)
end
else
for i=1:n1
    fij = QuadFunction(x{i,1},x{i,2},x{i,3});
    CR(i).addFunction(fij,'f');
end
U = PolyUnion(CR);
U.fplot('f','linewidth',linewidth,'color',color)
end; end; end;

```

Sub-Funciton: makeFeasible

```

function [x,p] = makeFeasible(xhat,phat,v,pA,pb,data)
% determines the closest feasible point when a solution to mp-qa is
% infeasible due to linearization of a constraint
%-----
%INPUT      TYPE      DESCRIPTION
%xhat:      row vector  infeasible variable values
%phat:      row vector  parameter values at infeasible point
%v:         matrix      vertices of parameter space
%pA:        matrix      parameter coefficients of linear
%            inequalities describing parameter space
%pb:        column vector RHS values of linear inequalities
%            describing parameter space
%data:      struct      mp-NLP information
%-----
%OUTPUT
%x:         row vector  closest feasible point
%p:         row vector  parameter values at feasible point
%-----
n = length(xhat);
m = length(phat);
pmin = min(v)';
pmax = max(v)';
obj = @(x)(norm(x-[xhat;phat],1));
x0 = [xhat;phat];
if isempty(data.A)
    A = [zeros(length(pb),n), pA];
    b = pb;
else

```

```

    A = [data.A, -data.E;
          zeros(length(pb),n), pA];
    b = [data.b;pb];
end
if isempty(data.Aeq)
    Aeq = [];
    beq = [];
else
    Aeq = [data.Aeq,-data.Eeq];
    beq = data.beq;
end
lb = [data.lb;pmin];
ub = [data.ub;pmax];
if isempty(data.constraints)
    const = [];
else
    const = @(x)NLPconst(x,n,data.constraints);
end
options = optimoptions('fmincon','Algorithm','interior-point','Display','Off');
u = fmincon(obj,x0,A,b,Aeq,beq,lb,ub,const,options);
p = u(n+1:end);
x = u(1:n);
end
function [c,ceq] = NLPconst(u,n,func)
%Nonlinear constraints for mp-nlp with parameters treated as variables
x = u(1:n);
p = u(n+1:end);
[c,ceq] = func(x,p);
end

```

Sub-Function: nonlinConstraints

```

function [c,ceq] = nonlinConstraints(x,p)

c = (-x(1)*x(2)*200e9)/(2000*p*(3*x(2)+sqrt(3)*x(1))) + 1;
ceq = [];

end

```

Appendix D

MATLAB FEA Solver For Problem 3.1.1.1

```
function [c,ceq] = truss_feasolver(x,t)
format long

global load
global E
global feacount
feacount = feacount + 1;

x(3) = x(2);
x(4) = x(2);
l = 1;          % Length of the shaft in m

v2_limit = 3e-3*1;    %vertical deflection constraint at tip
sigma1_limit = 4.83e-4*E; %compressive stress constraint in member 1
sigma3_limit = 8.74e-4*E; %tensile stress constraint in member 2

%fsym = [-2*p*cos(t);-2*p*sin(t);0;-p];    %extracted global force vector in symbolic
exp

% Number of members
ne = 4;
nen = 2;
ndof = 2;
edof = nen*ndof;
totdof = ne*ndof;

% Try to find angle wrt nodal co-ordinates
ang = [0 150 150 30];    %angle made by the bar with the horizontal in CCW
xc = [0;sqrt(3);0.866025;0];
yc = [0;0;0.5;1];
% Assign area for each bar
A = [x(1),x(2),x(3),x(4)];
Ien = [1 2; 2 3; 3 4; 1 3];
% To determine length of each bar
L = zeros(ne,1);
for i=1:ne
    delx = xc(Ien(i,2))-xc(Ien(i,1));
    dely = yc(Ien(i,2))-yc(Ien(i,1));
```



```

    L(i,1) = sqrt(delx^2+dely^2);
end
%Initialization
d = zeros(totdof,1);
def = zeros(ne,1);          %bar deflection
strain = zeros(ne,1);
stress = zeros(ne,1);
K = zeros(totdof,totdof);
F = zeros(totdof,1);
% LM matrix-connecting Ien to its dof
for e = 1 : ne                %(Loop over elements)
    for i = 1 : nen            %(Loop over number of element nodes)
        I = Ien(e,i);          %(Global node number)
        for a = 1 : ndof        %(Loop over number of dof/node)
            pp = ndof*(i - 1) + a;    %(Increment element equation number)
            LM(e,pp) = ndof*(I - 1) + a;    %(Fill Matrix with global equation number)
        end
    end
end
% Transformation matrix
celltransf = cell(ne,1);
% Finding stiffness matrix for each truss member and assembling it in global
% matrix
for i=1:ne
    k = A(i)*E/L(i);    %stiffness of each member
    transf = zeros(4,4);
    transf(1,:) = k*[cosd(ang(i))^2 cosd(ang(i))*sind(ang(i)) -cosd(ang(i))^2 -
    cosd(ang(i))*sind(ang(i))];
    transf(2,:) = k*[cosd(ang(i))*sind(ang(i)) sind(ang(i))^2 -cosd(ang(i))*sind(ang(i)) -
    sind(ang(i))^2];
    transf(3,:) = -transf(1,:);
    transf(4,:) = -transf(2,:);
    celltransf{i,1}(:, :) = (transf);
end
% Global K matrix formation
for e = 1:ne
    for pp = 1: edof          % Looping over tot no. of dof of elements
        P = LM(e,pp);
        for q = 1: edof
            Q = LM(e,q);
            K(P,Q) = K(P,Q) +(celltransf{e,1}(pp,q));
        end
    end
end
end

```

```

% Extraction of K matrix

h1 = K(3:6,:);
K_E = h1(:,3:6);    % Extracted K matrix
%Force matrix
F(3,1) = -2*load*sqrt(1-t^2); %remove comment
F(4,1) = -2*load*t;
% F(5,1) = -2*load*sqrt(1-t^2);
F(6,1) = -load;
F_E = F(3:6,1);
% Evaluation of def expression and value at free node where load 2p is
% applied
d_E = K_E\F_E;      % deflection at free node
d(3:6,1) = d_E;
%Deflection of the members
for e=1:ne
    for i=1:edof      % Looping over dof of each element
        P(i)=LM(e,i);
        u(i)=d(P(i));
    end
    u1 = u(1:2);
    u2 = u(3:4);
    def_mag = u2-u1;
    def(e) = def_mag*[cosd(ang(e));sind(ang(e))];
    strain(e) = def(e)/L(e);
    stress(e) = E*strain(e);
end
%Nodal Displacement constraint
c1 = max(abs(d)) - v2_limit;    %abs since deflection evaluated is both positive &
negative
%Stress constraints
[maxstress_t]=max(stress);
c2 = maxstress_t - sigma1_limit;    %To consider maximum tension stress
[maxstress_c]=min(stress);
c3 = abs(maxstress_c) - sigma3_limit;    %min to accomodate the maximum
compressive stress
c = [c1;c2;c3];
ceq = [];

```

Appendix E

MATLAB FEA Code for Problem 3.1.5.1

Master Code

```
function [c,ceq] = trussdynamics_feasolver(x,tpar)
format long
global E dens
global feacount
global phi_norm totdof fdof count W
global Fext t
feacount = feacount + 1;
%-----use for testing-----
% t=0.95 %-->0deg          % angle of load applied at tip wrt to horizontal axis in
parametric form
%          % teta angle for each member wrt horizontal axis (CCW)
% load = 1000;          % Magnitude of load applied
% E = 200e9;          % Young's Modulus in N/m^2
% x(1) = 0.5e-1;      % area of cross-section 1
% x(2) = 0.75e-1;      % area of cross-section 2
% dens = 8000;          % density of steel, kg/m^3
%-----use for testing-----
t = tpar;
x(3) = x(2);
x(4) = x(2);

l = 1;          % Length of the shaft in m

v2_limit = 3e-4*l;    % vertical deflection constraint at tip
sigma1_limit = 4.83e-4*E; % compressive stress constraint in member 1
sigma3_limit = 8.74e-4*E; % tensile stress constraint in member 2
angfreq1_allow_lb = 600; % minimum allowable first natural angular frequency
angfreq1_allow_ub = 1200; % maximum allowable first natural angular frequency

% Number of members
ne = 4;          % no. of elements
n = 4;          % no. of nodes
nen = 2;          % no. of nodes in each element
ndof = 2;          % no. of dof at each node
edof = nen*ndof;    % no. of dof in each element
totdof = ne*ndof;    % total dof in the system
```

```

fdof = 4;          %fixed bc;
freedof = totdof-fdof; %free bc;

% Try to find angle wrt nodal co-ordinates
ang = [0 150 150 30];          %angle made by the bar with the horizontal in CCW
xc = [0;sqrt(3);0.866025;0];
yc = [0;0;0.5;1];
% Assign area for each bar
A = [x(1),x(2),x(3),x(4)];
Ien = [1 2; 2 3; 3 4; 1 3];
% To determine length of each bar
L = zeros(ne,1);
for i=1:ne
    delx = xc(Ien(i,2))-xc(Ien(i,1));
    dely = yc(Ien(i,2))-yc(Ien(i,1));
    L(i,1) = sqrt(delx^2+dely^2);
end

%Initialization
def = zeros(ne,1);          %bar deflection
K = zeros(totdof,totdof);
M = zeros(totdof,totdof);

% LM matrix-connecting Ien to its dof
for e = 1 : ne              %(Loop over elements)
    for i = 1 : nen          %(Loop over number of element nodes)
        I = Ien(e,i);        %(Global node number)
        for a = 1 : ndof      %(Loop over number of dof/node)
            pp = ndof*(i - 1) + a;          %(Increment element equation number)
            LM(e,pp) = ndof*(I - 1) + a;    %(Fill Matrix with global equation number)
        end
    end
end

%Transformation cell
celltransf = cell(ne,1);

% Finding stiffness matrix for each truss member and assembling it in global
% matrix
for i=1:ne
    k = A(i)*E/L(i);          %stiffness of each member
    transf = zeros(4,4);
    transf(1,:) = k*[cosd(ang(i))^2 cosd(ang(i))*sind(ang(i)) -cosd(ang(i))^2 -
    cosd(ang(i))*sind(ang(i))];

```

```

transf(2,:) = k*[cosd(ang(i))*sind(ang(i)) sind(ang(i))^2 -cosd(ang(i))*sind(ang(i)) -
sind(ang(i))^2];
transf(3,:) = -transf(1,:);
transf(4,:) = -transf(2,:);
celltransf{i,1}(:, :) = (transf);
end
cellM = cell(ne,1);

%Formation of Local M matrix for each truss element
for i=1:ne
    mi = (dens*A(i)*L(i))/6;
    T = [cosd(ang(i)),sind(ang(i)),0,0; -sind(ang(i)),cosd(ang(i)),0,0;...
    0,0,cosd(ang(i)),sind(ang(i)); 0,0,-sind(ang(i)),cosd(ang(i))]; %transformation
matrix
    mbar = mi*[2,0,1,0; 0,2,0,1; 1,0,2,0; 0,1,0,2]; %local cood ele mass
matrix
    me =(T'*mbar*T); %Transformed elemeent mass
matrix
    cellM{i,1}(:, :) = me;
end

% Global K matrix formation
for e = 1:ne
    for pp = 1: edof % Looping over tot no. of dof of elements
        P = LM(e,pp);
        for q = 1: edof
            Q = LM(e,q);
            K(P,Q) = K(P,Q) +(celltransf{e,1}(pp,q));
            M(P,Q) = M(P,Q) +(cellM{e,1}(pp,q));
        end
    end
end

% Extraction of K & Matrices

K_h = K(3:6,:);
K_E = K_h(:,3:6); % Extracted K matrix

M_h = M(3:6,:);
M_E = M_h(:,3:6); % Extracted M matrix

%Find the natural frequency & mode shapes
[phi,lamda] = eig(K_E,M_E);

```

```

angfreq = sqrt(diag(lamda));
[angfreq_sort,n_sort] = sort(angfreq);
freq_sort = angfreq_sort/(2*pi);
for j=1:(totdof-fdof)
    phi_sort(:,j) = phi(:,n_sort(j));
end
mu = diag(phi_sort' * M_E * phi_sort);
for j=1:(totdof-fdof)
    phi_norm(:,j) = phi_sort(:,j)/sqrt(mu(j));
end
% Determination of q for a range of time for first k natural frequencies
kk = 1:4;
q = cell(length(kk),1);
t_int = linspace(0,0.015,25);
qd = zeros(length(kk),length(t_int));
phid = zeros(freedof,length(kk));
for count = 1:length(kk)
    I = kk(count);
    phid(:,count) = phi_norm(:,I);
    W = angfreq_sort(I);
    [time,q{count}] = ode45(@derivative,t_int,[0 0]);
end

for j = 1:length(t_int)
    for i = 1:length(kk)
        qd(i,j) = q{i}(j,1);
    end
end
%Determination of d for every time step at all nodes
d = zeros(freedof,length(t_int));
d_all = zeros(totdof,length(t_int));
d2x = zeros(length(t_int),1);
d2y = zeros(length(t_int),1);
d3x = zeros(length(t_int),1);
d3y = zeros(length(t_int),1);
for i=1:length(t_int)
    d(:,i) = phid*qd(:,i); % displacement at all free dof for each time step
    d_all(3:6,i) = d(:,i); % displacement at all dof
    d2x(i,1) = d(1,i); % x disp at node 2 for each instant in time (time step)
    d2y(i,1) = d(2,i); % y disp at node 2 for each instant in time (time step)
    d3x(i,1) = d(3,i); % x disp at node 2 for each instant in time (time step)
    d3y(i,1) = d(4,i); % y disp at node 2 for each instant in time (time step)
end

```

```

%Finding maximum nodal deflection at any time instant
d2x_max = max(abs(d2x));
d2y_max = max(abs(d2y));
d3x_max = max(abs(d3x));
d3y_max = max(abs(d3y));
dmax_v = [d2x_max;d2y_max;d3x_max;d3y_max];
dmax = max(dmax_v);

%Stress calculation for each member at each time step
strain = zeros(ne,length(t_int));
stress = zeros(ne,length(t_int));
for i = 1: length(t_int)
    for e=1:ne
        for j=1:edof % Looping over dof of each element
            P(j)=LM(e,j);
            u(j)=d_all(P(j),i);
        end
        u1 = u(1:2);
        u2 = u(3:4);
        def_mag = u2-u1;
        def(e,i) = def_mag*[cosd(ang(e));sind(ang(e))];
        strain(e,i) = def(e,i)/L(e);
        stress(e,i) = E*strain(e,i);
    end
end

%Find maximum stress in each member
tstress_max_v = max(stress);
tstress_max = max(tstress_max_v);
cstress_max_v = min(stress);
cstress_max = min(cstress_max_v);

%Nodal Displacement constraint
c1 = dmax - v2_limit; %abs since deflection evaluated is both positive &
negative

%Stress constraints
c2 = tstress_max - sigma1_limit; %max tensile stress constraint
c3 = abs(cstress_max) - sigma3_limit; %maximum compressive stress constraint
c4 = angfreq_sort(1) - angfreq1_allow_ub;
c5 = angfreq1_allow_lb - angfreq_sort(1);
c = [c1;c2;c3;c4;c5];
ceq = [];

```

Sub-Function: derivative

```
% Subfunction definition
% t: angle of Fext in the parametric form
function [z] = derivative(time,q)
    global t
    global totdof fdof phi_norm Fext W count;
    Fext = zeros(totdof-fdof,1);
    Fext(3,1) = -(10^7)*time*exp(-500*time)*sqrt(1-t^2); % x-component of force at
node 3
    Fext(4,1) = -(10^7)*time*exp(-500*time)*t; % y-component of force at node 3
    r = phi_norm'* Fext; [z] = [ 0 1; -W^2 0]*q + [ 0; r(count) ];
end
```

Sub-Function: trussdynamics_feasolver_nonpar

Remove t=tpar in Sub-Function trussdynamics_feasolver and change function name accordingly

Appendix F

MATLAB Code for ASM Algorithm for Problem 3.1.1.1

Master Code

```
format long
clear all
clc
% original non-normalized, non-segmented using SAM
global E dens
global load
global t
global X_count X_array F_array feacount feacount_nonpar
E = 200e9;
dens = 8000;          %lb/in^3
load = 1000;          %Load magnitue
tlow = 0.15;          %lb for parameter (load angle)
tup = 1;              %Ub for parameter (load angle)
n_int = 11;           %n_int+1 vertices
inc_t = (tup-tlow)/n_int; %increment value for non-parametric optimization

X_array = [];
F_array = [];
X_count = 1;          %counter for optimization calls
feacount = 0;          %counter for number of FEA calls
feacount_nonpar = 0; %counter for number of non-parametric FEA calls

%optimization problem structure for approximate MP algorithm
trussang_struct = struct;

%objective function handle
trussang_struct.objective = @(x,t)(8000*1*(3*x(1)+sqrt(3)*x(2)));

%linear constraint matrices (inequality and equality)
%Form: A*x <= b+E*t
%trussorg_struct.A = [A1cof1,A2cof2;-E,0;0,-E];
trussang_struct.A = []; % Member 3 as largest tension
trussang_struct.b = [];
trussang_struct.E = [];
trussang_struct.Aeq = [];
```

```

trussang_struct.Eeq = [];
trussang_struct.beq = [];

%nonlinear constraint function handle
trussang_struct.constraints = @(x,t)truss_feasolver(x,t);

%lower and upper bounds for decision variables
trussang_struct.lb = [1e-9;1e-9];
trussang_struct.ub = [0.5;0.5];

% Definition for parameter space, tolerance and problem data
pSpace = Polyhedron('lb',tlow, 'ub',tup);
tol = 0.001;
problem = trussang_struct;

% Calling main AMP function to find optimal parametric solution
[xstar,fstar,CRstar,metrics] = AMPmain(tol,pSpace,problem);
CRsize = length(CRstar);

%non-parametric calculation for comparison
l_t = tlow;
u_t = tup;
i=1;
for t = l_t:inc_t:u_t

Xl=[1e-9,1e-9];
Xf=[5e-1,5e-1];
X0 = (Xl+Xf)/2;
A=[];
B=[];
Aeq=[];
Beq=[];
options = optimoptions('fmincon','Algorithm','sqp','Display','off');
[c, ceq] = truss_feasolver_nonpar(X0);

%[X,Fval,Exitflag,Output,Lambda,Grad,Hessian] = fmincon(@objfun, Initial x, A, B,
Aeq, Beq, LB, UB, non-linear constraints)
[X,Fval,Exitflag,Output,Lambda,Grad,Hessian] = fmincon (@trussang_fmincon,X0, A,
B, Aeq, Beq, Xl, Xf, @truss_feasolver_nonpar, options);

X_v(i,1)=X(1);
X_v(i,2)=X(2);
Fval_v(i) = Fval;
i=i+1;

```

end

```
figure(1)
plot(l_t:inc_t:u_t,X_v(:,1),'-k*', 'MarkerSize',10);
hold on
plot(l_t:inc_t:u_t,X_v(:,2),'--go','MarkerSize',10)
hold on
% Plot of optimum design variable
for i=1:CRsize
figure(1)
plotParamSolution(xstar(i),CRstar(i),1,[],'red')
hold on
plotParamSolution(xstar(i),CRstar(i),2,[],'blue')
hold on
end

%Labeling parametric results
figure(1)
xlabel('t-angle of load','FontSize',14);
ylabel('Optimum area of cross-section in m^2','FontSize',14);
legend('x_1-nonpar','x_2-nonpar','x_1-ASM', 'x_2-ASM');
hold off
```

Sub-Function: AMPmain

```
function [xstar,fstar,CR,metrics] = AMPmain(tol,pSpace,problem)
%This function solves a general multi-parametric convex optimization
%problem (mp-nlp) using the approximate mp algorithm of Bemporad and
%Filippi (2006)
%-----
%INPUT          TYPE          DESCRIPTION
%tol:           scalar        error tolerance
%pSpace:        polyhedron     parameter space
%problemData:   struct        mp-NLP information
%-----
%OUTPUT
%xstar:         cell           optimal decision variable functions
%fstar:         cell           optimal objective function
%CR:            polyhedron vector partition of parameter space
%metrics:       struct         solution information
%-----
```

```

%METRICS FIELDS:
%error:    sum of largest single point errors for each simplex
%intError: approximation of integrated error over parameter space
%nlp:      total number of NLP problems solved
%tol:      maximum error tolerance used
%-----

%=====
=====
%partition parameter space into simplices
S = pSpace.triangulate;
%=====
=====

%=====
=====
%Construct a linear approximate of the solution for each simplex
RemainVolume = pSpace.volume;      %what is
this?% % % % % % % % % % % % % % 555
[xstar,fstar,CR,metrics] = mpSimplexApproximation(tol,S,problem,RemainVolume);
metrics.tol = tol;
%=====
=====
End

```

Sub-Function: mpSimplexApproximation

```

function[xOpt,fOpt,CROpt,metric,RemainVolume]=...
mpSimplexApproximation(tol,S,problem,RemainVolume)
%Constructs a linear approximation of the optimal decision functions and
%optimal value function for each simplex in S. Calculates the maximum error
%over each simplex and partitions it further if error is too large.
%-----
%INPUT      TYPE      DESCRIPTION
%tol:       scalar     error tolerance
%S:         polyhedron array   simplexes
%problem:   struct     mp-NLP information
%-----
%OUTPUT
%xOpt:      cell       optimal decision variable functions
%fOpt:      cell       optimal objective function
%CROpt:     polyhedron vector   partition of parameter space

```

```

%metric:      struct          solution information
%RemainVolume: scalar          size of parameter space that does
%              not yet have a solution
%-----
global X_count X_array F_array
%=====
=====
%get number of parameters and variables
%=====
=====
if X_count == 1
    pn = S(1).Dim;          %2 vertices at first iteration (plow and pup) % only for 1par
    xn = length(problem.lb);
elseif X_count > 1
    pn = 1;                %1 vertex corresponding to pmean thereafter %only for 1 par
    xn = length(problem.lb);
end
%=====
=====
%initialize struct fields, cells, and arrays
%=====
=====
metric.error = 0;
metric.intError = 0;
metric.nlps = length(S)*(pn+2);          %%What is this???
xOpt = cell(0);
fOpt = cell(0);
CRopt = [];
%=====
=====

for i = 1:length(S)
    vertices = S(i).V;

    M = [vertices,ones(pn+1,1)];
    X = zeros(xn,pn+1);
    fval = zeros(1,pn+1);

    if X_count == 1
        %determine optimal solution at each vertex
        int0 = problem.lb + .5*(problem.ub-problem.lb);
        for j = 1:pn+1
            [X(:,j),fval(j)] = optimizationProblem(int0,vertices(j,:)',problem);
        end
    end
end

```

```

        X_array(X_count:X_count+1,1) = [S(i).V(1);S(i).V(2)];    %assign parameter
bounds in 1st column
        F_array(X_count:X_count+1,1) = [S(i).V(1);S(i).V(2)];
        X_array(X_count:X_count+1,2:3) = X'; %assign decision variables along the row
for each column(param val)
        F_array(X_count:X_count+1,2) = fval;
        X_count = X_count + 2; %no. of param val (no. of optimization runs)
    else
        %Search tool to extract the opt dec var for the given parameter
        %value
        i1=find(X_array==S(i).V(1));
        i2=find(X_array==S(i).V(2));
        X(:,1)= X_array(i1,2:3);
        X(:,2)= X_array(i2,2:3);
        fval(1,1) = X_array(i1,2);
        fval(1,2) = X_array(i2,2);
    end

    %linear approximations of optimal value function and decision functions
    % Value Function:      zApprox = fval*(M^-1)[1;t]
    % Decision Function:   xApprox = X(M^-1)[1;t]
    Minv = M^-1;
    xApprox = (Minv*X)';
    zApprox = (Minv*fval)';

    %=====
    %=====
    %determine maximum error

    %=====
    %=====
    %computes maximum error for zApprox as well as
    %evaluating f at the linearly interpolated optimizers based on xApprox
    %see Bemporad 2006 paper for more details

    %initial solution: use center of critical region
    tCenter = mean(vertices)';
    X_array(X_count,1) = tCenter; %assign next param value for which opt is carried
out
    %%-----
    x0 = xApprox*[tCenter;1];
    % [xHat,tHat,error1] = computeError(problem,S(i),x0,zApprox);

```

```

[xHat,fHatb] = optimizationProblem(x0,tCenter,problem);
tHat = tCenter;
error1 = abs(fHatb-zApprox*[tHat;1])/fHatb;
X_array(X_count,2:3) = xHat'; %assign decision variables along the row for each
column(param val)
X_count = X_count + 1; % inc for next opt run

xBar = xApprox*[tHat;1];
error2 = abs(problem.objective(xHat,tHat) - problem.objective(xBar,tHat));

%=====
=====

CRvol = S(i).volume;

%Choose which type of error should be used for determining the quality
%of the solution:
%errorType = error1;
errorType = error2;

if (errorType < tol)
    xOpt = [xOpt;xApprox];
    fOpt = [fOpt;zApprox];
    CRopt = [CRopt;S(i)];
    metric.error = metric.error + errorType;
    metric.intError = metric.intError + errorType*CRvol;
    RemainVolume = RemainVolume - CRvol;
else

    S2 = splitSimplex(S(i),tHat);

    %start a new approximation for the new simplexes
    [x2,f2,CR2,metric2,RemainVolume] =
mpSimplexApproximation(tol,S2,problem,RemainVolume);

    metric.error = metric.error + metric2.error;
    metric.intError = metric.intError + metric2.intError;
    metric.nlps = metric.nlps + metric2.nlps;
    xOpt = [xOpt;x2];
    fOpt = [fOpt;f2];
    CRopt = [CRopt;CR2];
end
end

```

```

end
function [x,t,error] = computeError(problem,S,x0,zHat)
SA = S.A;
Sb = S.b;
m = size(SA,1);
n = length(problem.lb);
vlb = min(S.V)';
vub = max(S.V)';
A = [problem.A, -problem.E;
     zeros(m,n),SA];
b = [problem.b;Sb];
lb = [problem.lb;vlb];
ub = [problem.ub;vub];
obj = @(x)nlpObjective(x,n,zHat,problem.objective);
if isempty(problem.constraints)
    const = [];
else
    const = @(x)nlpConstraints(x,n,problem.constraints);
end

options = optimoptions('fmincon','Algorithm','sqp','Display','Off');
[x0,error] = fmincon(obj,x0,A,b,[],[],lb,ub,const,options);
% options = optimoptions('fmincon','Algorithm','interior-point','Display','Off');
% [x0,error] = fmincon(obj,x0,A,b,[],[],lb,ub,const,options);
% options = optimoptions('fmincon','Algorithm','sqp','Display','Off');
% [x0,error] = fmincon(obj,x0,A,b,[],[],lb,ub,const,options);
error = abs(error);
t = x0(n+1:end);
x = x0(1:n);
end

function f = nlpObjective(u,n,z,func)
%objective function for mp-nlp with parameters treated as variables
x = u(1:n);
p = u(n+1:end);
f = func(x,p)-z*[p;1];
end

function [c,ceq] = nlpConstraints(u,n,func)
%Nonlinear constraints for mp-nlp with parameters treated as variables
x = u(1:n);
p = u(n+1:end);
[c,ceq] = func(x,p);
End

```


Sub-Function: optimizationProblem

```
function [x, fval] = optimizationProblem(x0,p,data)
% solves the mp-NLP as a standard optimization problem at a particular
% parameter value p
%-----
% INPUT      TYPE      DESCRIPTION
%x0:         row vector  initial solution
%p:          row vector  parameter value to solve problem with
%data:       struct      mp-NLP information
%-----
% OUTPUT
%x:          row vector  solution
%fval:       scalar      objective value
%-----
if isempty(data.b)
    A = [];
    b = [];
else
    A = data.A;
    b = data.b + data.E*p;
end

if isempty(data.beq)
    Aeq = [];
    beq = [];
else
    Aeq = data.Aeq;
    beq = data.beq + data.Eeq*p;
end

lb = data.lb;
ub = data.ub;

if isfield(data,'quadObjective')      %%% What does this do???
    obj = @(x)(data.objective(x,p)+data.quadObjective(x,p));
else
    obj = @(x)data.objective(x,p);
end

if isempty(data.constraints)
    const = [];
else
```

```

    const = @(x)data.constraints(x,p);
end
%IMPORTANT: The choice of fmincon algorithm has a considerable affect on
%the performance of the parametric algorithm.

options = optimoptions('fmincon','Algorithm','sqp','Display','Off');
[x0,fval] = fmincon(obj,x0,A,b,Aeq,beq,lb,ub,const,options);

% options = optimoptions('fmincon','Algorithm','interior-point','Display','Off');
% [x0,fval] = fmincon(obj,x0,A,b,Aeq,beq,lb,ub,const,options);

% options = optimoptions('fmincon','Algorithm','sqp','Display','Off');
% [x0,fval] = fmincon(obj,x0,A,b,Aeq,beq,lb,ub,const,options);
x = x0;
end

```

Sub-Function: splitSimplex

```

function P = splitSimplex(CR,varargin)
%partitions the simplex CR by connecting each facet of CR with a
%point in CR
%uses specified point if given, otherwise uses the center
%-----
%INPUT          TYPE          DESCRIPTION
%CR:            polyhedron     parameter space
%varargin:      row vector     point to use to split region
%-----
%OUTPUT          TYPE          DESCRIPTION
%P:             polyhedron array parameter space partition
%-----

if isempty(varargin)
    center = mean(CR.V)';
else
    center = varargin{ 1 };
end

CR.minVRep;
V = CR.V;
m = size(V,1);
P(m) = Polyhedron([]);

for i = 1:m

```

```

V2 = V;
V2(i,:) = center';
Ptemp = Polyhedron(V2);
if Ptemp.isFullDim && (Ptemp.volume > 0)
    P(i) = Polyhedron(V2);
end
clear Ptemp
end
P(P.isEmptySet)=[];
End

```

Sub-Function: plotParamSolution

Refer Appendix C, plotParamSolution

Sub-Function: truss_feasolver_nonpar

Append the following to truss_feasolver
t(1) = t1;

Appendix G

MATLAB Code for Problem 3.1.2.1

Master Code

```
format long
clear all
clc
% original non-normalized, non-segmented using SAM
global E load
global t1 t2 %for non-parametric results
global feacount X_count feacount_nonpar
load =5000; %load magnitude
E = 200e9; % Youngs modulus in N/m^2
t1low = 0.1; %lb for t1
t1up = 1; %ub for t1
t2low = 0.1; %lb for t2
t2up = 1; %ub for t2
n_t1 = 15; %no. of vertices for t1
n_t2 = 15; %no. of vertices for t2
inc_t1 = (t1up-t1low)/n_t1;
inc_t2 = (t2up-t2low)/n_t2;

X_count = 0;
feacount = 0;
feacount_nonpar = 0;

%optimization problem structure for approximate MP algorithm
trussang_struct = struct;
%objective function handle
trussang_struct.objective = @(x,t)(8000*1*(3*x(1)+sqrt(3)*x(2)));
%linear constraint matrices (inequality and equality)
%Form: A*x <= b+E*t
%trussorg_struct.A = [A1cof1,A2cof2;-E,0;0,-E];
trussang_struct.A = [];
trussang_struct.b = [];
trussang_struct.E = [];
trussang_struct.Aeq = [];
trussang_struct.Eeq = [];
trussang_struct.beq = [];
%nonlinear constraint function handle
```

```

trussang_struct.constraints = @(x,t)truss_feasolver(x,t);
%lower and upper bounds for decision variables
trussang_struct.lb = [1e-7;1e-7];
trussang_struct.ub = [0.1;0.1];
% Definition for parameter space, tolerance and problem data
pSpace = Polyhedron('lb',[t1low;t2low], 'ub',[t1up;t2up]);
tol = 0.105;
problem = trussang_struct;

% Calling main AMP function to find optimal parametric solution
[xstar,fstar,CRstar,metrics] = AMPmain(tol,pSpace,problem);

CRsize = length(CRstar);

%non-parametric calculation for comparison
l_t1 = t1low;
u_t1 = t1up;
l_t2 = t2low;
u_t2 = t2up;
j=1;
n = length(l_t1:inc_t1:u_t1); %for t1
m = length(l_t2:inc_t2:u_t2); %for t2
tt1 = linspace(l_t1,u_t2,n_t1);
tt2 = linspace(l_t2,u_t2,n_t2);
for t2 = linspace(l_t2,u_t1,n_t2)
    i = 1; %re-initialization--corresponds to m
    for t1 = linspace(l_t1,u_t2,n_t1)
        Xl=[1e-5,1e-5];
        Xf=[0.5,0.5];
        X0=(Xl+Xf)/2;
        A=[];
        B=[];
        Aeq=[];
        Beq=[];
        options = optimoptions(@fmincon,'Algorithm','sqp','Display','off');%,'TolX',1e-
14,'TolCon',1e-9,'TolFun',1e-9,'MaxFunEvals',10000);
        [c, ceq] = truss_feasolver_nonpar(X0);

        %[X,Fval,Exitflag,Output,Lambda,Grad,Hessian] = fmincon(@objfun, Initial x, A,
B, Aeq, Beq, LB, UB, non-linear constraints)
        [X,Fval,Exitflag,Output,Lambda,Grad,Hessian] = fmincon (@trussang_fmincon,X0,
A, B, Aeq, Beq, Xl, Xf, @truss_feasolver_nonpar, options);

        X_v1(j,i) =X(1);

```

```

        X_v2(j,i)=X(2);
        F_v(j,i) = Fval;
        i=i+1;
    end
    j = j+1;          %counter--corresponds to n or t1
end

```

```

figure(1)
[t1_plot,t2_plot] = meshgrid(tt1,tt2);
%Plot of optimum design variable
for i=1:CRsize
    figure(1)
    plotParamSolution(xstar(i),CRstar(i),1,[],'red')
    hold on
    % surf(t1_plot,t2_plot,X_v1);
    % hold on
    figure(2)
    plotParamSolution(xstar(i),CRstar(i),2,[],'blue')
    hold on
    surf(t1_plot,t2_plot,X_v2);
    hold on
end
figure(1)
legend('x_1 - ASM');
xlabel('t_1 - Angle of load (P_1)','FontSize',14);
ylabel('t_2 - Angle of load (P_2)','FontSize',14);
figure(2)
legend('x_2 - ASM');
xlabel('t_1 - Angle of load (P_1)','FontSize',14);
ylabel('t_2 - Angle of load (P_2)','FontSize',14);

```

Sub-Function: modified mpSimplexApproximation

```

function[xOpt,fOpt,CROpt,metric,RemainVolume]=
...mpSimplexApproximation(tol,S,problem,RemainVolume)
global X_count;
%=====
=====
%get number of parameters and variables
%=====
=====

```

```

pn = S(1).Dim;
xn = length(problem.lb);
%=====
=====
%initialize struct fields, cells, and arrays
%=====
=====
metric.error = 0;
metric.intError = 0;
metric.nlps = length(S)*(pn+2);          %% What is this???
xOpt = cell(0);
fOpt = cell(0);
CROpt = [];
%=====
=====

for i = 1:length(S)
    length(S)
    vertices = S(i).V;
    M = [vertices,ones(pn+1,1)];
    X = zeros(xn,pn+1);
    fval = zeros(1,pn+1);

    %determine optimal solution at each vertex
    int0 = problem.lb + .5*(problem.ub-problem.lb);
    for j = 1:pn+1
        [X(:,j),fval(j)] = optimizationProblem(int0,vertices(j,:),problem);
        X_count = X_count + 1;
    end
    %linear approximations of optimal value function and decision functions
    % Value Function:    zApprox = fval*(M^-1)[1;t]
    % Decision Function: xApprox = X(M^-1)[1;t]
    Minv = M^-1;
    xApprox = (Minv*X)';
    zApprox = (Minv*fval)';

%=====
=====
    %determine maximum error

%=====
=====
    %computes maximum error for zApprox as well as

```

```

%evaluating f at the linearly interpolated optimizers based on xApprox
%see Bemporad 2006 paper for more details

%initial solution: use center of critical region
tCenter = mean(vertices)';
xBar = xApprox*[tHat;1];
error2 = abs(problem.objective(xHat,tHat) - problem.objective(xBar,tHat));

%=====
=====

CRvol = S(i).volume;

%Choose which type of error should be used for determining the quality
%of the solution:
errorType = error1;
%errorType = error2;

disp('*****')
disp(['Largest error at t = ' num2str(tHat)])
disp(['zApprox error is ' num2str(error1)])
disp(['f(xApprox) error is ' num2str(error2)])
disp(['Region Volume is ' num2str(CRvol)])
disp(['Total Volume Remaining is ' num2str(RemainVolume)])
disp('*****')
disp(' ')

if (errorType < tol)
    xOpt = [xOpt;xApprox];
    fOpt = [fOpt;zApprox];
    CROpt = [CROpt;S(i)];
    metric.error = metric.error + errorType;
    metric.intError = metric.intError + errorType*CRvol;
    RemainVolume = RemainVolume - CRvol;
else

    S2 = splitSimplex(S(i),tHat);

    %start a new approximation for the new simplexes
    [x2,f2,CR2,metric2,RemainVolume] =
mpSimplexApproximation(tol,S2,problem,RemainVolume);

    metric.error = metric.error + metric2.error;
    metric.intError = metric.intError + metric2.intError;

```



```

        metric.nlps = metric.nlps + metric2.nlps;
        xOpt = [xOpt;x2];
        fOpt = [fOpt;f2];
        CROpt = [CROpt;CR2];
    end
end
end

function [x,t,error] = computeError(problem,S,x0,zHat)
SA = S.A;
Sb = S.b;
m = size(SA,1);
n = length(problem.lb);

vlb = min(S.V)';
vub = max(S.V)';

A = [problem.A, -problem.E;
     zeros(m,n),SA];
b = [problem.b;Sb];

lb = [problem.lb;vlb];
ub = [problem.ub;vub];

obj = @(x)nlpObjective(x,n,zHat,problem.objective);

if isempty(problem.constraints)
    const = [];
else
    const = @(x)nlpConstraints(x,n,problem.constraints);
end

options = optimoptions('fmincon','Algorithm','sqp','Display','Off');
[x0,error] = fmincon(obj,x0,A,b,[],[],lb,ub,const,options);
error = abs(error);
t = x0(n+1:end);
x = x0(1:n);
end

function f = nlpObjective(u,n,z,func)
%objective function for mp-nlp with parameters treated as variables
x = u(1:n);
p = u(n+1:end);
f = func(x,p)-z*[p;1];

```

end

```
function [c,ceq] = nlpConstraints(u,n,func)
%Nonlinear constraints for mp-nlp with parameters treated as variables
x = u(1:n);
p = u(n+1:end);
[c,ceq] = func(x,p);
end
```

Sub-Function: truss_feasolver

```
function [c,ceq] = truss_feasolver(x,t)
format long
% t=0 %-->0deg          % angle of load applied at tip wrt to horizontal axis in
parametric form
%          % teta angle for each member wrt horizontal axis (CCW)
% load = 1000;          % Magnitude of load applied
% E = 200e9;            % Young's Modulus in N/m^2
% x(2)= 1.3125e-4;      % area of cross-section 1
% x(1)= 2e-4;           %area of cross-section 2
% t(1) = 0.5; %load angle at node 2
% t(2) = 0.5; %load angle at node 3
global E load scale_t
global feacount
x(3) = x(2);
x(4) = x(2);
t(1) = t(1);
t(2) = t(2);
l = 1;          % Length of the shaft in m

feacount = feacount + 1;
v2_limit = 3e-3*l; %vertical deflection constraint at tip
sigma1_limit = 4.83e-4*E; %compressive stress constraint in member 1
sigma3_limit = 8.74e-4*E; %tensile stress constraint in member 2

%fsym = [-2*p*cos(t);-2*p*sin(t);0;-p]; %extracted global force vector in symbolic
exp

% Number of members
ne = 4;
nen = 2;
ndof = 2;
```

```

edof = nen*ndof;
totdof = ne*ndof;

% Try to find angle wrt nodal co-ordinates
ang = [0 150 150 30]; %angle made by the bar with the horizontal in CCW
xc = [0;sqrt(3);0.866025;0];
yc = [0;0;0.5;1];
% Assign area for each bar
A = [x(1),x(2),x(3),x(4)];
Ien = [1 2; 2 3; 3 4; 1 3];
% To determine length of each bar
L = zeros(ne,1);
for i=1:ne
    delx = xc(Ien(i,2))-xc(Ien(i,1));
    dely = yc(Ien(i,2))-yc(Ien(i,1));
    L(i,1) = sqrt(delx^2+dely^2);
end

%Initialization
d = zeros(totdof,1);
def = zeros(ne,1); %bar deflection
strain = zeros(ne,1);
stress = zeros(ne,1);
K = zeros(totdof,totdof);
F = zeros(totdof,1);

% LM matrix-connecting Ien to its dof
for e = 1 : ne % (Loop over elements)
    for i = 1 : nen % (Loop over number of element nodes)
        I = Ien(e,i); % (Global node number)
        for a = 1 : ndof % (Loop over number of dof/node)
            pp = ndof*(i - 1) + a; % (Increment element equation number)
            LM(e,pp) = ndof*(I - 1) + a; % (Fill Matrix with global equation number)
        end
    end
end

%Transformation matrix
celltransf = cell(ne,1);

% Finding stiffness matrix for each truss member and assembling it in global
% matrix
for i=1:ne
    k = A(i)*E/L(i); %stiffness of each member
    transf = zeros(4,4);

```

```

transf(1,:) = k*[cosd(ang(i))^2 cosd(ang(i))*sind(ang(i)) -cosd(ang(i))^2 -
cosd(ang(i))*sind(ang(i))];
transf(2,:) = k*[cosd(ang(i))*sind(ang(i)) sind(ang(i))^2 -cosd(ang(i))*sind(ang(i)) -
sind(ang(i))^2];
transf(3,:) = -transf(1,:);
transf(4,:) = -transf(2,:);
celltransf{i,1}(:, :) = (transf);
end

% Global K matrix formation
for e = 1:ne
    for pp = 1: edof % Looping over tot no. of dof of elements
        P = LM(e,pp);
        for q = 1: edof
            Q = LM(e,q);
            K(P,Q) = K(P,Q) +(celltransf{e,1}(pp,q));
        end
    end
end

% Extraction of K matrix

h1 = K(3:6,:);
K_E = h1(:,3:6); % Extracted K matrix

%Force matrix
F(3,1) = -2*load*sqrt(1-t(1)^2);
F(4,1) = -2*load*t(1);
F(5,1) = -load*sqrt(1-t(2)^2);
F(6,1) = -load*t(2);
F_E = F(3:6,1);
% Evaluation of def expression and value at free node where load 2p is
% applied

d_E = K_E\F_E; % deflection at free node
d(3:6,1) = d_E;

%Deflection of the members
for e=1:ne
    for i=1:edof % Looping over dof of each element
        P(i)=LM(e,i);
        u(i)=d(P(i));
    end
    u1 = u(1:2);

```

```

    u2 = u(3:4);
    def_mag = u2-u1;
    def(e) = def_mag*[cosd(ang(e));sind(ang(e))];
    strain(e) = def(e)/L(e);
    stress(e) = E*strain(e);

end

%Nodal Displacement constraint
c1 = max(abs(d)) - v2_limit;      %abs since deflection evaluated is both positive &
negative
%Stress constraints
[maxstress_t]=max(stress);
c2 = maxstress_t - sigma1_limit;    %To consider maximum tension stress
[maxstress_c]=min(stress);
c3 = abs(maxstress_c) - sigma3_limit;    %min to accomodate the maximum
compressive stress
c = [c1;c2;c3];
ceq = [];
% %end
%end

```

Sub-Function: truss_feasovler_nonpar

Append the following to truss_feasolver

```
t(1) = t1;
```

```
t(2)= t2;
```

Appendix H

MATLAB Code for Problem 3.1.3.1

Master Code

```
format long
clear all
clc
% original non-normalized, non-segmented using SAM
%mpSimplexApproximation has error1 as relative%%%Check
%mpSimplexApproximation
global E
global t1 t2 %for non-parametric results
global feacount feacount_nonpar
E = 200e9;
int_t1n = 10;
int_t2n = 10;
t1low = 0.1; %lb for t1
t1up = 1; %ub for t1
inc_t1 = (t1up-t1low)/int_t1n;
t2low = 5000; %lb for p in N
t2up = 7000; %ub for p in N
inc_t2 = (t2up-t2low)/int_t2n;

feacount = 0;
feacount_nonpar = 0;
tic

%optimization problem structure for approximate MP algorithm
trussang_struct = struct;

%objective function handle
trussang_struct.objective = @(x,t)(8000*1*(3*x(1)+sqrt(3)*x(2)));

%linear constraint matrices (inequality and equality)
%Form: A*x <= b+E*t
%trussorg_struct.A = [A1cof1,A2cof2;-E,0;0,-E];
trussang_struct.A = [];
trussang_struct.b = [];
trussang_struct.E = [];
trussang_struct.Aeq = [];
```

```

trussang_struct.Eeq = [];
trussang_struct.beq = [];

%nonlinear constraint function handle
trussang_struct.constraints = @(x,t)truss_feasolver(x,t);

%lower and upper bounds for decision variables
trussang_struct.lb = [1e-7;1e-7];
trussang_struct.ub = [0.5;0.5];

% Definition for parameter space, tolerance and problem data
pSpace = Polyhedron('lb',[t1low;t2low], 'ub',[t1up;t2up]);
tol = 0.12;
problem = trussang_struct;

% Calling main AMP function to find optimal parametric solution
[xstar,fstar,CRstar,metrics] = AMPmain(tol,pSpace,problem);

CRsize = length(CRstar);

toc
%non-parametric calculation for comparison
l_t1 = t1low;
u_t1 = t1up;
l_t2 = t2low;
u_t2 = t2up;
j=1;
n = length(l_t1:inc_t1:u_t1); %for t1
m = length(l_t2:inc_t2:u_t2); %for t2
for t2 = l_t2:inc_t2:u_t2
    i = 1; %re-initialization--corresponds to m
    for t1 = l_t1:inc_t1:u_t1
        X0=[1e-9,1e-9];
        Xf=[5e-1,5e-1];
        A=[];
        B=[];
        Aeq=[];
        Beq=[];
        options = optimoptions('fmincon','Algorithm','sqp','Display','off');
        [c, ceq] = truss_feasolver_nonpar(X0);

        %[X,Fval,Exitflag,Output,Lambda,Grad,Hessian] = fmincon(@objfun, Initial x, A,
        B, Aeq, Beq, LB, UB, non-linear constraints)

```



```

l = 1;          % Length of the shaft in m

v2_limit = 3e-3*1;    %vertical deflection constraint at tip
sigma1_limit = 4.83e-4*E; %compressive stress constraint in member 1
sigma3_limit = 8.74e-4*E; %tensile stress constraint in member 2

%fsym = [-2*p*cos(t);-2*p*sin(t);0;-p];    %extracted global force vector in symbolic
exp
F = [-2*p*sqrt(1-t(1)^2);-2*p*t(1);0;-p];

% Number of members
n = 4;
nen = 2;
ndof = 2;
edof = nen*2;
totdof = n*ndof;
% Try to find angle wrt nodal co-ordinates
ang = [0 150 150 30];    %angle made by the bar with the horizontal in CCW
xc = [0;sqrt(3);0.866;0];
yc = [0;0;0.5;1];
% To determine length of each bar
L = zeros(n);
for i=1:4
L(i) = sqrt((xc(2)-xc(1))^2+(yc(2)-yc(1))^2);
end
% Assign area for each bar
A = [x(1),x(2),x(2),x(2)];
Ien = [1 2; 2 3; 3 4; 1 3];
%LM = zeros(tot_ne,edof)
% LM matrix-connecting Ien to its dof
for e = 1 : n                %(Loop over elements)
    for i = 1 : nen          %(Loop over number of element nodes)
        I = Ien(e,i);        %(Global node number)
        for a = 1 : ndof      %(Loop over number of dof/node)
            pp = ndof*(i - 1) + a;    %(Increment element equation number)
            LM(e,pp) = ndof*(I - 1) + a;    %(Fill Matrix with global equation number)
        end
    end
end
end
%Transformation matrix
celltransf = cell(4,1);
K = zeros(totdof,totdof);

% Finding stiffness matrix for each truss member and assembling it in global

```

```

% matrix
for i=1:4
k = A(i)*E/L(i);    %stiffness of each member
transf = zeros(4,4);
transf(1,:) = k*[cosd(ang(i))^2 cosd(ang(i))*sin(ang(i)) -cosd(ang(i))^2 -
cosd(ang(i))*sind(ang(i))];
transf(2,:) = k*[cosd(ang(i))*sind(ang(i)) sind(ang(i))^2 -cosd(ang(i))*sind(ang(i)) -
sind(ang(i))^2];
transf(3,:) = -transf(1,:);
transf(4,:) = -transf(2,:);
celltransf{i,1}(:, :) = (transf);
end

% Global K matrix formation
for e = 1:n
    for pp = 1: edof                                % Looping over tot no. of dof of elements
        P = LM(e,pp);
        for q = 1: edof
            Q = LM(e,q);
            K(P,Q) = K(P,Q) +(celltransf{e,1 }(pp,q));
        end
    end
end

% Extraction of K matrix

h1 = K(3:6,:);
K_E = h1(:,3:6);    % Extracted K matrix

% Evaluation of def expression and value at free node where load 2p is
% applied

def = K_E\F;        % deflection at free node
v2 = def(2);        % value of deflection at node where 2p is applied

% Evaluation of force on member 1 (b/w nodes 1 & 2)

force1 = (-2*p*sqrt(1-t(1)^2) - 2*sqrt(3)*p*t(1));    %for compressive stress

% Evaluation of force on member 2 (b/w nodes 1 & 2)

force3 = (p + 4*p*t(1));

c1 = abs(v2) - v2_limit;    %abs since deflection evaluated is negative

```

```
c2 = abs(force1) - sigma1_limit*x(2);      %abs since compressive stress evaluated is  
negative  
c3 = force3 - sigma3_limit*x(1);  
  
c = [c1;c2;c3];  
ceq = [];  
%end
```

Appendix I

MATLAB FEA Code for Problem 3.1.4.1

Master Code

```
format long
clear all
clc
%Areas that are same
%X3=X4=X8=X9; X6=X7=X10; X1; X2; X5
% original non-normalized, non-segmented using FEASAM
%Remember to rescale the objective
global E
global dens
global l          %in
global scale_l scale_obj
global load1_np load2_np t1_np t2_np
global feacount feacount_nonpar

feacount = 0;
feacount_nonpar = 0;
E = 10e7;          %psi
dens = 0.1;        %lb/in^3
l = 360;          %inch
p1low = 90000;      %lb
p1up = 100000;      %lb
p2low = 80000;
p2up = 95000;
t1low = 0.1;
t1up = 1;
t2low = 0.1;
t2up = 1;
inc_p = 1000;
inc_t = 0.1
%      scale_l = (p2up-p2low)/p2low;  %scaling factor for load values beyond 40000
scale_l = 1/10;
scale_obj = 1000;
p1low = scale_l*p1low;
p1up = scale_l*p1up;
p2low = scale_l*p2low;
p2up = scale_l*p2up;
```

```

tlow = 0.1;
tup = 1;
inc_p1 = (p1up-p1low)/10;
inc_p2 = (p2up-p2low)/10;
inc_t = 0.1;

%optimization problem structure for approximate MP algorithm
truss10bar_struct = struct;

%objective function handle
truss10bar_struct.objective = @(x,t)truss10bar_obj(x,t);

%linear constraint matrices (inequality and equality)
%Form: A*x <= b+E*t
truss10bar_struct.A = [];      % Member 2 as largest tension
truss10bar_struct.b = [];
truss10bar_struct.E = [];
truss10bar_struct.Aeq = [];
truss10bar_struct.Eeq = [];
truss10bar_struct.beq = [];

%nonlinear constraint function handle
truss10bar_struct.constraints = @(x,t)truss10bar_feasolver(x,t);

%lower and upper bounds for decision variables
truss10bar_struct.lb = 0.1*ones(10,1);
truss10bar_struct.ub = 20*ones(10,1);

% Definition for parameter space, tolerance and problem data
pSpace = Polyhedron('lb',[t1low,t2low,p1low,p2low], 'ub',[t1up,t2up,p1up,p2up]);
tol = 0.150;
problem = truss10bar_struct;

% Calling main AMP function to find optimal parametric solution
[xstar,fstar,CRstar,metrics] = AMPmain(tol,pSpace,problem);

CRsize = length(CRstar);

% Non-parametric solution
% non-parametric calculation for comparison
tic
ii=1; jj=1; kk=1; ll=1;
for t1_np = t1low:inc_t:t1up
jj=1;

```

```

for t2_np = t2low:inc_t:t2up
kk=1;
for load1_np = p1low:inc_p1:p1up
ll=1;
for load2_np = p2low:inc_p2:p2up
Xl=0.1*ones(10,1);
Xf=20*ones(10,1);
X0=(Xl+Xf)/2;
A=[];
B=[];
Aeq=[];
Beq=[];
options = optimoptions('fmincon','Algorithm','sqp','Display','off');
[c, ceq] = truss10bar_feasolver_nonpar(X0);

[X,Fval,Exitflag,Output,Lambda,Grad,Hessian] = fmincon (@truss10bar_obj,X0, A, B,
Aeq, Beq, Xl, Xf, @truss10bar_feasolver_nonpar, options);

X1_v(ll,kk) = X(1);
X2_v(ll,kk) = X(2);
X3_v(ll,kk) = X(3);
X5_v(ll,kk) = X(5);
X6_v(ll,kk) = X(6);
ll = ll+1
end
kk = kk+1
end
X1_vv{jj,ii}{:,:} = X1_v;
X2_vv{jj,ii}{:,:} = X2_v;
X3_vv{jj,ii}{:,:} = X3_v;
X5_vv{jj,ii}{:,:} = X5_v;
X6_vv{jj,ii}{:,:} = X6_v;
jj = jj+1
end
ii=ii+1
end
toc

```

Sub-Function: truss10bar_feasolver

```

function [c,ceq] = truss10bar_feasolver(x,par)
format long

```

```

%x(1)          %area of cross-section 1
%x(n)          %area of cross-section n
%par(1)        %load in lbf at node 2
%par(2)        %load in lbf at node 3
%par(3)        %angle made by load1 wrt horizontal in CCW
%par(4)        %angle made by laod2 wrt horizontal in CCW
% E=10000e3;    %psi
% l = 360;      %inch % Length of the bar
%x=[8.06;3.95;0.1;0.1;7.94;5.57;5.74;0.1;0.1;5.57];
global scale_1
global E dens load
global l
global feacount feacount_nonpar

feacount = feacount+1;          %counter for fea calls
A = x;                          %Area vector of size 10
load1 = par(3)/scale_1;
load2 = par(4)/scale_1;
t1 = par(1);
t2 = par(2);
v2_limit = 3e-3*l;             %vertical deflection constraint at tip
sigma1_limit = 4.83e-4*E; %compressive stress constraint in member 1
sigma3_limit = 8.74e-4*E; %tensile stress constraint in member 2

ne = 10;                      % Number of bars
n = 6;                        % Number of joints (nodes)
nen = 2;                      %For assembly
ndof = 2;
edof = nen*2;
totdof = n*ndof;
% Try to find angle wrt nodal co-ordinates
ang = [0,0,90,0,0,45,135,90,45,135]; %angle made by each bar(ascending
member no.) with the horizontal in CCW
xc = [0;l;2*l;2*l;l;0];
yc = [0;0;0;l;l;l];
Ien = [1 2; 2 3; 3 4; 5 4; 6 5; 1 5; 2 6; 2 5; 2 4; 3 5];
% To determine length of each bar
L = zeros(ne,1);
for i=1:ne
L(i) = sqrt((xc(Ien(i,2))-xc(Ien(i,1)))^2+(yc(Ien(i,2))-yc(Ien(i,1)))^2);
end
% Assign area for each bar

%LM = zeros(tot_ne,edof)

```

```

% LM matrix-connecting Ien to its dof
for e = 1 : ne % (Loop over elements)
    for i = 1 : nen % (Loop over number of element nodes)
        I = Ien(e,i); % (Global node number)
        for a = 1 : ndof % (Loop over number of dof/node)
            p = ndof*(i - 1) + a; % (Increment element equation number)
            LM(e,p) = ndof*(I - 1) + a; % (Fill Matrix with global equation number)
        end
    end
end
end
% Transformation matrix
celltransf = cell(ne,1);
K = zeros(totdof,totdof);
F = zeros(totdof,1);
d = zeros(totdof,1);
def = zeros(ne,1); % bar deflection
strain = zeros(ne,1);
stress = zeros(ne,1);

% Finding stiffness matrix for each truss member and assembling it in global
% matrix
for i=1:ne
    k = A(i)*E/L(i); % stiffness of each member
    transf = zeros(4,4);
    transf(1,:) = k*[cosd(ang(i))^2 cosd(ang(i))*sind(ang(i)) -cosd(ang(i))^2 -
    cosd(ang(i))*sind(ang(i))];
    transf(2,:) = k*[cosd(ang(i))*sind(ang(i)) sind(ang(i))^2 -cosd(ang(i))*sind(ang(i)) -
    sind(ang(i))^2];
    transf(3,:) = -transf(1,:);
    transf(4,:) = -transf(2,:);
    celltransf{i,1}(:, :) = (transf);
end

% Global K matrix formation
for e = 1:ne
    for p = 1: edof % Looping over tot no. of dof of elements
        P = LM(e,p);
        for q = 1: edof
            Q = LM(e,q);
            K(P,Q) = K(P,Q) +(celltransf{e,1}(p,q));
        end
    end
end
end

```



```

%Assembly of F matrix
F(3,1)=-load1*sqrt(1-t1^2); %cos(teta);      %horizontal load at node2 %teta defined
b/w horizontal axis and load in CCW
F(4,1)=-load1*t1;          %sin(teta);
F(5,1)=-load2*sqrt(1-t2^2);
F(6,1)=-load2*t2;
%-----
% Extraction of K matrix
% Enforce Boundary Conditions

%Essential displacement BC--(d1=d2=d11=d12=0)
h1 = K(3:10,:);          % Extraction of row matrix
K_E = h1(:,3:10);        % Extracted of column from the extracted rows

%Extraction of F matrix
F_E = F(3:10,1);
%Nodal Displacement
d_E = K_E\F_E;           % deflection at free node where 2p is applied symbolic exp
d(3:10,1) = d_E;
%-----
%Calculation of stress
%Deflection of the members
for e=1:ne

    for i=1:edof          % Looping over dof of each element
        P(i)=LM(e,i);
        u(i)=d(P(i));
    end
    u1 = u(1:2);
    u2 = u(3:4);
    def_mag = u2-u1;
    def(e) = def_mag*[cosd(ang(e));sind(ang(e))];
    strain(e) = def(e)/L(e);
    stress(e) = E*strain(e);

end
%Nodal Displacement constraint
c1 = max(abs(d)) - v2_limit;    %abs since deflection evaluated is both positive &
negative
%Stress constraints
[maxstress_t,i]=max(stress);
c2 = maxstress_t - sigma3_limit*x(i);    %To consider maximum tension stress
[maxstress_c,j]=min(stress);

```

```
c3 = abs(maxstress_c) - sigma1_limit*x(j);    %min to accomodate the maximum  
compressive stress  
c = [c1;c2;c3];  
ceq = [];end
```

Appendix J

MATLAB ASM Code for Problem 3.1.5.1

Master Code

```
format long
clear all
clc
% original non-normalized, non-segmented using SAM
%Dynamic load Fext is defined in the derivarive.m matlab file
global E dens
global X_count X_array F_array feacount feacount_nonpar
E = 200e9;
dens = 8000;          %lkg/m^3
tlow = 0.15;
tup = 1;
n_inc = 25;           %number of vertices -1
inc_t = (tup-tlow)/n_inc;

X_array = [];
F_array = [];
X_count = 1;
feacount = 0;
feacount_nonpar = 0;

%optimization problem structure for approximate MP algorithm
trussang_struct = struct;

%objective function handle
trussang_struct.objective = @(x,t)truss_obj(x,t);

%linear constraint matrices (inequality and equality)
%Form: A*x <= b+E*t_nonpar
%trussorg_struct.A = [A1cof1,A2cof2;-E,0;0,-E];
trussang_struct.A = []; % Member 3 as largest tension
trussang_struct.b = [];
trussang_struct.E = [];
trussang_struct.Aeq = [];
trussang_struct.Eeq = [];
trussang_struct.beq = [];
```

```

%nonlinear constraint function handle
trussang_struct.constraints = @(x,t)trussdynamics_feasolver(x,t);

%lower and upper bounds for decision variables
trussang_struct.lb = [1e-9;1e-9];
trussang_struct.ub = [0.5;0.5];

% Definition for parameter space, tolerance and problem data
pSpace = Polyhedron('lb',tlow, 'ub',tup);
tol = 0.1;
problem = trussang_struct;

% Calling main AMP function to find optimal parametric solution
[xstar,fstar,CRstar,metrics] = AMPmain(tol,pSpace,problem);
CRsize = length(CRstar);

%non-parametric calculation for comparison
l_t = tlow;
u_t = tup;
i=1;
global t
for t = l_t:inc_t:u_t

Xl=[1e-9,1e-9];
Xf=[5e-1,5e-1];
X0 = (Xl+Xf)/2;
A=[];
B=[];
Aeq=[];
Beq=[];
options = optimoptions('fmincon','Algorithm','sqp','Display','off');
[c, ceq] = trussdynamics_feasolver_nonpar(X0);

%[X,Fval,Exitflag,Output,Lambda,Grad,Hessian] = fmincon(@objfun, Initial x, A, B,
Aeq, Beq, LB, UB, non-linear constraints)
[X,Fval,Exitflag,Output,Lambda,Grad,Hessian] = fmincon (@trussang_fmincon,X0, A,
B, Aeq, Beq, Xl, Xf, @trussdynamics_feasolver_nonpar, options);

X_v(i,1)=X(1);
X_v(i,2)=X(2);
Fval_v(i,1) = Fval;
i=i+1;
end

```

```

figure(1)
plot(l_t:inc_t:u_t,X_v(:,1),'-k*');
hold on
plot(l_t:inc_t:u_t,X_v(:,2),'--go');
hold on
figure(2)
plot(l_t:inc_t:u_t,Fval_v(:,1),'--go');
hold on

```

Plot of optimum design variable

```

for i=1:CRsize
figure(1)
plotParamSolution(xstar(i),CRstar(i),1,[],'red')
hold on
plotParamSolution(xstar(i),CRstar(i),2,[],'blue')
hold on
figure(2)
plotParamSolution(fstar(i),CRstar(i),1,[],'blue')
hold on
end

```

%Labeling parametric results

```

figure(1)
xlabel('\phi - Angle of load P_{ext}','FontSize',14);
ylabel('Optimum area of cross-section in m^2','FontSize',14);
% legend('x_1-nonpar','x_2-nonpar','x_1-ASM','x_2-ASM');
legend('x_1-Non-parametric','x_2-Non-parametric','x_1-ASM','x_2-ASM');
hold off
figure(2)
xlabel('\phi - Angle of load P_{ext}','FontSize',14);
ylabel('Optimum mass of the truss structure in kg','FontSize',14);
% legend('non-parametric','ASM');
legend('ASM');
hold off

```

Appendix K

MATLAB ASM Code for Problem 3.2.1.1

Master Code

```
clear all
clc

%Multi-objective Honeycomb problem with epsilon constraint method
%Main objective - Minimization of Weight
% Total volume of the panel is fixed (L X H)
%Obj as constraints: G12(Flexural modulus) & nu12(Max strain rate)
%DC(x) - epsilon <= 0

%%%%%%%%Predefinitions
% Height of panel = 12.7 mm
% Length of the panel = 63.5 mm
% Number of unit cells in x direction,  $N_x = L/(2h\cos(\text{teta}))$ 
% Number of unit cells in y-direction:  $N_y = 2$ 
% teta = 30 deg

%%%%%%%%Constants
global scale_nu12
global dens E sig_yield
global L H h Nx Ny teta

scale_nu12 = 100;

dens = 8000e-9; %density in kg/mm^3 (steel)
E = 210e3; %Youngs modulus in N/mm^2
sig_yield = 200; %yield strength in N/mm^2

L = 63.5; %Length of the panel in mm
H = 12.7; %Height of the panel in mm
teta = 30; %cell angle in deg
Ny = 2; %Number of unit cells in the y-direction
h = H/(2*Ny*(1+sind(teta))); %height and length of the unit cell element in mm. 2.117
Nx = round(L/(2*h*cosd(teta))); %Number of unit cells in x direction
epsilon1_l = 4.7; %parametric lb G12 in N/mm^2
epsilon1_u = 7.7; %parametric ub G12 in N/mm^2
epsilon2_l = 0.01*scale_nu12; %parametric lb nu12
```

```

epsilon2_u = 0.03*scale_nu12; %parametric ub nu12
Honeycomb = struct;
x_lb = 0.01;      %lb of thickness in mm
x_ub = 0.12;      %ub of thickness in mm

Honeycomb.objective = @(x,p)G12_obj(x);

%linear constraint matrices (inequality and equality)
%Form: A*x <= b+E*p
Honeycomb.A = [];
Honeycomb.b = [];
Honeycomb.E = [];
Honeycomb.Aeq = [];
Honeycomb.Eeq = [];
Honeycomb.beq = [];

%nonlinear constraint function handle
Honeycomb.constraints = @(x,p)constraints(x,p);

%lower and upper bounds
Honeycomb.lb = x_lb;
Honeycomb.ub = x_ub;

% Definition for parameter space, tolerance and problem data
pSpace = Polyhedron('lb',[epsilon1_l;epsilon2_l], 'ub',[epsilon1_u;epsilon2_u]);
tol = 0.075;
problem = Honeycomb;

% Calling main AMP function to find optimal parametric solution
[xstar,fstar,CRstar,metrics] = AMPmain(tol,pSpace,problem);

CRsize = length(CRstar);

%Rescaling CRstar (parameter2)

% Plot of optimum design variable
for i=1:CRsize
figure(1)
plotParamSolution(xstar(i),CRstar(i),1,[],'red')
hold on
figure(2)
plotParamSolution(fstar(i),CRstar(i),1,[],'green')
hold on

```

```

end
figure(1)
xlabel('G_{12}^* in MPa','FontSize',14);
ylabel('\nu_{12}^* in %','FontSize',14);
zlabel('Optimum thickness of hexagonal unit cell in mm','FontSize',14);
figure(2)
xlabel('G_{12}^* in MPa','FontSize',14);ylabel('\nu_{12}^* in %','FontSize',14);
zlabel('Optimum weight of honeycomb panel in kg','FontSize',14);
Sub-Function: G12obj

```

```

function [obj] = G12_obj(x)
global dens
global h
global Nx Ny

% Length of the panel = 63.5 mm
% Height of the panel = 12.7 mm
% Number of unit cells in x direction,Nx = L/(2hcos(teta))
% x = cell thickness (th)- Decision variable
% t = cell height (h) - Parameter
% obj = weight of panel (W)
obj = dens*Ny*(Nx*7*h)*x;      %net length of 6 edges/unit cell = 7h % for teta=30,
Nx ~= 17
end

```

Sub-Function: constraints

```

function [g,H]=constraints(x,t)

% x = cell thickness (th)- Decision variable
% t(1) = Epsilon for Max flexure modulus
% t(2) = Epsilon for Max shear strain rate
% nu12 = Max shear strain rate
% G12 = Max flexure modulus
global scale_nu12
global E sig_yield
global h teta

% problem equality (none) and inequality nonlinear constraints

%Geometrical constraints
% t > 2 for manufacturing

```



```

% g1: Epsilon constraint on Max shear strain rate
t(1)
g(1,1) = ((E*x^3*(1+sind(teta)))/(3*cosd(teta)*h^3))/t(1) - 1;
t(2) = t(2)/scale_nu12;
% g2: constraint on the maximum shear flexure modulus
g(2,1) = (0.75*sig_yield*h)/(E*x*(1+sind(teta)))/t(2) - 1;
g
H=[];end

```

Appendix L

MATLAB Code for Problem 3.3.1

Master ASM Code

```
format long
clear all
clc
tic
% original non-normalized, non-segmented using FEASAM
%errortype in mpsimplex is relative error and searchtool enhanced
%feasolver updated to shape optimize both bottom and top surface
%Update y-coordinates for bottom layer(Because, lb & ub is for top surface)
%no of nodes in y direction = ne_ly +1;
%update x_ub based on ne_ly and l_x
%stress and deflection limits defined in feasolver
%dec_var == height of top surface nodes %decision variable
%ang == angle(parametric form) of tip load wrt horizontal measured CW%parameter
    %teta \
    %teta \
    %ang %teta \
%-----cant beam axis %%t=1 --> teta =90, t=sin(teta)
global E mu dens
global l_x l_y th ne_lx ne_ly
global load
global feacount X_array F_array X_count

feacount = 0;
X_array = [];
F_array = [];
X_count = 1;
%Initial discretization properties
l_x = 5;          %length of the beam in inch
l_y = 1;          %width of the beam in inch
th = 0.1;         %thickness of the beam in inch
ne_lx = 40;       % no. of cells in each row
ne_ly = 4;        % no. of cells in each column

%Material property definition
E = 29e6;         % Young's Modulus in psi
mu = 0.3;         %poisson's ratio
dens = 0.26;      %lb/in^3
tlow = 0.15;      %lbound of tip load in lb
```

```

tup = 1;          %ubound of tip load in lb
inc_p = 2;

%Loading conditions
load = 20;        % Tip load magnitude in lb

%optimization problem structure for approximate MP algorithm
Cantbeam_shape_struct = struct;

%objective function handle
Cantbeam_shape_struct.objective = @(x,t)Cantbeam_shape_obj(x,t);

%linear constraint matrices (inequality and equality)
%Form: A*x <= b+E*t
Cantbeam_shape_struct.A = [];      % Member 2 as largest tension
Cantbeam_shape_struct.b = [];
Cantbeam_shape_struct.E = [];
Cantbeam_shape_struct.Aeq = [];
Cantbeam_shape_struct.Eeq = [];
Cantbeam_shape_struct.beq = [];

%nonlinear constraint function handle
Cantbeam_shape_struct.constraints = @(x,t)Cantbeam_shape_feasolver(x,t);

%lower and upper bounds for decision variables
%not lesser than 0.55%top surface node %position ref %measured from original bottom
surface
%add 0.5 to the lb defined in problem definition
Cantbeam_shape_struct.lb = 0.666*ones(ne_lx+1,1);
%add 0.5 to the lb defined in problem definition%Here measured from original bottom
surface
Cantbeam_shape_struct.ub = 0.9999*ones(ne_lx+1,1);

% Definition for parameter space, tolerance and problem data
pSpace = Polyhedron('lb',tlow, 'ub',tup);
tol = 0.1;
problem = Cantbeam_shape_struct;

% Calling main AMP function to find optimal parametric solution
[xstar,fstar,CRstar,metrics] = AMPmain(tol,pSpace,problem);

CRsize = length(CRstar);

```

Master FEA code

```
function [c,ceq] = Cantbeam_shape_feasolver(dec_var,ang)

global E mu dens
global l_x l_y ne_lx ne_ly th
global load feacount
scale = 1;

% %temporary test
% dec_var = linspace(1,0.66,21)
% ang = 0.20;
% E = 29e6;           % Young's Modulus in psi
% mu = 0.3;           % poisson's ratio
% dens = 0.26;        % lb/in^3
% l_x = 10;           % length of the beam in inch
% l_y = 1;            % width of the beam in inch
% th = 0.1;           % thickness of the beam in inch
% ne_lx = 20;         % no. of cells in each row
% ne_ly = 4;          % no. of cells in each column
% load = 13;
% feacount = 0;
dec_var = 2*dec_var;    % getting the symmetric geometry
feacount = feacount + 1;
length(dec_var);

% Direction for load is corrected in equation found below
% Weight also calculated here in first part of Jacobian calc

t = ang;               % parameter
% stress limit definition
defl_limit = 3e-3*l_x;  % vertical deflection constraint at tip
sigmac_limit = (4.83e-4*E)*scale; % compressive stress constraint in member 1
sigmat_limit = (8.74e-4*E)*scale; % tensile stress constraint in member 2

tot_ne = ne_lx*ne_ly; % no. of cell elements in the domain(updated)
n_lx = ne_lx + 1;      % no. of nodes in each row
n_ly = ne_ly + 1;      % no. of nodes in each column
dl_x = l_x/ne_lx;      % length of each element in row direction
% dl_y = l_y/ne_ly;     % length of each element in column direction
% dl_y = dec_var/ne_ly; % updated dl_y to vector for compatibility of cells after changing
top nodes
```

```

dl_y = dec_var/ne_ly; %updated dl_y to vector for compatibility of cells after changing
top and bottom nodes
n_l = (n_lx * n_ly); %total number of nodes in the initial domain(updated)
y_outer = dec_var+(1-dec_var)/2; %y-coordinates for outer most nodes

node=zeros(n_l,2);
I = 1; %initializing for lower surface
for i=1:n_lx %assigning y-coordinates for lower surface
    node(I,1)=(i-1)*dl_x;
    node(I,2) = 1-y_outer(i); %y-coordinate
    I = I+1;
end

J = I-n_lx; %counter to extract y coor values of nodes in previous lower layer

% Co-ordinates determination for original rectangular domain
for j=2:n_ly % from bottom to top, horizontal lines
    for i=1:n_lx
        node(I,1)=(i-1)*dl_x; %x-coordinate
        node(I,2)= node(J,2)+(dl_y(i)); %y-coordinate %updated for changed
        nodeposition%nonsym
        I = I + 1;
        J = J + 1;
    end
end

%Bring back I to node no. starting from left side of top row
I=I-n_lx; %I is tot_nodeno. + 1, Calculation taken care of

%Appending nodes on the outermost surface(for shape change)
for i = 1:n_lx %looping over number of nodes along horizontal-axis
    node(I,2) = y_outer(i); %y-coordinate
    I = I+1;
end

% Formation of element node connectivity vector Ien
Ien=zeros(tot_ne,4);
e=1; % Square Element counter
l=1; % To automate 1st node number for each ele
J=1; % 1st node number for each element counter
inc = n_lx; % To find 3rd node number for each square element from 2nd ele
for j=1:ne_ly

```

```

Ien(e,1) = J;
for i=1:ne_lx
    Ien(e,1) = l;
    Ien(e,2) = Ien(e,1) + 1;
    Ien(e,3) = Ien(e,2) + inc;
    Ien(e,4) = Ien(e,3) - 1;
    l = l+1;
    e = e+1;
end
J = J+n_lx;
l = J;
end

nen = 4;
ndof = 2; % no. dof per node
edof = nen*ndof; % total no.of dof for each ele
tot_dof = n_l*ndof; % total no. of dof for mesh
I = 0;
for e = 1 : tot_ne % (Loop over elements)
    for i = 1 : nen % (Loop over number of element nodes)
        I = Ien(e,i); % (Global node number)
        for a = 1 : ndof % (Loop over number of dof/node)
            p = ndof*(i - 1) + a; % (Increment element equation number)
            LM(e,p) = ndof*(I - 1) + a; % (Fill Matrix with global equation number)
        end
    end
end

% Defining Guass quadrature rule
m2=2; % no. of gauss points in x direction
n1=2; % no. of guass points in y direction
[xsi,wx]=GLTable(m2); %xsi, W is weight func & since m=n
[eta,we]=GLTable(n1); %eta,
% K matrix, F matrix formation
x=zeros(4,1);
y=zeros(4,1);
D = (E/(1-(mu*mu)))*[1 mu 0; mu 1 0; 0 0 0.5*(1-mu)];
K=sparse(tot_dof,tot_dof);
for e=1:tot_ne
    m1=zeros(edof,edof);
    for i=1:4
        x(i)=node(Ien(e,i),1);
        y(i)=node(Ien(e,i),2);
    end
end

```

```

xy=[x(1),y(1);x(2),y(2);x(3),y(3);x(4),y(4)];
for i=1:m2
    for j=1:n1
        % B_cap or shape function partial derivatives wrt xsi and eta
        N1x=-1/4*(1-(eta(j))); % N1,xsi
        N2x=-N1x; % N2,xsi
        N3x=1/4*(1+(eta(j)));
        N4x=-N3x;
        N1e=-1/4*(1-(xsi(i)));
        N2e=-1/4*(1+(xsi(i)));
        N3e=-N2e;
        N4e=-N1e;
        % B matrix for determination of K
        B_temp = [N1x,N2x,N3x,N4x;N1e,N2e,N3e,N4e];
        J_trans = B_temp*xy;
        %J_trans_inv = inv(J_trans);
        Bd = J_trans\B_temp; % Same as J_trans_inv*B_cap
        B = [Bd(1,1) 0,Bd(1,2) 0,Bd(1,3) 0,Bd(1,4) 0; 0 Bd(2,1), 0 Bd(2,2), 0 Bd(2,3), 0
Bd(2,4); Bd(2,1) Bd(1,1),Bd(2,2) Bd(1,2), Bd(2,3) Bd(1,3), Bd(2,4) Bd(1,4) ];
        A = th*(B'*D*B)*det(J_trans)*wx(i)*we(j);
        m1 = m1 + A;
    end
end
K(LM(e,:),LM(e,:))= K(LM(e,:),LM(e,:))+ m1;
end
K;
% Determination of F matrix for each element
fe = zeros(edof,1);
s=0;
F=sparse(tot_dof,1);
for e=1:tot_ne
    m2 = zeros(nen,nen);
    for i=1:4
        x(i)=node(Ien(e,i),1);
        y(i)=node(Ien(e,i),2);
    end
    xy=[x(1),y(1);x(2),y(2);x(3),y(3);x(4),y(4)];
    p_count=1;
    for p=1:nen
        fe_node = zeros(ndof,1);
        for q=1:nen
            sum=0;
            for i=1:2
                for j=1:2

```

```

N1 = (1/4)*(1-xxi(i))*(1-eta(j));
N2 = (1/4)*(1+xxi(i))*(1-eta(j));
N3 = (1/4)*(1+xxi(i))*(1+eta(j));
N4 = (1/4)*(1-xxi(i))*(1+eta(j));
N = [N1;N2;N3;N4];
% for calculation of J_transpose
N1x=-1/4*(1-(eta(j))); %N1,xxi
N2x=-N1x; %N2,xxi
N3x=1/4*(1+(eta(j)));
N4x=-N3x;
N1e=-1/4*(1-(xxi(i)));
N2e=-1/4*(1+(xxi(i)));
N3e=-N2e;
N4e=-N1e;
B_temp1 = [N1x,N2x,N3x,N4x;N1e,N2e,N3e,N4e];
J_trans = B_temp1*xy;
%calculation of m11 by adding at each gauss points
sum = sum + ((N(p))*(N(q))*wx(i)*we(j)*th*det(J_trans));
end
end
m2(p,q) = m2(p,q) + sum;
end
fb = -0.28; %psi per node
for r=1:nen
    fe_node = fe_node + m2(p,r)*[0;fb];
end
fe(p_count,1) = fe_node(1,1);
fe(p_count+1,1) = fe_node(2,1);
p_count =p_count+2;
end
F(LM(e,:),1) = F(LM(e,:),1)+ fe(:,1);
end
F(tot_dof-1) = load*sqrt(1-t^2) + F(tot_dof-1); %load*cos(ang) horizontal right
F(tot_dof) = -load*t + F(tot_dof); %load*sin(ang) vertically down

% Ftot = F + Ftract;
% Ftot

% Extraction of K and F matrix to determine the temperature (Eliminate
% singularity
%K matrix extraction
K_Eh=zeros(tot_dof-2*n_ly,tot_dof);
j=1;
z=3;

```



```

for i=1:n_ly % Sequential extraction of elements in each row along the column
% Row extraction
K_Eh(j:i*2*(n_lx-1),:) = K(z+(i-1)*(2*n_lx) :i*(2*n_lx),:);
j=j+2*(n_lx-1);
end
j=1;
z=3;
for i=1:n_ly % Sequential extraction of elements in each row along the column
% Column extraction from the rows extracted
K_E(:,j:i*2*(n_lx-1)) = K_Eh(:,z+(i-1)*(2*n_lx) :i*(2*n_lx));
j=j+2*(n_lx-1);
end
% F matrix extraction
F_E=zeros(tot_dof-2*n_ly,1);
j=1;
z=3;
for i=1:n_ly
    F_E(j:i*2*(n_lx-1),1)=F(z+(i-1)*(2*n_lx):i*(2*n_lx),1);
    j=j+2*(n_lx-1);
end
% Determination of Displacement
d_E = K_E\F_E;
displ = zeros(tot_dof,1);
j=1;
z=3;
for i=1:n_ly
    displ(z+(i-1)*2*n_lx : i*2*n_lx,1) = d_E(j:i*2*(n_lx-1),1);
    j=j+2*(n_lx-1);
end
displ;

% Stress & Strain in each Element
Elemental_Strain = zeros(3,tot_ne);
Elemental_Stress = zeros(3,tot_ne);
for e = 1:tot_ne
    eta = 0;
    psi = 0;
    eta_n = [-1,1,1,-1];
    psi_n = [-1,-1,1,1];
    DispF = zeros(8,1);
    a = -(1/4)*(1-eta);
    b = (1/4)*(1-eta);
    c = (1/4)*(1+eta);
    d = -(1/4)*(1+eta);

```

```

p = -(1/4)*(1-psi);
q = -(1/4)*(1+psi);
r = (1/4)*(1+eta);
s = (1/4)*(1-psi);
for i=1:4
x(i)=node(Ien(e,i),1);
y(i)=node(Ien(e,i),2);
end
xy=[x(1),y(1);x(2),y(2);x(3),y(3);x(4),y(4)];
B1 = [a,b,c,d;p,q,r,s];
J = B1*xy;
DeterminantJ = det(J);
%InverseJ = inv(J);
B = J\B1;
a1 = B(1,1); b1 = B(1,2); c1 = B(1,3); d1 = B(1,4);
p1 = B(2,1); q1 = B(2,2); r1 = B(2,3); s1 = B(2,4);
B = [a1,0,b1,0,c1,0,d1,0;0,p1,0,q1,0,r1,0,s1;p1,a1,q1,b1,r1,c1,s1,d1];
for i = 1 : 8
L_disp = displ(LM(e,i));
DispF(i,1) = DispF(i,1)+L_disp;
end
Strain = B*DispF;
Stress = D*Strain;
Elemental_Strain(:,e) = Elemental_Strain(:,e)+Strain;
Elemental_Stress(:,e) = Elemental_Stress(:,e)+Stress;
% Calculation of principal and von-mises
avg = (Elemental_Stress(1,e) + Elemental_Stress(2,e))/2;
R = sqrt(((Elemental_Stress(1,e) - avg))^2+(Elemental_Stress(3,e))^2);
P1 = avg+R;
P2 = avg-R;
Von_Mises(e) = sqrt((P1^2) - (P1*P2) + (P2^2));
end
Maximum_Von_Mises = max(Von_Mises);

% Post processing
Ele_Strain = zeros(3,tot_ne);
Ele_Stress = zeros(3,tot_ne);
E_Von_Mises = zeros(tot_ne,1);
for e = 1:tot_ne
eta = 1;
psi = 1;
eta_n = [-1,1,1,-1];
psi_n = [-1,-1,1,1];
DispF = zeros(8,1);

```

```

a = -(1/4)*(1-eta);
b = (1/4)*(1-eta);
c = (1/4)*(1+eta);
d = -(1/4)*(1+eta);
p = -(1/4)*(1-psi);
q = -(1/4)*(1+psi);
r = (1/4)*(1+eta);
s = (1/4)*(1-psi);
for i=1:4
    x(i)=node(Ien(e,i),1);
    y(i)=node(Ien(e,i),2);
end
xy=[x(1),y(1);x(2),y(2);x(3),y(3);x(4),y(4)];
B1 = [a,b,c,d;p,q,r,s];
J = B1*xy;
DeterminantJ = det(J);
%InverseJ = inv(J);
B = J\B1;
a1 = B(1,1); b1 = B(1,2); c1 = B(1,3); d1 = B(1,4);
p1 = B(2,1); q1 = B(2,2); r1 = B(2,3); s1 = B(2,4);
B = [a1,0,b1,0,c1,0,d1,0;0,p1,0,q1,0,r1,0,s1;p1,a1,q1,b1,r1,c1,s1,d1];
for i = 1 : 8
    L_disp = displ(LM(e,i));
    DispF(i,1) = DispF(i,1)+L_disp;
end
E_Strain = B*DispF;
E_Stress = D*E_Strain;
Ele_Strain(:,e) = Ele_Strain(:,e)+E_Strain;
Ele_Stress(:,e) = Ele_Stress(:,e)+E_Stress;
% Calculation of principal and von-mises
avg = (Ele_Stress(1,e) + Ele_Stress(2,e))/2;
R = sqrt(((Ele_Stress(1,e) - avg)^2)+(Ele_Stress(3,e)^2));
P1 = avg+R;
P2 = avg-R;
E_Von_Mises(e) = sqrt((P1^2) - (P1*P2) + (P2^2));
end
E_Maximum_Von_Mises = max(E_Von_Mises);

c(1) = E_Maximum_Von_Mises - sigmac_limit; %stress constraint
c(2) = max(abs(displ)) - defl_limit; %deflection limit
ceq = [];
c;

```

Appendix M

MATLAB FEA Code to Extract Optimized Beam Shape

```
%function [out] = OptParEvaluate(xstar)

%Update legend based on number of critical regions. Only 5 legends can be
%there right now
%Function to evaluate parametric results and plot
%pass a parameter value and find shape from evaluating optimal parametric
%solution from corresponding critical region
dec_var = 1:(ne_lx+1);
xaxis_i = 0;
xaxis_f = 5;
yaxis_i = -2;
yaxis_f = 2;
%Note if 4 critical regions comment str5

%finding the optimal node positions for each angle(p) listed for every critical region
for k=1:length(CRstar)
    t = (CRstar(k).V(1,1)+CRstar(k).V(2,1))/2;
    for i=1:length(t)
        for j=1:(ne_lx+1) %length(dec_var)
            xu(i,j) = xstar{1}(j,1)*t(i) + xstar{1}(j,2); %xu(i,j) = all top y node pos for t(i);
row i vector
            xl(i,j) = 1-xu(i,j);
            end
            fobj(k,i) = fstar{k}(1,1)*t(i) + fstar{1}(1,2);
            end
            xcell{k,1}(:, :) = xu; %xcell(:,1) contains xu(i,j) for each Critical
region
            xcell{k,2}(:, :) = xl;
        end
    end

% Plot optimal beam shape for each listed angle(p) within every critical region
plot_type = ["r*-"; "g+-"; "b*-"; "c+-"; "m*-"];
dec_varplot = linspace(xaxis_i,xaxis_f,ne_lx+1); %scaled to show true beam length
figure(1)
for k=1:length(CRstar)
    t = (CRstar(k).V(1,1)+CRstar(k).V(2,1))/2;
    for i=1:length(t)
        plot(dec_varplot,xcell{k,1}(i,:),plot_type(k,2:4));
        hold on
    end
end
```

```

end
xlabel('Length of the beam in inch','FontSize',14);
ylabel('Height of the beam in inch','FontSize',14);
title('Optimal shape of the beam, \epsilon - 1%');
axis([xaxis_i,xaxis_f, yaxis_i, yaxis_f]);

end
str1 = sprintf('CR1: t = %f',(CRstar(1).V(1,1)+CRstar(1).V(2,1))/2);
str2 = sprintf('CR1: t = %f',(CRstar(2).V(1,1)+CRstar(2).V(2,1))/2);
str3 = sprintf('CR3: t = %f',(CRstar(3).V(1,1)+CRstar(3).V(2,1))/2);
str4 = sprintf('CR4: t = %f',(CRstar(4).V(1,1)+CRstar(4).V(2,1))/2);
% str5 = sprintf('Critical region: %f < p < %f',CRstar(5).V(1,1),CRstar(5).V(2,1));
legend(['1: ' str1], ['2: ' str2], ['3: ' str3], ['4: ' str4]);

for k=1:length(CRstar)
    t = (CRstar(k).V(1,1)+CRstar(k).V(2,1))/2;
    for i=1:length(t)
        plot(dec_varplot,xcell{k,2}(i,:),plot_type(k,2:4));
        hold on
    end
end

end

```

REFERENCES

- [1] Acevedo J., and Pistikopoulos E.N., 1997. "A Multiparametric Programming Approach for Linear Process Engineering Problems under Uncertainty", *Industrial and Engineering Chemistry Research*, Vol 36, pp. 717-728.
- [2] Dominguez, L., and Pistikopoulos, E.N., 2011. "Recent Advances in Explicit Multiparametric Nonlinear Model Predictive Control", *Industrial and Engineering Chemistry Research*, Vol 50, pp. 609-619.
- [3] Dua, V., and Pistikopoulos, E.N., 1998. "Optimization Techniques for Process Synthesis and Material Design under Uncertainty", *Transactions of Institute of Chemical Engineers*, Vol 76, Part A, pp. 408-416
- [4] Haftka, R.T., and Gurdal, Z., 1992. "Elements of Structural Optimization", Kluwer Academic Publishers, Boston, pp. 167-169
- [5] Bemporad, A., and Filippi, C., 2006. "An Algorithm for Approximate Multiparametric Convex Programming", *Computational Optimization and Applications*, Springer Science, Vol 35, pp. 87-108.
- [6] Acevado J., and Pistikopoulos E.N., 1996. "A Parametric MINLP Algorithm for Process Synthesis Problems under Uncertainty", *Industrial and Engineering Chemistry Research*, Vol 35, pp. 147-158.
- [7] Dominguez, L., and Pistikopoulos, E.N., 2013. "A Quadratic Approximation-Based Algorithm for the Solution of Multiparametric Mixed-Integer Nonlinear Programming Problems", *American Institute of Chemical Engineers Journal*, Vol 59, No 2, pp. 483-495.
- [8] Dua, V., and Pistikopoulos, E.N., 2003. "Parametric Optimization in Process Systems Engineering: Theory and Algorithms", *Proceedings of Indian National Science Academy*, 69 A, pp. 429-444.
- [9] Pistikopoulos, E.N., Georgiadis, M.C., Dua V., 2007. "Parametric Programming and Control: From Theory to Practice", 17th European Symposium on Computer Aided Process Engineering, Elsevier.
- [10] Narciso, D.A.C., Faisca, N.P., Pistikopoulos, E.N., 2008. "A Framework for Multiparametric Programming and Control – An Overview", *IEEE International Engineering Management Conference*.
- [11] Gal, T., and Nedoma, J., 1972. "Multiparametric Linear Programming", *Management Science*, Vol 18, No. 7, pp. 406-422.
- [12] Dua V., and Pistikopoulos, E.N., 1999. "Algorithms for the Solution of Multiparametric Mixed-Integer Nonlinear Optimization Problems", *Industrial and Engineering Chemistry Research*, Vol 38, pp. 3976-3987.

- [13] Johansen, T.A., 2002. "On Multi-parametric Nonlinear Programming and Explicit Nonlinear Model Predictive Control", In Proceedings of the 41st IEEE Conference on Decision and Control, Las Vegas, pp. 12-15.
- [14] Pistikopoulos, E.N., Dua, V., 1998. "Planning Under Uncertainty, a Parametric Optimization", Proceedings of 3rd International Conference on Foundations of Computer-Aided Process Operations, Editors (J.F. Békny and Blau G.E.), pp-164
- [15] Pistikopoulos, E.N., Dua, V., Bozinis, N.A., Bemporad, A., and Morari, M., 2002. "On-Line Optimization Via Off-Line Parametric Optimization Tools", Computers and Chemical Engineering, Vol 26, pp.175-185.
- [16] Bemporad, A., Borrelli, F., and Morari, M., 2002. "Model Predictive Control of Linear Programming – The Explicit Solution", IEEE Transactions On Automatic Control, Vol 47, No. 12, pp.1974-1985.
- [17] Papalexandri, K.P., and Dimkou, T.I., 1998. "A Parametric Mixed-Integer Optimization Algorithm for Multiobjective Engineering Problems Involving Discrete Decisions", Industrial and Engineering Chemistry Research, Vol 37, pp. 1866-1882.
- [18] Ju, J., Summers, J.D., Ziegert, J., and Fadel, G., 2009. "Design of Honeycomb Meta-materials for High Shear Flexure", ASME International Design Engineering Technical Conferences & Computers and Information In Engineering Conference, California, USA.
- [19] Sakizlis V., Kakalis N.M.P., Dua V., Perkins J.D., Pistikopoulos E. N., 2004. "Design of robust model-based controllers via parametric programming", Automatica, Elsevier, Vol-40, pp 189 – 201.
- [20] Herceg, M., Kvasnica, M., Jones, C.N., and Morari, M., 2013. "Multi-Parametric Toolbox 3.0", European Control Conference, Zurich, Switzerland, pp.502-510.
- [21] Leverenz J., 2015. "Network Target Coordination For Multiparametric Programming", PhD thesis, Mathematical Sciences Department, Clemson University, Clemson, SC.
- [22] Sobieski, J.S., Kodiyalam, S., and Yang, R.Y., 2001. "Optimization of Car Body Under Constraints of Noise, Vibration, and Harshness (NVH), and Crash", Structural Multidisciplinary Optimization, Springer-Verlag, Vol 22, pp. 295-306.
- [23] Terlaky, T., and Curtis, F.E., 2010. "Modelling and Optimization: Theory and Applications", Proceedings in Mathematics and Statistics, Springer, Vol 21, pp. 77 – 104.
- [24] Ryu, J., and Pistikopoulos E.N., 2001. "Solving Scheduling Problems under Uncertainty using Parametric Programming", Proceedings of 6th IFAC Symposium on Dynamics and Controls of Process systems

- [25] MATLAB R2014, The MathWorks Inc.
- [26] ANSYS® Academic Research, Release 15.0
- [27] ModeFRONTIER 4.4.2, ESTECO.
- [28] Abaqus 6.12, Dassault Systems, Simulia, User's Manual.
- [29] Tondel P., Johansen, T.A., and Bemporad A., 2003. "An Algorithm For Multi-parametric Quadratic Programming and Explicit MPC Solutions", *Automatica*, Elsevier, Vol 39, pp. 489-497.
- [30] Chankong, V., and Haimes, Y. Y., 1983. "Multiobjective Decision Making: Theory and Methodology", *North-Holland Series in System Science and Engineering*, Vol 8, pp. 140-176.