OPTIMAL DESIGN OF A PARALLEL BEAM SYSTEM WITH ELASTIC SUPPORTS TO MINIMIZE FLEXURAL RESPONSE TO HARMONIC LOADING

by

BRET R. HAUSER

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2015

Dedication

To my wife, Karen

and my parents

.

Acknowledgements

I am most grateful for the wisdom and guidance shown to me by my Supervising Professor, Bo P. Wang. His interest, critique and constant encouragement have been a source of inspiration for me, and have made this research effort a very valuable process.

I would also like to express my appreciation to Professors Wen Chan and Kent Lawrence of UTA's Department of Mechanical and Aerospace Engineering for serving on my review committee.

My deepest gratitude goes to my father, Victor, who taught and inspires me to think big, not shy from big challenges and to learn new things. To my mother, Bettye, who taught me that with enough determination I could achieve those goals. And to my wife, Karen, whose daily love and support enable me to do what I was taught.

April 8, 2015

Abstract

OPTIMAL DESIGN OF A PARALLEL BEAM SYSTEM WITH ELASTIC SUPPORTS TO MINIMIZE FLEXURAL RESPONSE TO HARMONIC LOADING

Bret R. Hauser, M.S.

The University of Texas at Arlington, 2015

Supervising Professor: Bo P. Wang

Mechanical systems subject to vibration are prevalent across many industries. Harmonic problems can be especially challenging to optimize due to the likelihood that the response will be multi-modal; influenced by system natural frequencies throughout the design space. Further, analysis of these systems often involves large and complex computer models which require significant resources to execute. A parallel beam system, as evaluated with Finite Element Modeling (FEM), is used in this work to demonstrate a proposed method of identifying an optimum in a constrained, multi-modal environment suitable for Expensive Black Box Functions.

The presented method leverages benefits of a combined approach where the domain is first surveyed for potential areas of optimal response using a method of Steepest Feasible Descent (SFD), followed by a search in the optimal region using direct search methods. The method of SFD is made useful for constrained models by a penalty system including both deterministic and programmatic methods. A sensitivity-based search vector method also helps to manage situations where significant difference in magnitude exists among the design variables. Evidentiary support for these key program elements is provided using standardized test functions. The effectiveness of the method is also demonstrated by seeking a minimum flexural response for a parallel beam system subject to elastic support and response constraints.

Table of Contents

Acknowledgements iv

Abstract v

List of Illustrations x

List of Tables xiii

Chapter 1 Introduction 1

Background and Motivation 1

Objectives 3

Chapter 2 Overview of Problem under Study 5

Problem Description 5

Visualization of Response 8

Chapter 3 A Review of Theory and Literature 10

Optimization of Structural Dynamic Systems by Maximizing Fundamental Frequency 10

Dynamic Vibration Absorber Systems 17

Undamped DVA 17

Damped DVA - Theory 20

Damped DVA – Practical Application 24

Optimization of Smooth, Unimodal Functions 25

First-Order Methods 26

Second-Order Methods 29

Optimization of Non-Smooth, Multimodal Systems 29

Direct Methods 30

Genetic Algorithm 30

Particle Swarm Optimization 31

Sequential Quadratic Programming 33

Optimization of Expensive Black Box Functions 33

Chapter 4 Model Development and Validation 36

Modelling and Validation Plan 36
Theoretical Solutions - Dynamic 38
Theoretical Solutions - Static 41
Single Beam Model Validation 42
Description of 1D & 3D FEM Models 42
Mesh Sensitivity Results and Validation of FEM to Theoretical Models 44
Parallel Beam Model Validation (1D vs. 3D Harmonic) 51
Selected Model Description for Problem under Study 54
Chapter 5 Development and Characterization of the Optimization Algorithms 55
Identification of Candidate Methodologies 55
Down-Select Technique and Description of Test Functions 58
Down-Select Technique 58
Test Function Descriptions 58
De Jong’s Function 59
Rosenbrock’s Function 60
Rastrigin’s Function 61
Schwefel’s Function 62
Description of Optimization Program and Key Subroutines 63
Method of Steepest Feasible Descent from Global Sampling Points 65
Selection of a Feasible Search Direction 66
One-Dimensional Line Search Strategy for a Constrained Problem 71
Selection of Feasible Start Points 76
SQP Method from Global Sampling Points 76
Combination of Steepest Feasible Descent and SQP Methods 77
Use of a ‘Single-Objective Optimization’ Tool for ‘Multi-Objective Optimization’ 77
Algorithm Confirmation 78
One-Dimensional Line Search Effectiveness 78

Performance as an Unconstrained Search against Test Functions 78

Performance as a Constrained Search against Test Functions 87

Performance Test Parameter Definition – Overall Effectiveness and Efficiency 91

Method of Steepest Feasible Descent from Global Sampling Points 95

Performance as an Unconstrained Search against Test Functions 95

Performance as a Constrained Search against Test Functions 99

SQP Method from Global Sampling Points 101

Performance as an Unconstrained Search against Test Functions 101

Performance as a Constrained Search against Test Functions 105

Combination of Methods from Global Sampling Points 107

Performance as an Unconstrained Search against Test Functions 107

Performance as a Constrained Search against Test Functions 111

Selected Optimization Technique for Problem under Study 113

Chapter 6 Optimization of Parallel Beam Problem 120

Parametric Model Definition – Use of Super-Elements 120

Optimization Plan 122

Use of a ‘Single-Objective Optimization’ Tool for ‘Multi-Objective Optimization’................ 122

Development of the Objective Function 125

Development of the Constraint Functions 126

Additional Considerations 129

Results and Discussion 131

Chapter 7 Summary and Conclusions 145

Chapter 8 Opportunities for Further Research 150

Appendix A FEM Model Validation Data 151

Appendix B Flowcharts of Key Program Subroutines 160

Appendix C Full Program Listing 169

Appendix D Tabulated Results 216

References 223

Biographical Information...227

List of Illustrations

Figure 2-1 Illustration of problem under study 5
Figure 2-2 Visualization of maximum tip deflection over partial design space 9
Figure 3-1 Classical SDOF system with damping 10
Figure 3-2 FRF as a function of damping - classical SDOF system 12
Figure 3-3 Undamped dynamic vibration absorber 18
Figure 3-4 FRF of main and absorber masses - undamped DVA, μ=0.2 19
Figure 3-5 Damped dynamic vibration absorber 20
Figure 3-6 FRF of damped DVA with multiple damping magnitudes, μ=0.2 22
Figure 4-1 Close-up of 1D (beam element) model - nodal spacing = 0.050" 43
Figure 4-2 Close-up of 3D (solid element) model - nodal spacing $\cong$ 0.0125" 43
Figure 4-3 Mesh sensitivity and validation results - static deflection, cantilever beam 45
Figure 4-4 Mesh sensitivity and validation results - static deflection, simply supported beam 45
Figure 4-5 Mesh sensitivity and validation results - mode frequency, cantilever beam 47
Figure 4-6 Mesh sensitivity and validation results - mode frequency, simply supported beam 47
Figure 4-7 Theoretical results - mode shape, simply supported beam 49
Figure 4-8 FEM results (1D model, 200 NPI) - mode shape, simply supported beam 49
Figure 4-9 Theoretical results - mode shape, cantilever beam 50
Figure 4-10 FEM results (1D model, 200 NPI) - mode shape, cantilever beam 50
Figure 4-11 Parallel beam harmonic analysis model constraints and boundary conditions 51
Figure 4-12 Harmonic mesh convergence error - 1D elements 53
Figure 4-13 Harmonic mesh convergence error - 3D elements 54
Figure 5-1 De Jong's function in 2D 60
Figure 5-2 Rosenbrock's function in 2D 61
Figure 5-3 Rastrigin's function in 2D 62
Figure 5-4 Schwefel's function in 2D 63

Figure 5-5 Program flowchart - main program flow 65

Figure 5-6 Identification of constraint boundaries 75

Figure 5-7 1-D line search, De Jong's function 80

Figure 5-8 1-D line search, Rosenbrock's function 81

Figure 5-9 1-D line search, Rastrigin's function (search vector #1) 82

Figure 5-10 1-D line search, Rastrigin's function (search vector #2) 83

Figure 5-11 1-D line search, Rastrigin's function (search vector #3) 84

Figure 5-12 1-D line search, Schwefel's function (search vector #1) 85

Figure 5-13 1-D line search, Schwefel's function (search vector #2) 86

Figure 5-14 Example of penalty function test result, De Jong's function 88

Figure 5-15 Comparison of penalty methods - De Jong's test function 90

Figure 5-16 Starting point locations - De Jong's test function 93

Figure 5-17 Starting Point locations Rosenbrock's test function 93

Figure 5-18 Starting point locations - Rastrigin's test function 94

Figure 5-19 Starting point locations - Schwefel's test function 94

Figure 5-20 Test result, steepest feasible descent - De Jong's function 96

Figure 5-21 Test result, steepest feasible descent - Rosenbrock's function 96

Figure 5-22 Test result, steepest feasible descent - Rastrigin's function 97

Figure 5-23 Test result, steepest feasible descent - Schwefel's function 97

Figure 5-24 Test result, constrained steepest feasible descent - De Jong's function 99

Figure 5-25 Test result, constrained steepest feasible descent - Rosenbrock's function 100

Figure 5-26 Test result, constrained steepest feasible descent - Rastrigin's function 100

Figure 5-27 Test result, constrained steepest feasible descent - Schwefel's function 101

Figure 5-28 Test result, direct method (SQP) - De Jong's function 102

Figure 5-29 Test result, direct method (SQP) - Rosenbrock's function 102

Figure 5-30 Test result, direct method (SQP) - Rastrigin's function 103

Figure 5-31 Test result, direct method (SQP) - Schwefel's function 103

Figure 5-32 Test result, constrained direct method (SQP) - De Jong's function 105

Figure 5-33 Test result, constrained direct method (SQP) - Rosenbrock's function 106

Figure 5-34 Test result, constrained direct method (SQP) - Rastrigin's function 106

Figure 5-35 Test result, constrained direct method (SQP) - Schwefel's function....................... 107

Figure 5-36 Test result, combined method - De Jong's function.. 108

Figure 5-37 Test result, combined method - Rosenbrock's function 109

Figure 5-38 Test result, combined method - Rastrigin's function 109

Figure 5-39 Test result, combined method - Schwefel's function 110

Figure 5-40 Test result, constrained combined method - De Jong's function 111

Figure 5-41 Test result, constrained combined method - Rosenbrock's function 112

Figure 5-42 Test result, constrained combined method - Rastrigin's function 112

Figure 5-43 Test result, constrained combined method - Schwefel's function 113

Figure 5-44 Additional direct method test, Schwefel's function, unconstrained 117

Figure 6-1 Super-element models - schematic layout... 122

Figure 6-2 Harmonic tip response for optimal configuration of parallel beam............................ 134

Figure 6-3 Harmonic tip response for start point #6 135

Figure 6-4 Mode shape of family variant '2', 82.5 Hz 136

Figure 6-5 Mode shape of family variant '3', 42.5 Hz 136

Figure 6-6 SQP Variable Convergence - Length Dimensions.. 138

Figure 6-7 SQP Variable Convergence - Spring Stiffnesses 139

Figure 6-8 SQP Variable Convergence - Damping Coefficients 139

List of Tables

Table 2-1 Input and response variable details for problem under study 7

Table 4-1 Output metrics for harmonic model comparison 52

Table 4-2 Acceptable nodal densities for study of parallel-beam.. 54

Table 5-1 Input parameters for test of Sn derivation methodology 69

Table 5-2 Resulting Sn vectors for test of Sn derivation methodology 69

Table 5-3 Comparison of penalty methods, radius of path optimums from center 89

Table 5-4 Program parameters - test function evaluations 92

Table 5-5 Constraint region parameters - constrained test function evaluations.......................... 95

Table 5-6 Test function result - method of steepest feasible descent.. 98

Table 5-7 Test function result - direct method (SQP).. 104

Table 5-8 Test function result - combined method.. 110

Table 5-9 Test function result - summary of unconstrained results 114

Table 5-10 Additional unconstrained test function result - Schwefel's function 116

Table 5-11 Comparison of Optimization Results to Literature Results 119

Table 6-1 Normalization factors used for response variables.. 124

Table 6-2 Weighting factors used for response variables 125

Table 6-3 Starting points used in the parallel beam optimization problem 130

Table 6-4 Program parameters – parallel-beam optimization problem...................................... 131

Table 6-5 Results data – summary of function evaluation count 132

Table 6-6 Design variable values for optimal configuration of parallel beam.............................. 133

Chapter 1

Introduction

Background and Motivation

Mechanical systems subject to vibration are prevalent across many industries including aerospace, mining, machine tools, medical devices, consumer appliance, building construction, etc. Although very different in their application, these industries sometimes share the need to minimize one or more aspects of flexural deformation given some time-variant input load and with consideration for a set of design constraints.

Multiple options are available to the designer as to how best to optimize such a system in order to avoid operation at or near a natural frequency. For example, one option would be to modify some property of the system (geometry, material, etc.) in order that the structure's first (lowest) natural frequency is greater than the range of intended operation. In so doing, the structure is 'assured' that it will not operate at or near a natural frequency, and will not pass through one as the operating frequency ramps up and down prior to and following the intended operation.

Sometimes however, design constraints exist which prevent the lowest natural frequency from exceeding the operating range of the device. For example, the system may have design constraints that force it to operate at or near a natural frequency(s) at least some of the time. In these cases, it is important that the flexural response of the system also be considered at and near the natural frequency(s) in order to assure safe operation. One common approach used by designers to limit flexural response at and near these natural frequencies is to incorporate either a damped or undamped Dynamic Vibration Absorber (DVA). Such systems can be very effective at minimizing the flexural response and commonly involve the addition of spring-mass elements (with or without damping) 'tuned' to absorb system energy at a particular natural frequency(s) of interest.

A subset of mechanical vibration problems further exist whereby a 'family' or 'platform-system' design is built around a common 'base structure'. Such a family-style design adds challenge to the design process because the base system incorporates a variety of additive sub-systems which may differ from each other in form, fit and/or function, but all of which must function within (often common) design parameters. Examples of such family systems include cantilever-type drilling platforms (oil, water, etc.), machine tools intended to support a broad family of fixed- and adjustable-boring tools, or the support structure of a multi-purpose engine test cell. In the case of these family system examples, the ‘end-effect’ sub-system is dynamically loaded and adds complication to the design process in that the overall system design has a much broader scope with respect to vibrational response. In these instances, the ‘base’ structure must be designed to account for (potentially) multiple natural frequency responses due to the changing geometric configurations. Also, geometric and other design constraints for the platform-style product may limit the variables available for optimization. In fact, these design limitations may preclude some variables which have the most significant impact on improving the system from being optimized at all. For example, the mounting geometry to support the family variant ‘end effector’ may be ‘fixed’ in its design, regardless of the vibrational needs of the various family variant end effectors. Here, a variety of natural frequency responses among the family variants is possible and a simple DVA design may not be adequate (because of the various configurations and natural frequencies involved). Nevertheless, the system response including all of the family variants must be optimized to a safe level, therefore, the art of system optimization must include appropriate compromise among the family variants.

When seeking an optimum design for flexural system response (minimum deformation, minimum stress, etc.), the mathematical form of the response must be considered. If the response varies smoothly for the given input parameters, as a parabolic bowl for example, then the response meets the criteria for a smooth, unimodal function and a variety of optimization methods may be efficiently used. If, as in the case of our example, a vibratory deformation response is to be minimized in a frequency range that includes natural frequency response(s), then the response is no longer smooth and instead likely to be multimodal. To further complicate matters, family-style products often produce multiple natural frequencies within the intended operating range. If an optimization strategy that is appropriate for smooth, unimodal responses were used, then a local minimum would be found based upon the assumed start condition. The identified local minimum though may not be a suitable ‘global’ minimum because the unimodal-based optimization strategy cannot ‘cross’ the various resonant response peaks from one area of ‘local minimum’ to the next. Without consideration for this limitation to the optimization process, suitable trade-offs may not be appropriately identified between input variables and responses; causing design limitations in the effectiveness, efficiency and/or safety of the ultimate system.

Finally, the effective simulation of many such family-style systems involve models (computer simulation, test configurations, etc.) that are sufficiently large and involve enough complexity that they have burdensome temporal or financial cost for each solution pass. Further, the mathematical form of the response for these systems is typically unknown to the designer prior to exploration of the response; therefore the most efficient optimization strategy is likely to not be immediately obvious and to present a significant challenge to the overall design process. These models then meet the criteria of an ‘expensive black box’ function and require that the cost of an Optimization exercise be considered as a potentially significant obstacle in the overall process.

In the practice of mechanical system design, some combination of these factors frequently exists such that the overall optimization process is encumbered. Global search algorithms are documented in literature that attempt to solve such problems and identify a ‘global optimum’ solution. These are often complicated and less than ideal for systems with complex responses or for those which meet other criteria for ‘black-box’ systems. As such, the ‘design cost’ of utilizing these more complicated optimization algorithms as applied to a diversity of design problems increases; and, further, can become overly burdensome on many product design efforts resulting in a general lack of use and failure to find a needed optimal solution.

Objectives

In this work, an optimization strategy is sought that is an effective compromise between finding an acceptable ‘global optimum’ and the overall design cost for harmonic response of family-style systems; specifically, one that may be modeled as a system of parallel beams. That is, an optimization methodology is explored which has limited complexity, and therefore has potential for efficient implementation on a variety of design problems, while at the same time being effective in producing an acceptable global optimum solution given a limited number of solution passes. Finite Element Analysis (FEA) of the harmonically-loaded system is used in this work and serves as an example of a black-box style system where useful details about the form of the response are not known prior to the optimization effort.

Chapter 2

Overview of Problem under Study

Problem Description

For simplicity of study, the family-style sample problem is defined to be a system of two parallel beams, one of which is common for all variants (the base structure) and a second that varies in geometry to form the ‘family’ of ‘end-effects’. Loading is applied to the system at the free end of the top (variable, end-effect) bar. Figure 2-1 illustrates the architecture of the problem.

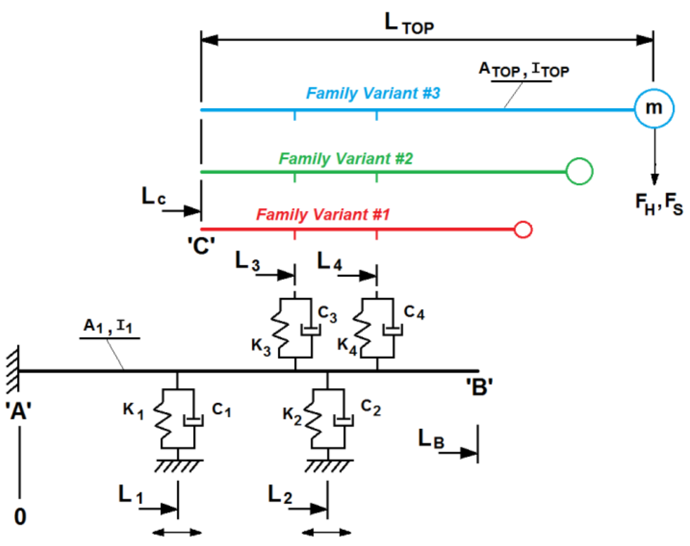


Figure 2-1 Illustration of problem under study

The specific problem to be solved through this work then, is to minimize the flexural response of the upper beam’s ‘tip mass’ given harmonic loading across a specified range of frequencies and subject to design variable constraints, consideration for the ‘optimal’ family response among all variants as well as a pre-defined ‘minimum stiffness’ for the system.

The base (bottom) beam is defined to be cantilevered, but also supported to ground by two (2) mid-span spring-damper mechanisms whose locations are variable among the family system (positions L_1 and L_2). That is, the locations of these lower supports as well as the value of their respective spring stiffness and damping values are variables to be optimized in the study. The overall length of the base beam (L_B) as well as its cross-sectional properties (A_1, I_1) and material properties are 'fixed' values and consistent for all family variants.

The top beam is attached to the bottom via two (2) spring-damper connections (positions L_3 and L_4). The placement of these 'inter-body' connections are defined to be common (fixed) among all family variants, but variable in stiffness & damping value. That is, the stiffness and damping value for each of these inter-body supports are variables to be optimized for optimal family response whereas the location of these supports are not. Three (3) different top beam configurations are included in the overall study to form the 'family'. The axial location of the top beam (L_c), cross-sectional properties (A_{Top}, I_{Top}) and material properties are fixed and common among all family variants. However, the overall length (L_{Top}) and size of the lumped mass (m), differ for each of the (3) family variants. These are 'fixed values' for each family variant and are not variables for optimization as part of the study.

Two types of loading are considered for the system. The first is a harmonic load (F_H) placed at the distal or tip-end of the second beam, common among all design variants. The second is a static load (F_S), also placed at the tip-end of the top beam but used to characterize a static deflection as a constraint for the Optimization process. The use of this statically driven constraint is intended to prevent the algorithm from identifying a solution which minimizes harmonic response at the cost of impractically low system stiffness. Note that gravity loading on the bar is neglected for the purposes of this optimization study.

Output (response) variables of interest are products of deformation of the top (variant) beam. Specifically, these response metrics include:

- Tip (free-end) displacement of top beam
    - Static response
    - Maximum integrated (discrete) sum and range of harmonic responses among frequencies within the operating range (at 0□ phase angle for each frequency polled)

- Maximum harmonic deflection at any point along the top beam among frequencies within the operating range

A summary of input values and response variables is provided in Table 2-1 below.

Table 2-1 Input and response variable details for problem under study

Input Values		Family Variant		
		1	2	3
Fixed Values				
Bottom Beam				
Length	L_B	72 in		
Cross-Section	A_1, I_1	Solid, Circular – OD = 0.500 in		
Material Properties *Young's Modulus*	E_B	$29x10^6$ psi		
Poisson's Ratio	μ_B	0.29		
Mass Density	ρ_B	$7.324x10^{-4}$ Lb_m/in^3		
Top Beam				
Location	L_c	36 in		
Support Locations	L_3	45 in		
	L_4	63 in		
Cross-Section	A_{Top}, I_{Top}	Solid, Circular – OD = 0.250 in		
Material Properties *Young's Modulus*	E_T	$29x10^6$ psi		
Poisson's Ratio	μ_T	0.29		
Mass Density	ρ_T	$7.324x10^{-4}$ Lb_m/in^3		
Force Magnitude	F_H, F_s	0.5 Lb_f		
Frequencies Studied	f_{ii}	10~100 Hz, 2.5 Hz increments		
Length	L_{Top}	48 in	54 in	60 in
Lumped Mass	m	$4.8x10^{-5}$ lb_m	$3.8x10^{-4}$ lb_m	$1.3x10^{-3}$ lb_m
Variables (to be Optimized)				
Bottom Beam				
Support #1	L_1	$0 < L_1 < L_2$		
	K_1	$500 < K_1 < 50000$ Lb_f/in		
	C_1	$0.005 < C_1 < 0.500$ *(damping ratio)*		
Support #2	L_2	$L_1 < L_2 < L_B$		
	K_2	$500 < K_2 < 50000$ Lb_f/in		
	C_2	$0.005 < C_2 < 0.500$ *(damping ratio)*		
Top Beam				
Support #3	K_3	$500 < K_3 < 50000$ Lb_f/in		
	C_3	$0.005 < C_3 < 0.500$ *(damping ratio)*		

Table 2.1 - *Continued*

Support #4	K_4	$500 < K_4 < 50000$ Lb_f/in
	C_4	$0.005 < C_4 < 0.500$ *(damping ratio)*
Response Variables		
Static Load		
Tip Deflection		
Harmonic Load		
Tip Deflection – sum of harmonic responses among frequencies (defined in detail later)		
Tip Deflection – range of harmonic response among frequencies (defined in detail later)		

Visualization of Response

Figure 2-2 illustrates the complexity of the response for tip deflection of the parallel beam problem (Figure 2-1) as a function of two (2) of the input variables. Specifically, a 'preliminary' model using the commercially available FEA software ANSYS ® (SAS IP, Inc.) is used here to explore variation in the location of the two (2) ground supports to simply illustrate the potential for multimodality of the harmonic tip responses. As shown, many 'peaks' exist within the design space where resonant conditions exist. Intermingled within these areas of maximum lie several areas of potential minimum where an acceptable global minimum may exist. Further, the location and interrelationship of these resonant areas serve to 'isolate' the potential areas of minimum from one another. Because the surface cannot be considered a smooth, unimodal surface over the design space, optimization algorithms intended for such are not reliable for use here in order to find a global optimum. Therefore, the need for a method that includes this flexibility is needed.

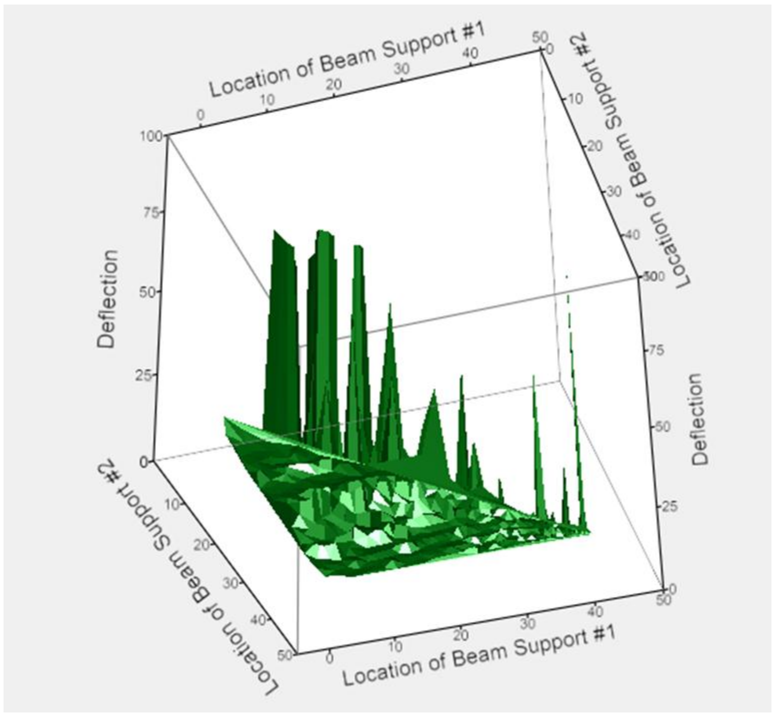


Figure 2-2 Visualization of maximum tip deflection over partial design space

Chapter 3

A Review of Theory and Literature

Optimization of Structural Dynamic Systems by Maximizing Fundamental Frequency

It is well understood that all physical systems capable of storing energy also have a defined set of natural frequencies at which that system will naturally vibrate.[1] Mechanical systems, such as those discussed within the scope of this paper are principally elastic systems, being comprised of spring, mass and damper characteristics and as such include the capability of dynamic response; whereby energy is transferred from a potential state to a kinetic state and back. There exists then a set of frequencies at which these systems will naturally vibrate without a continued external periodic force. If, however, an external periodic forcing function is applied to these systems then a vibrational response is created. If the forcing function frequency coincides with the natural frequency of the system then the system is said to be in 'resonance' and the amplitude of the vibrational response can become much larger than the amplitude of the forcing function. In fact, if no damping exists as part of the system then the response amplitude at resonance mathematically becomes infinite.

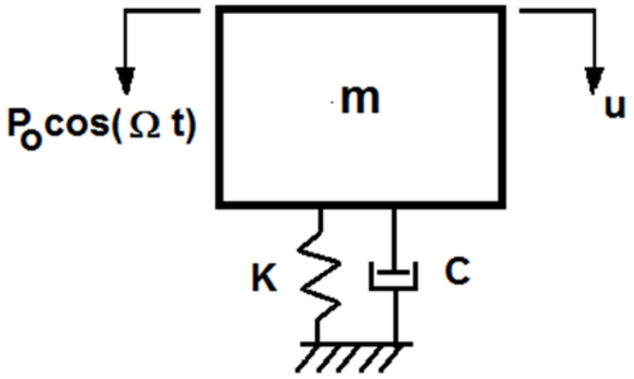


Figure 3-1 Classical SDOF system with damping

Considering a classical, Single Degree of Freedom (SDOF) model including spring, mass and dashpot elements as shown in Figure 3-1, the Equation of Motion is [2]

$$m\ddot{u} + c\dot{u} + ku = p_o \cos(\Omega t) \tag{3-1}$$

And the steady state solution is

$$u_p = \mathrm{U}\cos(\Omega t - \alpha) \tag{3-2}$$

where:

$$\mathrm{U} = \mathit{steady\ state\ amplitude}$$

$$\alpha = \tan^{-1}\left[\frac{c\Omega}{k - m\Omega^2}\right] = \mathit{phase\ lag\ due\ to\ damping} \tag{3-3}$$

$$\omega_n = \sqrt{\frac{k}{m}} = \mathit{undamped\ natural\ frequency} \tag{3-4}$$

The Frequency Response Function (FRF) or 'Magnification Factor' for a sinusoidal forcing function (Ω) applied to a system with natural frequency (ω_n) [2], frequency ratio (r) and damping ratio (ζ) is

$$\frac{U(r)}{U_o} = \frac{1}{\sqrt{(1 - r^2)^2 + (2\zeta r)^2}} \tag{3-5}$$

where:

$$\mathrm{U}_o = \mathit{static\ displacement\ of\ the\ mass}\ (m) \mathit{under\ static\ load}\ (p_o)$$

$$\mathrm{r} = \frac{\Omega}{\omega_\mathrm{n}} \tag{3-6}$$

$$\zeta = \frac{c}{c_{cr}} \tag{3-7}$$

As can be deduced from Equation (3-5), a forcing function (Ω) equal to a natural frequency of the system (ω_n) without the influence of damping (ζ=0) produces a singularity and a mathematically infinite value for the frequency response function. As defined here, damping represents the ability of a system to dissipate energy. In reality of course, a condition of zero damping does not practically exist since all mechanical systems within the scope of this work include at a minimum some amount of material damping. Therefore, the amplitude of vibrational response of a system (frequency response factor) is determined by the forcing function frequency in relation to the natural frequency(s) of the system as well as the degree of damping present.

Figure 3-2 illustrates the FRF as a function of frequency ratio (r) for multiple values of damping ratio (ζ).

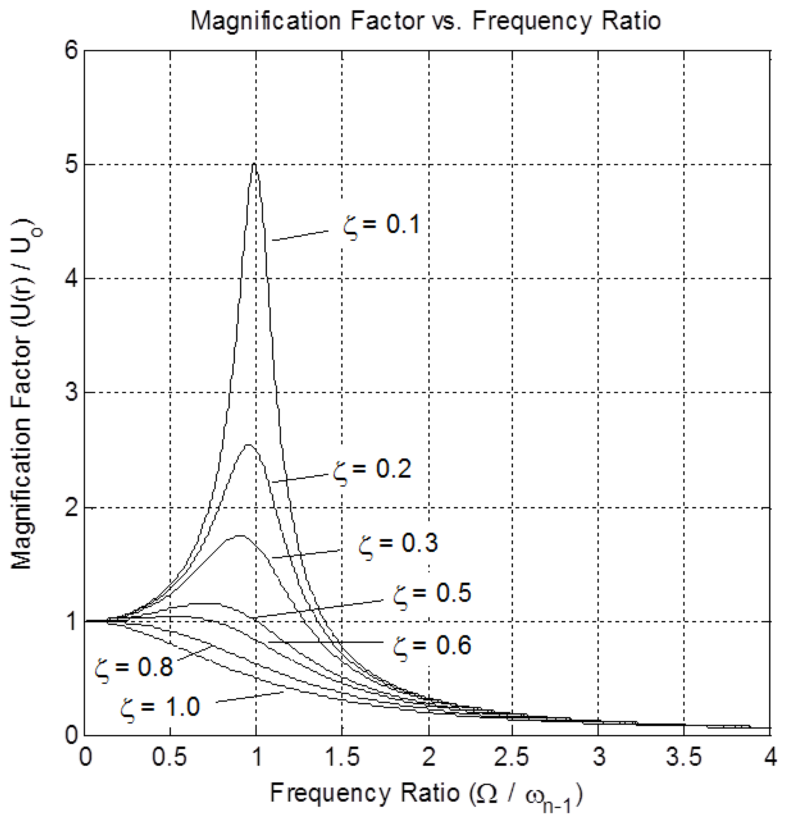


Figure 3-2 FRF as a function of damping - classical SDOF system

A preferred and common practice is to design a physical structure such that the forcing frequency is significantly below the fundamental (lowest) natural frequency of the system. Norton [1] recommends a factor of at least 3 or 4 for this safety factor whereas other texts recommend designing the structure such that the fundamental frequency is as much as 15 to 20 times the frequency of the forcing function [3]. Unfortunately, this is not always possible given constraints of the design.

A related form of dynamic response is defined as a 'self-excited vibration [4] [5] [1]. Self-excited vibrations (SEV) are those dynamic responses where systems begin to vibrate 'of their own accord' based on a motion such that when the motion stops, the response motion also stops.

Shaft whirl in an electric motor is a classic example of SEV and due in large part to a rotating shaft with eccentric mass which may cause the system to deform much like a jump-rope being swung by children [1] or other mode shapes. Another way of describing SEV is that of a free-vibration but with negative damping [4]. SEV however is not limited to rotating structures and may exist in lateral vibrations as well. Billah and Scanlan [6] propose that SEV or negative damping may be responsible for the Tacoma Narrows bridge failure of 1940.

The forced vibration response of a system (as is the subject of this paper) differs from SEV in that the response motion exists independently of the forcing motion. That is, when the forcing function discontinues the response motion continues, decaying to zero only if the system has positive damping characteristics. Both responses, however, are dependent upon (characterized by) the natural frequency(s) of the system [4] and as such should be considered in the overall design process as discussed previously. This paper directly considers the lateral vibrations of beam systems due to forced vibration and not SEV. Because of this, SEV will not be discussed in further detail other than to say that, in general, since both vibrational phenomena are rooted in the natural frequency characteristics of the structure; the techniques described in this paper to identify an optimum combination of stiffness, mass and damping for a system in order to minimize response to forced vibration may also have application to SEV as well.

With consideration for the lateral vibration of beams, much research has been conducted on means by which to raise the fundamental natural frequency to a maximal value by the optimal placement and sizing of intermediate, lateral elastic beams. Akesson and Olhoff [7] described that a minimum stiffness could be identified for optimally located intermediate elastic supports to maximize the fundamental frequency of a cantilever beam.

They initially describe, based on Courant's maximum-minimum principle [8], how rigid supports may be positioned at the vibrational zero of a cantilever beam to raise the fundamental frequency. By Courant, a structure with *n* additional supports will raise its *j*th natural frequency ω_j to between the *j*th and (*j*+*n*)th frequencies. That is, for a cantilever beam system a fundamental eigenfrequency (ω_o) exists where the mode shape is well-known as an oscillatory general bending of the beam about the neutral axis. No vibrational node or 'zero' other than the connection with the support wall exists at this fundamental frequency. The beam's second mode (ω_1) is a higher order flexure where at a specific axial location, the beam 'pivots' at the neutral axis at a 'zero' point or a vibrational node. At this axial location, the vibrating beam does not depart vertically from the static lateral position. If a single rigid support (n=1) is located at this axial position (vibrational node of the (j+n)th frequency), then the new fundamental frequency of the cantilever beam raises to the value of the original second mode. That is, $\omega_o^1 = \omega_1^0$. Similarly, rigid supports placed at the 2 nodes or 'zeroes' of the third natural frequency (ω_2) of the initial system will raise the fundamental frequency to that original third eigenfrequency.

Akesson and Olhoff further demonstrate that similar improvements can be made using elastic instead of rigid supports. A critical value of stiffness exists for this support(s) whereby if it is placed at a node of the next higher eigenvalue, the fundamental frequency is raised to that next eigenfrequency value as if it were a rigid support. If the stiffness value of this support is further raised to be above this critical value, no impact to further increase the natural frequency is obtained. Stiffness values below this critical value however do not achieve the full eigenfrequency increase as experienced with the critical value of stiffness.

The placement of the elastic support (of varying stiffness) can similarly be modified to raise the fundamental natural frequency to the value of the original second eigenvalue. That is, Akesson and Olhoff demonstrate that an optimum value (for proper placement of the support) exists for the increase in fundamental frequency with the critical value of stiffness. Deviations from this optimum location, whereby the support is positioned further axially 'down' the beam, result in mode switching where the two modes in question interchange their roles as 'first and second modes'.

Finally, Akesson and Olhoff describe that differing optimums exist in the placement of the support depending on whether the support's stiffness is greater than or equal to or less than the critical support stiffness. That is, if the support stiffness is greater than or equal to the critical stiffness, then the optimum location of the support is identified by the absence of support reaction force. As described above, a rigid support placed at a vibrational node is an optimum location. Since a vibrational node by definition has no lateral displacement in the cantilever beam problem, a support placed at this location would experience no reaction force. However, if the support stiffness is less than the critical stiffness value then a reaction force will be associated, regardless of the support location (vibrational node or not). However, for this sub-critical stiffness value, the optimal support position is described to be that placement that provides a zero slope to the mode shape at the support location. To summarize then, the optimal placement of a support with critical or super-critical stiffness is identified by a reaction force of ‘0’ and the optimal placement of a support with sub-critical stiffness is identified by a mode-shape slope of ‘0’ at the support location.

Wang, Jiang and Zhang [9] follow this work and attempt to devise an approach to programmatically optimize the location of support positions to maximize the fundamental frequency of structures regardless of their stiffness value. In their paper, authors Wang, et al. describe a method for using design sensitivity analysis to find the effects of design variables (support location and stiffness) upon the fundamental frequency of a structure, and then optimize those variables using a heuristic optimization procedure called evolutionary shift. An equation for design sensitivity is derived using the derivatives of a natural frequency with respect to the support location, including terms for both reaction force and mode slope. An initial support location (for a support of defined stiffness) is identified and then moved gradually to assess its impact on design sensitivity. An optimum solution is identified as that position where design sensitivity has a value of ‘0’; that is, either the reaction force goes to zero in the case that the support stiffness is critical or super-critical, or the mode-slope goes to zero in the case that the support stiffness is sub-critical. One reported challenge to the procedure is a ‘mode-shift behavior’ discussed by Akesson and Olhoff where certain support locations cause an ‘interchange’ of their roles as first and second modes. Another challenge reported by Wang, et al. is related to their implementation of FEA as part of the process. They discuss that in the practical application of their process, the optimal location of a support is often not at a defined FE node but rather between nodal locations and within the span of a defined element. Rather than creating an alternate FE mesh, the authors choose to interpolate the design sensitivity function between the established FE nodes in order to find the precise optimal location for the support.

Wang also uses this general methodology to optimize support locations for the static deflection of structures due to force [10] and moment [11] loading with similar success and challenges.

Recognizing that it is impossible within a practical world to implement a support with infinite stiffness, Wang, Friswell and Lei [12] use the closed form equation for design sensitivity proposed above [9] to investigate a minimum effective stiffness for intermediate supports considering various end conditions. Further, they illustrate that the support(s) need to be stiffer in order to be applicable for maximizing higher-order frequencies. Note that in this work, optimization algorithms are not used to identify the preferred solution(s) in preference for the application of closed form solutions. In so doing, the frustrations experienced in the previous works [9] [10] are avoided.

Similar results are demonstrated by Zhu and Zhang [13] by slightly different means. In their work, Zhu and Zhang implement a topology optimization method to maximize the fundamental natural frequencies of beams and plates. Using a Solid Isotropic Material with Penalty (SIMP) model, they demonstrate that intermediate beam supports of sub-critical spring stiffness result in maximal fundamental frequencies if placed more distally than if the spring stiffness is a critical or super-critical value.

Consideration for the works described in this section infers that the maximization of a fundamental natural frequency for a beam system is enhanced by the fact that the response function (frequency) can be treated as a 'smooth surface'. That is, the closed form solution developed by Wang et al. [9] as well as the graphical results depicted by Akesson and Olhoff [7] indicates that the resonant frequency(s) varies without significant mathematical discontinuity as a function of support location and stiffness. This 'simplifies' the optimization effort in that many techniques are then available to the designer. Further, optimizing the design of a system by 'relocating' the fundamental natural frequency to be greater than the anticipated operating frequency is a beneficial and desired practice. However, this is not always possible.

Dynamic Vibration Absorber Systems

Sometimes design constraints exist such that one or more natural frequencies cannot be avoided within the anticipated operating range. In the case that damping is not explicitly added, the resulting frequency response at the resonance point becomes a mathematical discontinuity to the response where theoretically infinite response is reached. Of course, no system in the real world ever has '0' damping, but in practice the damping properties of many metals and other materials are low enough that, without additional explicit use of damping elements or other techniques, the overall 'limitation' effect is negligible. For these conditions, the response at conditions of resonance can become extremely large and a source for product failure. The existence of this 'mathematical discontinuity' within the response also frustrates the optimization effort. One approach to limit the maximum frequency response at and near a resonant condition is the use of damped or undamped dynamic vibration absorbers (DVA).

Undamped DVA

Vibration texts including Den Hartog [4], Craig [2] and Piersol [5] commonly describe the use of a Dynamic Vibration Absorber (DVA) as a device which can be used to mitigate the vibrational response of a system in resonance. Sometimes also referred to as a Tuned Vibration Absorber (TVA), the device was first invented by Frahm [14] in 1911 and makes use of a secondary spring-mass system which is sized appropriately to 'absorb' the energy of the primary system during resonant conditions.

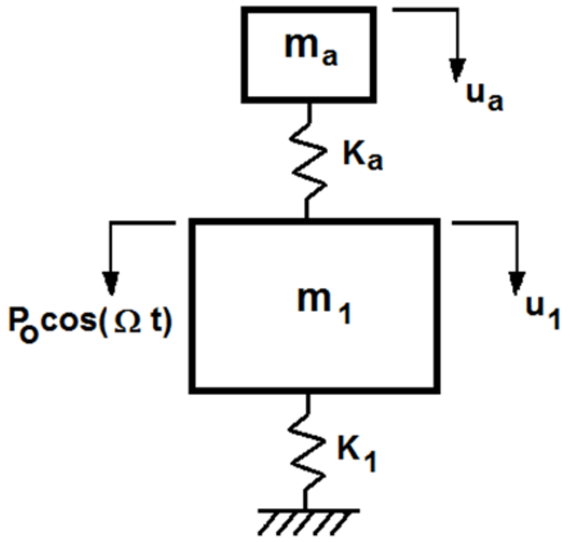


Figure 3-3 Undamped dynamic vibration absorber

Recalling the simplified 1-DOF spring-mass system of Figure 3-1 and Eq. (3-4), a resonant condition is created as a function of the mass and spring stiffness. In the application of a DVA, a second mass of significantly smaller size (m_a) is attached to the free side of the main mass via a secondary spring (k_a). The secondary (DVA) spring stiffness is sized together with the secondary (DVA) mass in order to match the resonant frequency of the primary spring-mass. In so doing, the absorber spring-mass is excited at the resonant frequency with a defined, finite displacement while the displacement of the main mass is limited by the system to '0'. The Equation of Motion of the resulting 2 Degree of Freedom system in matrix format is [2]

$$\begin{bmatrix} m_1 & 0 \\ 0 & m_a \end{bmatrix}\begin{Bmatrix} \ddot{u}_1 \\ \ddot{u}_a \end{Bmatrix} + \begin{bmatrix} k_1 + k_a & -k_a \\ -k_a & k_a \end{bmatrix}\begin{Bmatrix} u_1 \\ u_a \end{Bmatrix} = \begin{Bmatrix} p_o \\ 0 \end{Bmatrix}\cos(\Omega t) \tag{3-8}$$

Which results in the following FRF for the main and absorber masses respectively. [2]

$$\frac{U_1}{U_o} = \frac{1 - r_a^2}{\left[1 + \mu\left(\frac{\omega_a}{\omega_p}\right)^2 - r_1^2\right][1 - r_a^2] - \mu\left(\frac{\omega_a}{\omega_p}\right)^2} \tag{3-9}$$

$$\frac{U_a}{U_o} = \frac{1}{\left[1 + \mu\left(\frac{\omega_a}{\omega_p}\right)^2 - r_1^2\right][1 - r_a^2] - \mu\left(\frac{\omega_a}{\omega_p}\right)^2} \tag{3-10}$$

where:

$\mathrm{U}_o = static\ displacement\ of\ the\ primary\ mass\ (m_1)\ under\ static\ load\ (p_o)$

$$\mathrm{r}_1 = \frac{\Omega}{\omega_{\mathrm{n\_1}}} \quad , \quad \mathrm{r}_a = \frac{\Omega}{\omega_{\mathrm{n\_a}}} \tag{3-11}$$

$$\omega_{n\_1} = \sqrt{\frac{\mathrm{k}_1}{\mathrm{m}_1}} \quad , \quad \omega_{n\_a} = \sqrt{\frac{\mathrm{k}_a}{\mathrm{m}_\mathrm{a}}} \tag{3-12}$$

$$\mu = \frac{m_a}{m_1} \tag{3-13}$$

Although a DVA works well to limit the deformation of the main mass exactly at the resonant frequency, a strong detriment of the system is that two (2) new system natural frequencies are created on either side of the original resonance as shown in Figure 3-4.

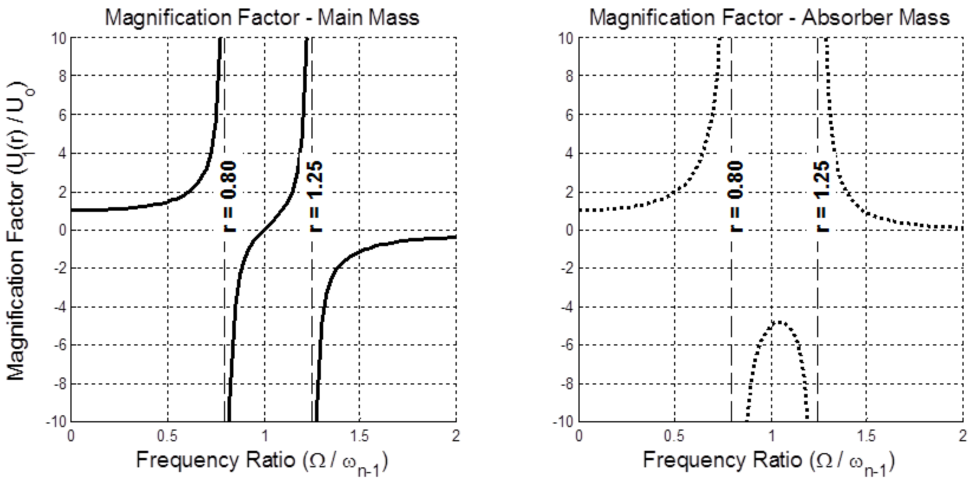


Figure 3-4 FRF of main and absorber masses - undamped DVA, μ=0.2

As discussed by Den Hartog [4] and illustrated in Figure 3-4, a simplified 2-DOF DVA system can expect the new natural frequencies to be at approximately 0.8*ω_n and 1.25*ω_n for an absorber mass sized to be 20% of the primary mass. If the absorber mass were to grow to as much as 50% of the primary mass, the new resonant frequencies expand only a few percent further from the original resonance condition; but as the absorber mass shrinks below 20% the two (2) natural frequencies converge on a single natural frequency (of the primary mass). This limits the effectiveness of such a device to those applications where the expected operating range is relatively constant and centered on the the resonant frequency of the primary mass.

Damped DVA - Theory

Den Hartog describes a useful addition to the undamped DVA through the addition of an explicit damping element. In his text [4], Den Hartog describes the addition of a dashpot element between the primary and secondary masses of the example 2-DOF system to not only reduce the maximum amplitude of the response at and near resonance, but to significantly expand its effectiveness with respect to frequency. This is illustrated in Figure 3-5.

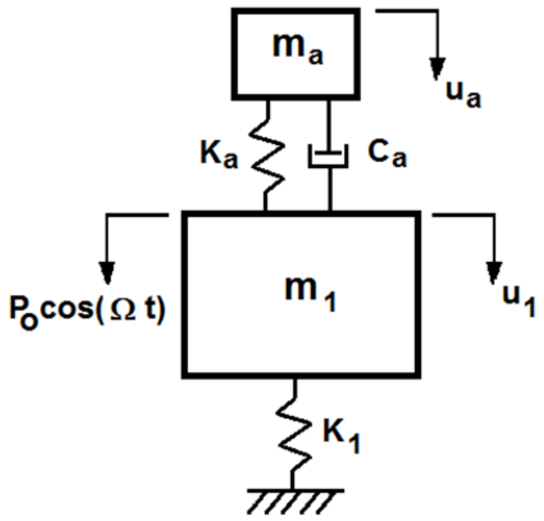


Figure 3-5 Damped dynamic vibration absorber

The Equation of Motion for this system becomes [4]

$$\begin{bmatrix} m_1 & 0 \\ 0 & m_a \end{bmatrix} \begin{Bmatrix} \ddot{u}_1 \\ \ddot{u}_a \end{Bmatrix} + \begin{bmatrix} c & -c \\ -c & c \end{bmatrix} \begin{Bmatrix} \dot{u}_1 \\ \dot{u}_a \end{Bmatrix} + \begin{bmatrix} k_1 + k_a & -k_a \\ -k_a & k_a \end{bmatrix} \begin{Bmatrix} u_1 \\ u_a \end{Bmatrix} = \begin{Bmatrix} p_o \\ 0 \end{Bmatrix} \cos(\Omega t) \tag{3-14}$$

which results in the following FRF for the main mass.

$$\frac{U_1}{U_o} = \sqrt{\frac{(2\zeta g f)^2 + (g^2 - f^2)^2}{(2\zeta g f)^2 (g^2 - 1 + \mu g^2)^2 + [\mu f^2 g^2 - (g^2 - 1)(g^2 - f^2)]^2}} \tag{3-15}$$

where:

$$\zeta = \frac{c_a}{2m\omega_{n\_1}} \tag{3-16}$$

$$g = \frac{\Omega}{\omega_{n\_1}} \tag{3-17}$$

$$f = \frac{\omega_{n\_a}}{\omega_{n\_1}} \tag{3-18}$$

The influence of the damper can be understood physically as follows. When the damper value is negligible (c=0), the 2-DOF system is the same as the undamped system described previously and with two (2) resonant peaks exist; one on either side of the resonant frequency of the main mass system (depending upon the value of the absorber mass). When the damper value is at the other theoretical extreme however (c=∞), the main and absorber masses are effectively 'locked' together and behave as one with a new, single resonant condition near but slightly lower than the original 'main-system' resonance due to the increase in mass. This new single resonant peak however is still located between the two (2) resonant frequencies of the system where c=0. As the damper value varies between c=0 and c=∞, the response amplitude decreases from ∞ to a finite value. Further, as the value moves from c=∞ toward c=0, the response curve transitions from a single peak toward a 2-peak response; trending toward the 2 infinitely-large responses when c=0. This is illustrated in Figure 3-6.

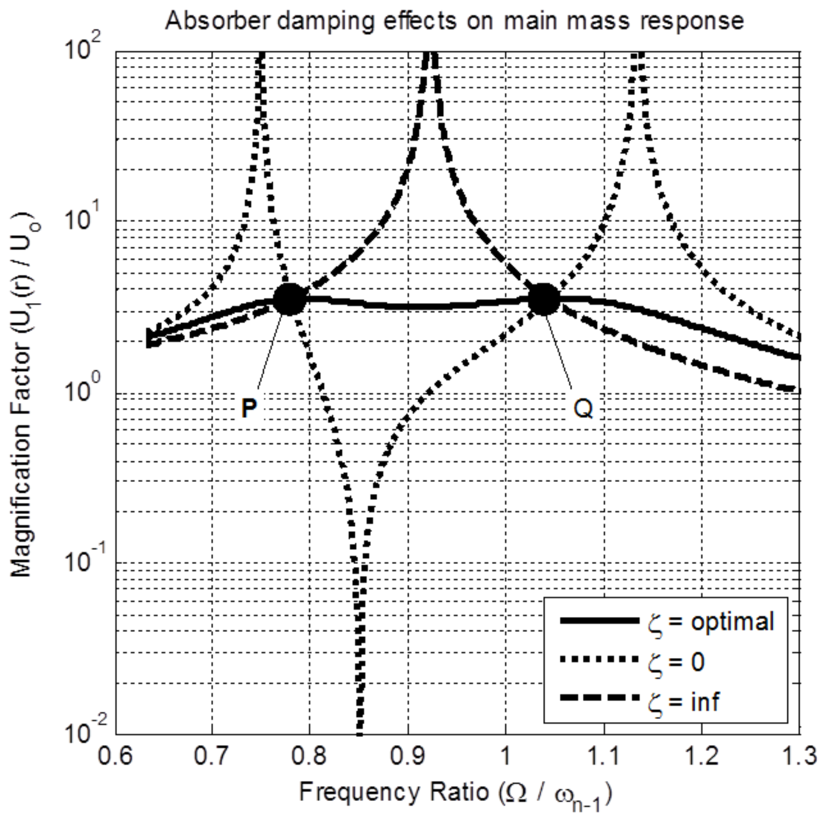


Figure 3-6 FRF of damped DVA with multiple damping magnitudes, μ=0.2

Interestingly, all response curves for $0 < c < \infty$ intersect at two (2) frequency points that Den Hartog terms 'P' and 'Q'. They lie on either side of the resonant peak for $c=\infty$, but within the 'outer' resonant peaks corresponding to c=0. Den Hartog explains that at some (finite) value for c within this range, an optimum exists whereby the maximum response amplitude is minimized and the slope of the amplitude curve is '0' at both points P and Q. Further, he showed that this optimal value for damping can be identified by differentiating the steady-state response equation with respect to r where r is defined as the ratio of the operating frequency to the natural frequency of the main mass and spring.

Ozguven and Candir [15] apply the use of damped DVA's to a cantilever beam in order to suppress the first and second resonant responses. Specifically, they pursue two (2) damped DVA's attached to the free-end of a cantilever beam to which a harmonic load is applied, also at the free end. The free-end is chosen for the location of both dampers because, as noted by Ozguven and Candir, a DVA is most effective if attached to the point where maximum response occurs at the resonance of interest. Procedurally, Ozguven and Candir first derive closed-form solutions for the steady state amplitude of the beam displacement and then use these equations to solve the min-max problem using an iterative numerical technique. Initially, the properties of the first absorber are then determined in order to tune it to the first resonant condition without consideration for the second absorber, and then the second absorber's parameters to address the second resonance independent of the first absorber. A second step is to then optimize the parameters of the first absorber over the frequency range of interest while keeping the properties of the second absorber fixed. Once that optimum is found then those properties for the first absorber are 'fixed' and the optimization repeated for the second absorber over the pre-defined frequency range. The process is repeated until an acceptable convergence threshold is reached, indicating that optimum values for both DVA's are attained.

Interestingly, Ozguven and Candir found that the optimum parameters of the first absorber are appreciably affected by the existence of the second absorber, while the converse was not true. Further, they found that the existence of structural damping has little effect upon the overall system so long as that value is 'light'. In their work, Ozguven and Candir utilized a structural damping ratio of 0.01, which they considered to be 'light'.

Yang et al. [16] extend the application of a damped DVA to application on a vibrating plate and through a frequency band. Of particular interest is their observation that coupling among structural modes affects the optimal location of the DVA's. More specifically, Yang et al. point out that for control of modes that are well-separated from neighboring resonant frequencies, the optimal location of the DVA is found to be the maximum vibration locations for both narrow band and broadband vibration reduction. However, modes where the resonant frequencies are relatively close together provide interactions that strongly influence the optimal placement of the DVA, including negative effects on off-target modes. A genetic algorithm (GA) optimization routine is used in their study to facilitate these optimal placements.

Damped DVA – Practical Application

Multiple 'practical' applications of damped DVA's were found during the literature study. Three (3) are mentioned here with particular relevance to the current study.

Russell [17] investigated the effect of structural damping and a novel use of a damped DVA to reduce the sensation of sting experienced in the top hand of a player holding a baseball or softball bat. In this work, Russell compares the analysis of a baseball bat to a free-free beam. Given the 'node' and 'anti-node' locations from a free-free beam analysis, Russell notes that when the bat is conventionally gripped then the bottom hand is located near an anti-node of the first bending mode and at a node of the second bending mode. The reverse is true however for the top hand. That is, the top hand is located at the node of the first mode and the anti-node of the second mode. Simply stated, Russell notes that the left and right hands are both placed near anti-nodes (maximal deflection of the bat); for each of the first two (2) modes respectively. It is this location of the hands (near the anti-nodes) that is a significant contributor to the sensation of 'sting' for an 'incorrectly-hit' ball. To this end, he notes that anti-nodes exist at the extreme distal (barrel) end of the bat for both first and second modes. Russell's paper is primarily a report on empirical testing of multiple bat construction methods for an undisclosed manufacturer, but does conclude that a relationship exists between the effects of damping, the damped DVA and the sting experienced by players. Specifically, he cites a correlation between physical testing to determine the modal damping rate of each bat type and a subjective evaluation of sting experienced by players. Further, he concludes that the bat incorporating the damped DVA, when tuned to the 2nd mode, was most successful in reducing sting for the player.

Hao et al. [18] report the application of a damped DVA to an electric grass trimmer in order to suppress detrimental hand-arm vibration. Similar in principle to the goals of this investigation, they sought to reduce the vibration experienced at the handle of the device subject to harmonic loading applied at the free end by optimally locating and tuning a damped DVA. Optimal location and tuning of the DVA was found through experimentation, resulting in a decrease of measured acceleration at the handle in excess of 67%.

Finally, Pierson [19] addresses a tangential topic with the incorporation of magnetorheological fluids to serve as a ‘controllable support mechanism’ for support of feed stock in high-volume computer-numerically-controlled (CNC) turning machines. The problem that he attempts to address is the efficiency of the change-over process when the size of bar stock must be changed from job-to-job. In the current technology, such a changeover requires that the closely toleranced guide channel for the feed stock also be changed to an appropriate size for the new bar diameter. His attempt is to utilize a support system that is more generally sized such that it can be used to support a wider variety of bar stock sizes while still effectively mitigating vibrational resonances in the feed stock at and near machine frequencies. His proposal is to use magnetorheological fluids together with appropriately placed electromagnets to alter the effective stiffness of the fluid and thereby overall support system as a means of manipulating the resulting natural frequency. The scope of his work does not present a final solution, but does suggest promise in the idea as well as some practical considerations for further work. For example, he observes that a rigid body moving through a volume of magnetorheological fluid does not fit into the well-studied categories of flow, shear and squeeze-flow mode. Further, the difficulty in predicting natural frequencies of a bar within a field where viscous forces are present is not well-studied either; the latter being a significant contribution of his work. Although Pierson’s research is at best a tangential topic to that presented in this work, it is an interesting idea and worthy of note here.

Optimization of Smooth, Unimodal Functions

As inferred by Equation (3-4) for physical systems and discussed by Wang et al. [9] and Akesson and Olhoff [7], an objective function of fundamental natural frequency can be considered a smooth function, and either convex or concave in nature. That is, significant discontinuities to the response do not exist as they do with harmonic response near a resonant frequency because mass cannot physically diminish to zero (0) so as to create a mathematical singularity. This simplifies the task of optimization for the natural frequency value because a wider array of tools is appropriate. Further, the constraint functions are also free of local discontinuities so long as they exist as side constraints (due to geometric bounds, material property bounds, etc.). Vanderplaats [20] states that if the objective and constraint functions can both be shown to be convex, then a single optimum exists for the objective function and that optimum is the global optimum. Multiple methods are available to the designer to identify the global optimum of a smooth, convex (unimodal) function, including zero-order, first-order and second-order methods.

First-Order Methods

First-order methods are usually more efficient than zero-order methods because they utilize gradient information as opposed to solely the function values [20]. Among the simplest and most well-known of first-order methods is the method of Steep Descent. In this technique for unconstrained problems, the response is evaluated at an assumed starting location and a search direction is established that is consistent with the largest gradient, or the direction of steepest descent. Given this search direction and the starting point, a 1-D search routine is then performed in the direction of steepest descent until a minimum value is found. [20] (Note that in the case where a maximum value is sought, the 'negative' of the minimum is conventionally sought.) The method of Steep Descent is not a particularly efficient method in comparison to other first-order methods [20] and, because of this, is not commonly used for a full optimization method. Nevertheless, it has been investigated and used successfully.

Fliege and Svaiter [21] for example proposed using the method of steep descent for unconstrained multicriteria optimization as well as adaptation of the method of feasible directions for use in constrained cases. In their work, they show that these methods converge to a point satisfying certain first-order necessary conditions for Pareto optimality [21] that are parameter free. In so doing, Fliege and Svaiter demonstrate first-order methods that are free from ordering information or weighting factors, simplifying an approach to multicriteria optimization. Although successful in their demonstration, the authors conclude that since the method is a first-order method, it should be considered only as 'first steps' toward an overall efficient method rather than as efficient methods unto themselves.

Several one-dimensional (1-D) or ‘line’ search methods are also available to be used in conjunction with a 1st order Optimization method. One effective method [20] is that of polynomial approximation. By this method, a one-dimensional polynomial of the response is created in the direction of the search vector and the location of the minima thus identified. An advantage of this method is that, given a ‘good’ curve fit to the data, a global minimum along the search vector (for a multimodal curve) can be identified with some certainty as compared with a local minimum that might be otherwise misunderstood to be the global minimum. A disadvantage of this method may be the amount of computing resource needed to evaluate the 1-D curve as a function of length depending upon search strategy used.

An alternative to the polynomial approximation is an approximation method, or ‘inexact line search condition’ sometimes referred to as the Armijo method. Nocedal [22] describes that a sufficient decrease in the objective function (along the descending search direction) can be described by the inequality

$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k \nabla f_k^T p_k \tag{3-19}$$

where x_k is the starting point for the search, α is the step length, p_k is the (descending) search direction, $\nabla f_k^T p_k$ is the directional derivative and with the constant $c_1 \in (0,1)$. This inequality, also known as the Armijo condition, states that a ‘sufficient decrease’ in the objective function ‘f’ is obtained for a given step such that the result lies below the curve indicated by the RHS of (3-19) for an assumed value of c_1. In practice, the value of this constant is generally very small and may be on the order of 10^{-4} or so.

Although the Armijo inequality does constitute a ‘sufficient decrease’ along the search direction to establish a potential minima (in that it ‘guarantees’ a reduction of the function value), it does not ensure that the result is a global or even local minimum along the search vector and risks that an unnecessarily short step length is identified. A result of this could be a large number of search steps. Nocedal [22] suggests that an additional condition may be employed to evaluate whether a minima has been identified in the search. A curvature condition is proposed as

$$\nabla f(x_k + \alpha_k p_k)^T p_k \geq c_2 \nabla f_k^T p_k \tag{3-20}$$

where the LHS of Equation (3-20) is the derivative of the LHS of Equation (3-19) and the constant $c_2 \in (c_1, 1)$ is an assumed value (usually closer to 1 than c_1) used to help identify a 'sufficient' difference between the slope (LHS of Equation (3-20)) and the initial slope at the point x_k. Equations (3-19) and a modified form of Equation (3-20) whereby both sides are taken as the absolute values are commonly known as the Wolfe conditions [22] and are useful in identifying both a sufficient decrease (per Armijo) and a minimally acceptable slope (positive or negative as noted by the inclusion of absolute value). This is an improvement over the simple implementation of the Armijo condition (3-19) but still fails to differentiate a minimum as a local or global feature.

Although both of these methods (Armijo and Wolfe conditions) hold the potential to speed the evaluation process along the search vector in comparison with the polynomial approximation, the polynomial approximation holds the potential to be a more accurate evaluation of the search vector with improved discrimination between local and global minimums.

A draw-back to the Steepest Descent method is that no information is carried forward from one iteration to another [20]. That is, as a minimum is identified along the 1-D search vector for each iteration, the process starts 'afresh' in that a new search direction is identified based on the surface gradient (only) at the new start point. Vanderplaats [20] recommends that two (2) alternative approaches exist as modifications of the Steepest Descent method, but which yield improved convergence efficiencies. The Conjugate Direction method [23] identifies an initial search direction and 1-D search in the same method as described above for Steepest Descent, but modifies subsequent search directions by adding a 'factor' of the previous iteration's search vector to the newly calculated vector of steepest descent at the current iteration's start point. Alternatively, the Variable Metric Method [24] modifies the search vector based on the inverse of the Hessian matrix as the optimization progresses; the Hessian matrix being a matrix of second-partial derivatives of the function with respect to the design variables. Since Variable Metric Methods have convergence criteria similar to second-order methods, they are often called quasi-

Newton methods and represent a potentially significant improvement in convergence rate as compared with basic Steepest Descent methods.

Second-Order Methods

Second order methods improve upon first order methods by adding Hessian matrix information as part of the optimization routine. As such, these methods are usually more efficient than the lower-order methods.[20]

Newton's method is considered to be the classical second-order method. [20] It derives the search vector based upon the Taylor series expansion of the response and incorporates both gradient and Hessian matrices directly. As described by Vanderplaats [20], this is both a blessing and a limitation of the method. Where the Hessian matrix can be practically calculated, the Newton method is 'almost always the preferred approach' due to the improved efficiencies. However, multiple factors may prohibit the calculation of the Hessian matrix in real-world problems. For example, closed-form equations for many problems are not readily available and calculation of the Hessian matrix using finite-difference methods can be too costly to be practical in many cases. Further, the use of the Newton method relies on a Hessian matrix that is non-singular and positive definite. If the Hessian is not positive-definite, then a non-convex problem exists and multiple local-minimums exist; a condition that can result in oscillation in the solution . If the Hessian is singular, or nearly so, then that indicates that the objective is linear in one or more of the design variables and the solution for the search direction vector may become so ill-conditioned that the results are not valid. For all of these reasons, Vanderplaats advises that second-order methods are less frequently used as a direct solution of complex real-world problems and more commonly as a component in the development of first-order methods which may be more easily employed.

Optimization of Non-Smooth, Multimodal Systems

As illustrated previously for the harmonic problem of this work, many real-world problems involve optimization of a response which is non-smooth, multimodal or both. As a result, 'higher order' information such as gradient and hessian functions is not available within a practical context. Further, the application of ‘purely’ first-order approaches such as the method of Steep Descent, while applicable, are not particularly efficient with respect to the speed of their convergence. Therefore, alternate methods are needed.

Direct Methods

One class of techniques is commonly described as Direct Methods. Kolda et al. [25] describe that Direct Search methods are traditionally best-known as unconstrained optimization techniques that do not explicitly use derivatives. Because of this, they recommend that Direct Search methods may be particularly appropriate for Simulation Based Optimization where the process of obtaining derivatives for gradient-based methods may, at the very least be difficult, even when the underlying objective and constraint functions are smooth (continuously differentiable). Kolda et al. review that Direct Methods enjoyed popularity as early as the 1960’s where they found favor with application to digital computer use. For example, an early direct search method is described as a Compass Search where ordinal steps (North, South, East and West) are taken from an assumed starting point to identify a reduction in the function. Should one be found, then that point becomes the new iterate and a subsequent step is made in kind. If a reduction is not found, then the ordinal steps are taken at ½ step increments until the step size meets a predetermined minimal value indicating convergence.

Vanderplaats [20] discusses three (3) more modern Direct Search techniques applicable to Constrained Functions; Genetic Search, Particle Swarm and Sequential Quadratic Programming.

Genetic Algorithm

Genetic Algorithms (GA) [26] attempt to mimic natural evolutionary processes in that candidate designs are represented by binary sequences that are altered with respect to their ‘fitness’ to the objective function. Key features of a GA algorithm [20] include the use of function values only, the ability to handle discrete variables and a requirement that it include evaluation of a very large number of function evaluations. If the objective function can be described by a response equation (surrogate function of a more complex simulation result for example), then solution of each function evaluation can be very fast making the GA approach an attractive alternative for multimodal problems.

Wang and Chen [27] applied a GA method to the optimization of support location of beams. In their work, Wang and Chen effectively used GA to identify optimal placement of both elastic and rigid supports to maximize the fundamental natural frequency of beams with a variety of boundary conditions without the use of gradient information.

Particle Swarm Optimization

Particle Swarm Optimization (PSO) methods [28] are another technique which attempts to find the solution to optimization problems by mimicking the effects of nature. In fact, Vanderplaats compares PSO to modelling '… a swarm of bees or flock of birds seeking food' [20]. Although PSO is designed to solve unconstrained problems, it can be applied to minimization of constrained problems by modifying the objective function into a 'pseudo-objective function' which includes a penalty-value, based upon violation of the defined constraints. In general, an initial population of starting points (or 'particles') is established, each with a randomly assigned velocity vector. The pseudo-objective function is evaluated at its new position (original position plus velocity * time step) and the particle with minimum pseudo-objective function is identified. A new velocity vector is computed for the population which utilizes an 'inertia' parameter applied to the new velocity vector as well as 'trust' parameters applied to terms which involve the identified particle with minimum pseudo-objective function. A subsequent time step is then performed, all particles are re-evaluated and the process repeated until convergence is obtained through minimized movement of the particles.

Banks et al. [29] state that PSO has been in existence since approximately 1997 and, as a 'natural computing paradigm' has enjoyed formative research particularly in the fields of social modelling, computer graphics and simulation and animation of natural swarms or flocks of animals. They cite that an attribute shared by many of the 'natural computing' paradigms is the ability to deal with noisy or incomplete data. Further, research is cited by Banks et al. that PSO finds solutions near optima faster than GA but is more prone to premature convergence than

Evolutionary Algorithms. This is termed by Banks et al. as ‘Swarm Stagnation’ which, although desired as a rapid convergence outcome, is a potential drawback of PSO in that the swarm can easily stagnate around a solution without pressure to continue exploration.

Vesterstrøm and Thomsen [30] compared the effectiveness of Differential Evolution (DE), Particle Swarm Optimization and Evolutionary Algorithms (EA) on a series of 34 standardized benchmark programs. They conclude that in general, the DE algorithm outperforms both PSO and EA in terms of consistent speed of convergence and ability to find the optimum. The DE algorithm is another in the group of methods that attempt to utilize natural processes as a ‘search method’. [30]   An array of initial start points (termed ‘individuals’) is assumed and evaluated and then offspring are created using the weighted difference of parent solutions. Importantly, Vesterstrøm and Thomsen conclude that the PSO routine was particularly problematic with test cases involving noisy functions and this aspect is recommended for further investigation.

Laskari et al. [31] investigate PSO as a means of optimizing minimax problems in comparison to Sequential Quadratic Programming (SQP) methods (discussed later in this chapter). Key conclusions are that PSO is shown to be an effective method for solving minimax problems, but may be outperformed by SQP methods which were shown to be much faster for less complex problems. The authors [31] note however that PSO is very easily implemented and has a key advantage that gradient information is not required as is required by the SQP method. As a result, it is not affected by discontinuities in the objective function as discussed previously. Further, they recommend that if the problem to be solved is a ‘black box function’ (where ‘only function values are provided’ [31] as opposed to gradient, hessian or other information), as may be the case with optimization of computer simulation results, then PSO may be a good alternative as an initial search tool with continued optimization performed by the more efficient gradient-based methods such as SQP.

As with GA above, Vanderplaats [20] advises that PSO methods require a very large number of function evaluations. In the event that a simplified surrogate model of the response is available for evaluation, the optimization event may be able to fit within an acceptable bound of computational effort. If however a surrogate model is not practical (as might be the case for finite element based computer simulations) the PSO method may not be an efficient alternative with respect to other methods.

Sequential Quadratic Programming

Sequential Quadratic Programming (SQP) [32] is described by Vanderplaats [20] as another in the category of Direct Methods suitable for optimization of constrained functions. In many methods, including the method of Steep Descent discussed earlier, an initial search direction is chosen based upon gradient information obtained in the vicinity of the iteration's start point and a 1-D search then performed to find a potential optimum along that vector. In the SQP method [20], the selection of the search vector is treated as a sub-problem with a quadratic objective function and linear constraint functions shown as:

$$\text{Minimize: } \quad \mathrm{Q(S)} = \mathrm{F(X)} + \nabla F(\mathrm{X})^T S + \frac{1}{2} S^T B S \tag{3-21}$$

$$\text{Subject to: } \quad \nabla g_j(\mathrm{X})^T S + \delta_j g_j(\mathrm{X}) \leq 0 \quad j = 1, m \tag{3-22}$$

$$\nabla h_k(\mathrm{X})^T S + \bar{\delta} h_k(\mathrm{X}) = 0 \quad k = 1, l \tag{3-23}$$

Vanderplaats describes that here, the design variables are the components of S and the matrix B is initially the identity matrix, but is updated at each subsequent iteration to approach the Hessian matrix of the Lagrangian.

The commercial software program MATLAB ® (The MathWorks, Inc.) includes SQP as an algorithm option within its constrained nonlinear optimization command 'fmincon' [33]. MATLAB ® help documentation regarding the use of fmincon [33] states that gradient information may be defined by the user as an input to the command. If gradient information is not provided by the user however, it is estimated 'automatically' within the fmincon function by the finite difference method.

Optimization of Expensive Black Box Functions

Much work is conducted and published in the area of optimization of 'Expensive Black Box Functions'. As previously defined [31], a Black Box Function is one where only function values of the response are available. This may be in contrast to knowledge of the response that might lead to gradient or hessian information for example. This is commonly the case with computer simulations such as Finite Element Analysis (FEA) and Computational Fluid Dynamics (CFD) techniques or experimental evaluations of physical prototypes. Knowles [34] describes a set of characteristics that are typically associated with an Expensive Black Box Function (EBBF). They include (but are not limited to):

- The time taken to perform one evaluation is of the order of minutes or hours
- Only one evaluation can be performed at one time (no parallelism is possible)
- The total number of evaluations to be performed is limited by financial, time or resource constraints.
- No realistic simulator or other method of approximating the full evaluation is readily available.

Due to the nature of the EBBF problem, solution techniques are often focused on stochastic techniques. The use of surrogate models is one common approach which aids in the optimization of this class of problems. [35] [36] (By definition, use of a surrogate model transforms the problem from an EBBF to a simpler problem where additional optimization tools are appropriate.) Jones et al. [35], for example, suggest a surrogate response surface (RS) approach based in 'kriging' models and 'Bayesian global optimization' techniques which has three (3) major advantages in comparison with other EBBF techniques.

- The technique often requires the fewest function evaluations of competing methods because effective interpolation and extrapolation is made possible over 'large distances'.
- The approach provides a credible stopping rule based on confidence intervals provided by a statistical model of the RS.
- The approach provides a fast approximation to the computer model as compared to the original evaluation itself.

An obvious limitation of the RS method is one of accuracy of the surrogate model with respect to the primary model. In the event that the computer simulation results in a relatively smooth function, an effective RS can be developed and validated in a relatively few number of function evaluations; making such a technique highly desirable for EBBF's. However, if (as is the case with many harmonic analyses of mechanical systems) the RS is more complex and includes multimodality and / or discontinuities a great many more function evaluations are required in order to produce a surrogate model of acceptable quality.

Another technique for optimizing on EBBF's is the use of mode-pursuing sampling methods. In their paper, Wang et al. [37] describe the mode-pursuing sampling method as a technique that 'systematically generates more sample points in the neighborhood of the function mode while statistically covering the entire search space'.

Although the motivation for this work includes consideration for EBBFs, a full inclusion of methods specific to their solution is beyond the selected scope. Rather, a 'simplified alternative' is sought which is in keeping with and provides some amount of 'balance' between more traditional optimization techniques and consideration for problems involving expensive function solves.   A description of the selected technique(s) is included in a later chapter.

Chapter 4

Model Development and Validation

Modelling and Validation Plan

Chapter 2 describes a sample problem for purposes of this study which is comprised of a 'family' of mechanisms involving two parallel beams; one (bottom beam) that is common to all family variants and one (top beam) which is unique to each family variant. In addition to cantilever support, the bottom (common) beam is supported by two intermediate elastic supports to ground, each of which includes both spring and damper components. The top (variable) beam is a 'free-free' beam with respect to end conditions, but is supported by two intermediate elastic supports (spring and damper components) to the base beam. A lumped mass and applied force (static and harmonic analyses) are applied to the tip end of the top beam. The system is illustrated in Figure 2-1.

A primary focus of this study is to compare and contrast the effectiveness of alternative strategies for optimization of the parameters related to the elastic supports using Finite Element Methods (FEM) as the means for solving the structural analysis problem. Therefore, it is important to validate that the FEM models are correct and appropriate prior to their use in the optimization effort. This model validation effort is the subject of this chapter and focuses on two (2) primary means of comparisons which are listed here and explained in greater detail below.

1) comparison of 1D (beam) & 3D (solid) FEM results for a simplified single-beam model against theoretical solutions for both static deflection and modal analysis
2) comparison of 1D & 3D FEM results for harmonic loading of the parallel-beam problem as a means of identifying the most efficient modeling method for later use in the optimization studies

Mesh density studies are included as part of each of these processes in order to support the relative conclusions.

The first part of the validation effort focuses on evaluation of a single beam model. This simplification from the parallel-beam is considered in order to simplify the development of a theoretical solution; against which the validation comparison can be made. That is, a theoretical solution to the parallel-beam problem (with intermediate elastic supports) is a complex effort and outside the selected scope of this project. Rather, the focus of this project is a comparison of alternative optimization techniques in combination with FEM structural analysis. The simpler two-step validation approach described above is sufficient to support correctness of the FEM solution and is therefore used here for comparison of the FEM modal and static results to theoretical solutions.

Theoretical solutions for both natural frequency and static load analyses for the first part of this validation exercise are based on classical Euler-Bernoulli beam theory. It is recognized that a limitation exists in this approach in that Euler-Bernoulli beam theory does not account for shear deformation or rotary inertia as may become significant for short, stubby beams.[2] Methods such as Timoshenko beam theory may be employed to incorporate both of these elements in the theoretical analysis and would make the solution more robust for beams of low aspect ratio, but given that the Problem under Study has a relatively large aspect ratio, this more complex analytical method will not be employed. Rather, the first part of the validation exercise focuses on comparison of the FEM results for mode shape, mode frequency and static deflection to theoretical results using classical Euler-Bernoulli beam theory.

The second part of the validation effort evaluates the level of detail required in the FEM model. Specifically, can an appropriate (analytically correct) result be reached using the more simple method of 1D meshing as compared with a more burdensome 3D technique. This comparison is made specifically using the parallel beam system under harmonic loading. The theoretical calculation of harmonic response for continuous systems is complex and beyond the selected scope of this work. Harmonic model validation is, in general, a combination of modal and structural elements and therefore is validated to a minimal degree by validation of the two component analyses (above). However, a significant difference exists between the performance of a 1D and 3D modeling technique with respect to mesh convergence and model efficiency. Therefore, the second validation effort compares the response of the FEM 1D and 3D models to each other in a harmonically loaded, parallel beam system. This validation effort replicates the model under study in this larger work and serves to substantiate (or refute) that a 1D modelling technique can be utilized rather than a 3D technique in order to improve the speed of the computations. This determination will be made based on a review of the results and the convergence of their responses.

Theoretical Solutions - Dynamic

The theoretical solution to transverse vibration of Euler-Bernoulli beams is described in detail in the text by Craig and Kurdila [2] and summarized below. The method is derived using Newton's Second Law and makes the following assumptions:

- Beam deformation is planar and in the principle plane of definition for the beam.
- The beam is oriented within this plane such that its neutral axis is the x-axis within the plane of definition.
- Cross sections of the beam which are originally perpendicular to the neutral axis (undeformed state) remain perpendicular to the neutral axis in the deformed state; i.e. transverse shear deformation is neglected.
- The material is both linearly elastic and homogeneous at and throughout any cross section.
- Transverse stresses (σ_y and σ_z) are negligible in comparison to the axial stress σ_x.
- Rotary inertia of the beam may be neglected in the moment equation.
- Mass density is constant at each cross section such that the center of mass coincides with the geometric centroid of any given cross section of the beam.

The general equation of motion then for the transverse vibration of a single beam is

$$\frac{\partial^2}{\partial x^2}\left(EI\frac{\partial^2 v}{\partial x^2}\right) + \rho A\frac{\partial^2 v}{\partial t^2} = p_y(x,t), \qquad 0 < x < L \tag{4-1}$$

where the first term on the left hand side represents the elemental bending moment aspect of Newton's Law and the second term represents the elemental shear force for a beam of length L. For free vibration as is appropriate to consideration of natural frequency response here, the right hand term goes to zero and the Equation of Motion reduces to

$$\frac{\partial^2}{\partial x^2}\left(EI\frac{\partial^2 v}{\partial x^2}\right) + \rho A\frac{\partial^2 v}{\partial t^2} = 0, \qquad 0 < x < L \tag{4-2}$$

Assuming harmonic motion of the form V(x)cos(ω_t-α) where α is the phase lag due to damping, the eigenvalue equation for free vibration of a uniform beam becomes

$$\frac{\partial^4 V}{\partial x^4} - \lambda^4 V = 0 \tag{4-3}$$

where:

$$\lambda^4 = \omega^2\frac{\rho A}{EI} \tag{4-4}$$

The general solution to the fourth-order differential equation in (4-3) is given as [2]

$$V(x) = C_1\sinh(\lambda x) + C_2\cosh(\lambda x) + C_3\sin(\lambda x) + C_4\cos(\lambda x) \tag{4-5}$$

Values for the four constants (C1 thru C4) are obtained by incorporating boundary conditions specific to the beam under study. Two (2) boundary conditions are selected as being sufficient to validate the FEM model to theoretical computations. First a beam which is simply supported at both ends is considered and then a cantilever beam is considered for the second case. Although only parallel beam problem does not include a case where the bottom beam is simply supported at both ends, it is included here in order to add confidence in the validation of the FEM model.

For the uniform beam of length (L) with simply supported end conditions at each end, the boundary conditions are [2]

$$V(0) = 0$$

$$\frac{\partial^2 V}{\partial x^2}|_{x=0} = 0$$

$$V(L) = 0 \tag{4-6}$$

$$\frac{\partial^2 V}{\partial x^2}|_{x=L} = 0$$

Through application of boundary conditions (4-6) for the simply supported beam, it is shown by Craig and Kurdila [2] that a nontrivial solution to the general solution (4-5) exists only if sin(λL)=0 Therefore, the characteristic equation yields eigenvalues (λ_r) as follows [2]:

$$\lambda_r = \frac{r\pi}{L} \tag{4-7}$$

where r indicates the order of the mode.

Use of these eigenvalues, then, yields the following solutions for natural frequency (ω_r) and mode shape (ϕ_r) for a uniform beam of length (L), simply supported at each end. [2]

$$\omega_r = \left(\frac{r\pi}{L}\right)^2 \sqrt{\left(\frac{EI}{\rho A}\right)} \tag{4-8}$$

$$\phi_r(x) = \sin\left(\frac{r\pi x}{L}\right) \tag{4-9}$$

For the second constraint case (cantilever), the boundary conditions for a beam of length (L) are [2]

$$V(0) = 0$$

$$\frac{\partial V}{\partial x}|_{x=0} = 0$$

$$\frac{\partial^2 V}{\partial x^2}|_{x=L} = 0 \tag{4-10}$$

$$\frac{\partial^3 V}{\partial x^3}|_{x=L} = 0$$

Application of boundary conditions (4-10) for the cantilever beam is shown by Craig and Kurdila [2] to yield a characteristic equation as

$$\cos(\lambda L)\cosh(\lambda L) + 1 = 0 \tag{4-11}$$

No simple expression for the roots of this characteristic equation is available so, as referenced by Craig and Kurdila [2], numerical solutions yield the following eigenvalue expressions which are in turn used to define expressions for the first four natural frequencies (ω_r). The first four mode shapes $\boldsymbol{y_r}\left(\frac{x}{L}\right)$ of the cantilever beam are given by Blevins [38]. Constants $\lambda_r L$ are given by Craig and Kurdila [2] and σ_r by Blevins [38].

$$\omega_r = \frac{(\lambda_r L)^2}{L^2}\sqrt{\left(\frac{EI}{\rho A}\right)} \tag{4-12}$$

$$f_r = \frac{\omega_r}{2\pi} \tag{4-13}$$

$$y_r\left(\frac{x}{L}\right) = cosh\left(\frac{\lambda_r x}{L}\right) - cos\left(\frac{\lambda_r x}{L}\right) - \sigma_r\left(sinh\left(\frac{\lambda_r x}{L}\right) - sin\left(\frac{\lambda_r x}{L}\right)\right) \tag{4-14}$$

where:

$$\lambda_1 L = 1.8751\,, \quad \lambda_2 L = 4.6941\,, \quad \lambda_3 L = 7.8548\,, \quad \lambda_4 L = 10.996 \tag{4-15}$$

$$\sigma_1 = 0.734095514,\ \ \sigma_2 = 1.018467319,\ \ \sigma_3 = 0.999224497,\ \sigma_4 = 1.000033553 \tag{4-16}$$

Theoretical Solutions - Static

Deflection of static beams, both cantilevered and simply supported, are well known and documented in literature. Therefore a review of their derivation will not be included here. Rather, the resulting closed form solutions are provided below for point-loaded beams as found in various ‘beam tables’ and texts including Den Hartog. [39]

Tip (maximum) deflection of cantilevered beam of length (L):

$$\delta_{tip} = \frac{PL^3}{3EI} \tag{4-17}$$

Center (maximum) deflection of beam of length (L) simply supported at both ends:

$$\delta_{center} = \frac{PL^3}{48EI} \tag{4-18}$$

Note that as with the larger (parallel beam) optimization study, the weight of the bar due to gravity is not considered here.

Single Beam Model Validation

Description of 1D & 3D FEM Models

The top beam for Family Variant ‘1’ is selected as the model for the ‘single-beam’ FEM validation exercise. Specific dimensional and material properties are given in Table 2-1. As mentioned previously, two (2) constraint cases are considered for this validation exercise:

1) the beam is loaded and supported as a cantilever
2) simply supported at each end with mid-span load.

Load values for the two (2) cases are also taken from Table 2-1. The lumped mass at the free end of the top beam is not necessary or considered in this validation exercise, which is consistent with the assumptions of the theoretical models discussed previously. Finally, both 1D (beam) and 3D (solid) models of the modal analysis are constrained to limit motion to the primary plane only. That is, both lateral translation and axial rotation are constrained to ‘0’ in order to be in keeping with the theoretical models.

Figure 4-1 and Figure 4-2 illustrate a close-up view of the 1D (beam) and 3D (solid) models respectively. Note that although beam elements are used, the commercial FEM program used (ANSYS ® v14.5.7) illustrates the 1D (beam) model in a quasi-3-dimensional fashion as noted in the illustration below (Figure 4-1). However, the beam elements have units of length only, with cross-sectional geometric properties obtained via section property definition. The 3D (solid) model by contrast discretizes the solid body into ‘solid’ elements, including both brick and tetrahedral shapes.

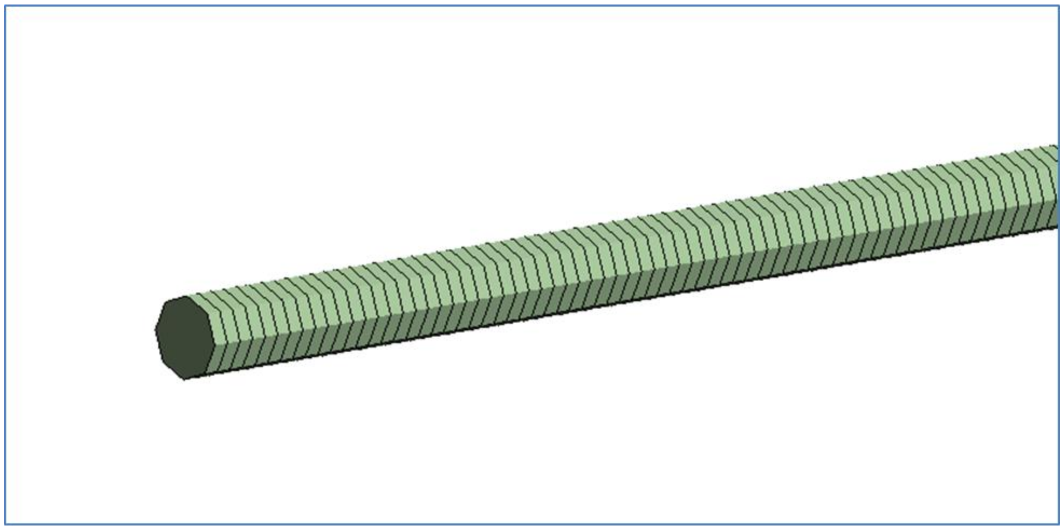


Figure 4-1 Close-up of 1D (beam element) model - nodal spacing = 0.050"

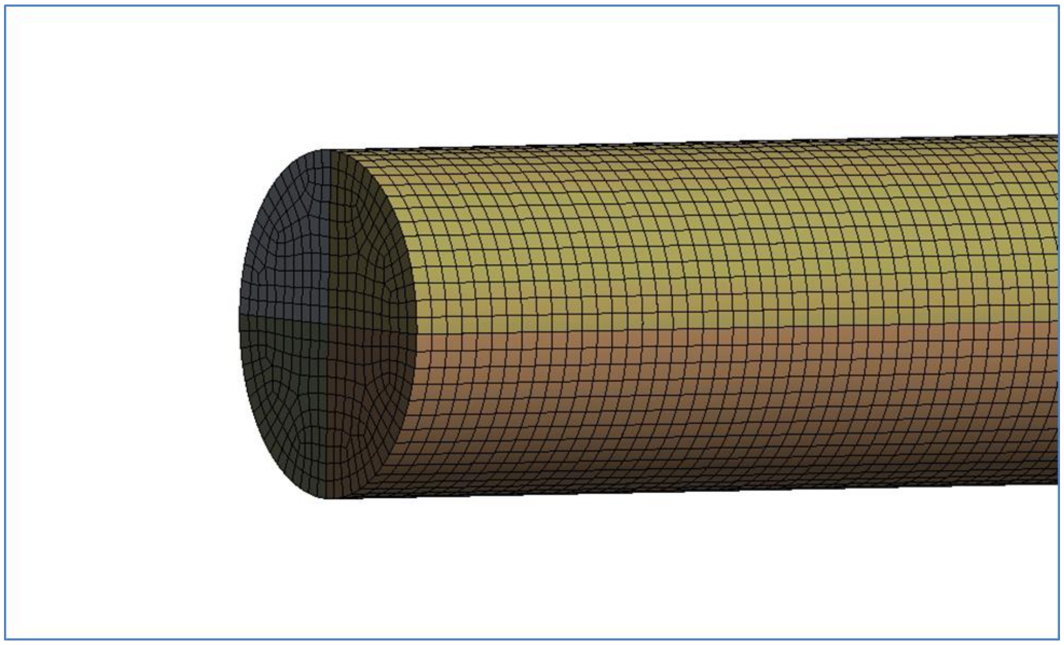


Figure 4-2 Close-up of 3D (solid element) model - nodal spacing ≅ 0.0125"

For purposes of this investigation, the 1D (beam) element type used was ANSYS' ® 3-D, 2-Node beam, BEAM188; based on linear shape functions and uses one point of integration along the length. It is recognized that a higher order element type (such as cubic) would require far fewer elements to reach a converged response. However, the use of linear elements with a nodal density appropriate for solution convergence supports the needs of this study and illustrates the use of the optimization tools in an 'Expensive Black Box' style model. The shape functions associated with ANSYS'® BEAM188 [44] used as a linear beam are provided in the equation below for reference.

$$
\begin{aligned}
u &= \frac{1}{2}\left[u_l(1-s) + u_j(1+s)\right] \\
v &= \frac{1}{2}\left[v_l(1-s) + v_j(1+s)\right] \\
w &= \frac{1}{2}\left[w_l(1-s) + w_j(1+s)\right] \\
\theta_x &= \frac{1}{2}\left[\theta_{xl}(1-s) + \theta_{xj}(1+s)\right] \\
\theta_y &= \frac{1}{2}\left[\theta_{yl}(1-s) + \theta_{yj}(1+s)\right] \\
\theta_z &= \frac{1}{2}\left[\theta_{zl}(1-s) + \theta_{zj}(1+s)\right]
\end{aligned}
\tag{4-19}
$$

Mesh Sensitivity Results and Validation of FEM to Theoretical Models

Validation of the FEM model results is obtained by comparison of both static deflection and modal results to the theoretical results obtained using the equations from the preceding section. As part of this process, mesh sensitivity analyses are conducted for both static and modal results in order to define the appropriate mesh density at which the FEM models converge. For purposes of this exercise, an error value of 1% (between FEM and theoretical result) is selected as a threshold of 'goodness'. This value is selected for the purposes of this study as an appropriate compromise between accuracy and overall model size and corresponding computing requirements.

Mesh sensitivity results as well as a comparison to the theoretical solutions for static deflection are summarized in Figure 4-3 and Figure 4-4 for the Cantilever and Simply-Supported constraint conditions respectively. Data supporting these figures are provided in Appendix A as Tables A-1 and A-2.

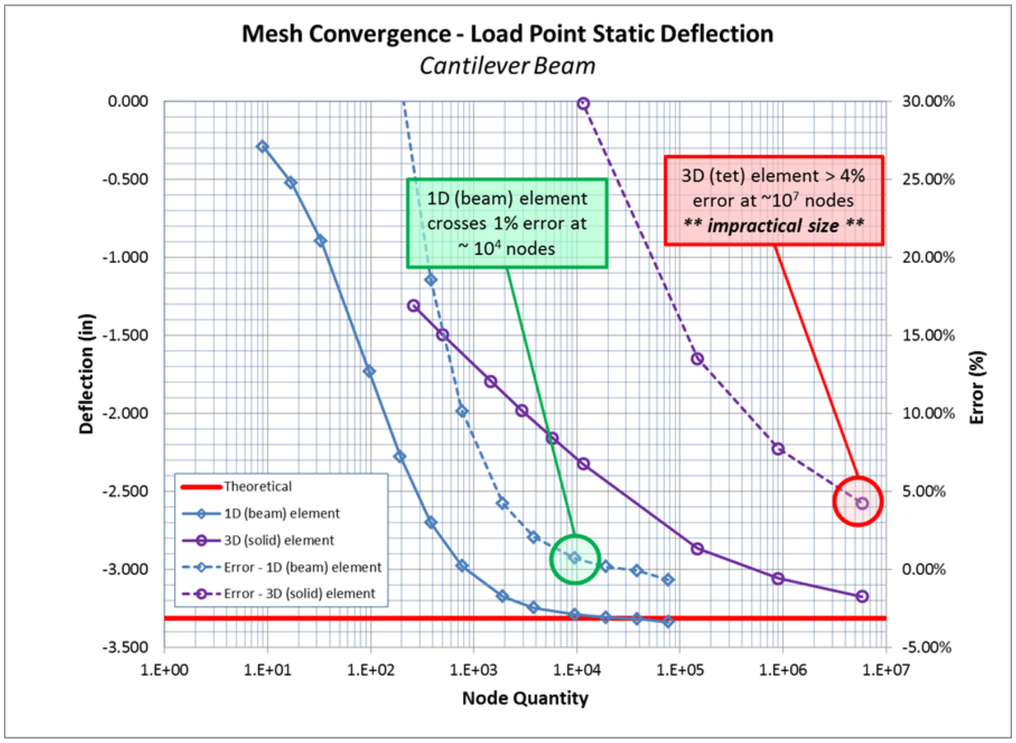


Figure 4-3 Mesh sensitivity and validation results - static deflection, cantilever beam

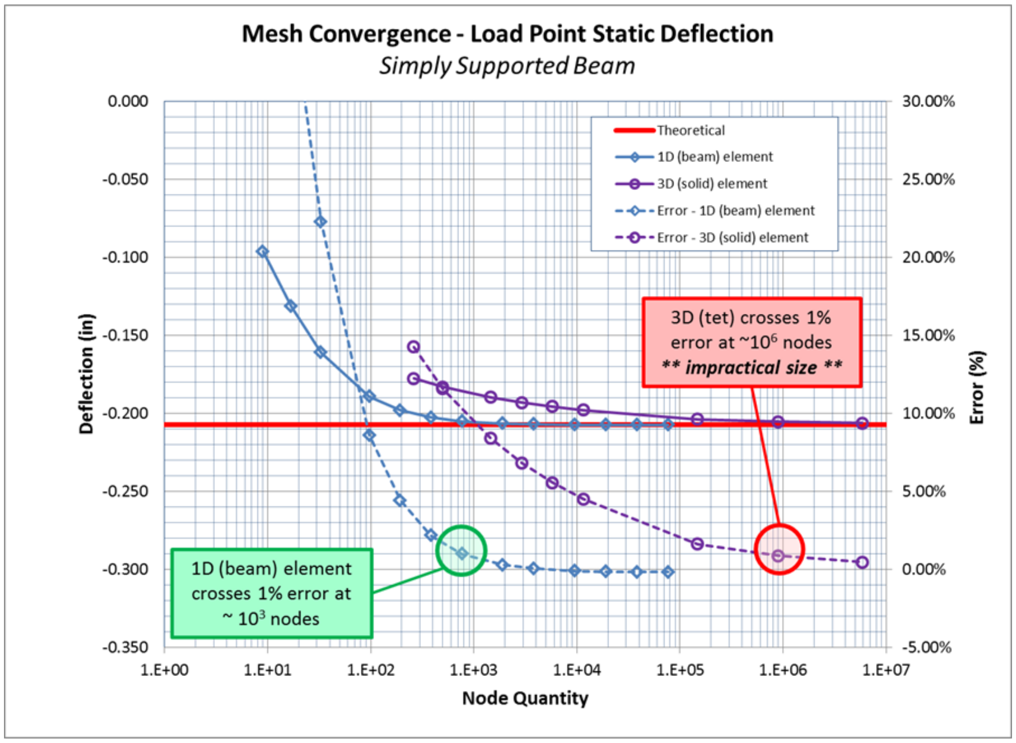


Figure 4-4 Mesh sensitivity and validation results - static deflection, simply supported beam

Both the 1D (beam) and 3D (solid) models of the cantilever and simply supported beams indicate convergence toward the theoretical solution(s). However, the 1D model does so with significantly fewer total nodes than the 3D model in both cases. As shown in the data above, the 1D model achieves 1% error with approximately 9600 total nodes for the cantilever beam and approximately 800 total nodes for the simply supported beam. This equates to approximately 200 nodes per inch (NPI) and 16.7 NPI for the cantilever and simply supported beams respectively. The 3D model, however, requires many more nodes; in excess of 10^6 total nodes ($\cong 2.08\text{x}10^4$ NPI) for the simply supported beam and more than 10^7 nodes for the cantilever. As noted in the figure though, with 10^7 nodes, the cantilever beam fails to reach the acceptability threshold of 1% effort. It is recognized that good FEM practice includes methods to selectively modify the nodal density in different areas of the geometry in order to reduce the required overall number of nodes, but it is anticipated that the 3D model will still require significantly more nodes than the beam modeling method. Further, the repetitive modeling method required by the planned optimization effort makes it impractical to ‘optimize’ the nodal densities for each geometry and a more generalized meshing strategy is preferred.

Comparison of the cantilever and simply supported mesh discretization results then show a significant difference in the minimum mesh densities required. For both 1D and 3D beams, the cantilever end condition requires significantly higher mesh density than the simply supported condition to achieve convergence. This is attributed to the fact that the relatively large deformation of the cantilever beam requires smaller elements (higher nodal density) in order to accurately model the result. The 3D models in general either required such a high nodal density as to be impractical (with respect to computing resources). Therefore, a conclusion is drawn from these results that a 1D element type with nodal density of at least 200 NPI is appropriate to achieve the goals of the study for static deflection.

Mesh sensitivity results for the modal evaluations (frequency value) are summarized in Figure 4-5 and Figure 4-6 for the Cantilever and Simply-Supported constraint conditions respectively. Data supporting these figures are provided in Appendix A as Tables A-3 thru A-6.

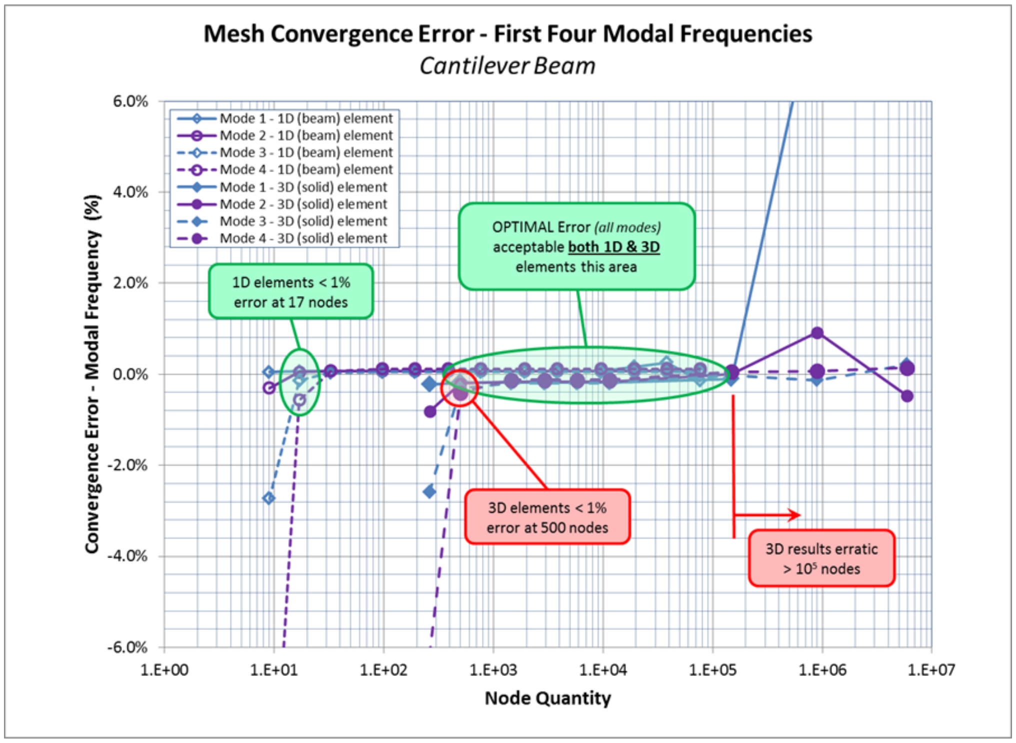


Figure 4-5 Mesh sensitivity and validation results - mode frequency, cantilever beam

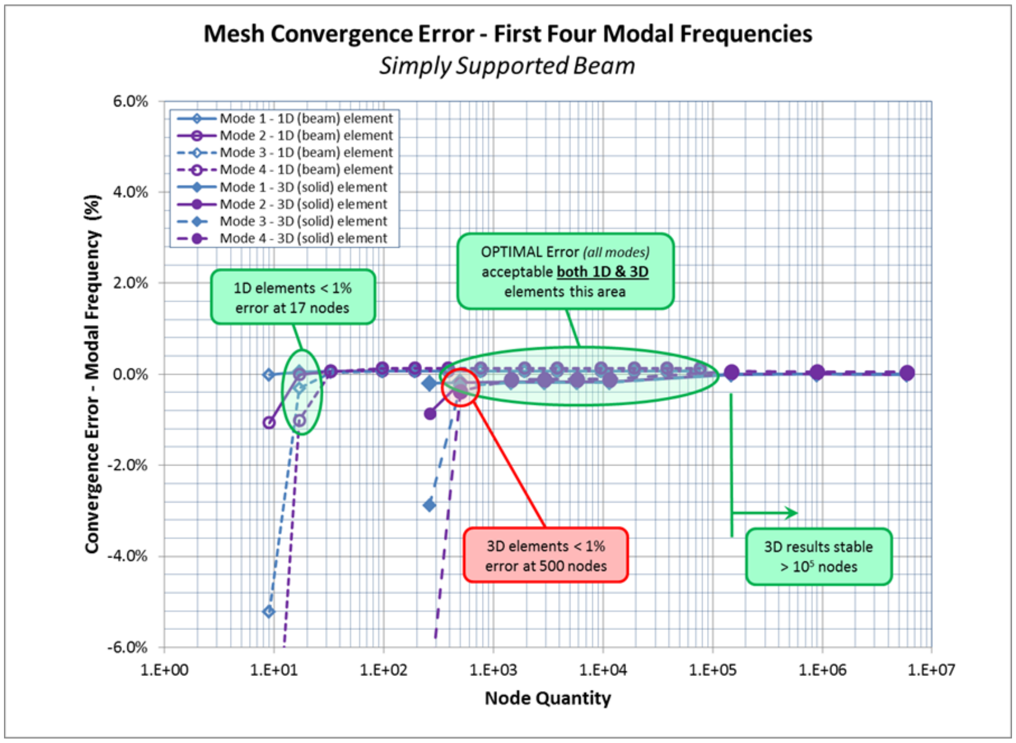


Figure 4-6 Mesh sensitivity and validation results - mode frequency, simply supported beam

As with the static deflection results above, both the 1D (beam) and 3D (solid) models of the cantilever and simply supported beams indicate convergence toward the theoretical solution. Also as with the static results, the modal results for the 1D model shows convergence with significantly fewer total nodes than does the 3D model. However, the minimum nodal density required for convergence is lower for the modal analysis than the structural. That is, the 1D model of the cantilever beam for modal results achieves 1% error with approximately 17 total nodes or 0.35 NPI (vs 200 NPI for the static analysis). The 3D model requires at least 500 nodes (10.42 NPI) for the modal frequency result as compared to $2.08\text{x}10^4$ NPI for the static analysis. The simply supported model has identical results for the modal analysis (as compared to the cantilever beam) with a minimum 17 total nodes for the 1D method and 500 nodes for the 3D method. Note that these values indicate the nodal density required to achieve the error threshold (1%) for all four (4) modes.

Interestingly, the modal frequency predictions for the cantilever beam with 3D elements diverge above approximately 10^5 nodes (20,833 NPI) but remain stable at this high nodal density for the simply supported case. Note that the 1D elements did not show such a discrepancy at high nodal densities; rather, 1D modal frequency predictions remained within 1% of the theoretical values for both cantilever and simply supported beams.

These results indicate that acceptable results may be achieved with nodal densities greater than 0.35 NPI for 1D elements and between 10.42 and 20,800 NPI for 3D elements. Given the selected scope of this work, it is sufficient to accept an upper bound on nodal densities of 3D modal models of 20,800 NPI and not pursue a precise understanding of the divergence at higher densities.

Theoretical results for mode shape of the simply supported models are established in Equation (4-9) above and presented graphically in Figure 4-7 below. By comparison, mode shape results for the 1D FEM solution using a (minimally acceptable) nodal density of approximately 200 NPI are presented in Figure 4-8. As shown, the FEM results for mode shape match the theoretical results. The 3D results for mode shape are identical as well, although they are not depicted here since 3D element types will not be used for the parallel beam study (given the above validation comparisons).

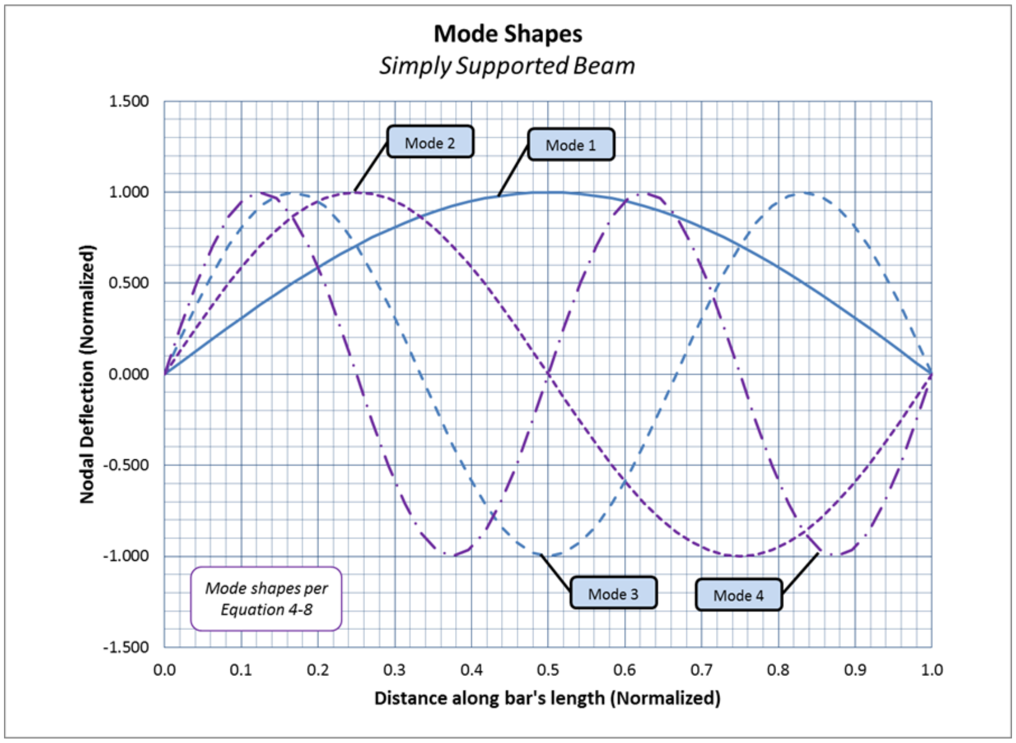


Figure 4-7 Theoretical results - mode shape, simply supported beam

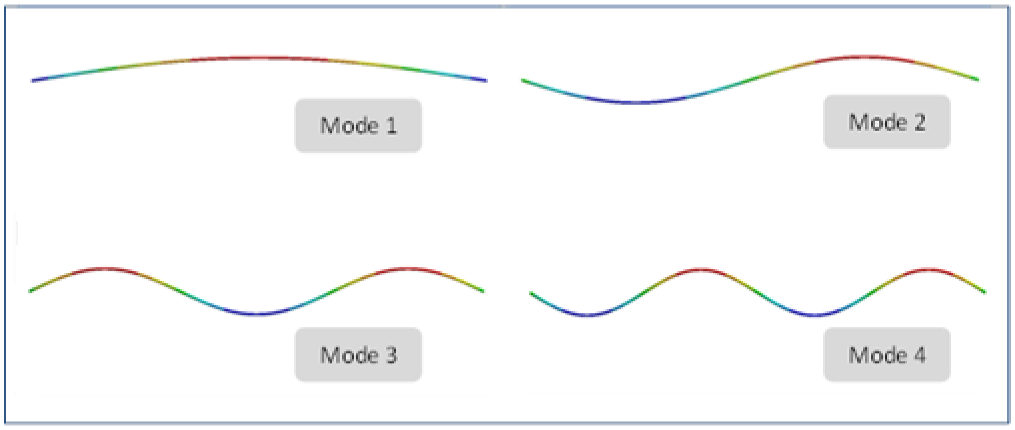


Figure 4-8 FEM results (1D model, 200 NPI) - mode shape, simply supported beam

Theoretical results for mode shape of the cantilever case are established in Equation (4-14) above and presented graphically in Figure 4-9 below. Similar results for the 1D FEM solution using a (minimally acceptable) nodal density of approximately 200 NPI are presented in Figure 4-10 for comparison. Here too, the FEM mode shape results for 1D model of a cantilever beam match the theoretical results and therefore validate this aspect of the FEM model with 1D element type. Although not depicted here, the 3D results for mode shape match as well.

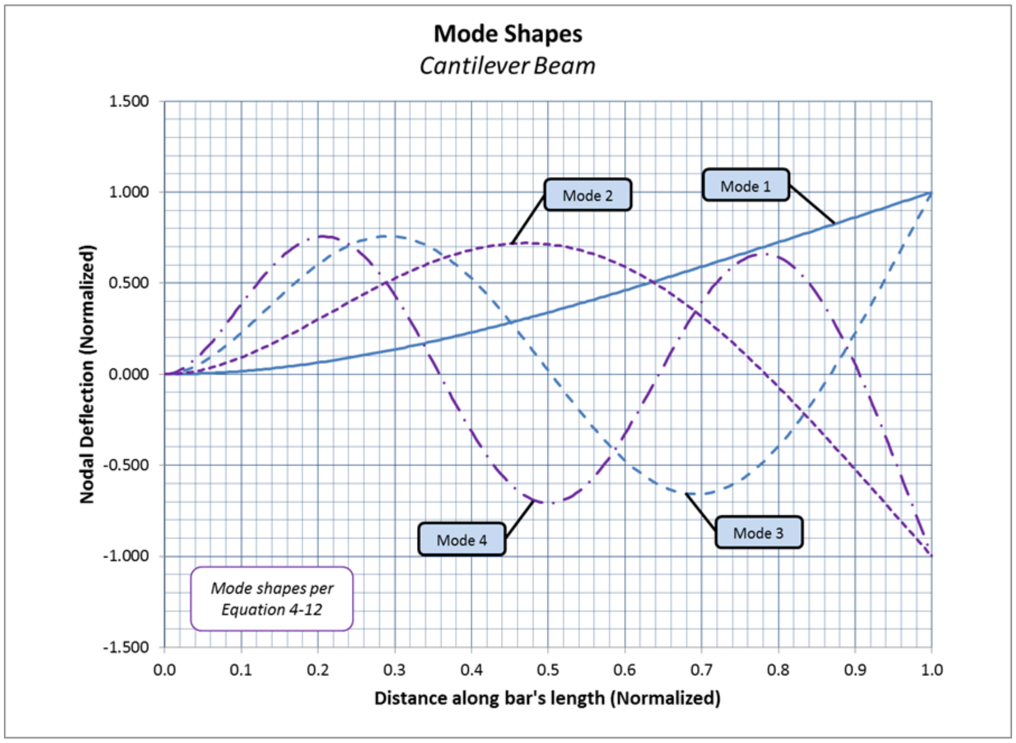


Figure 4-9 Theoretical results - mode shape, cantilever beam

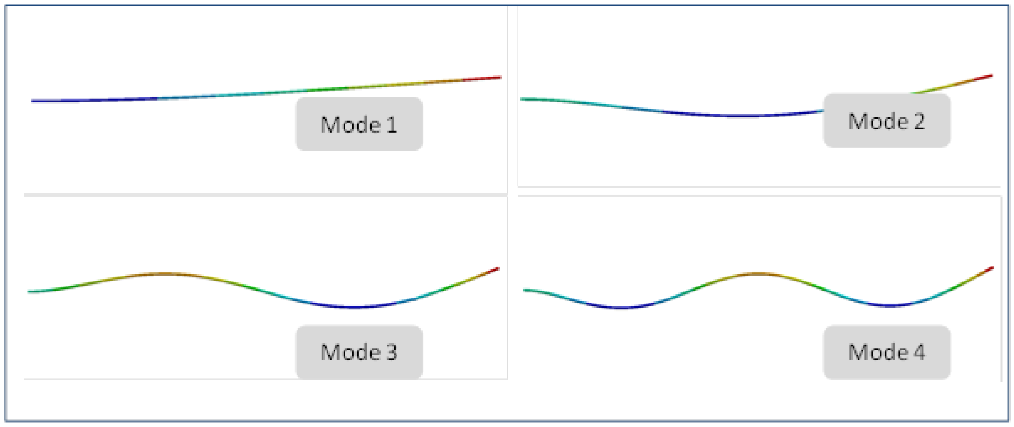


Figure 4-10 FEM results (1D model, 200 NPI) - mode shape, cantilever beam

Parallel Beam Model Validation (1D vs. 3D Harmonic)

In addition to the single beam validation efforts (for static deflection and natural frequency), a comparison between 1D and 3D modeling methods was conducted for the parallel beam model in order to compare and contrast the relative responses under harmonic loading. Although not a true ‘validation’ per se (since a comparison is not made to theoretical results) this comparison serves to help support a decision for 1D vs. 3D modelling method for the parallel beam Problem under Study and eventual optimization exercise. This is necessary because the optimization exercise is anticipated to involve a large number of FEM solves and efficiency during that process is important in order to minimize the overall time required given computing resources at hand.

The parallel beam model used in this validation comparison is as depicted in Figure 2-1 with values specified per Table 2-1 for Family Variant #1 (48” long top beam). Figure 4-11 illustrates the model boundary conditions and loads. As shown, a harmonic load (0.5 lb_f) is applied to the free-end of the top beam. The bottom beam is primarily cantilevered (fixed 3 DOF translation, 3 DOF rotation) with mid-span supports as shown. The left end of the top beam is fixed in 2DOF rotation and axial translation only to allow for ‘free’ movement within the plane of the analysis as described previously. In addition, all nodes of the model are constrained to prevent lateral translation and thereby limit motion to planar. The analysis is conducted from 10 – 100 Hz in 2.5 Hz increments.

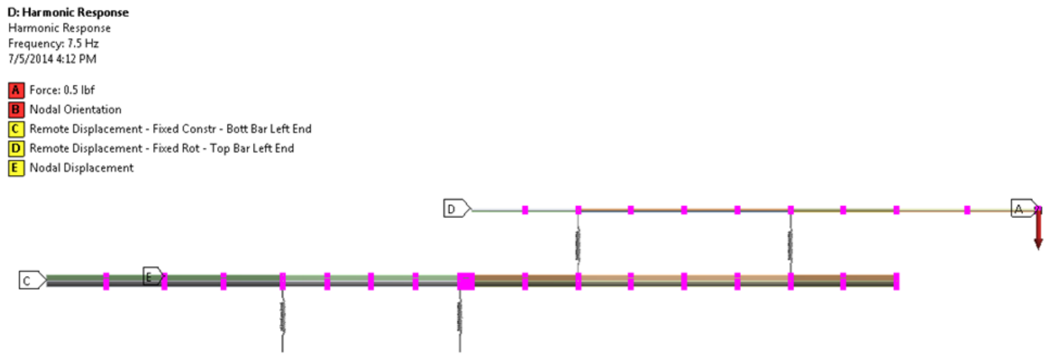


Figure 4-11 Parallel beam harmonic analysis model constraints and boundary conditions

Two (2) types of output metrics are considered for the harmonic model validation effort; displacement metrics of the top beam's free tip (load point) and the reaction forces of the four (4) support springs.

The, displacement response of the top beam's load point is considered via two (2) metrics; by summing the response at each frequency and also by computing the range of harmonic displacements across the range of frequency. This is consistent with output metrics that are to be optimized for the Parallel Beam problem (discussed in detail in a later chapter) and represent for the purposes of this validation study the system response through a variety of conditions.

Spring reaction forces are not considered as part of the Parallel Beam optimization study, but are included here as another type of metric from which to gain insight into the differences and similarities of the two (2) modeling methods. For the validation effort, spring reaction force is considered by summing the forces for each spring across the range of frequencies (independently) in order to gain insight across a variety of conditions in a similar manner to tip deflection (above). The list of output metrics to be compared then is summarized in Table 4-1 below.

Table 4-1 Output metrics for harmonic model comparison

Sum – Load Point Displacements (vertical) Across Frequencies
Range - Load Point Displacements (vertical) Across Frequencies
Sum – Forces Across Frequencies (Spring 1)
Sum – Forces Across Frequencies (Spring 2)
Sum – Forces Across Frequencies (Spring 3)
Sum – Forces Across Frequencies (Spring 4)

Figure 4-12 and Figure 4-13 illustrate the convergence error of the 1D and 3D harmonic models respectively. Data supporting these figures are provided in Appendix A as Tables A-7 and A-8. For these purposes, 'convergence error' is the difference (%) in the result of two adjacent nodal densities. As shown, the 1D model has convergence error < 1% through the entirety of mesh densities studied (70 – $2x10^5$ total nodes or approximately 1.4 – 1600 NPI). The 3D model on the other hand is converged to < 1% only for total node count above approximately

40,000 (approximately 342 NPI). Note that both 1D and 3D models show a trend toward increasing error at higher node densities (above 1600 & 3083 NPI respectively). As with similar trends for the modal and static deflection studies above, this trend is attributed to numerical inconsistencies of the large models and is not explored further within the scope of this work.

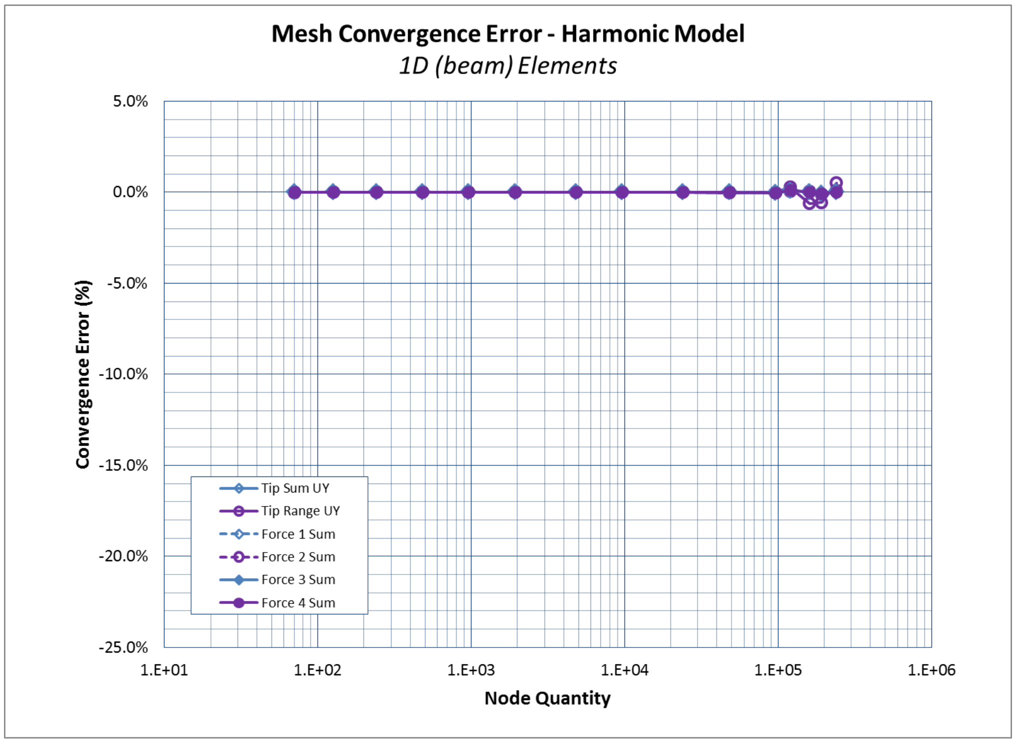


Figure 4-12 Harmonic mesh convergence error - 1D elements

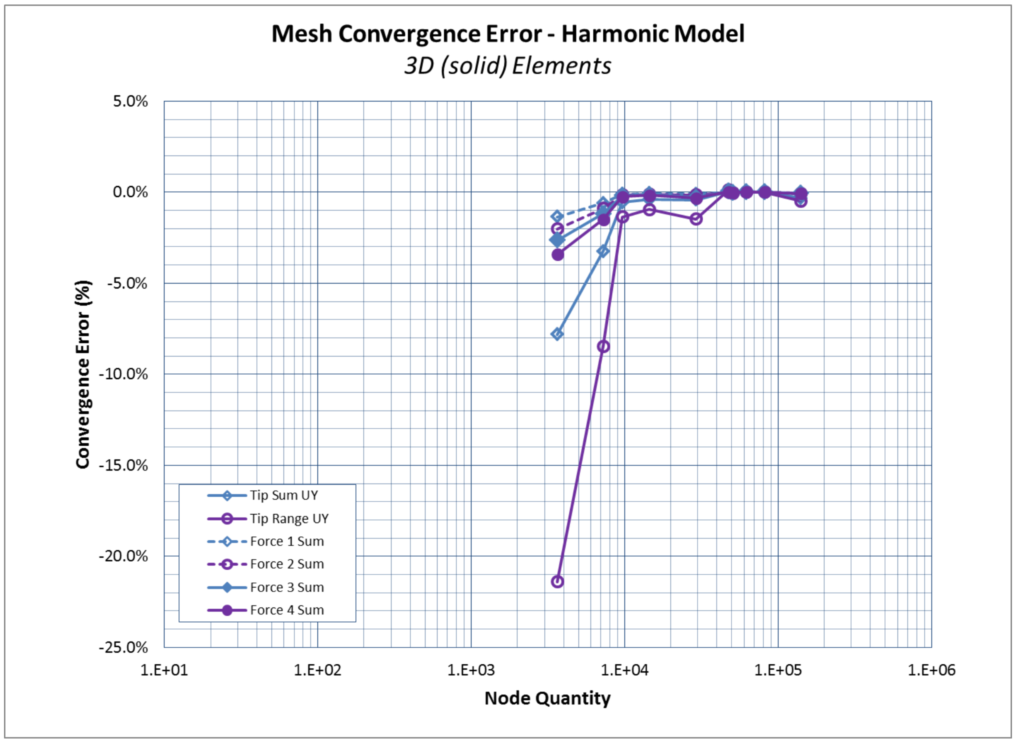


Figure 4-13 Harmonic mesh convergence error - 3D elements

Selected Model Description for Problem under Study

Table 4-2 summarizes the range of acceptable nodal densities (to achieve error < 1%) per the mesh sensitivity results for static, modal and harmonic analyses.

Table 4-2 Acceptable nodal densities for study of parallel-beam

	1D Elements		3D Elements	
	Minimum	Maximum	Minimum	Maximum
Static	200	None observed	2.1x104	None observed
Mode Frequency	0.35	None observed	10.4	20,833
Harmonic	1.4	None observed	342	None observed

It is concluded then that a 1D element model with nodal density of at least 200 NPI is appropriate to balance model size and result accuracy and should be used in the later optimization efforts.

Chapter 5

Development and Characterization of the Optimization Algorithms

Identification of Candidate Methodologies

An objective of this work is to seek an optimization method that is an effective compromise between finding an 'absolute global optimum' solution and the overall cost of the effort as applied to harmonic response problems for platform-style systems. As discussed in a previous section of this report, a wide variety of optimization methods are available; each having pro's and con's with respect to application to the problem at hand. Key considerations are summarized as follows.

- First-order search methods (e.g.; Steep Descent) are known to be efficient in the initial stages of optimization but loose efficiency as the search nears the optimum. Because of their nature, they require gradient information about the surface which is not directly available for Black Box type functions.
- Direct search methods on the other hand (e.g.; Genetic Algorithm, Particle Swarm, Sequential Quadratic Programming, etc.) do not require gradient information about the response which makes them attractive for Black Box Functions. Note that SQP methods can incorporate gradient information to advantage, but it is not required in order to determine the descent direction. Rather, SQP treats the selection of the search vector as an optimization 'sub-problem'.
- Direct search methods are reported in literature as being effective for use in searching for global optimums, especially for multimodal responses. This is particularly important for problems involving harmonic response, where natural frequencies may exist within the operating range causing a multimodal nature to the response.
- Direct methods, however, are reported to require a high number of function evaluations to achieve the desired result. This contradicts the objective of finding a

search method applicable to Expensive Black-Box Functions where each function solution may be very costly.

- Laskari, et al. [31], report that the Particle Swarm direct method outperformed SQP for all but the 'less complex problems'. Noting that PSO (and other direct methods) do not require gradient information, they recommend that these methods might be appropriate for Expensive Black Box functions; especially for finding a 'good approximation of the solution' and then 'continuing with a faster gradient based method, such as SQP'.

Laskari, et al.'s recommendation is the inspiration for the method explored in this work; an initial 'coarse global' search for starting points sampled from across the entire design space, followed by use of an SQP direct-search method in the local region showing the most optimal result from the 'coarse global' search. Their recommendation, however, is based upon use of a PSO direct search method as the initial 'coarse search' given that PSO was the primary subject of their paper.

As discussed previously, Direct methods are known to require a large number of function evaluations during the course of their search. Since this is in conflict with a goal of this work to define a strategy appropriate to large models, a modification to Laskari, et al.'s recommendation is considered for this investigation. That is, to utilize the first-order method of Steepest Descent for the global search, followed by an SQP (local search) method starting from the most attractive global result. Since "Steepest Descent' is, strictly speaking, an unconstrained search algorithm and the problem is a constrained one, a modification is made to account for effects of various constraints. For purposes of this work, the resulting process is termed the method of 'Steepest Feasible Descent' to clarify the distinction. The classical method of Steep Descent, and by extension Steepest Feasible Descent, is known to be particularly effective (efficient) in identifying an approximation to the optimum result within the first few 'jumps', but is particularly inefficient as a local search tool near the optimal point. SQP however, is valued for its efficiency as a local search tool, but is less effective at transitioning ‘ridges’ and other local maxima features of a multimodal response in order to find a ‘global optimum’.

The preferred method for this work then, is to utilize the efficiency of the first (very few) jumps of a Steepest Feasible Descent methodology to find the general area of potential minimum, followed by SQP as the more efficient local search tool. A (perhaps) simpler approach might be to simply evaluate a number of sample points throughout the design space (without a specific directional algorithm) as a method to finding the best regions for a subsequent local search. Success in this approach however would be largely attributed to the ‘luck’ of having the ‘right’ starting points since no real intelligence is utilized during such a search. By incorporating the first few jumps of a Steepest Feasible Descent methodology however, it is theorized that the additional ‘intelligence’ will result in a more intelligent estimation of global minimum; with a limited number of function evaluations. By following this Steepest Feasible Descent initial search with a more refined SQP local search, it is theorized that the best of both methods will contribute to a better result than either method employed individually.

Although the combined method appears to be an efficient overall strategy, its effectiveness on a multimodal response (as opposed to a ‘straight-forward’ application of either the Steepest Feasible Descent or SQP methods) is not immediately clear and is therefore a subject of comparison in this work. Specifically, three (3) optimization methodologies are explored and compared here in order to identify an ‘optimal’ methodology for later use on the parallel beam problem. The candidate methods are:

a) Use of Steepest Feasible Descent method (only), starting from a given number of points sampled from across the design space. As with the ‘combined’ method (below), this Steepest Feasible Descent search will be limited to a very few ‘jumps’ for each starting point in order to evaluate the efficiency of this approach.

b) Use of SQP (only) starting from each of a sampling of starting points from across the entire design space. In this way, SQP is used as a ‘global’ search tool and identifies a potential result for each starting point based upon selected convergence criteria.

Use of SQP in this way evaluates its ability to cross ‘ridges’ and other local maxima in a multimodal response in order to find an acceptable global minimum.

c) The combined use of the method of Steepest Feasible Descent as a coarse ‘pre-search’ for the sampling of points across the design space followed by a more refined SQP method starting at the single most optimal result of the pre-search. In this way, it is hoped that the most-efficient aspects of each approach will be exploited and the less-efficient aspects of each likewise avoided in the overall method.

Down-Select Technique and Description of Test Functions

Down-Select Technique

The three (3) candidate optimization methodologies each have strengths and weaknesses related to the Problem under Study based on the theory and results of the literature review. Without specific comparative data however, a firm recommendation cannot be made as to the ‘best’ method to be used for this application. Therefore, a comparative study and data-based selection of the most appropriate method from among the candidates needs to be conducted. In order to draw strong conclusions, this ‘down-selection’ is based upon a comparison of performance from among the candidate methods against standard test functions for which optimum theoretical values are known. Two (2) key metrics are identified for this comparison. They are:

- Total number of function evaluations used to identify the global optimum value (efficiency).
- Comparison of the identified optimum for each candidate technique as compared to the theoretical (global) minima of the test function (efficacy).

Based upon this comparison, an ‘optimal’ method will be selected and later utilized on the parallel beam problem.

Test Function Descriptions

Test functions are needed that have known theoretical (global) optimums in order to compare the relative efficiency and efficacy of each candidate optimization method. Since the problem to be solved is a harmonic analysis where multimodal responses are expected, it is appropriate that the test functions also include a variety of modalities. Also, the parallel beam problem is defined as a multi-variable problem. Therefore, the test functions need to incorporate this capability as well. For the purpose of this comparison however, their use here will be limited to 2 dimensions in order to aid in visualization of the program responses. Given these considerations, four (4) standard test functions [40] are selected with known theoretical global optimums, varying degrees of modality and the capability to scale with respect to the number of input variables. They are listed here in increasing order of 'difficulty' and discussed more fully below. The test functions are:

- De Jong's Function
- Rosenbrock's Valley
- Schwefel's Function
- Rastrigin's Function

De Jong's Function

This function represents a simple test function which is parabolic in nature. As such, the surface is both unimodal and convex making it among the easiest to solve. De Jong's function in 2-D is given below and illustrated in Figure 5-1.

$$f(x_1, x_2) = {x_1}^2 + {x_2}^2 \tag{5-1}$$

Test Area: $-5.12 \leq x_i \leq 5.12, \quad i = 1,2$

Theoretical Minimum at $(x_1, x_2) = (0,0); \quad f(x_1, x_2) = 0$

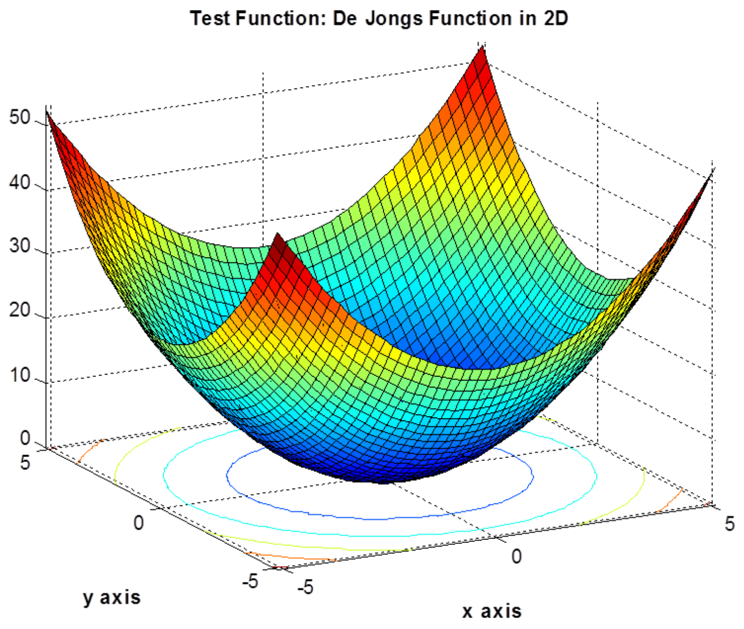


Figure 5-1 De Jong's function in 2D

Rosenbrock’s Function

Sometimes referred to as Rosenbrock’s Valley, this function represents a somewhat more complicated test surface in that it introduces multi-modality over part of the domain. As can be seen in Figure 5-2, an elongated ‘valley’ exists on either side of a bisecting plane within which lies a global optimum. Although the function appears to be symmetric about the bisecting plane, the two crescent-shaped ‘minima’ areas are slightly different such that only one global optimum exists. This, together with the fact that the valley ‘floor’ has little differentiation from the global optimum makes this function particularly challenging. Rosenbrock’s function in 2D is defined as

$$f(x_1, x_2) = 100 * (x_2 - x_1^2)^2 + (1 - x_1)^2 \tag{5-2}$$

Test Area: $-2.048 \leq x_i \leq 2.048, \quad i = 1,2$

Theoretical Minimum at $(x_1, x_2) = (1,1); \quad f(x_1, x_2) = 0$

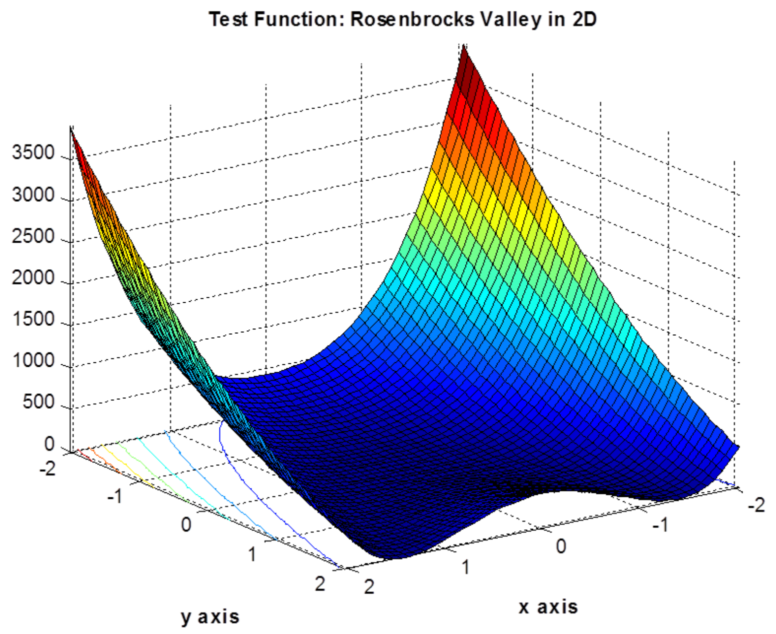


Figure 5-2 Rosenbrock's function in 2D

Rastrigin's Function

Rastrigin's test function is a significantly more complex test surface than either of the previous functions in that many local minima exist within the standard domain. This makes it a good test surface for routines needing to assess multimodal behaviors. However, it is expected to be more complicated (i.e.: a greater number of local minima) than the multimodal surface of the parallel beam problem (and thus a conservative test function). Since the function is based on De Jong's, the surface is (generally speaking) symmetric about the two planes through x_i=0; creating a global optimum at the origin. Rastrigin's function in 2D is defined as

$$f(x_1, x_2) = 10 * [x_1^2 - 10\cos(2\pi x_1)] + [x_2^2 - 10\cos(2\pi x_2)] \tag{5-3}$$

Test Area: $-5.12 \leq x_i \leq 5.12, \quad i = 1,2$

Theoretical Minimum at $(x_1, x_2) = (0,0); \quad f(x_1, x_2) = 0$

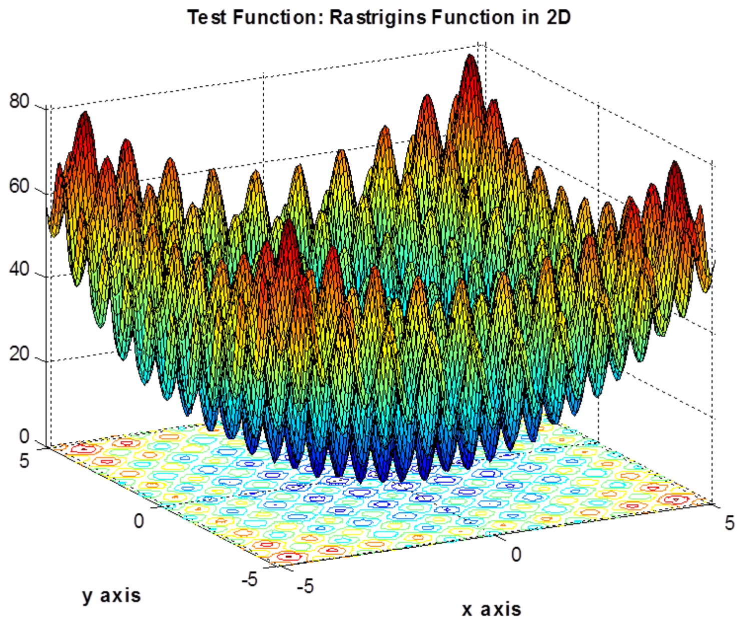


Figure 5-3 Rastrigin's function in 2D

Schwefel's Function

Schwefel's function is similar to Rastrigin's in that several local minima exist within the design space. Unlike Rastrigin's however, Schwefel's function is not symmetric. As shown in Figure 5-4, the global minimum is geometrically distant from the origin; providing a challenge to optimization routines that might favor symmetry. Schwefel's function in 2D is defined as

$$f(x_1, x_2) = (418.9829 * 2) - \left[x_1 * \sin\left(\sqrt{|x_1|}\right)\right] + \left[x_2 * \sin\left(\sqrt{|x_2|}\right)\right] \tag{5-4}$$

Test Area: $-500 \leq x_i \leq 500, \quad i = 1,2$

Theoretical Minimum at $(x_1, x_2) = (420.9687,420.9687); \quad f(x_1, x_2) = 0$

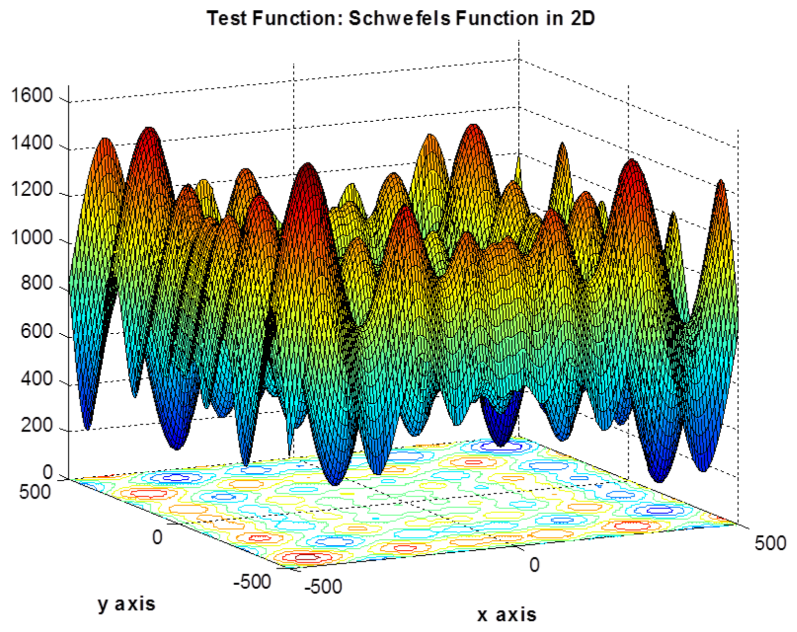


Figure 5-4 Schwefel's function in 2D

Description of Optimization Program and Key Subroutines

An optimization program was written from scratch in MATLAB ® as part of this overall project. The Steepest Feasible Descent subroutine(s) are based on classical methods, but include a few enhancements that are believed to enhance the effectiveness. The direct search subroutine leverages an existing function (fmincon) available within MATLAB ®, but otherwise was also written from scratch as part of this project.

In general, the program calls an appropriate 'external' solver to evaluate the function at the design variable conditions and then returns the results for consideration of the next optimization move (change in design variable values). This external solver could either be an ANSYS ® FEA project, an external MATLAB ® Objective Function or an internal subroutine to evaluate one (1) of the above described test functions. For the parallel-beam problem, the internal subroutine calls ANSYS ® as the external solver with reference to a predefined FEM 'project' including geometry, material, boundary condition and other relevant specifications. Input variables are passed from the primary code to ANSYS ® for the particular 'design point solution', and results then passed from ANSYS ® back to the primary MATLAB ® code. Use of the external MATLAB ® function is similar in that function evaluation is handled by the external MATLAB ® function. The ‘function solver’ for the four (4) test functions is written as a subroutine internal to the main program simply because this functionality is intended to be a ‘verification’ alternative for the user and, as such, has limited scope with respect to its use. That is, it is not intended that the user be able to modify the test function algorithm(s) other than to select among their use. Regardless of the solver used however, the program allows the user to select from among the three (3) candidate optimization methods:

a) Global Optimization via Steepest Feasible Descent Method
b) Global Optimization via Direct (SQP) Method
c) Global Optimization via Steepest Feasible Descent followed by Direct Optimization via SQP

The ‘top-level’ program flow to negotiate among these options is diagrammed in Figure 5-5. Detail describing each of the major program elements follows and key subroutines are charted in Appendix B. A full program listing is provided in Appendix C.

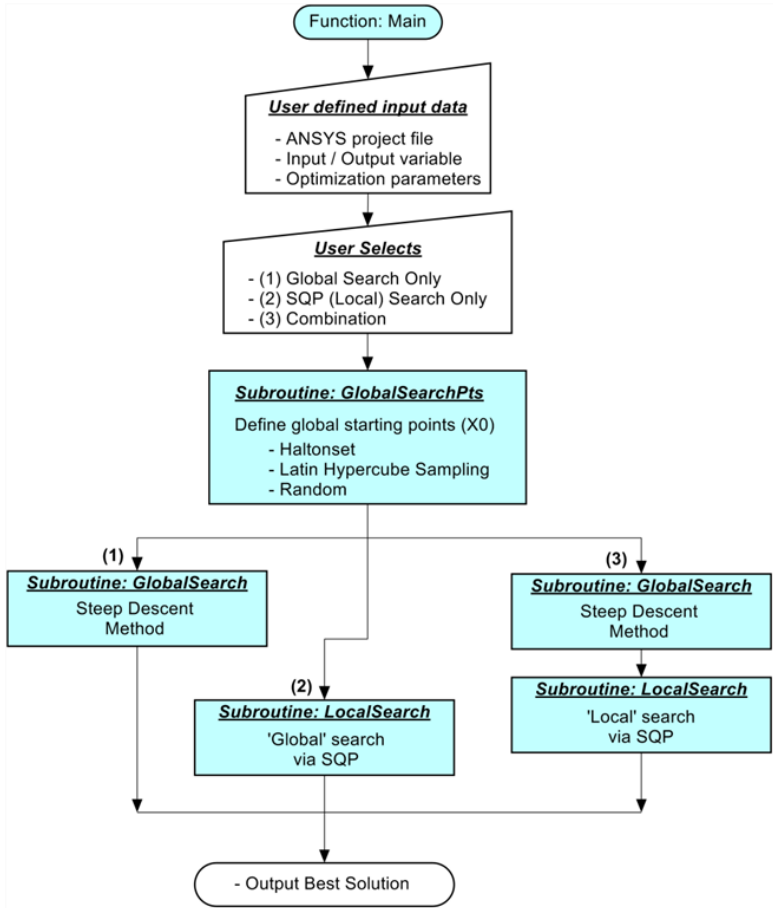


Figure 5-5 Program flowchart - main program flow

Method of Steepest Feasible Descent from Global Sampling Points

The classical first-order search method of Steepest Descent is an unconstrained optimization technique [20] that relies on the identification of a descent gradient (search vector) from a given starting point, along which a 1-D optimization is then conducted to find a minimum value. This process, termed here as one 'jump', is then repeated until either a pre-defined number of jumps is reached or convergence criteria is met.

Three (3) significant challenges exist to the effective implementation of this approach. They are:

1. Selection of the search direction given consideration for a Black Box type function and the effects of constraints

2. A 1-D line search technique that accounts for both multimodal responses and the effects of constraints; including side-bounds and other constraints on design variables as well as result-oriented constraints
3. Appropriate selection of start points for the global search

Discussion of each of these challenges and the implemented solutions follow.

Selection of a Feasible Search Direction

The first-order search technique of Steep Descent is not in itself a method for solving constrained problems. The search direction in the classical implementation [20] is simply derived as the vector with the steepest descending gradient based upon first-order information of the response at the starting point (X0).

The (parallel beam) Problem under Study meets the criteria of a 'Black Box function' and, as such, an equation for the response is not known prior to the optimization attempt. Because of this, the search direction is established via Finite Difference Approach where a linearized equation of the response is established for a very small 'area' surrounding each starting point (X0). The size of this 'area' is determined by offsetting each design variable (independently) by a small value. In theory, and when each of the design variables are of the same order of magnitude, this process works very well. In practice however, optimization problems may require solution where the design variables are of significantly different orders of magnitude. Such is the case with the parallel beam problem where stiffness values can be of the order of 10^5 lbf/in while physical dimensions are less than 100 in and damping coefficients may be as low as 10^{-3}. In this case, a constant value for the offset value on each design variable (from X0) may skew the sensitivity feedback of the derived Sn. That is, if an offset magnitude is chosen to be small enough to be appropriate for the damping coefficient for example (order 10^{-3}), it will represent an insignificant difference to the spring stiffness (order 10^5). In this case, the change to the spring stiffness would result in negligible change to the solution, where the same step-size change to the damping coefficient may result in a much more significant effect. As a result, the computed Sn vector would incorrectly favor a descent direction for the damping coefficient with little or no component in the direction of spring stiffness; even if the response gradient actually may favor a steeper gradient for the spring stiffness when viewed from a ‘larger scale’.

Two (2) options to correct this problem are identified and considered for use here. First, the step size could be defined as a function of the defined range in the given design variable. That is, as the step from X0 for each (independent) design variable is considered, the step size could be established as a percentage of the range of that design variable. This is given in the equation below.

$$\Delta_x = \alpha_r \left(x_{Upper\ Bound} - x_{Lower\ Bound} \right) \tag{5-5}$$

where:

$$\alpha_r = percentage\ of\ the\ range$$

A second option would be to utilize a percentage of the average of the bounds for the given design variable per the following equation.

$$\Delta_x = \alpha_a \left(\frac{x_{Upper\ Bound} - x_{Lower\ Bound}}{2} \right) \tag{5-6}$$

where:

$$\alpha_a = percentage\ of\ the\ average$$

In general, the effect of either of these two (2) approaches would be an improvement over the constant step size described previously. However, the ‘averaged’ approach of Equation (5-6) more directly addresses the fundamental issue of identifying the order of magnitude of the design variable and is therefore selected for inclusion in the program. A value of 0.5% is selected for α_a in order to balance the need for a large enough step to be meaningful while remaining small enough to consider only ‘local’ effects around X0. This becomes particularly important when the response is highly multi-modal as is discussed later.

To confirm this theory, a simple test is derived using a Classical SDOF system with damping (Figure 3-1). The goal is to substantiate that:

- A large order of magnitude difference among variables can influence the resulting Sn vector.
- The sensitization method proposed by Equation (5-6) results in a more appropriate search vector than a common step method under such circumstances.

To conduct the test, a simple optimization problem is created to minimize the FRF (Equation (3-5)) given side-bound conditions only for each of the three (3) design variables. Side bounds are established for two (2) sets of conditions: 1) minimal order of magnitude difference between the variables and 2) large order of magnitude difference among the variables in order to test that the sensitized method of determining Sn is (or is not) correct.

First, the idea is challenged that a ‘common’ step size for each design variable is insufficient when large differences in the order of magnitude exist; that is, the first test supports that the use of a common step size in such instances is ‘a problem’. To do this, each of two (2) variable sets are solved using the ‘common’ step size method, one where the variable magnitudes are relatively close to one another and another where there is significant difference in the magnitudes among the variables. A comparison of the Sn resulting from each method will be used to support a statement that the common step size method is (or is not) problematic in such circumstances..

Secondly, the variable set with large order of magnitude differences is to be solved by both methods for deriving the Sn vector; that is, by both the ‘common step size’ method and the ‘sensitized Sn’ method of Equation (5-6). From these results it will be demonstrated that, for large order of magnitude differences, a difference in the resulting Sn vector exists (or not) based upon method. By evaluating the results of both tests it will be demonstrated that a) a problem exists to be solved and b) which method of computing Sn is more appropriate when large size differences exist among the variables.

Input parameters for the test are given in the following table. Note that Data Set (1) has minimal difference among the variable’s magnitudes whereas Data Set (2) has more pronounced differentiation. For each run, the program is asked to solve an external MATLAB ® optimization function where the objective function is the result of Equation (3-5) for a single forcing function frequency (Ω). (The use of a single forcing frequency as opposed to a range as is typical for FRF studies manages the size of the problem while achieving the objectives of the example.) Constraints are limited to the side bound constraints on the variables for simplicity.

Table 5-1 Input parameters for test of Sn derivation methodology

Parameter	Data Set (1)		Data Set (2)	
	Minimum	Maximum	Minimum	Maximum
Mass	1.0 kg	10 kg	10 kg	100 kg
Spring rate	10 N/m	50 N/m	10000 N/m	50000 N/m
Damping ratio	0.01	0.50	0.01	0.50

Results from the comparative runs are provided below. For each run, the Sn vector is computed by comparing the start point and end points, and then normalizing to a unit vector length.

Table 5-2 Resulting Sn vectors for test of Sn derivation methodology

Run	Computation Method	Sn Vector Components		
		mass	stiffness	damping
1	Common – Data Set 1	0.9733	-0.2294	0
2	Common – Data Set 2	0	0	0
3	Sensitized – Data Set 2	0.1166	0	0.9932

The comparison between runs 1 and 2 supports that a ‘common step size’ method (of deriving the Sn vector) is less effective when large variation exists in the relative orders of magnitude among the variables. The data set where limited variation exists (Run 1) resulted in a non-zero Sn vector, which is desirable. The data set with larger variation (Run 2), however, resulted in a ‘stalled’ analysis where the Sn vector was (incorrectly) determined to be 0.

The comparison between runs 2 and 3 support that the ‘sensitized step size’ method (of deriving the Sn vector) is more productive (and desirable) when large differences exist in the relative orders of magnitude among the variables. Given that both runs 2 and 3 utilize data set 2 (with the larger difference in magnitudes), run 3 detected that a descent vector exists with components of both mass and damping, as opposed to the stalled condition of run 2 using the ‘common step size’. The correctness of the sensitized step method (run 3) over the other runs is further supported by the theory of the Frequency Response Function (Equation (3-5), Figure 3-2) where the value of the damping ratio is shown to be a dominant factor. Neither Run 1 nor Run 2 identified damping ratio to be a dominant factor; an inconsistency between these results and the theory.

Given the results of Table 5-2, it is concluded that the ‘common step size’ method can be problematic in the correct identification of the Sn vector when there is large differences in magnitude among the input variables. Further, the data supports that the ‘sensitized step size’ can result in a more correct answer. Therefore, the sensitized method is selected for use in this project; resulting in the descent gradient (Sn) along which the steep descent 1-D line search is to be conducted. For purposes of this algorithm, it is assumed that the starting point (X0) is in feasible space. Consideration however must be given to feasibility in the direction of the Sn vector where some modification of the vector may (or may not) be required in order to remain in feasible space. This consideration of modification to the Sn vector is discussed below, and start point selection is discussed more fully in a later section. Determination of the Feasible Descent Vector is one of the key subroutines charted in Appendix B.

Once a candidate Sn has been identified based on a finite difference analysis of the area surrounding X0, the effect of constraints (side bounds and other design-variable constraints) must also be considered in order to establish that the Sn which is selected actually points to feasible space. Note that only design-variable constraints are considered in the potential modification of Sn. Result-oriented constraints would require solutions which, to this point, have not yet occurred. Result-oriented constraints are considered later in the evaluation of the 1D line search and are discussed below.

To consider the feasibility of the Sn vector to design-variable constraints, the distance from X0 to the nearest input constraint bound (in the direction of Sn) is first established and compared to the distance along Sn to the nearest input constraint bound. That is, a ratio is developed along the direction of Sn comparing the distance to the nearest constraint in the direction of travel vs. the total distance 'across the design space – both directions' along Sn. If this ratio falls below a predefined threshold, then the starting point X0 is considered to be 'against a constraint wall' and modification of the direction of Sn is needed to remain within feasible space.

Redirection of Sn to avoid the constraint is accomplished by identifying that component direction of Sn which has minimum distance to the constraint(s) and then setting that component of the vector to '0'. In this way, Sn is modified to the steepest descending vector 'along' the constraint boundary in order to remain within feasible space. Once this first modification to direction is made, the ratio of distance to the nearest constraint (in the new direction) vs. total distance is repeated to determine if additional redirections are needed. In the case that X0 is actually in a 'corner' with respect to design-variable constraints, then the algorithm returns a vector Sn with each component value equal to '0' and an error-handling routine identifies that no further search can be made from this X0. In this event, the function value at X0 is evaluated and recorded as the result of the given 'jump'. The effectiveness of this methodology is demonstrated in a later section.

One-Dimensional Line Search Strategy for a Constrained Problem

A goal of line search techniques is to find a 'sufficient' minimum along the search path with a minimum of computational effort. Armijo's-based methods (discussed previously) as well as the derivative methods are common solutions because they 'estimate' the minima with a minimum of function evaluations. These methods are particularly effective for simple, convex responses but are not effective where the response is multimodal.

Because of the expectation that the response surface will be multimodal for the harmonic problem at hand, a polynomial approximation method is used as part of the Steepest Feasible

Descent method here. The 1-D line search method is also one of the key subroutines charted in Appendix B.

In general, the 1-D line search algorithm searches for a minimum along Sn given a predefined limit for the maximum number of function evaluations (along each Sn search), accuracy of the curve fit (both R^2 and error at each solved point) and with consideration for feasibility of the results. A maximum search distance is computed from the starting point (X0) to the nearest design variable constraint (side-bound or other). Within this span, an initial sampling of points is evenly distributed and solved and the resulting data-set fit to a polynomial of order specified by the user. Based upon the curve fit of the 1-D polynomial to the data, a regression coefficient (R^2) is computed. A theoretical minima is computed for the polynomial and error evaluated between this predicted minimum and the value for the solved point with lowest function value result. If either the R^2 or error values are worse than the predefined acceptability criteria, then an additional point is established and evaluated at (or very near) to the location of the predicted minima value (from the 1D polynomial evaluation) and the process repeated. As points are added to the matrix, the polynomial order is increased to improve the accuracy of the curve fit up to a maximum user-specified order value. This maximum polynomial order is important because curve fits with large polynomial orders often result in erroneous fits due to numerical conditioning errors, etc. In practice, a maximum polynomial value of ‘4’ was observed to yield good results. Values significantly above ‘4’ were more likely to result in numerical conditioning errors. Also, the minimum distance between the new point relative to the predicted minimum location is controlled to prevent numerical conditioning errors due to a number of points being too close together.

The goal, then, is to find the location for the minima along the 1D polynomial, meeting acceptance criteria and with the fewest number of function evaluations given limitations on the order of the polynomial to avoid numerical conditioning errors. Given enough solved points along the search path, and for unimodal or ‘low-‘modal functions a solution can be found relatively quickly (with a few number of function solves). However, for search paths which are highly multimodal (and with consideration for the cost of each function evaluation of an EBBF) a successful 1-D line search can become a challenge. Therefore, a numerical strategy is implemented as follows in the case where acceptance criteria are not met within a user-specified maximum number of function evaluations along each search vector. That is, once the maximum number of points has been reached along a given search path, the strategy shifts from that of adding points near the anticipated minima to that of eliminating points far from the observed minimum in an effort to improve the accuracy of the curve fit in the area of interest. The criterion for 'keeping' a point in this instance is that it lies within ± 25% of the maximum span along Sn (from X0 to the nearest constraint bound). The user selectable value for the maximum number of points to evaluate along Sn is a compromise between increased solution times and the accuracy of the curve fit (and resulting predicted minima). In practice, acceptable results were achieved when between 15 and 20 maximum function evaluations were utilized along each Sn vector.

As with the selection of the Feasible Descent Search Vector, the effect of constraints must be considered for the 1-D line search algorithm. Here, however, all constraints (both side, input and output) are considered. Given that the Sn vector has been selected with consideration for design-variable constraints though, the emphasis here is on the results-oriented constraints. One popular approach found in literature is to minimize the objective function using the unconstrained 1-D search method, but apply a penalty function as constraints become violated. Chen and Chi [41] for example, utilize a penalty function to improve their particle swarm algorithm by applying a penalty to particles that are at the upper or lower bound of the design variable definition.

There are multiple ways in which to apply a penalty to the unconstrained objective function. For example, the penalty can be applied as an 'exterior' function whereby no penalty is incurred so long as all constraints are satisfied, but some penalty is assessed as a function of the constraint where a violation exists. [20] An example of this is shown in Equation (5-7) where an objective function f(x) is penalized by squaring the constraint function together with a penalty multiplier (α_p) if it fails to be satisfied (g>0).

$$f(x)_{pen} = f(x) + \alpha_p \sum_{j=1}^{n} \left(max\left[0, g_j(x)\right]^2 \right) \tag{5-7}$$

Vanderplaats explains that the motivation for squaring the constraint violation is to provide a slope of zero at the constraint boundary for the resulting penalized objective function. In this way, the transition from feasible to infeasible space is smoothed and the 1-D line search methodology thereby simplified. An alternative to the penalty method of Equation (5-7) is to replace the 'squared constraint value' with a linear value. The motivation for this approach is that for constraint violations near the boundary, although the 'squared' term does smooth the transition, the resulting penalty effectiveness may be minimized excessively for small positive constraint values. The consequence of this could be reduced effectiveness of the method to define the constraint border for multi-modal responses. Therefore, a variation of the penalty equation is also considered where the constraint value remains linear. This is given below as Equation (5-8).

$$f(x) = f(x) + \alpha_p \sum_{j=1}^{n} \left(max\left[0, g_j(x)\right] \right) \tag{5-8}$$

The comparison regarding the effectiveness of these two (2) methodologies with respect to a constrained 1-D line search is demonstrated in a later section.

The inclusion of a penalty (without other consideration), however, may not be sufficient to adequately define infeasible space during the 1-D line search process. As shown in Figure 5-6 below, the effect of the penalty function to accurately identify the constraint boundary can be limited. In theory, the penalty function should alter the curve of the function value such that the resulting minimum is outside of infeasible space. However, in this example the penalized function value's minimum remains in an infeasible region because the effect of the penalty value is not strong enough near the constraint bound to adequately change the shape of the curve. Note that this particular example uses Rastrigin's function (multi-modal) with a non-linear penalty method per Equation (5-7). This same behavior, to one degree or another, was observed during program development regardless of penalty method employed. Of course, the magnitude of the effect is influenced by both penalty method and multiplier, but whatever the selection, it cannot be guaranteed prior to an optimization run that such a situation will not exist.

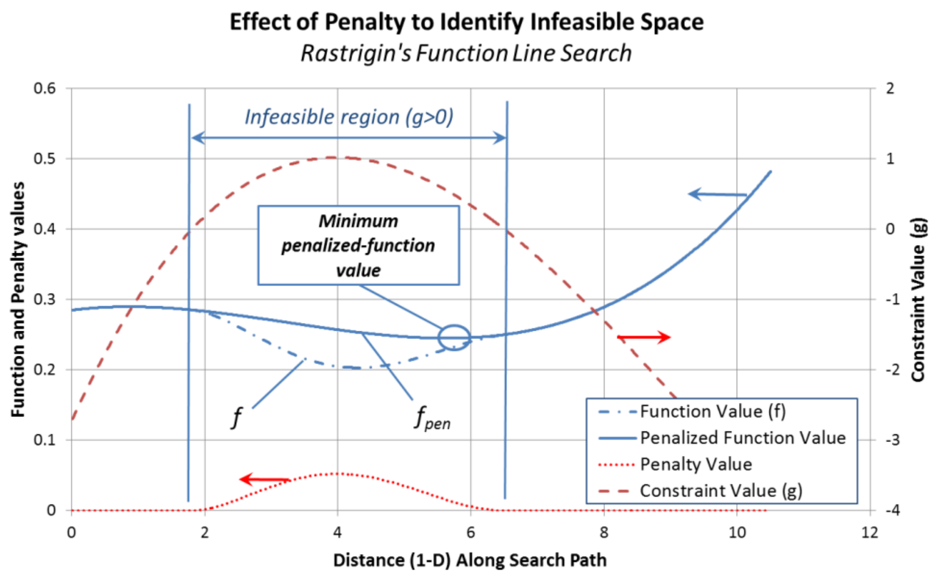


Figure 5-6 Identification of constraint boundaries

Therefore, in addition to application of a penalty to the function value, a numerical technique is also employed to help guarantee that the minima resulting from the 1-D line search is within feasible space. If the proposed minimum is found to be infeasible, then polynomial curves are fit to both penalized function and constraint values along the length of the 1-D search path. Predicted values of both constraint and penalized function are both surveyed at a high density (on the order of 1000 points along the 1-D curve) and the minimum (predicted) function value with feasibility (g<=0) is selected as the minimum for that 1-D line search. (Note that both function and constraint value evaluations during this process are 'predicted' values based upon the curve fits and do not represent additional function evaluations in order to minimize computing resource requirements in consideration for EBBF's.) This process is included in the flowchart of the One-Dimensional Line Search subroutine (Function: OneDSearch) in Appendix B. The effectiveness of this technique, in addition to selection of the appropriate penalty function (Equations (5-7 vs. (5-8) and penalty multiplier (α_p) is discussed in a later section.

Selection of Feasible Start Points

The definition of the global starting points is, in general, conducted by first establishing a user-defined number of start points which span the design space for each of the design variables between the specified lower- and upper-bounds. Each of these candidate points is then evaluated to filter out any points which are not in feasible space based design-variable constraints that may exist (side-bounds and other). The resulting matrix of starting points (X0's) is then submitted to the appropriate search routine. Determination of the Feasible Start Points is one of the key subroutines charted in Appendix B

It is recognized that one common way to conduct computer experiments is to utilize a Latin Hypercube Sampling (LHS) technique to optimize the distribution of the sampling points through the design space [42]. However, since LHS is within itself an optimization technique, some small differences in results may exist from run-to-run. Given that this report focuses heavily upon comparison among the optimization techniques, it is desired that the starting points (X0) be both identical among each run and (as much as possible) evenly distributed across the design space. Therefore, a relatively large number of start points using a haltonset distribution is utilized here. The MATLAB ® haltonset function [33] is a 'pseudo' randomization of the input variables across the design space, but in a repeatable manner whereby given the same design space definition and quantity of points, the definition of the set of candidate points will be identical from run-to-run. Note that for the MATLAB ® haltonset routine, the first point is always the origin. This gives an unfair advantage to algorithms attempting to solve symmetric test functions (e.g.; De Jong's and Rastrigin's). Therefore, for the purposes of this report and the comparisons being made, this first value of haltonset is omitted programmatically for all runs in order to remove potential biases.

SQP Method from Global Sampling Points

The implementation of the Direct Search methodology of SQP is relatively straightforward given the use of the MATLAB ® language. That is, the MATLAB ® function 'fmincon' is utilized with the 'SQP' modifier as a means to implement the SQP search process. Unlike the method of

Steepest Feasible Descent, the Direct Search Method of SQP is a constrained search methodology. That is, the effects of the constraints are handled directly by the algorithm. However, it was determined through trials (data not presented here) that the fmincon algorithm was more effective at preventing infeasible results by including the penalty function (described in the Steepest Feasible Descent section above) as part of the Objective Function. That is, the same penalty function is included in the Objective Function for both the Steepest Feasible Descent and SQP algorithms. In this way, the SQP method gave the most robust results. The effectiveness of this approach is demonstrated in a later section.

Combination of Steepest Feasible Descent and SQP Methods

The combination of Steepest Feasible Descent as an initial (coarse) search methodology followed by the SQP (Direct) method is theorized to yield a better result in terms of fewest number of function evaluations overall in order to achieve the most accurate answer for multimodal surfaces. Implementation of the process is straightforward with respect to the two (2) major algorithm components described above. Given the feasible start points, the Steepest Feasible Descent search is conducted with a user-specified maximum number of ‘jumps’ (2 or 3 jumps recommended). At the conclusion of this routine, the best (minimum Objective Function) feasible result is identified and then used as a single start point for the SQP search. As described above, both the Steepest Feasible Descent and SQP methodologies incorporate the same penalty modifier as part of the Objective Function for the most robust constraint handling solution. The effectiveness of this approach is demonstrated in a later section.

Use of a ‘Single-Objective Optimization’ Tool for ‘Multi-Objective Optimization’

The algorithms described here (both Steepest Feasible Descent and SQP) are Single-Objective Optimization tools [20]. That is, they are appropriate for optimization of a single objective. They can, however, be utilized for Multi-Objective Optimization by combining the elements of various ‘individual’ single objectives into a single ‘combination’ objective function. For example, if for the parallel beam problem it was desired to stabilize harmonic tip deflection, one might consider multiple aspects such as minimizing the tip deflection over a range of frequencies as well as minimizing the range of those deflection results. One method of 'consolidating' these objectives is to create one objective function by computing a discrete integration of the results over the frequency range as a metric of 'tip deflection' and another result which is the simple range among the results. By minimizing the sum of both results, a pseudo Multi-Objective analysis is conducted; however given that a single objective function results, the problem can be solved using Single-Objective Optimization tools such as described above.

As described by Vanderplaats [20] it may also be necessary to consider weighting factors for the components of this multi-faceted optimization function; especially if the magnitudes of the components are significantly different. If, in another example, the goal were to minimize both deflection and stress then it would be most effective to normalize each to a similar magnitude value. Otherwise, the optimization algorithm would disproportionately 'focus' its efforts to minimize the larger magnitude response at the expense of the other. Normalization of constraint functions should also be considered for similar reasons.

Given that the test functions are Single-Objective Optimizations, these considerations do not apply. However, they do apply to the parallel beam problem. This technique is discussed further in the development of both Objective and Constraint functions for the parallel beam problem in a later section.

Algorithm Confirmation

One-Dimensional Line Search Effectiveness

The effectiveness of the 1-D line search algorithm is evaluated separately for unconstrained and constrained searches using the previously described test functions. Discussions for each case occur separately below.

Performance as an Unconstrained Search against Test Functions

The effectiveness of the 1-D line search strategy is assessed using the four (4) test functions described previously. Specifically, for each of the test functions data is obtained from the programmatic 1-D search for a given search vector. This data is plotted and analyzed, and compared to the programmatic results. In this way, the effectiveness of the strategy is assessed and a statement regarding effectiveness of the method can be made. Each of the four (4) test functions are used in this assessment but, given the simplicity of the 1-D polynomial for both De Jong's and Rosenbrock's functions, only one search vector is discussed for each. Because the other two functions are multimodal and therefore more challenging to the search routine, multiple search vectors are considered for both Rastrigin's and Schwefel's functions in order to illustrate various potential scenarios. A total of seven (7) 1-D search path evaluations are presented here for the unconstrained case. For each test, the following were established:

- maximum number of points along Search Vector = 21
- minimum R^2 =0.9995
- abs(max error) = 5%
- initial (minimum) polynomial order = 3
- maximum polynomial order = 4

As might be expected, acceptance criteria were achieved quickly for the 1-D line search of De Jong's function, and with fewer than the maximum allowable number of points. Figure 5-7 illustrates this result.

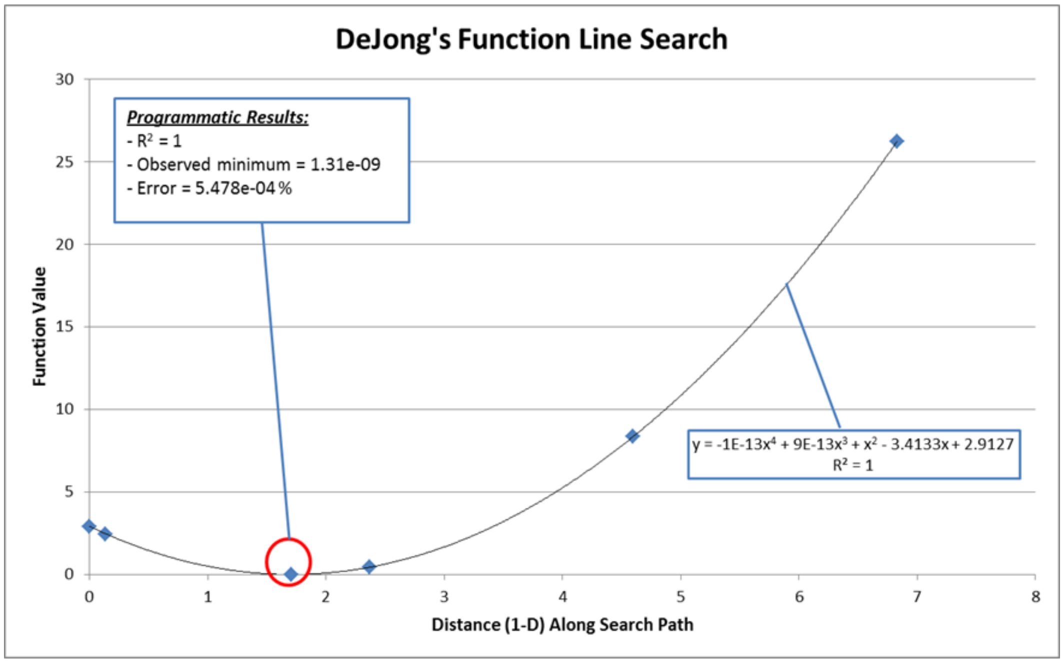


Figure 5-7 1-D line search, De Jong's function

As shown, only 6 total points were used by the program to establish a minimum with <0.0006% error between predicted and observed values. The regression coefficient was also found to be the same (R^2=1.0) for both programmatic and external evaluations of the polynomial. These results are acceptable.

Results from the Rosenbrock’s function are similar to that of De Jong’s due to the unimodal nature of the curve. As shown in Figure 5-8 only 6 total points were (again) used by the program to identify the minimum with near-zero error between predicted and observed values. The regression coefficient was also found to be acceptable for both programmatic and external evaluation of the polynomial. These results are also acceptable.

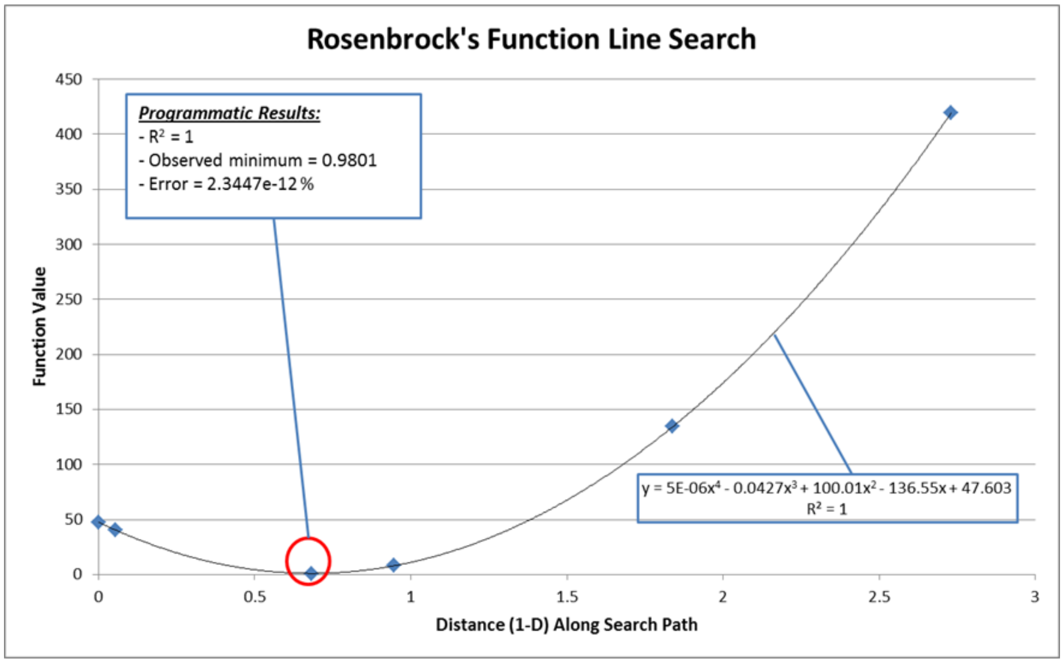


Figure 5-8 1-D line search, Rosenbrock's function

Rastrigin's function is multimodal in nature and, because of that, three (3) test cases are presented. As shown in Figure 5-9, the first case (search vector) resulted in an identified minimum that is roughly mid-span between X0 (x=0) and the nearest input constraint (x≅ 3.6). The full allowable quantity of points (n=21) were utilized during this line search and, as shown in the figure, the program identified an optimum at x≅1.36 and clustered points in that area. When all data are considered as a whole, the resulting correlation is poor (R^2=0.69). However, as the program removed data that are far from the identified minimum (after reaching the maximum number of points) a much improved curve fit emerged. As expected, the correlation coefficient (R^2=0.9953) agrees between the refined programmatic analysis and external evaluation and an error value of -3.075% exists between the predicted minimum from the polynomial evaluation and observed results. Although this error meets the acceptance criteria, the value of R^2 does not. The programmatic use of these two (2) metrics for acceptance of the curve fit is only used to stop the line search prior to reaching the maximum number of points. In a case such as this, the criterion was not met but the maximum number of points was reached. Programmatically, the response in this case is to accept the smallest of the (solved) function results and note the values of both R^2 and error in the output. The test case here is considered successful in that the data along the search vector are analyzed with equivalent results, both programmatically and by external evaluation. If such a case were to happen during an actual optimization run (acceptance criteria not met) then use of the smallest (solved) function result rather than polynomial prediction helps to ensure accuracy of the results. The use of the polynomial approximation method during the 1-D search however, adds to the 'appropriate selection' of additional points by identifying potential areas of minimum.

It is recognized also that due to the multimodality of the function, other minimums may exist and may be even be more attractive than the one which is identified. This is particularly true when the threshold for maximum number of points in the search is low. However, even though the potential exists, the program will identify a minimum that is 'sufficient' for the purposes of this coarse search without expending an extreme amount of computing resources which is acceptable.

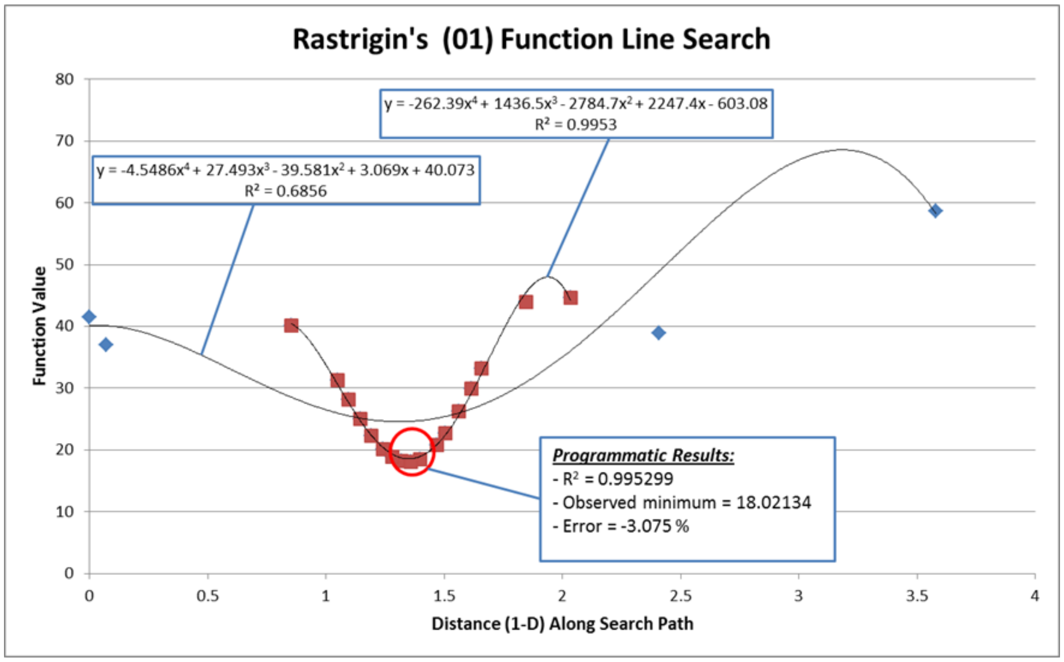


Figure 5-9 1-D line search, Rastrigin's function (search vector #1)

A second search vector for Rastrigin's is illustrated in Figure 5-10. Note that the only difference between this line search and the one for Rastrigin's above is the selection of a different search vector starting from a different X0. This time, the minimum was identified much closer to the X0 position; such that the initial descending 'vector' at X0 contributes to the identification of the minimum. That is, no 'ridge' separates X0 from the minimum. As with the first vector of Rastrigin's, the full quantity of points (n=21) were utilized during this line search and in the process the program identified an optimum at $x \cong 0.43$; clustering points in that area. When all data are considered as a whole for this search, the resulting correlation is less than deemed acceptable (R^2=0.92). However, as the program removed data that are far from the identified minimum a much improved curve fit (R^2=0.9998) emerged. An error value of -0.41% exists between the predicted minimum from the polynomial evaluation and observed results. These results are deemed to be acceptable.

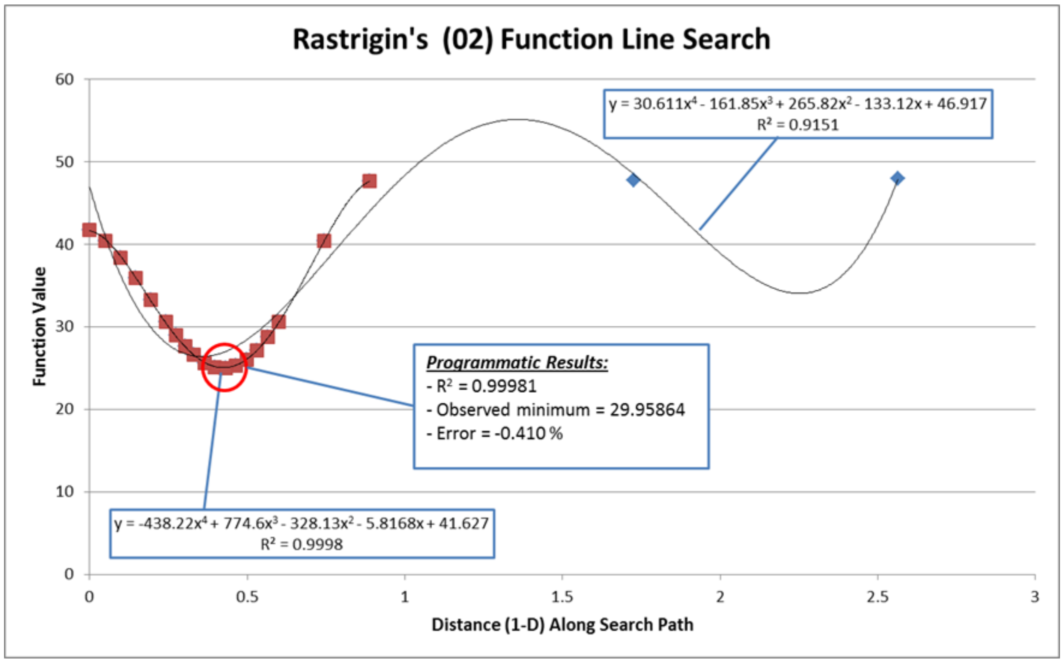


Figure 5-10 1-D line search, Rastrigin's function (search vector #2)

A third search vector for Rastrigin's (again from a different X0) is illustrated in Figure 5-11. This time, the minimum was identified much closer to the maximum (constraint bound) end of the line search. In fact, the curve appears to be highly multi-modal and the region evaluated by the program (around the predicted minimum) is very narrow. The full quantity of points (n=21) were utilized during this line search which (as a whole) result in a poor correlation coefficient (R^2=0.72). However, the programmatically refined fit yielded a much improved curve fit (R^2=0.99998) and an error value of -0.08%. This test case is another example where the identified minimum may not be the absolute minimum along the search vector. It does, however, represent a 'sufficient' minimum within the constraint a predefined maximum number of function evaluations. Therefore, these results are also deemed to be acceptable.

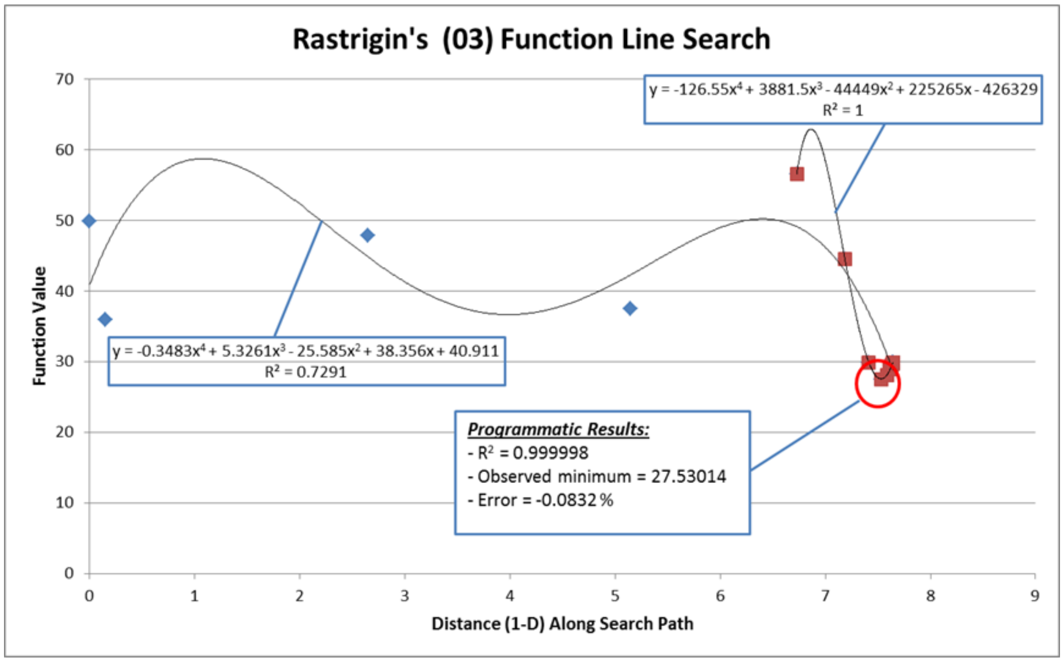


Figure 5-11 1-D line search, Rastrigin's function (search vector #3)

Schwefel's function is also multimodal in nature and, because of that, two (2) test cases are presented. As with the test cases for Rastrigin's above, these test cases for Schwefel differ in the selection of X0 and the direction of search vector in order to illustrate variety in the types of situations potentially encountered. The first of these, presented in Figure 5-12, appears to be 'simpler' than the Rastrigin's searches discussed above. All 21 points were utilized in the search, but the initial correlation with all of the data is marginal (R^2=0.96). As such, it does not meet the acceptance criteria. Because of that, the program entered the 'refinement' subroutine and the resulting curve fit was improved for both correlation coefficient (R^2=0.99999) and error (-0.009%). As shown, both programmatic and external analysis of the refined data agrees regarding the correlation coefficient. These results are acceptable.

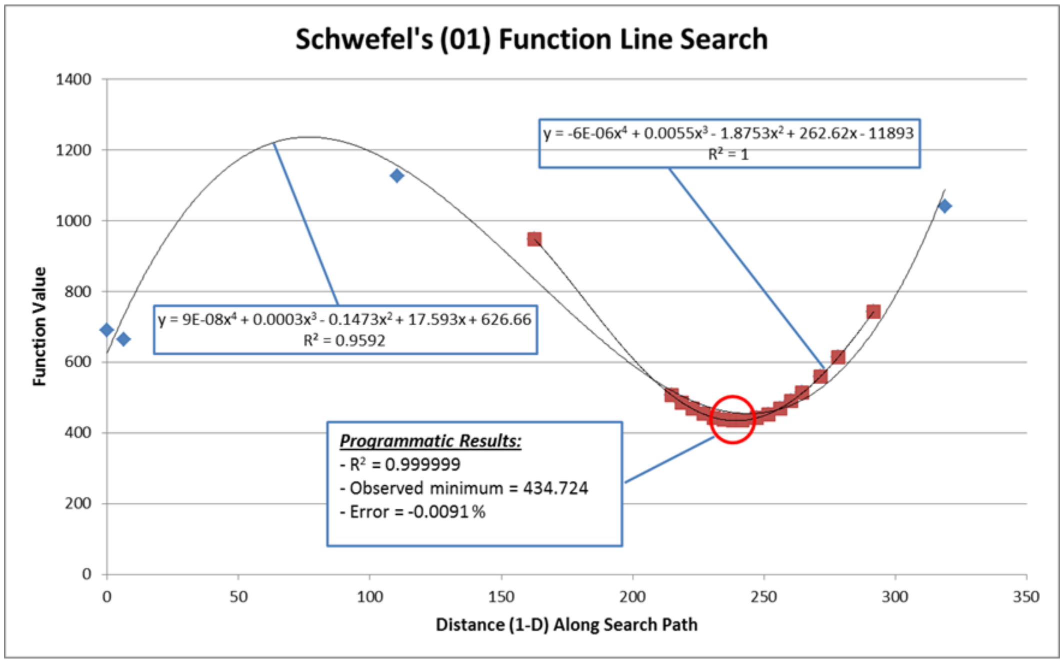


Figure 5-12 1-D line search, Schwefel's function (search vector #1)

The second search vector for Schwefel's function also resulted in a multi-modal 1D line search and is presented in Figure 5-13. The maximum of 21 points were again utilized in the search, resulting in effectively no correlation to the data set as a whole (R^2=0.194). The refinement routine improved the results (R^2=0.613, error = -12.28%) but failed to meet the acceptability criteria. However, both programmatic and external analysis of the refined data set result in the same correlation coefficient and, to that end, the test of the algorithm is deemed a success.

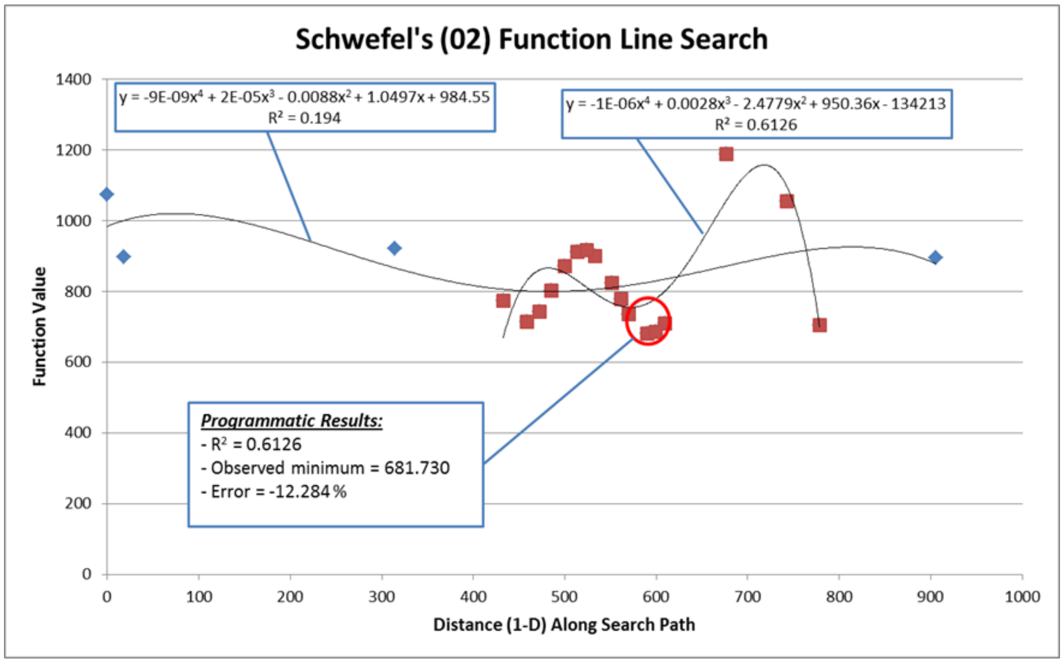


Figure 5-13 1-D line search, Schwefel's function (search vector #2)

It is recognized that significant improvement in the curve fit could be made by refining the data set even more (restricting data to a smaller region bounding the potential minimum). In addition, the data infers that a lower minimum may exist between the last two evaluated points (near the constraint bound – between x=800 and x=900). Such fidelity in the 1-D model, however, comes at a high price in terms of computational cost for EBBF where each function solution may take several minutes or hours to achieve. For this evaluation, given that acceptability criteria were not met, the identified minima from the search is the minimum (actual) function solve result rather than a predicted minimum from the polynomial curve. As such, a minimum is returned even though it may not be the absolute minimum that could be identified with further (more costly) investigation. The acceptability of whether or not such a minimum represents a 'sufficient' minimum in terms of the goal of identifying an area of regional minimum through the steepest feasible descent method with an acceptably few number of function solves is another question. This is evaluated via performance testing (to identify the theoretical minima of the test functions) in a later section.

Performance as a Constrained Search against Test Functions

In order to demonstrate the effectiveness of the 1-D algorithm for constrained searches, evaluate and choose between the two penalty function methods of Eq. (5-7) vs. (5-8) and ‘size’ the penalty multiplier (α_p), a test was conducted using De Jong’s test function. De Jong’s was selected for this effort because of the simplicity and symmetric nature of the function. The preferred method resulting from this test will be confirmed and discussed later using the other test functions as well.

As shown in Figure 5-14 below, a simple ‘zone of infeasibility’ was established for the test. This zone is a circle of defined radius and centered at the origin and shaded for clarity in Figure 5-14. Given that the minimum of De Jong’s function is at the origin, this forces the routine to find a minimum value which exists in feasible space, ‘outside’ of the infeasible area. In actuality, any location directly on the circular boundary functions as a correct minimum value for this test case. The test, then, becomes how well the given method identifies the boundary. Major parameters for the test cases are as follows.

- Steep Descent from each of 75 starting points (X0) using Haltonset
- Limited to 2 jumps per convergence path
- Non-feasible range = Radius 2.0, centered at (0,0)
- Penalty function per Equation (5-7 vs. (5-8 - squared vs. linear constraint values)
- Penalty Multiplier range 0.10 ~ 10.0

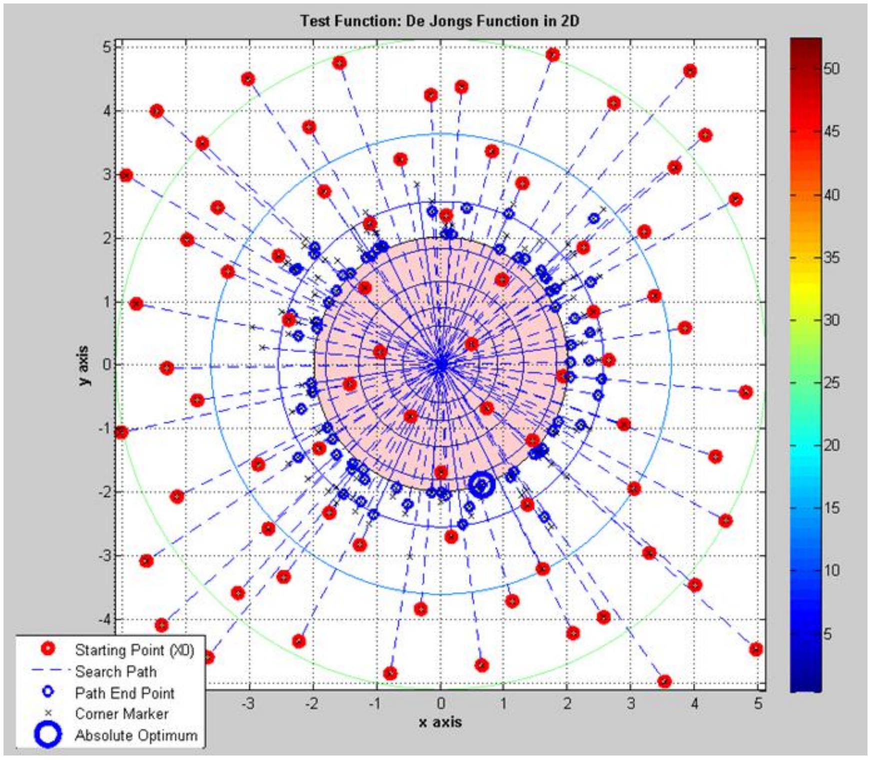


Figure 5-14 Example of penalty function test result, De Jong's function

The goal of this test then is to compare the two penalty function methods and also vary the value of the penalty multiplier to determine if:

1. the penalty function allows a minimum to be found within the infeasible boundary
2. how close to the boundary are the identified minimums located

In an 'ideal' result for this test case, the identified minimum values would lie exactly on the constraint boundary; in this case, at a radius of 2.0 from the origin. With an infinite number of 'jumps', the Steepest Feasible Descent method would likely converge to optimums directly at this boundary regardless of the penalty method employed.. By limiting the maximum number of jumps to two (2) however, the effectiveness of the penalty method is better challenged

To evaluate the effectiveness of a given algorithm, a 'radius' is computed from the origin of the design space to the resulting end point from each starting point's path (identified minimum from each start point). A 'maximum' and 'minimum' radius value is reported from among all of the convergence paths (all of the X0's) for each test case. In this way, the range of end point locations (relative to the origin) and overall effectiveness of the tested technique are easily compared. A perfect result of course would be that all end points would be located on the boundary (Radius = 2.0) and with no difference between ‘maximum’ and ‘minimum’ location. Results of the penalty algorithm test are provided in Table 5-3 below and illustrated in Figure 5-15.

Table 5-3 Comparison of penalty methods, radius of path optimums from center

Penalty Multiplier Value (αp)	‘Squared’ Constraint Value Eq. (5-7)		‘Linear’ Constraint Value Eq. (5-8)	
	Minimum Radius	Maximum Radius	Minimum Radius	Maximum Radius
0.001	2.00067	2.38871	2.00067	2.39084
0.003	2.00067	2.38861	2.00067	2.38967
0.005	2.00067	2.38683	2.00067	2.38791
0.010	2.00067	2.25501	2.00067	2.27474
0.025	2.00007	2.25501	2.00067	2.25501
0.050	2.00007	2.25501	2.00005	2.25501
0.100	2.00005	2.25501	2.00005	2.25501
0.250	2.00005	2.25501	2.00005	2.25501
0.500	2.00005	2.34921	2.00005	2.41282
2.500	2.00086	3.41399	2.00078	2.49872
5.000	2.00086	3.40547	2.00086	2.61456
10.000	2.00086	3.40159	2.00086	2.68848

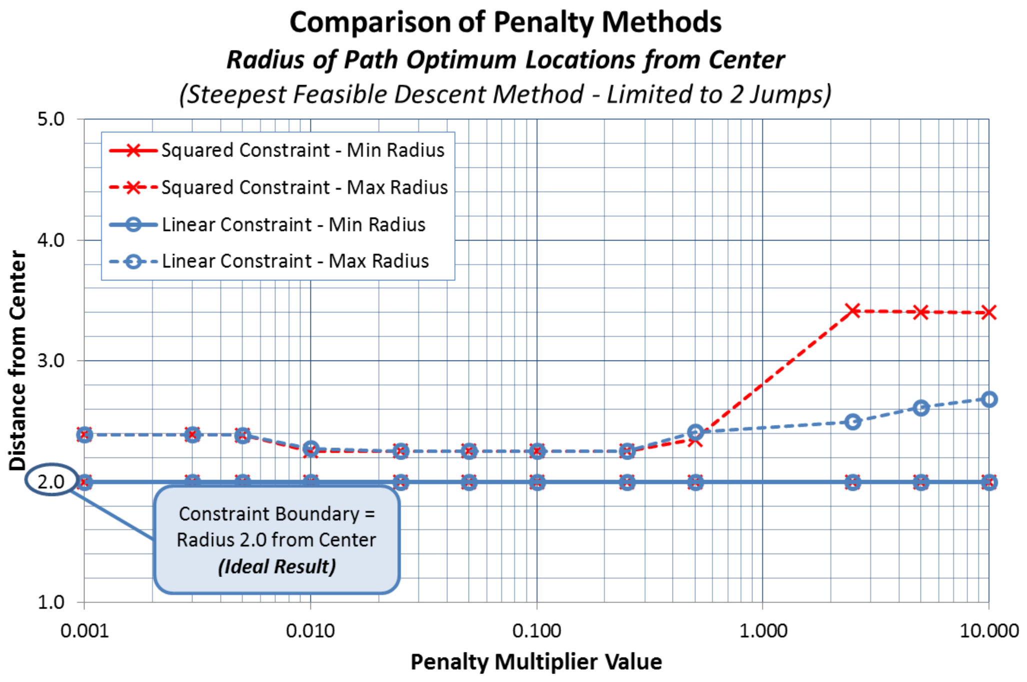


Figure 5-15 Comparison of penalty methods - De Jong's test function

As shown, none of the test cases resulted in an identified optimum that was inside the infeasible zone (radius < 2.0). In fact, minimum radius values for both penalty methods resulted in very nearly ideal results for all multiplier values tested. Given this result, a selection of any of the penalty multiplier values within the tested range would be an acceptable choice.

The maximum radius results are different however, with a notable optimal range observable for penalty multiplier values less than 0.50. If the penalty multiplier value is too low, it diminishes the effectiveness of the penalty equation somewhat as can be seen in Equations (5-7) and (5-8) and illustrated for multiplier values below 0.010. At the other extreme, large multiplier values (greater than 1.0) result in an increase in error for the 'max radius', particularly for the constraint values which are squared as part of the penalty equation Eq. (5-7).

Given these observations, together with the potential benefit to curve smoothness of the 'squared constraint' penalty method Eq. (5-7); a decision is made to utilize the 'squared constraint' penalty method for the parallel beam problem. A penalty multiplier value of 0.250 is selected as an acceptable value in the observed 'optimal region' (Figure 5-15); large enough to have a significant effect upon the applied penalty, but not so large as to cause increased error in the results.

Therefore, by applying Equation (5-7) to the Objective Function with a Penalty Multiplier of 0.250, the unconstrained method of Steepest Feasible Descent can be effectively utilized as a tool for constrained optimization. This is confirmed in a later section of this report where the method is challenged against constrained regions for each of the four (4) test surfaces.

Performance Test Parameter Definition – Overall Effectiveness and Efficiency

The optimization program was evaluated for accuracy and efficiency against all four (4) test functions using each of the three (3) candidate methodologies and considering both unconstrained and constrained scenarios. For each test, the design space was defined as 2-DOF with a range for each design variable in keeping with the standard test function 'specification' described previously. [40] Key results for the unconstrained tests are accuracy and efficiency of the overall solution by means of four (4) factors. These factors are used then to compare and contrast the candidate algorithms (Steepest Feasible Descent only, SQP only and the Combined approach). The key factors are:

- Coordinate location of the identified best solution vs. the theoretical solution.
- Function value of the identified best solution vs. the theoretical solution.
- Number of function evaluations conducted.
- Number of solutions found within a target range bounding the coordinate location of the theoretical solution. Specifically this 'Results Region Bound' is defined as ± 1.0% either side of the coordinate location as a percent of the maximum value of that coordinates design space. Specific values for this 'Results Region Bound' are provided in the table below for each test function.

Specific input parameters common to unconstrained tests for all three (3) methodologies are also listed in Table 5-4.

Table 5-4 Program parameters - test function evaluations

Number of Start Points	75
Start Point Definition	Haltonset
Max Jumps (Steep Descent)	2
Max Jumps (SQP method)	250
Min 1D Polynomial order	3
Max 1D Polynomial order	4
Max no. of 1D points	15
Min R2 allowable	0.9995
1D error allowable	5%
Minimum Des. Var. step size	0.001
Penalty Method	‘Squared Constraint Method’ - Eq. (5-7)
Penalty multiplier value	0.250

Design Space Definition		
Test Function	Design Space Range	Results Region Bounds
De Jong’s	x= ± 5.120, y= ± 5.120	x= ± 0.0512, y= ± 0.0512
Rosenbrock’s	x= ± 2.048, y= ± 2.048	x= ± 0.0205, y= ± 0.0205
Rastrigin’s	x= ± 5.120, y= ± 5.120	x= ± 0.0512, y= ± 0.0512
Schwefel’s	x= ± 500, y= ± 500	x= ± 5.00, y= ± 5.00

Starting points for each of the test functions are depicted graphically in the following figures. Although each is selected via haltonset, the scaling is appropriate to the varying design spaces.

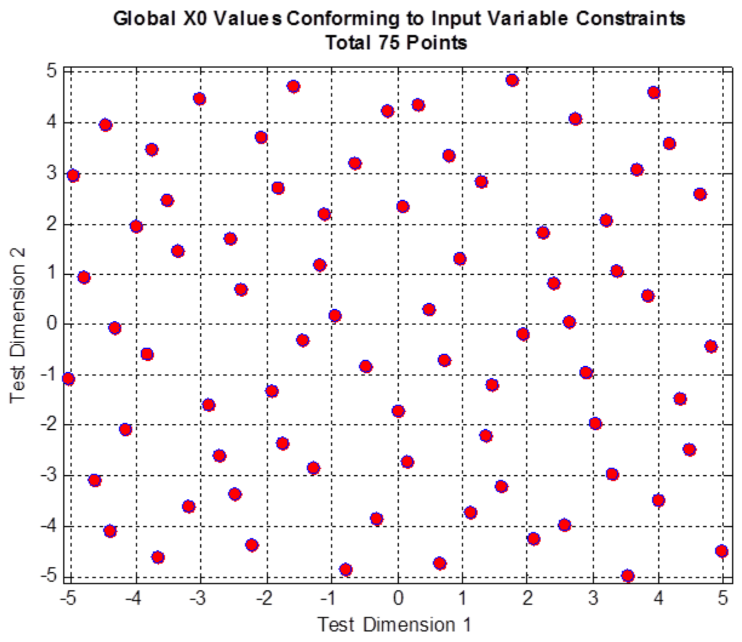


Figure 5-16 Starting point locations - De Jong's test function

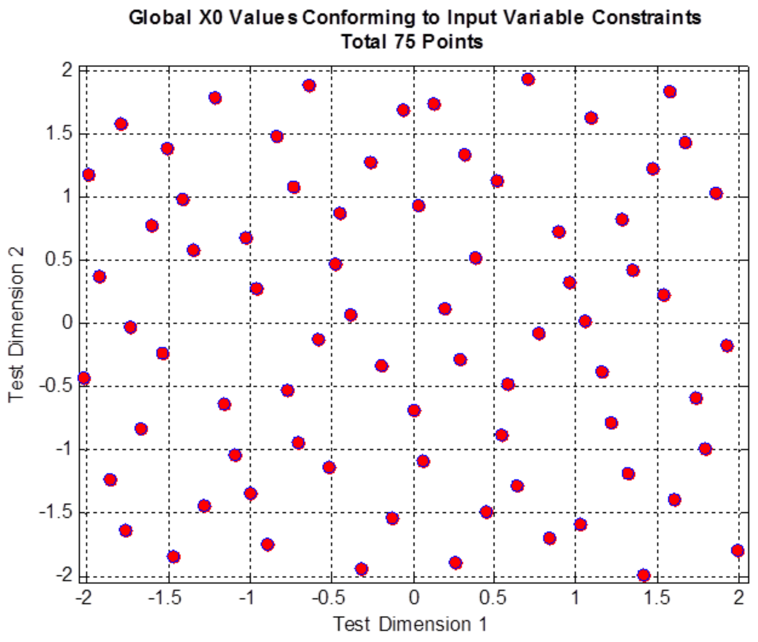


Figure 5-17 Starting Point locations Rosenbrock's test function

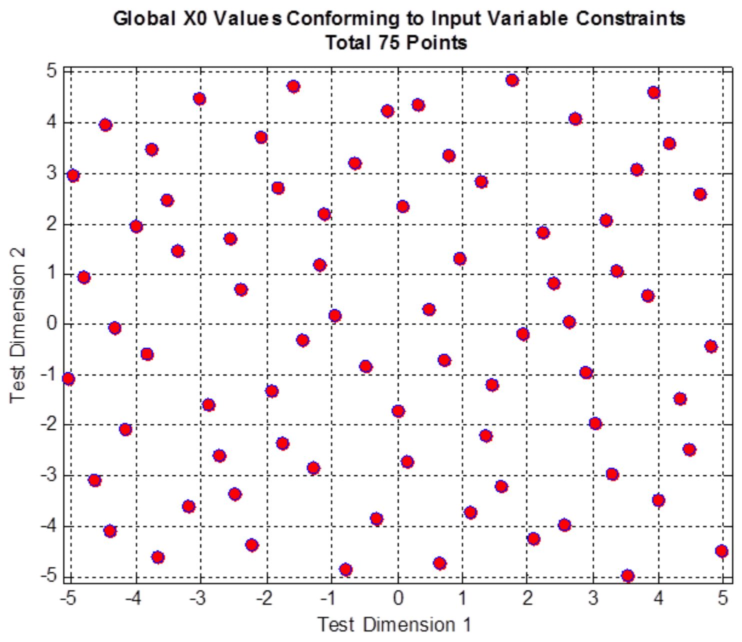


Figure 5-18 Starting point locations - Rastrigin's test function

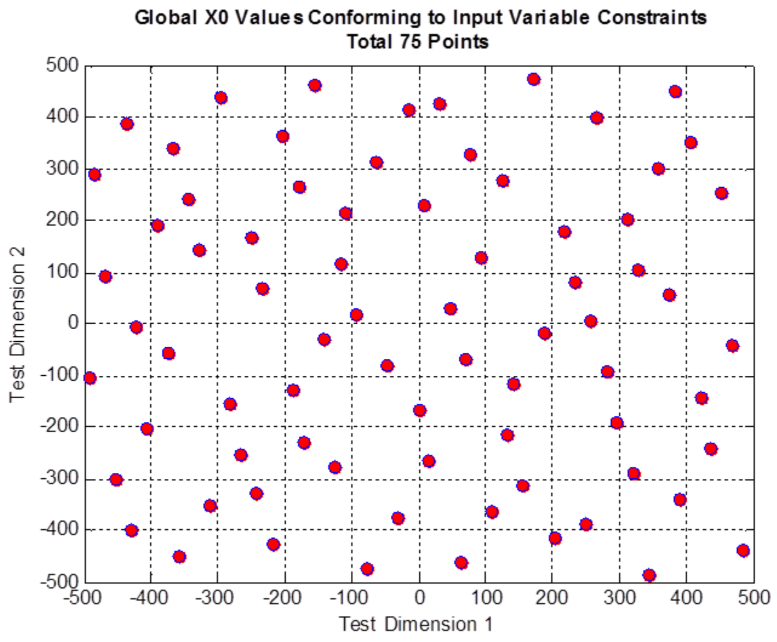


Figure 5-19 Starting point locations - Schwefel's test function

Input parameters for the constrained tests (including starting points) are identical to those of the unconstrained tests above with the exception that a circular region of infeasibility is defined around the theoretical optimum solution for each test function. In this way, the program is challenged to find the best feasible solution lying outside of the constrained area. This constraint, however, is only applied 'after' the initial selection of starting points. That is, although the constraint applies to the location of the point, it is not applied during the initial selection of the starting points; rather only to the position of the resultant points. In this way, the program is challenged with finding feasible results for starting points are defined both in feasible and infeasible space. Successful results for the constrained evaluations are that solutions are identified in feasible space (only). Constraint boundary definitions for the four (4) test functions are described in the following table.

Table 5-5 Constraint region parameters - constrained test function evaluations

Test Function	Constraint Region Definition	
	Center	Radius
De Jong's	(0,0)	2.0
Rosenbrock's	(1,1)	0.5
Rastrigin's	(0,0)	2.0
Schwefel's	(420,420)	100

Results for both unconstrained and constrained tests for each methodology (Steepest Feasible Descent only, SQP Method only and Combined) are provided and discussed in the sections below using each of the four (4) test functions.

Method of Steepest Feasible Descent from Global Sampling Points

Performance as an Unconstrained Search against Test Functions

Tests were conducted against each test function for the Method of Steepest Feasible Descent from the Global Start Points without any further optimization. That is, for this test the optimum solution was identified from the defined maximum number of jumps for each start point and a 'global' optimum identified from among those results. Graphical results are depicted for each test function in the following figures and illustrate the convergence paths followed for each global start point as well as the location of the best (lowest function result) overall value. Numerical results are tabulated below as well.

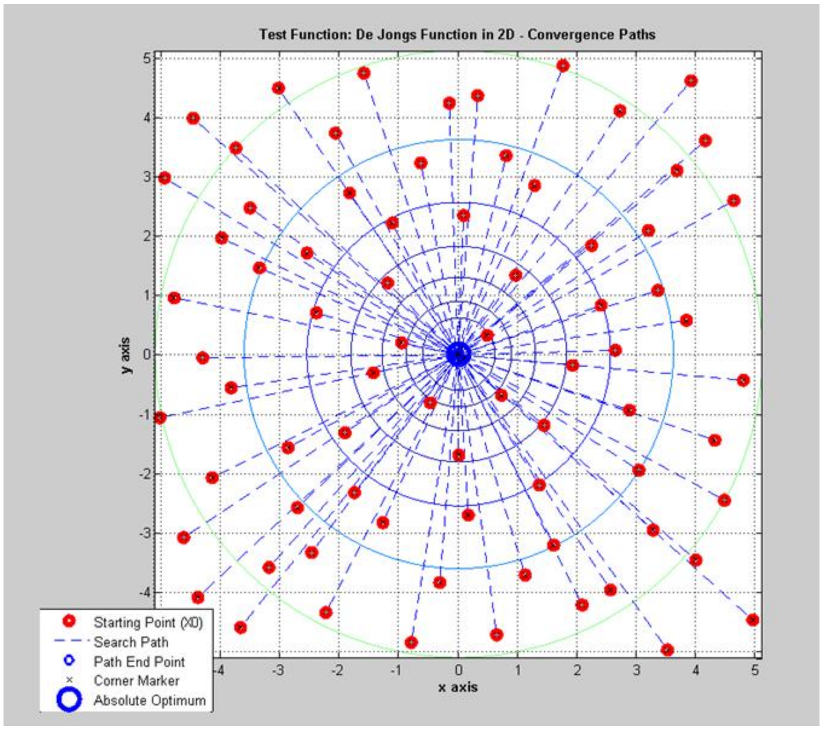


Figure 5-20 Test result, steepest feasible descent - De Jong's function

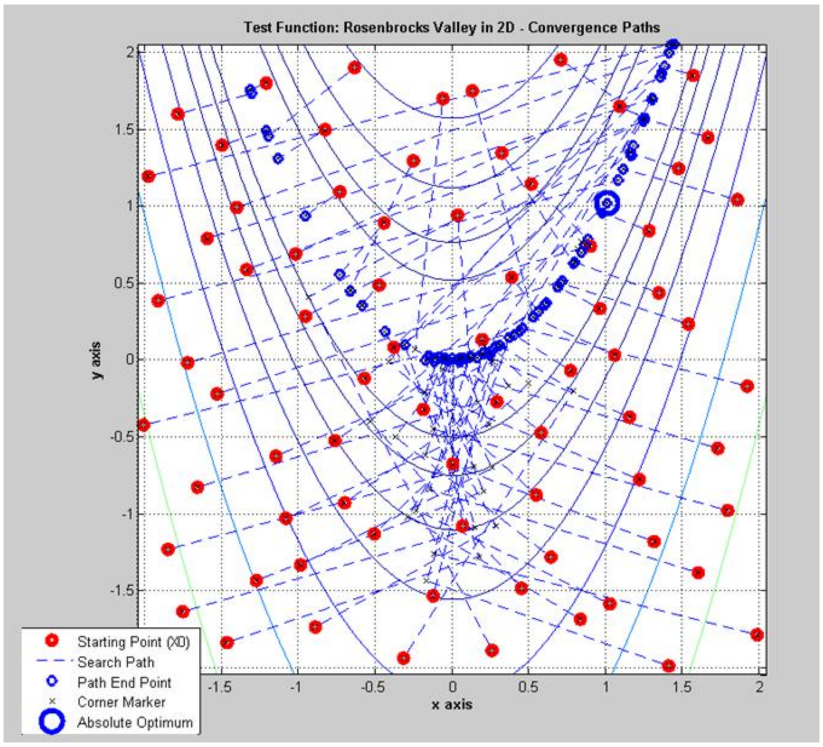


Figure 5-21 Test result, steepest feasible descent - Rosenbrock's function

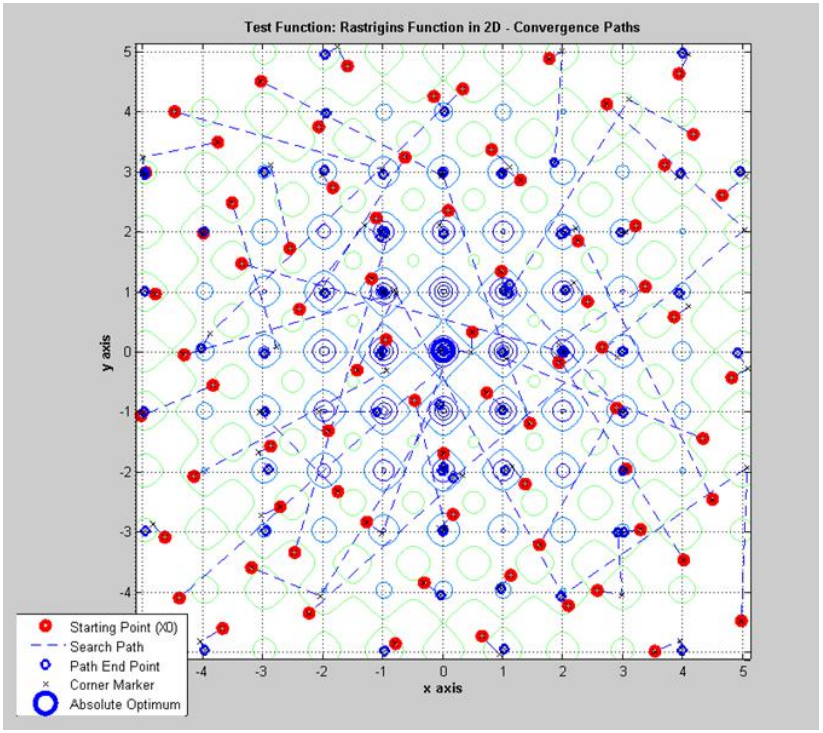


Figure 5-22 Test result, steepest feasible descent - Rastrigin's function

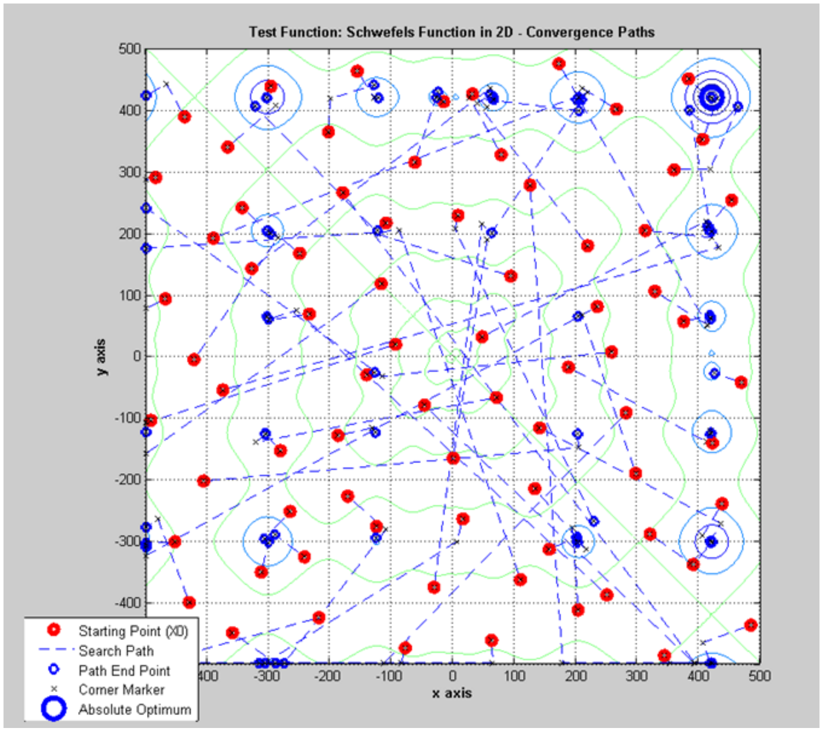


Figure 5-23 Test result, steepest feasible descent - Schwefel's function

Table 5-6 Test function result - method of steepest feasible descent

	Test Function			
	De Jong's	Rosenbrock's	Rastrigin's	Schwefel's
Coordinate Location of Overall Minimum				
Theoretical	(0,0)	(1,1)	(0,0)	(420.9687,420.9687)
Test Result	(4.2e-4, -4.5e-4)	(1.0067, 1.0135)	(-0.0070, 0.0117)	(421.109, 419.456)
Function Value of Overall Minimum				
Theoretical	0	0	0	0
Test Result	0	0	4.6e-4	2.0e-4
Number of Function Evaluations				
Test Result	1131	1190	2233	1869
Number of Solutions within 'Results Region				
Test Result	75	1	2	2

The data indicates that the Method of Steepest Feasible Descent from Global Start Points succeeded in finding optimums very near to the theoretical optimum for all four (4) test functions. All of the start points for the De Jong's function resulted in optimums within the ± 1.0% 'target range'. One (1) for Rosenbrock's and two (2) each for Rastrigin's and Schwefel's also fell within the target range. Another interesting observation is that for the relatively 'simple' test functions of De Jong and Rosenbrock, similar numbers of function values were needed (1131 and 1190). The more complicated Schwefel's function required more (1869) whereas the test function with the highest degree of multimodality (Rastrigin's) required the most function evaluations (2233).

These results support that the method of Steepest Feasible Descent does find at least one (1) optimum solution within the toleranced region for each of the test surfaces. A fuller benefit of this test is realized by the comparison of these results to those of the other two (2) methods (Direct Search only and the Combined method) which is presented later in this chapter.

Performance as a Constrained Search against Test Functions

Results for the constrained searches using the Method of Steepest Feasible Descent (only) from Global Search Points are provided in the following plots. As mentioned previously, the purpose of this test is to confirm (or refute) that the algorithm is capable of identifying optimums only within feasible space, regardless of whether the start point resides in feasible space or not. Note that, like the previous plots which include constrained regions, the area of non-feasibility is shaded.

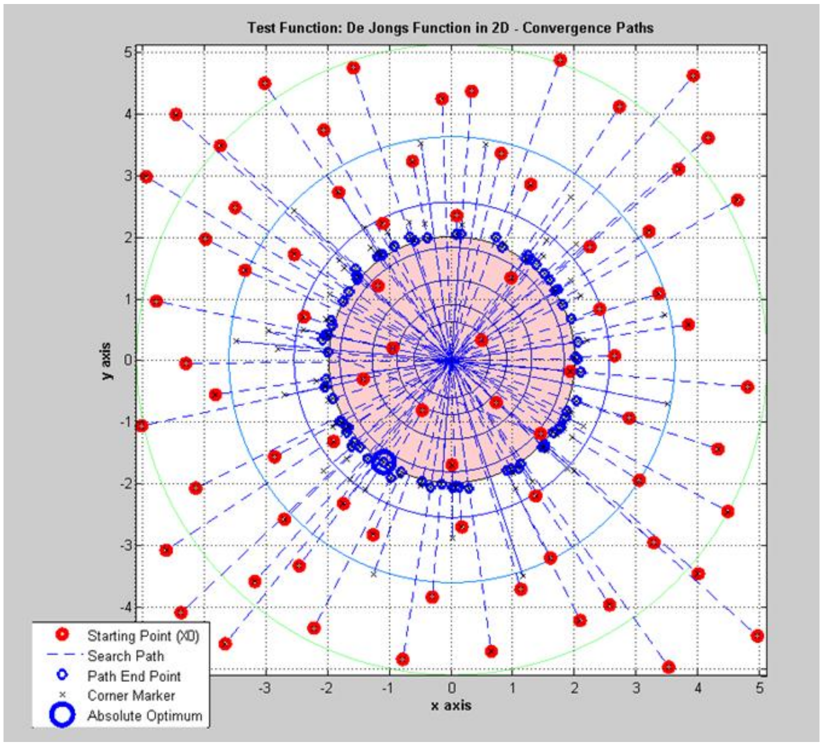


Figure 5-24 Test result, constrained steepest feasible descent - De Jong's function

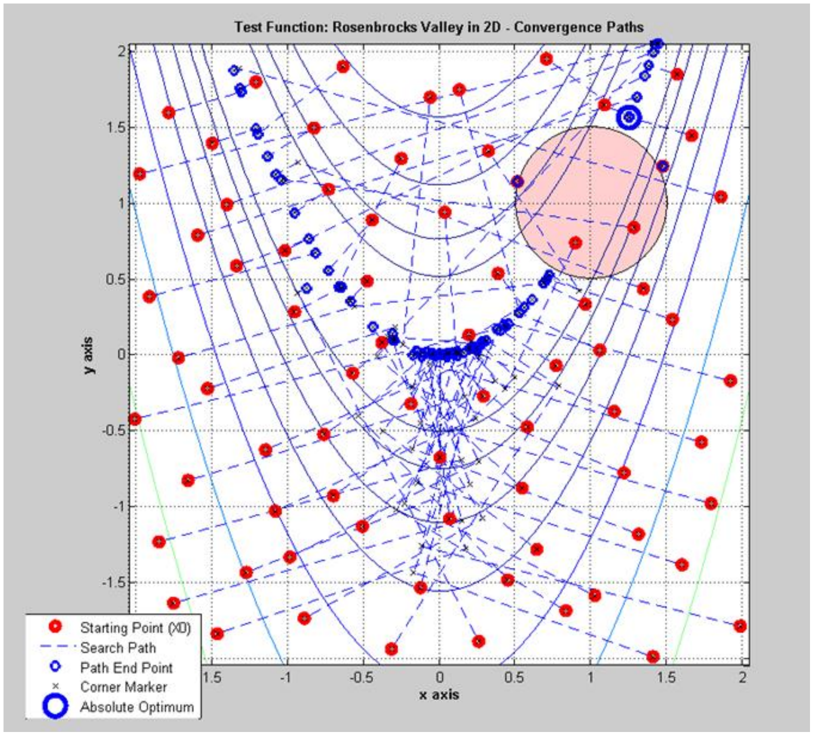


Figure 5-25 Test result, constrained steepest feasible descent - Rosenbrock's function

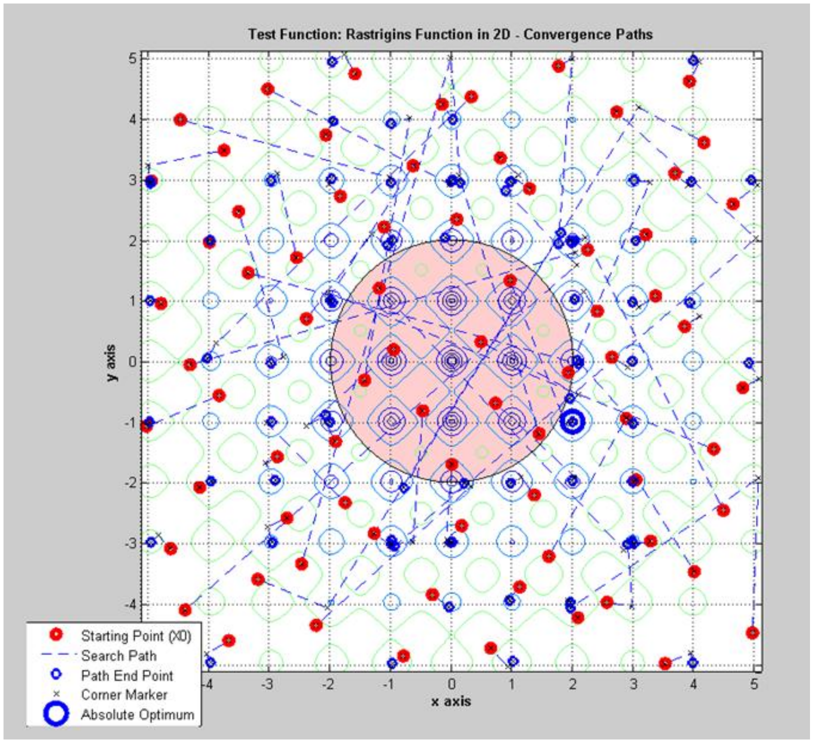


Figure 5-26 Test result, constrained steepest feasible descent - Rastrigin's function

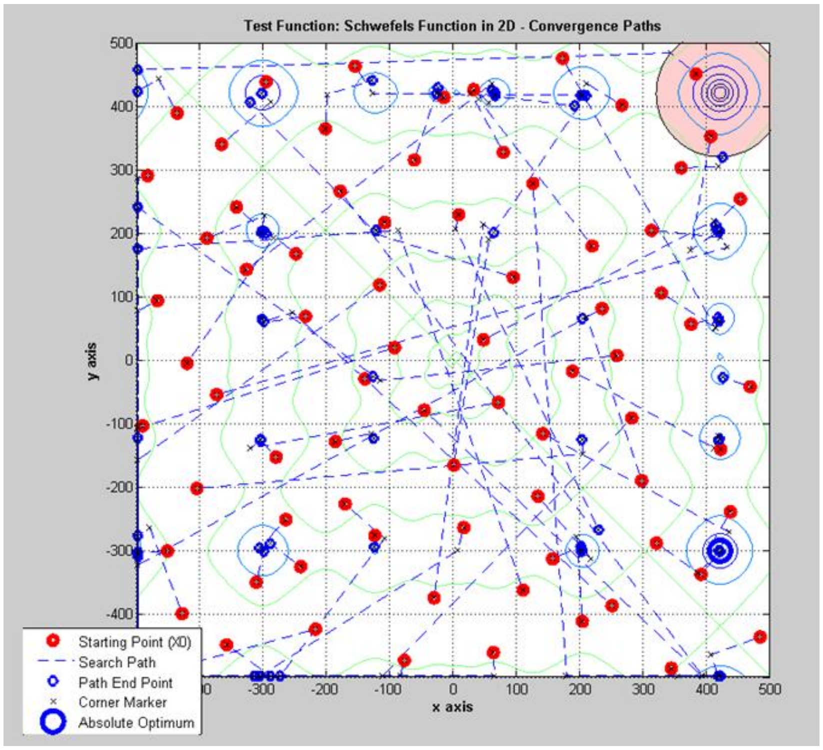


Figure 5-27 Test result, constrained steepest feasible descent - Schwefel's function

These data, then, indicate that the penalty algorithm used in conjunction with the Method of Steepest Feasible Descent from Global Search Points works as desired and prevents results from being identified within infeasible space; whether the starting point was defined within feasible space or not.

SQP Method from Global Sampling Points

Performance as an Unconstrained Search against Test Functions

Tests were also conducted against each test function for the Direct Method (using SQP) from the Global Start Points without any further optimization. That is, an optimum solution was identified from each start point and a 'global' optimum identified from among those results. Graphical results are depicted for each test function in the following figures showing the path followed from start to end for each global start point as well as the location of the best (lowest function result) overall value. Numerical results are tabulated below as well.

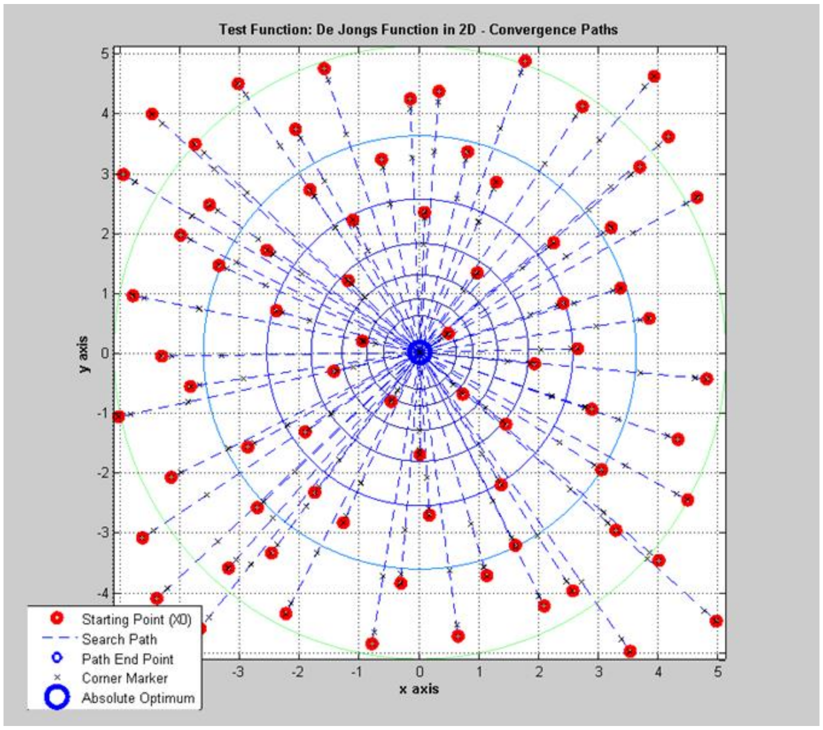


Figure 5-28 Test result, direct method (SQP) - De Jong's function

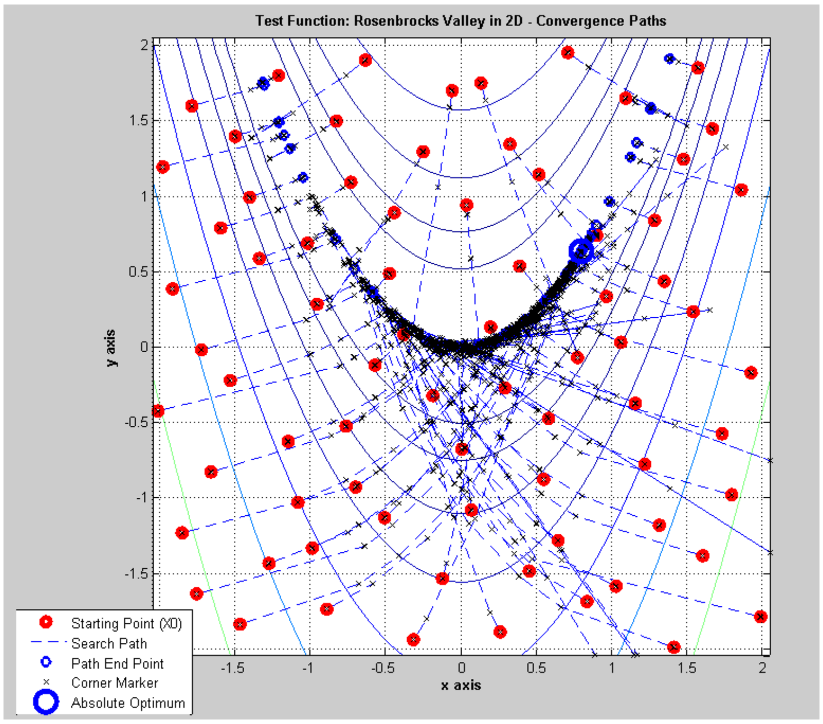


Figure 5-29 Test result, direct method (SQP) - Rosenbrock's function

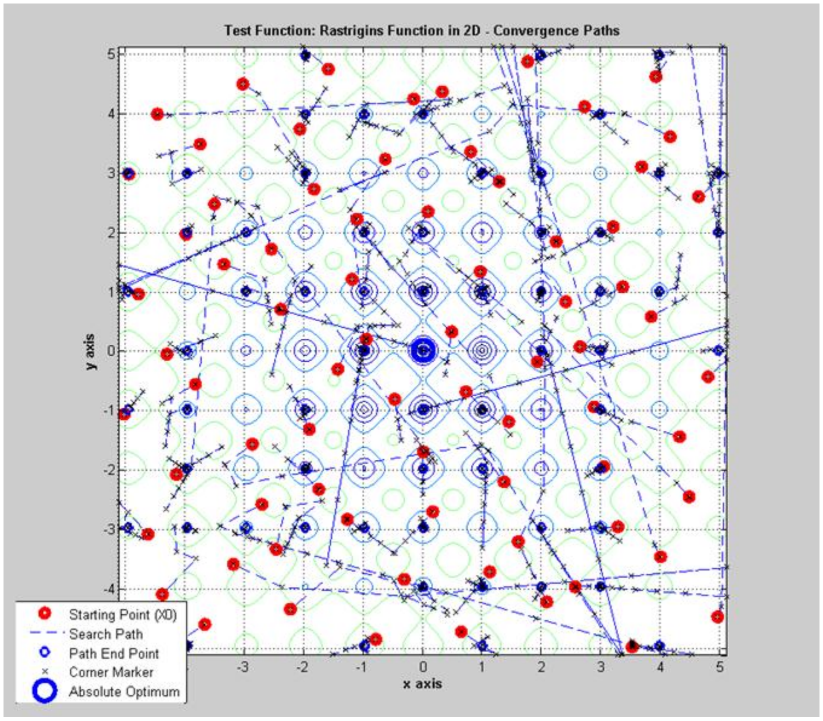


Figure 5-30 Test result, direct method (SQP) - Rastrigin's function

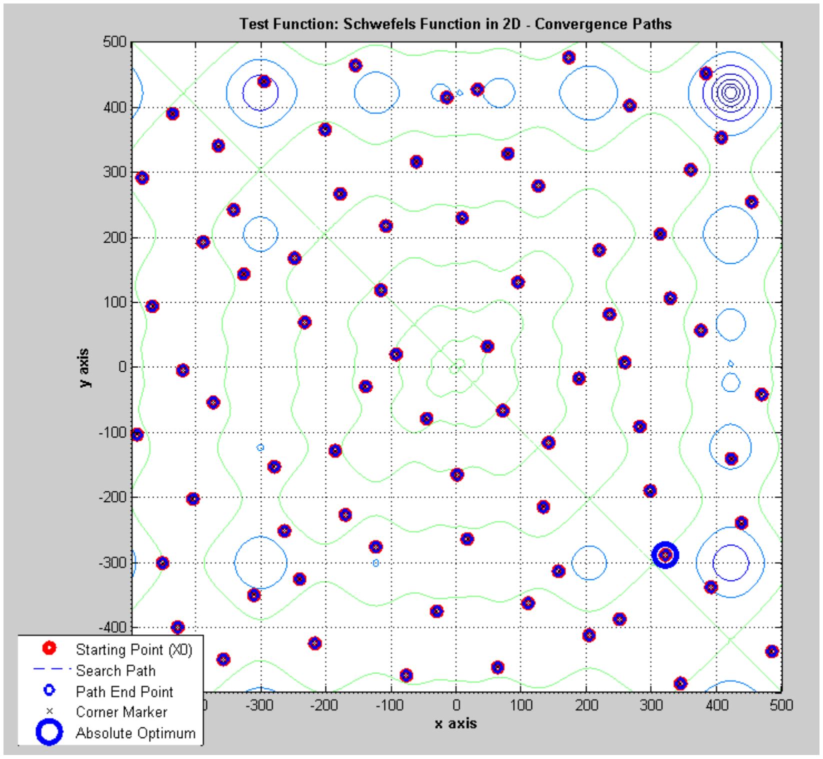


Figure 5-31 Test result, direct method (SQP) - Schwefel's function

Table 5-7 Test function result - direct method (SQP)

	Test Function			
	De Jong's	Rosenbrock's	Rastrigin's	Schwefel's
Coordinate Location of Overall Minimum				
Theoretical	(0,0)	(1,1)	(0,0)	(420.9687,420.9687)
Test Result	(7.4e-6,1.0e-3)	(0.9793, 0.9597)	(4.7e-5, 1.0e-3)	(-296.88, 438.27)
Function Value of Overall Minimum				
Theoretical	0	0	0	0
Test Result	2.03e-8	1.55e-7	2.7e-6	0.107
Number of Function Evaluations				
Test Result	1344	4863	2118	450
Number of Solutions within 'Results Region				
Test Result	75	0	1	0

The data indicates that the Direct Method using SQP from Global Start Points was successful in finding the theoretical optimum (within the toleranced region) for De Jong's function for all start points, but for only one (1) of the start points for Rastrigin's function. None of the start points resulted in the theoretical optimum for either Rosenbrock's or Schwefel's although the identified minimum was much closer (to the theoretical result) for the Rosenbrock's Valley than for the Schwefel's function. As shown in the data, the minimum identified for the Schwefel's function was not close to the theoretical optimum.

Another interesting point is that the number of function evaluations for the Schwefel's function was, comparatively speaking, quite low at only 450 solves. This, in comparison to 1344 ~ 4863 for the other three (3) test functions, indicates that the Direct Method with SQP converged to a solution within the minimum step size quickly and the algorithm failed to identify significant gradients on Schwefel's function that could have been explored.

The results indicate that use of the Direct Method with SQP was successful for the unimodal surface of De Jong, but should be used with care for multi-modal surfaces (e.g.;

Rastrigin's or Schwefel's) or surfaces with small gradients such as Rosenbrock's Valley. It is recognized that if a minimum step size value had been chosen smaller than 0.001, the SQP method would likely have been more accurate for at least some of the test functions, but at the expense of a greater number of function solves. This is demonstrated and confirmed in a later section.

Performance as a Constrained Search against Test Functions

Results for the constrained searches using the Direct Optimization Methods with SQP (only) from Global Search Points are provided in the following plots. These data help support the efficacy of the penalty algorithm employed. As mentioned previously, successful results prevent any optimum from being identified within infeasible space.

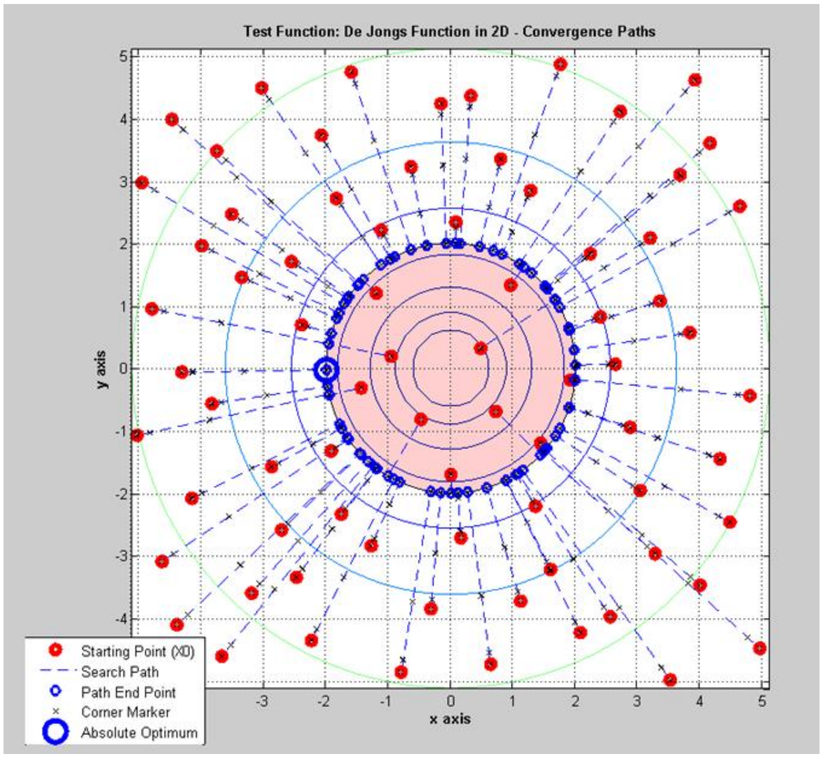


Figure 5-32 Test result, constrained direct method (SQP) - De Jong's function

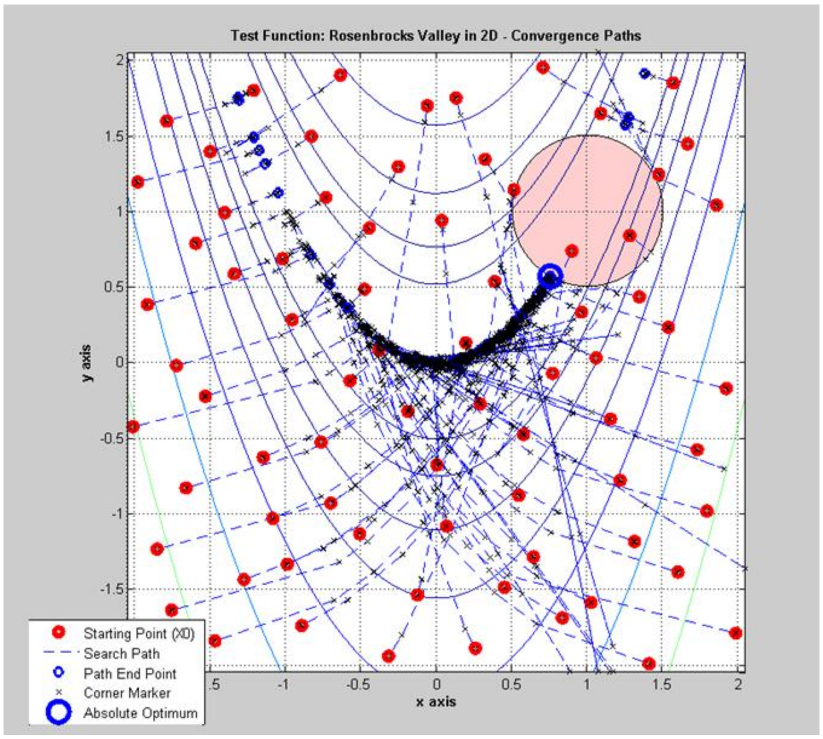


Figure 5-33 Test result, constrained direct method (SQP) - Rosenbrock's function

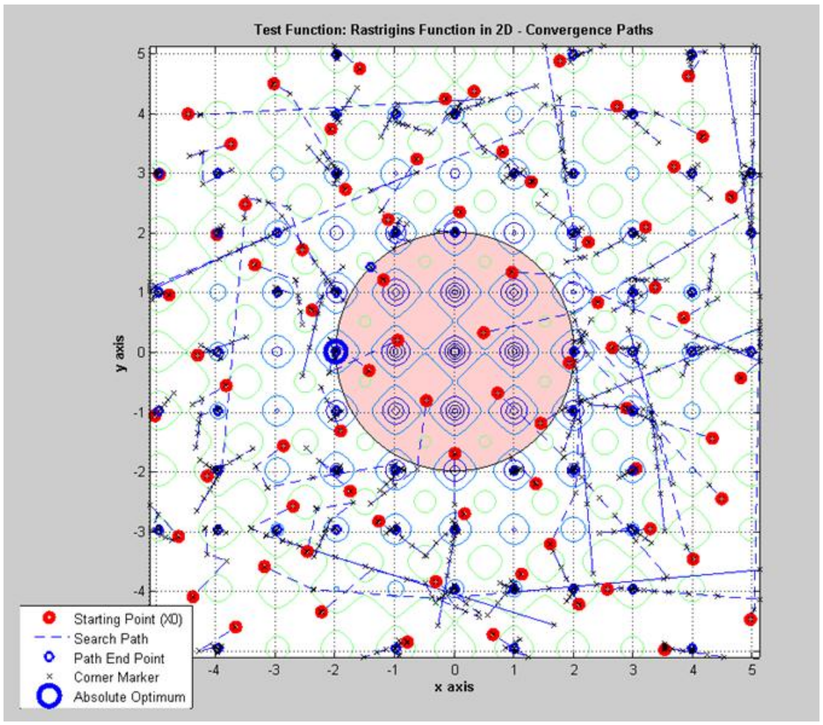


Figure 5-34 Test result, constrained direct method (SQP) - Rastrigin's function

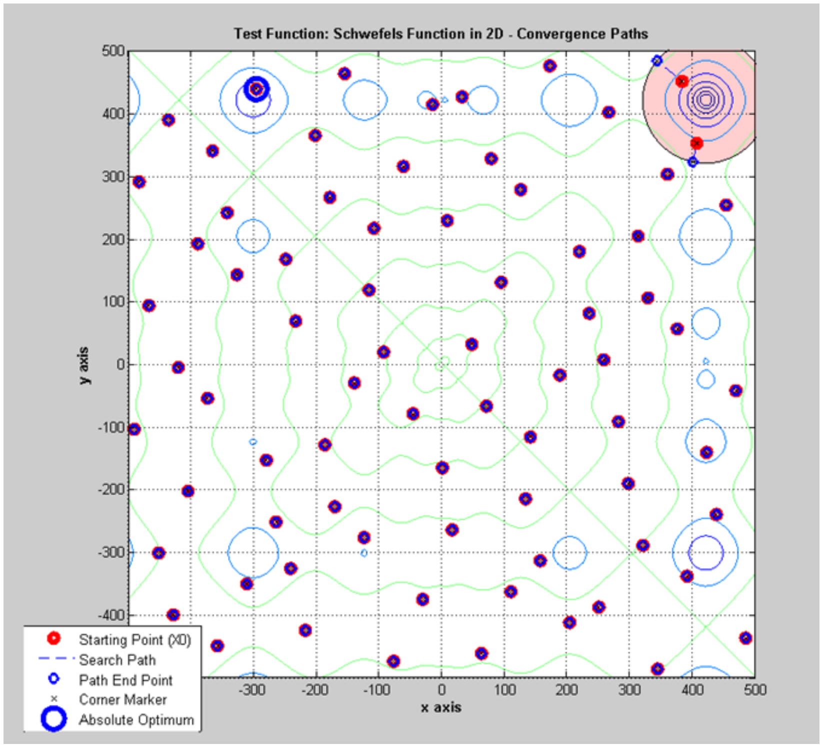


Figure 5-35 Test result, constrained direct method (SQP) - Schwefel's function

As with the Steepest Feasible Descent results previously reported, these data for the Direct Method with SQP indicate that the penalty algorithm works as desired and prevents results from being identified within infeasible space; whether the starting point was defined within infeasible space or not. The fact that no local minimums were identified within the infeasible region confirms the acceptability of the implemented SQP method with respect to constraint handling.

Combination of Methods from Global Sampling Points

Performance as an Unconstrained Search against Test Functions

Finally, tests were conducted against each test function for the ‘Combination Method’, which first finds a minimum for each of the Global Sampling Points, identifies the best ‘global’ interim solution and then applies the Direct Method with SQP to that as a starting point. As before, graphical results are depicted for each test function showing the path followed from beginning to end for each start point as well as the location of the best (lowest function result) overall value. Numerical results are tabulated below as well.

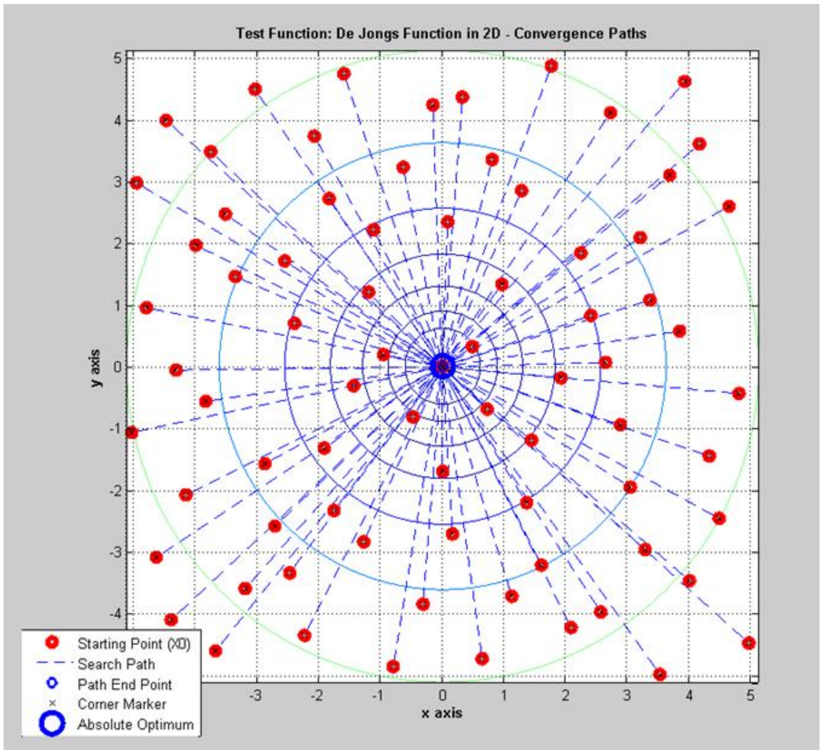


Figure 5-36 Test result, combined method - De Jong's function

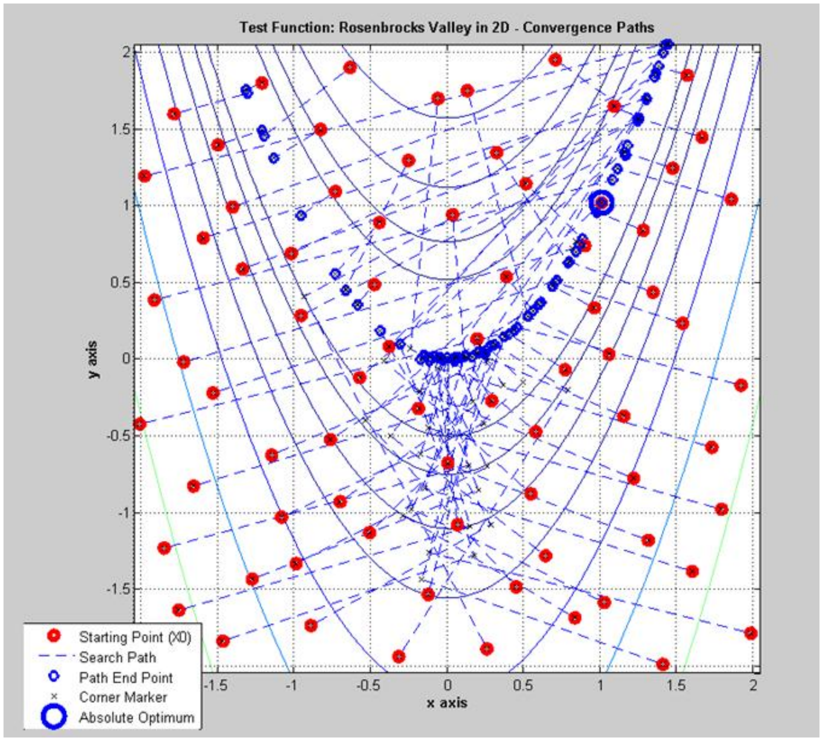


Figure 5-37 Test result, combined method - Rosenbrock's function

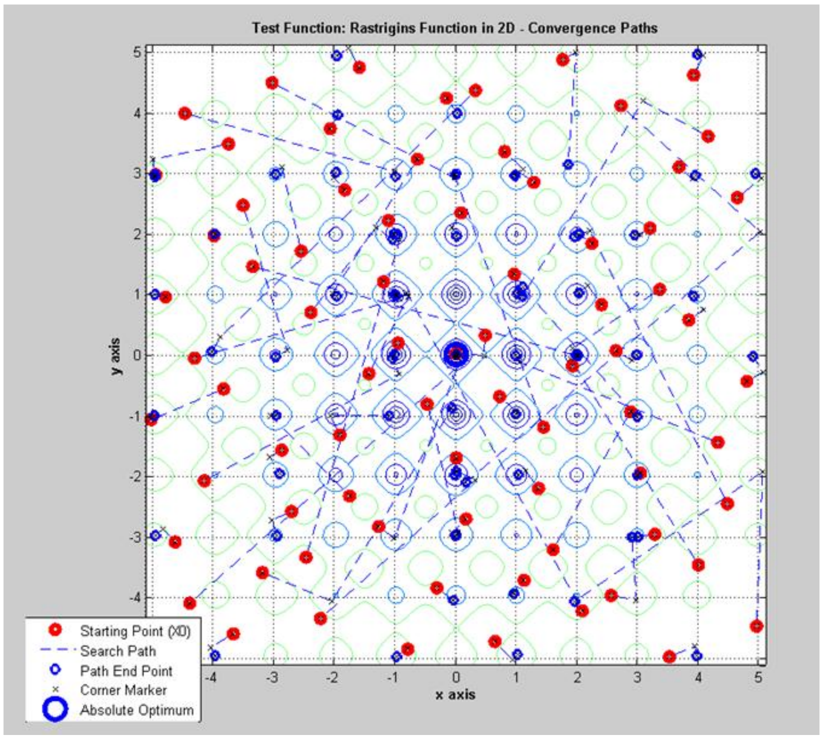


Figure 5-38 Test result, combined method - Rastrigin's function

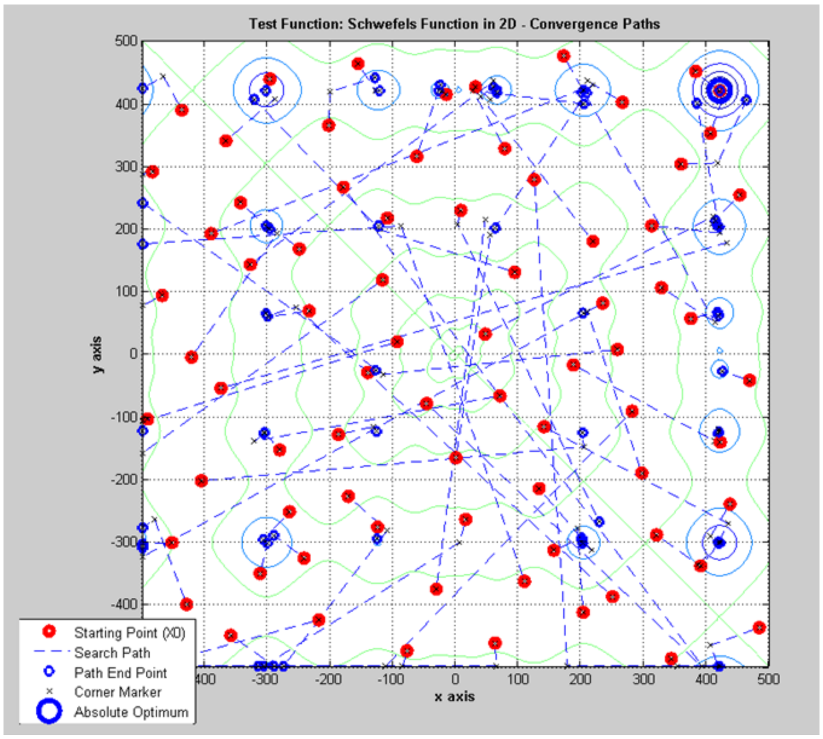


Figure 5-39 Test result, combined method - Schwefel's function

Table 5-8 Test function result - combined method

	Test Function			
	De Jong's	Rosenbrock's	Rastrigin's	Schwefel's
Coordinate Location of Overall Minimum				
Theoretical	(0,0)	(1,1)	(0,0)	(420.9687,420.9687)
Test Result	(3.9e-4, -1.4e-3)	(1.0065, 1.0135)	(6.3e-4, 7.7e-4)	(421.109, 419.458)
Function Value of Overall Minimum				
Theoretical	0	0	0	0
Test Result	4.3e-8	1.7e-8	2.5e-6	1.9e-4
Number of Function Evaluations				
Test Result	1137	1194	2250	1875
Number of Solutions within 'Results Region				
Test Result	1[(1)]	1[(1)]	1[(1)]	1[(1)]

1) *Note that since the final (SQP) search originates from a single point, only 1 result is possible.*

The data indicates that the Combined Method was successful at obtaining a ‘global’ minimum within the toleranced region for all test cases. These results are acceptable to demonstrate that the algorithm functions acceptably with respect to all four (4) test surfaces when evaluated with the Combined (Steepest Feasible Descent plus Direct SQP) search methodology. A comparison of the efficiency and efficacy of this method as compare with the other two (2) candidates is provided and discussed in a later section. As with the Direct Method previously, it is recognized that if a minimum step size value had been chosen smaller than 0.001, the result would likely have been more accurate for at least some of the test functions, but at the expense of a greater number of function solves. This is demonstrated and confirmed in a later section.

Performance as a Constrained Search against Test Functions

Following are plots (similar to those above for the other search methodologies) for the Combination Search Methodology on each test function to challenge the algorithms ability to prevent infeasible results.

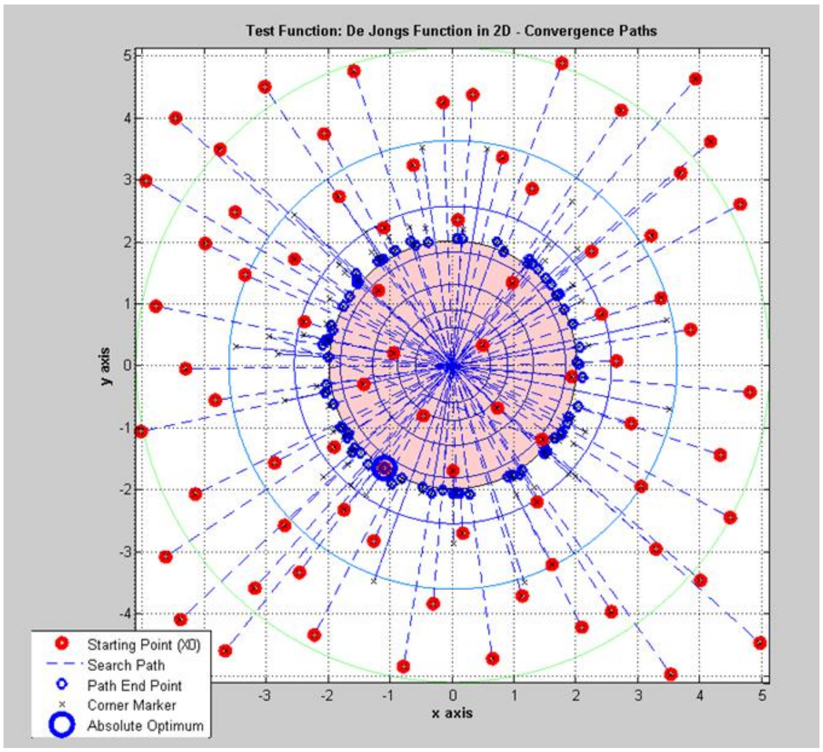


Figure 5-40 Test result, constrained combined method - De Jong's function

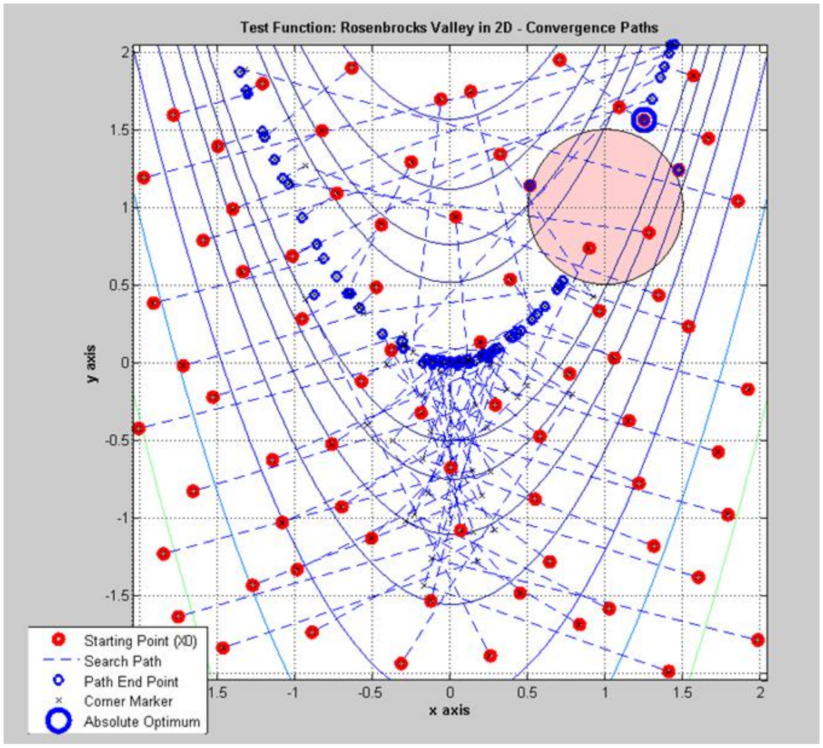


Figure 5-41 Test result, constrained combined method - Rosenbrock's function

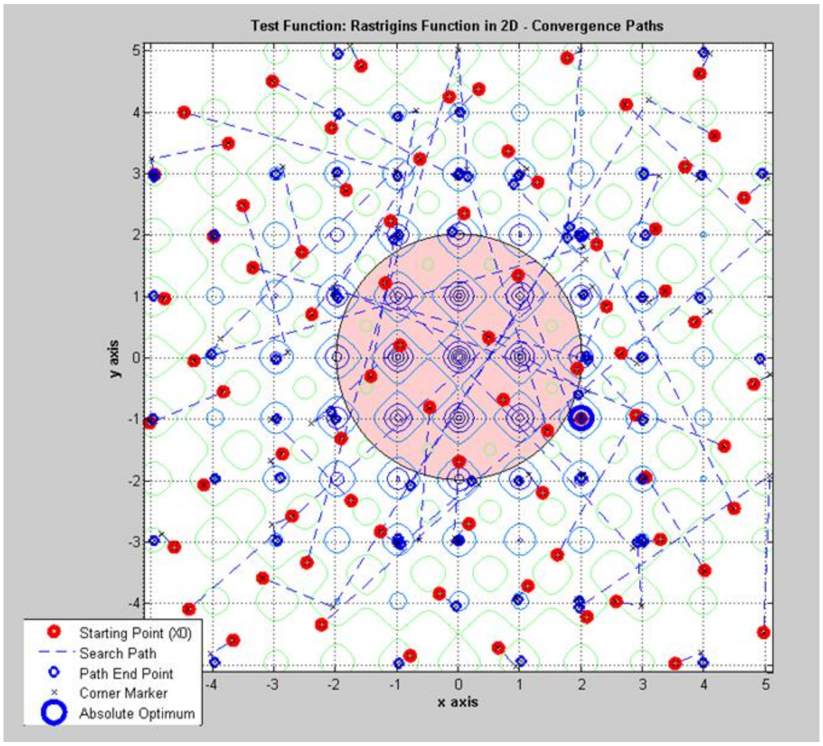


Figure 5-42 Test result, constrained combined method - Rastrigin's function

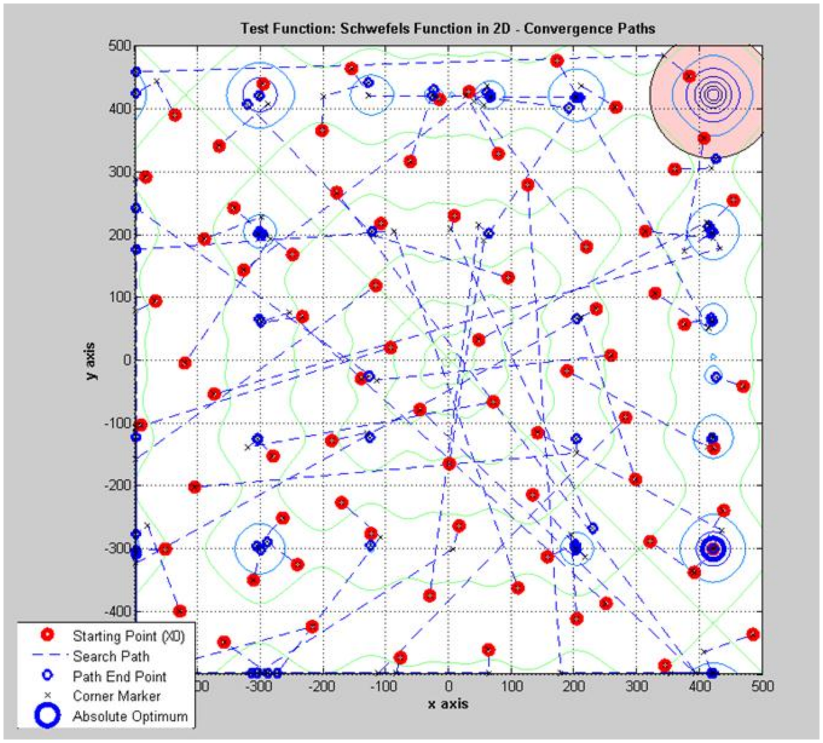


Figure 5-43 Test result, constrained combined method - Schwefel's function

These Combined Search Method results also support that the penalty algorithm works as desired and prevents results from being identified within infeasible space. This confirms the acceptability of the Combined method with respect to constraint handling.

Selected Optimization Technique for Problem under Study

In the previous section, the results of each of the search methods was evaluated with respect to finding a correct global optimum (vs. known theoretical optimums) for each of the four (4) test functions. In this section, the efficacy and efficiency of the methods are compared in order to support a statement as to which is the most effective and efficient methodology for application to the parallel beam problem. For convenience, the results of the test runs of the previous section are summarized in the following table.

Table 5-9 Test function result - summary of unconstrained results

	Test Function			
	De Jong’s	Rosenbrock’s	Rastrigin’s	Schwefel’s
Coordinate Location of Overall Minimum				
Theoretical	(0,0)	(1,1)	(0,0)	(420.969,420.969)
Steep Descent	(4.2e-4, -4.5e-4)	(1.0067, 1.0135)	(-0.0070, 0.0117)	(421.109, 419.456)
Direct (SQP)	(7.4e-6,1.0e-3)	(0.9793, 0.9597)	(4.7e-5, 1.0e-3)	(-296.88, 438.27)
Combined	(3.9e-4, -1.4e-3)	(1.0065, 1.0135)	(6.3e-4, 7.7e-4)	(421.109, 419.458)
Function Value of Overall Minimum				
Theoretical	0	0	0	0
Steep Descent	0	0	4.6e-4	2.0e-4
Direct (SQP)	2.03e-8	1.55e-7	2.7e-6	0.107
Combined	4.3e-8	1.7e-8	2.5e-6	1.9e-4
Number of Function Evaluations				
Steep Descent	1131	1190	2233	1869
Direct (SQP)	1344	4863	2118	450
Combined	1137	1194	2250	1875
Number of Solutions within ‘Results Region				
Steep Descent	75	1	2	2
Direct (SQP)	75	0	1	0
Combined	1(1)	1(1)	1(1)	1(1)

1) Note that since the final (SQP) search originates from a single point, only 1 result is possible

The Steepest Feasible Descent and Direct methods (used independently) produced similar results in terms of accuracy (efficacy) and number of function evaluations (efficiency) for both De Jong’s and Rastrigin’s solutions. In both cases, the SQP method was slightly more accurate than SFD in terms of both the coordinate location and function value of the identified optimum. This is to be expected since the SFD method was limited to two (2) ‘jumps’ as is more appropriate for a ‘coarse’ search method. Another interesting observation is that for the Rastrigin’s function, even given the 2-jump limitation, the SFD method found an optimum within the toleranced results region for two (2) start point searches, whereas only one (1) landed within the toleranced region for the SQP method. This is interesting because a primary challenge presented by Rastrigin's is the multitude of local minima throughout the design space separated by 'ridges'. This supports, to a limited degree, the original hypothesis that SQP may have more difficulty moving across ridges than Steepest Feasible Descent. The difference in this example however is small.

A more pronounced difference between these two (2) independent methods is observable with the Rosenbrock's Valley results. Here, the SFD method found an optimum very near to the theoretical optimum (with a '2-jump' limit) for one (1) of the start points, whereas none of the start points for the SQP method ended within the toleranced 'Results Region'; although the SQP solution was 'close' to the region of interest. Further, the SQP method required significantly more function evaluations than the Steepest Feasible Descent (4863 vs. 1190) to reach this conclusion. For the Rosenbrock's Valley test, then, Steepest Feasible Descent is shown to be both more effective and more efficient than the Direct method with SQP.

Perhaps the most striking difference between the two (2) independent methods is observable with the Schwefel's function. As a reminder, Schwefel's is similar to Rastrigin's in that multiple local minima exist, but with the added challenge that the theoretical minima is geometrically distant from the center of the design space. For this test function, the SFD method resulted in two (2) global start points finding a result within the allowable tolerance surrounding the theoretical minimum. The SQP method however failed to place any within the same results region and, in fact, found an 'optimal' result far from the theoretical solution. A closer look at the plot (Figure 5-31) indicates that for this test, the SQP method resulted in little if any movement among any of the start points for Schwefel's function. This is supported by the significantly fewer function evaluations used by SQP. Detailed review of the convergence data (not presented here) reveals that for this test, the algorithm stopped because the size of the step fell below the defined acceptability threshold (minimum design variable step size = 0.001); that is, the routine was quickly satisfied with the findings for each start point within the limits of the step size threshold. Given this observation, it is theorized that a significantly smaller design variable step size limit would improve the optimization results by allowing the algorithm to continue its search. To prove this theory, a subsequent test was run for the SQP method on Schwefel's function with a Minimum Design Variable step size set to 0.00001 (two (2) orders of magnitude smaller than the original test) and different results were observed. This time, the theoretical optimum solution was found but at the cost of a significantly greater number of function evaluations. Results of this additional test are as follows:

Table 5-10 Additional unconstrained test function result - Schwefel's function

Input:	
Min Design Variable step size	0.00001
Optimization Method	Direct Method (independently)
Test Surface	Schwefel's
Theoretical Global Optimum	(420.9687,420.9687)
Results:	
Location of Identified Minimum	(420.9687,420.9687)
Function Value of Identified Minimum	1.697e-8
Number of Function Evaluations	3963
Number of Solutions within Results Region	2

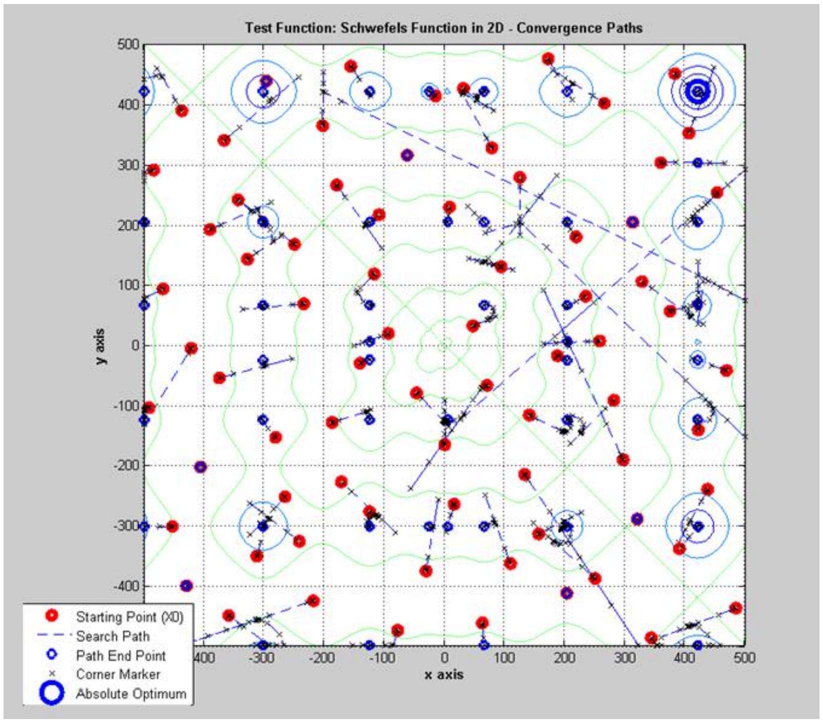


Figure 5-44 Additional direct method test, Schwefel's function, unconstrained

These data confirm the theory that the reason for the (original) failure of the Direct Method (SQP) to find the theoretical optimum on an unconstrained Schwefel's function was due to the minimum step size stopping criteria being too large. By making this value much smaller, the theoretical optimum was found by two (2) global starting points (vs. no acceptable solutions for the original search). However, more than eight (8) times the function evaluations were required in the process as compared with the Steepest Feasible Descent method (3963 vs. 450). Therefore, it can be concluded that for the more conservative (larger) minimum step size, the SFD Method is more effective than the SQP Method with Schwefel's function. If, however, the minimum step size is decreased for the SQP search, both methods are similar with respect to effectiveness (each finding theoretically correct solutions for 2 of the starting points) but the Steepest Feasible Descent is significantly more efficient than the Direct method. In a larger sense, this same consideration for stopping criteria can be applied to all of the methods as well as consideration for the maximum number of jumps specified for SFD searches; each with the potential to improve accuracy of the overall result, but at the expense of an increased number of function solves. As noted in Table 5-4, the stopping criterion for the SFD search was a design variable step size limit of 0.001”. By contrast, the stopping criterion for the second (successful) SQP search was a limit of 0.00001” (Table 5-10). When compared to the design space size for Schwefel’s function (x= ± 500, y= ± 500 per Table 5-4), the SFD search was successful with a design variable stopping size of 1e-4% (of the range in design space) whereas the SQP method required a stopping size of 1e-6**%.** This is an important consideration for EBBF functions where the cost of each function solve is significant.

Comparison of the combined method results (to the SFD and SQP individual methods) tests the theory that there is utility in leveraging benefits of the two individual methods as a means to maximize both efficacy and efficiency. A review of the data from De Jong’s surface (Table 5-9) shows that the combined method did find the correct answer within approximately the same small tolerance as either of the individual methods. Efficiency was also very similar between the combined and individual methods, supporting a conclusion that no significant benefit exists for the combined method on the relatively simple De Jong’s function. This is not surprising given the function’s simplicity. Note that, as indicated in Table 5-9 only one (1) solution exists within the toleranced ‘Results Region’ for the Combined method. This is because only one (1) start point is utilized in the SQP part of the search; starting from the optimum result of the ‘global’ Steepest Feasible Descent search.

A similar comparison for each of the other three (3) test functions yields a different, yet common theme. The combined method results in optimums which match the more accurate of either of the individually applied methods and with little more than the fewest function solves between them. That is, the combined method is both as accurate and as efficient as the better of the individual methods for these, more complicated, test surfaces. As mentioned for the SFD vs. SQP comparison above, this data supports that the ‘Combined’ method is more tolerant of a coarse design variable step size as a stopping criteria than the SQP method used independently. This is a significant observation with consideration for use with EBBF’s.

It is also interesting to compare the performance of the search methods to similar results from literature. Kao and Zahara [43] performed comparable tests using Rosenbrock's function with two (2) design variables in their investigation of a Continuous Genetic Algorithm (CGA). Their results are shown together with results from SFD, SQP and 'Combined' methods in Table 5-11 below.

Table 5-11 Comparison of Optimization Results to Literature Results

	Results From This Work (Values Repeated from Table 5-9)			Results from Literature using Continuous Genetic Algorithm [43]
	SFD Independently	SQP Independently	'Combined' Method	
Number of Function Evaluations	1190	4863	1194	960
Error (%) – Global Optimum Coordinate Location vs. Theoretical	1.51 %	4.42 %	1.50 %	0.40 %

A conclusion is formed that by combining the two search techniques in the proposed way, the best of each method is leveraged to achieve the best result. Therefore, the combined method is more suitable for Expensive Black Box functions than either method used independently because it finds an accurate result with the fewest number of function evaluations. As shown for the Rosenbrock's function, this reduction in function evaluations can be significant (4863 for Direct vs. 1190 for Combined ≅ 75% reduction). Further, the proposed method is shown to be less sensitive to coarse design variable step size as a stopping criterion. Both of these observations can be very important if each function solve requires several minutes, hours or more to accomplish. Given these observations, the combined method is selected for use on the parallel beam problem.

Chapter 6

Optimization of Parallel Beam Problem

The previous two (2) chapters discuss development of the key elements of this research effort; 1) finite element modeling of the harmonically loaded parallel beam system and 2) the best optimization method to be used in finding a global solution to a multimodal surface. In this chapter, these elements are brought together and the harmonic parallel beam problem is solved to find the combination of input parameters which result in a minimum harmonic response.

Parametric Model Definition – Use of Super-Elements

The parallel beam problem (illustrated in Figure 2-1) is comprised of three (3) model variants. As defined in Table 2-1, these family variants exist in parallel to form the overall problem definition, differing by the length of the upper bar (L_{top}) and the magnitude of the lumped mass (m). Ultimately, an optimum set of input parameters (common to all family variants) is sought which produces a minimum 'compromised' response across the variants.

One challenge presented by the problem definition is a method by which to model the variation of the bottom beam's supports along the entire length of the beam given a fixed placement for the two (2) inter-beam supports. That is, the geometric location of the inter-beam supports is fixed with respect to the origin and not subject to redefinition during the optimization process. The location of the lower beam's supports to ground, however, is not. Therefore, the relative location of the inter-beam supports with respect to the lower beam's supports will change during the analysis. Although the act of meshing the beam(s) is conducted via an automatic meshing routine within the finite element program (ANSYS ®), the connectivity of the spring supports to the upper and lower beams must be defined with respect to their relative locations. If an optimization model begins with both lower springs located between Point 'A' and the inter-body spring at L_3, then the automatic mesh definition is dependent upon a 'super-element' model definition that specifies one super-element for the lower beam between Point 'A' and L_1, a second super-element from L_1 to L_2 and a third super-element from L_2 to L_3. If the next iteration requires that the bottom support at L_2 move to the right of L_3, then the definition of the super-element from

L_1 to L_2 and then L_2 to L_3 is no longer correct. Instead, the super-element definition needs to be re-written to be from L_1 to L_3 and then L_3 to L_2.

Although there are potentially multiple ways to programmatically handle this detail, a relatively straightforward method of multiple model definitions is selected here. That is, by examination of the problem parameters it can be shown that six (6) potential configurations exists to cover all potential super-element configurations. If a unique ANSYS ® model (computer file) is written for each super element configuration, then depending upon the placement of the spring supports (driven by the optimization code), the appropriate ANSYS ® model (super-element configuration) can be called. These six (6) super-element configurations are depicted schematically in Figure 6-1 below. The super-element models are consistent between (i.e.; do not change among) the three (3) family variants of the problem. Therefore, by modeling these six (6) super-element configurations parametrically and providing the correct variable definitions according to the family variant (top beam length and lumped mass size.), then the problem is simplified for analysis.

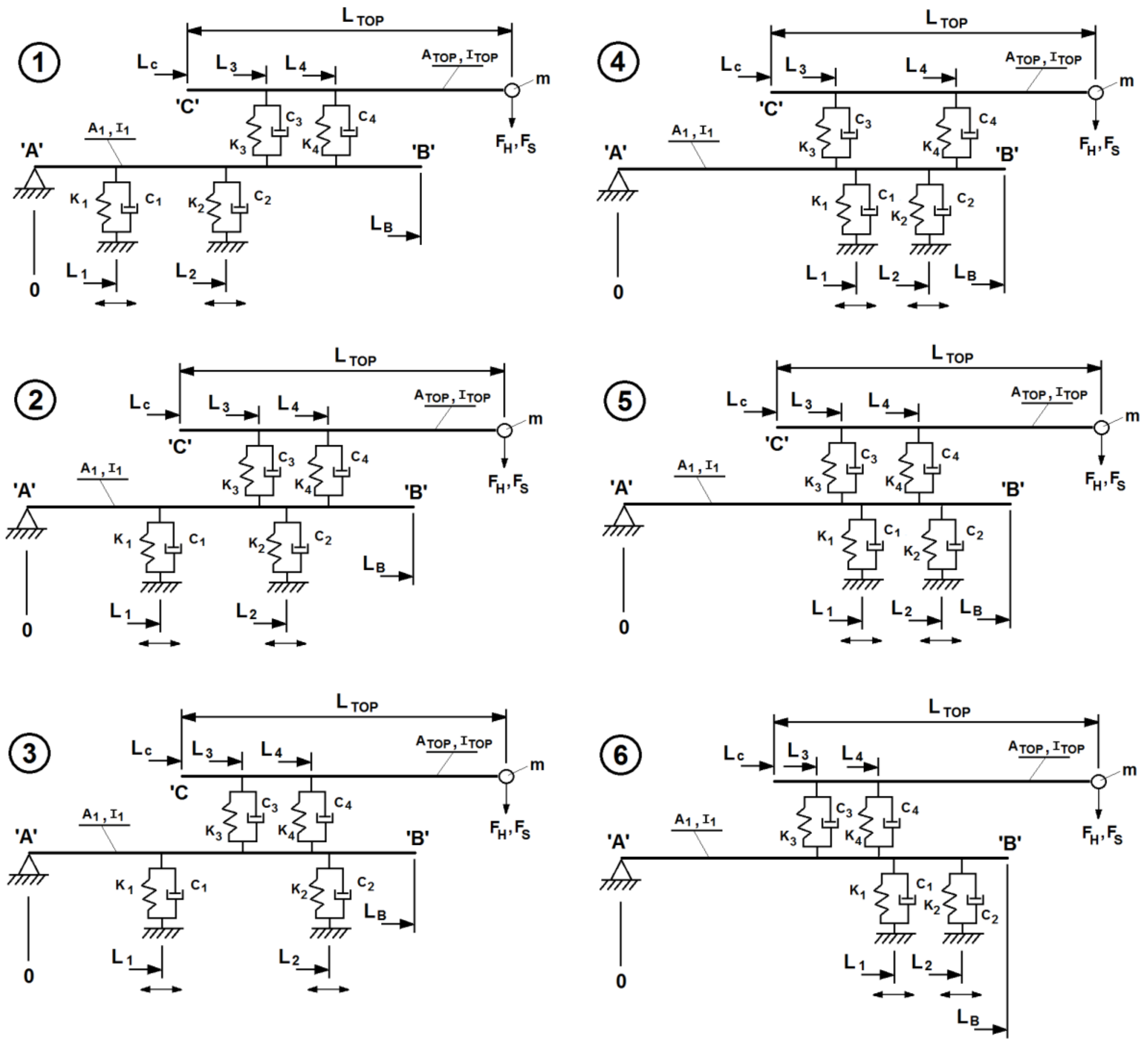


Figure 6-1 Super-element models - schematic layout

Optimization Plan

Use of a 'Single-Objective Optimization' Tool for 'Multi-Objective Optimization'

The problem statement for the parallel beam problem was defined in Chapter 2 as being comprised of both static and harmonic response variables. Tip deflection response due to a static loading is used as a constraint response and will be discussed in a later section of this chapter. Harmonic deflection responses (also due to tip loading) comprise the remainder of the problem statement and combine to form the variable (objective function) to be minimized.

Within this latter ‘group’, two (2) harmonic responses per family variant are considered as part of the total objective function. They are:

1. The sum of tip displacement due to the harmonic loading: Note that this is a discrete sum of the harmonic responses among the range of frequencies studied as shown in Eq. (6-1). It is similar to a discrete integration of the area under the FRF curve, but does not include the ‘dimension’ of frequency as would be considered in an actual integration. An integrated result is not necessary in this case since all analyses are conducted across the same frequency range and with the same frequency spacing.

$$d_{tip-sum} = \sum_{f} d_f \tag{6-1}$$

$$where\ d_f = harmonic\ displacement\ at\ frequency\ f$$

$$d_f = \sqrt{d_{real,f}^2 + d_{imaginary,f}^2}$$

$$f = 10 \sim 100\ Hz, step = 2.5\ Hz$$

   By minimizing this scalar sum of tip displacements, the overall magnitude of harmonic response (across the frequency range) is minimized. However, due to natural frequency effects it is quite possible that a solution with a ‘low’ summed deflection response could still have a large response at some particular frequency(s) and therefore be undesirable as a practical solution. Consideration for this eventuality is motivation for the second part of the objective function.

2. Consideration for minimizing the range of tip displacement due to harmonic loading aids in optimizing the system by seeking to ‘flatten’ the tip displacement across frequencies. When combined with the summed metric, the tip deflection response can be better optimized for minimal response across all frequencies. This scalar is given in Eq. (6-2).

$$d_{tip-range} = \max(d_f) - \min(d_f) \tag{6-2}$$

$$where\ f = 10 \sim 100\ Hz, step = 2.5\ Hz$$

By treating the two (2) harmonic responses as scalar components of a single objective function, the single-objective optimization tools described previously can be utilized to solve a multi-objective optimization problem. In order to ensure that both components are considered appropriately in the process, each of the responses are normalized against a scalar constant that represents a generally expected ‘average’ magnitude for the given response. The result is that each component of the objective function now has an order of magnitude that is similar and one response is not ‘favored’ over the other in the optimization process due to differences in scale. The magnitude of these normalization constants was selected by separate test cases of a couple of exemplary conditions which is not discussed further here. The exact magnitude of the normalization factors is not critical since the constant is applied to all results in an effort to scale them to similar magnitudes. The normalization factors used for this investigation are:

Table 6-1 Normalization factors used for response variables

Response Variable	Normalization Factor
Sum of tip displacements across frequencies - Eq. (6-1)	300
Range of tip displacements across frequencies - Eq. (6-2)	100

The multi-objective purpose of the single objective function is furthered by adding a weighting factor to each of the two components in order to consider their relative importance in the overall optimization process. The single-objective optimization function then becomes

$$OF_{SO} = \alpha_1(d_{tip-sum}) + \alpha_2(d_{tip-range}) \tag{6-3}$$

where $\alpha 1$ = Weighting factor – sum of tip displacements
$\alpha 2$ = Weighting factor – range of tip displacements

The weighting factors used are given in Table 6-2.

Table 6-2 Weighting factors used for response variables

Response Variable		Weighting Factor
$\alpha 1$	Sum of tip displacements across frequencies - Eq. (6-3)	1.0
$\alpha 2$	Range of tip displacements across frequencies - Eq. (6-2)	0.5

In this way, the summed harmonic tip displacements over the range of frequencies are given a higher priority than the range of displacements. The weighting factor values selected here are somewhat arbitrary, but serve to exemplify the development of a single objective function as a solution for a multi-objective problem.

Development of the Objective Function

A goal of the study is to find the combination of input variables which provide for the 'best' response for the family system. In addition to the use of a single objective function as described above, consideration must be made for the response of each of the family variants as components of the objective function in order to optimize the 'system' response. It is assumed here that equal weighting should be given to the response from each of the three (3) family variants. Therefore, as each of the family variants are evaluated for a given set of variables (support locations, spring stiffnesses and damping coefficients) the overall objective function simply becomes the sum of the individual objective functions for each variant. This is given in Equation 6-4 below.

$$OF = \sum_{i=1}^{n} (OF_{so})_i \tag{6-4}$$

$$for \;\; n = \text{total 3 family variants}$$

Finally, a penalty value is added to the overall objective function as defined in Eq. (5-7). In this way, the resulting penalized objective function considers the following:

- Multi-objective optimization of both sum and range of harmonic tip deflection across the range of frequencies studied.

- Relative importance (weighting) of the sum and range responses of the system
- Consideration for the overall system response; including multiple family variants
- Penalty effects for designs resulting in infeasible responses to established constraints

Development of the Constraint Functions

Two (2) types of constraints are considered in this analysis; those related to the values of the input (design) variables and those related to the output (results) of the simulation.

Input constraints are, as the term implies, those constraints that apply solely to the values of the various input variables. These include side bounds as well as other constraints imposed on the system. For the parallel beam problem, side bounds include the upper and lower limits associated with the placement of the bottom beam's supports (L_1 & L_2) as well as upper and lower bounds for each of the four (4) spring stiffnesses and four (4) damping coefficients. Values for these side-bound constraints are given in Table 2-1 as illustrated in Figure 2-1.

Programmatically, these side-bound constraints are converted into a set of constraint equations for each of the various inputs. Each side-bound constraint is an 'inequality constraint' in that any value less than or equal to zero (0) represents feasibility. Development of an exemplary constraint equation for the lower-bound (LB) of an input variable is given as follows:

$$x \geq LB$$

$$\therefore \ 0 \geq LB - x$$

$$\therefore g_{LB} = 1 - \frac{x}{|LB|} \tag{6-5}$$

Note that in Eq. (6-5), the equation is normalized against the value of the LB. In this way, similarity among all constraint equations is included for improved results during the optimization process (similar in concept to previous discussions regarding response variables). If the constraint equations were not normalized, then the differing magnitudes of the constraint results would have an unintended consequence of 'weighting' during optimization. Further, for side-bound constraints where the value of the side-bound may be a negative value (such as with the test surfaces addressed previously), it is important to normalize against the absolute value of the

LB so as not to invert the sign of the actual constraint result. A similar process exists for the upper-bound (UB) constraints but is not repeated here due to its straightforward nature. These side-bound constraint equations are implemented in the optimization code programmatically for each of the input variables with respect to both upper and lower bounds by the process of Eq. (6-5). A total of 20 side-bound constraint equations exist for the parallel beam problem; consisting of upper and lower equations for each of ten (10) input variables.

In addition to the side-bound constraints, seven (7) other input constraints are imposed upon the model. These include constraints that govern the location of the bottom beam's supports (L_1 & L_2) given the physical limitations that none of the four (4) spring supports can lie 'on top of each other'. In fact, given an assumed physical geometry of the supports, some minimum amount of distance must exist between them in order for the geometry of the design to be practical. It is assumed for the purposes of this exercise that this minimum distance is 1.5 inches. Each of these constraint equations are developed in the same manner as presented in Eq. (6-5) above. 'Non side-bound' constraints and equations are given in Eq. (6-6) below.

L_1 and L_3 differ by at least 1.5" $$g_1 = 1 - \frac{|L_1 - L_3|}{1.50}$$

L_1 and L_4 differ by at least 1.5" $$g_2 = 1 - \frac{|L_1 - L_4|}{1.50}$$

L_2 and L_3 differ by at least 1.5" $$g_3 = 1 - \frac{|L_2 - L_3|}{1.50}$$

L_2 and L_4 differ by at least 1.5" $$g_4 = 1 - \frac{|L_2 - L_4|}{1.50} \qquad (6\text{-}6)$$

L_1 is offset from Point 'A' by at least 1.5" $$g_5 = 1 - \frac{L_1}{1.50}$$

L_2 is offset from Point 'B' by at least 1.5" $$g_6 = \frac{L_2 - L_b}{1.50} + 1$$

L_2 is greater than L_1 and is offset by at least 1.5" $$g_7 = \frac{L_1 - L_2}{1.50} + 1$$

Each of these ‘non side-bound’ constraints are also considered as inequality constraints in the model whereby a negative value (or ‘0’) resulting from the constraint equation indicates a feasible result. For this problem, no equality constraints were utilized.

Result-based (or output) constraints were also considered for the model. As described previously, the goal of the process is to minimize harmonic response, but with consideration for some minimum stiffness of the system such that a maximum static deflection value is not exceeded. Therefore, for each combination of design variable values, and for each of the three (3) family variants a static analysis was conducted where the deflection of the tip of the upper beam was evaluated. This resulting deflection value (for each family variant) was then compared to the established maximum allowable deflection value in order to create a constraint equation for this output result.

For this evaluation, an allowable maximum static deflection of 2.0 inches (given the static loading of 0.5 Lb_f defined in Table 2-1) was established. Given this value, it can be expected that configurations with a maximum cantilever are more likely to exceed the static threshold and others (where L_1 & L_2 are more centrally located for example) may have feasible results with respect to static deflection. The results-based inequality constraint for static deflection is developed in Eq. (6-7) as follows:

$$2.0 \geq |deflection_{static}|$$

$$\therefore\ 0 \geq |deflection_{static}| - 2.0$$

$$\therefore\ g_{result\_i} = \frac{|deflection_{static}|}{2.0} - 1 \qquad (6\text{-}7)$$

$$where\ i\ =\ \text{each of the 3 family variants}$$

Programmatically, each of the individual, inequality constraint equation results (20 side-bound, 7 ‘non side-bound’ and 3 static deflection results) are combined into a single row vector for passage to the various subroutines of the optimization program. As mentioned previously, no equality constraints were used in this problem. Had they been used, they would have been collected into a similar vector for passage as well.

Additional Considerations

One limitation of the Steepest Feasible Descent algorithm derived and implemented here is that, during the 1-D line search, the maximum length of the line search is determined by the distance from the starting point (X0) to the nearest design variable constraint bound. In the case that only side-bounds exist on the design variables, this is sufficient and allows the fullest possible exploration of the design space along the search vector (Sn) without further consideration. However, if additional design variable constraints exist within the domain then the maximum distance may be shortened accordingly. That is, these additional, intermediary design variable constraints could be incorrectly sensed as an upper or lower bound by the subroutine that looks for the maximum distance across the design space. In this event, the maximum search distance would be limited to the space between X0 and the first such intermediary constraint. The consequence of this issue is that additional feasible space may still exist beyond the recognized 'nearest constraint bound' that would not be recognized by the algorithm for exploration and a potential global minimum not discovered.

Such is the case for the parallel beam problem here where the support locations are constrained from existing within a tolerance band of one another. For example, if X0 is defined such that both L_1 and L_2 exist to the 'left end' of the lower bar (near point A), and the search path dictates positive movement of both L_1 and L_2 (to the right), then the maximum search distance will be limited to the first support location constraint that is encountered (L_2's interaction with L_3). This would allow exploration of the design space to the left of L_3, but would omit feasible space to the right of L_3 that should also be explored. Such a limitation could prevent a global optimum from being discovered if it existed to the right of L_3. As discussed later under 'Opportunities for Further Research', this is an area where improvement can be made to the current program.

Until such improvements can be incorporated, a 'workaround' was implemented for use with this parallel beam analysis. Starting points were selected such that some existed in each of the six (6) geometric configurations (ref. Figure 6-1). In this way, it is possible for a 'global' minimum to be identified in any of the six (6) configurations for the Steepest Feasible Descent Method. From that point, the direct search method (SQP) would begin and could traverse any of the configurations; since the 1-D search distance is not defined in the same way for SQP as for the SFD Method.

The haltonset method of point selection was maintained as the method for starting point selection, but for this analysis a sufficient number of points was identified that four (4) starting points could be identified for each of the six (6) configurations. Programmatically, a loop was created using haltonset point selection in increasing size and the 'first four' starting points that fell into each of the six (6) configurations (and was also feasible with respect to other design variable constraints) was selected. The maximum of four (4) points per configuration, or total 24 starting points, was selected in an effort to both demonstrate the effectiveness of the overall method to identify an optimum while giving consideration to the overall computing time required (given that the parallel beam problem takes significant time to solve each function evaluation). The starting points identified and used in the analysis are provided in Table 6-3 below. Other key program parameters are given in Table 6-4.

Table 6-3 Starting points used in the parallel beam optimization problem

Super-Element Model (Figure 6-1)	Support Locations (in)		Spring Stiffnesses (Lb_f/in)				Damping Coefficients			
	L_1	L_2	K_1	K_2	K_3	K_4	C_1	C_2	C_3	C_4
1	9.000	32.000	40100	28786	18500	15731	0.1215	0.1092	0.0911	0.0733
	22.500	26.667	4460	22724	45500	38577	0.2962	0.2655	0.2202	0.1757
	2.250	42.667	16340	16663	23409	12216	0.4709	0.4218	0.3494	0.2781
	29.250	37.333	28220	10602	1318	35062	0.1523	0.0845	0.4785	0.3805
2	18.000	48.000	20300	14643	9500	8115	0.0632	0.0571	0.0480	0.0391
	31.500	58.667	44060	2520	14409	4601	0.4127	0.3697	0.3063	0.2440
	11.250	53.333	8420	44949	41409	27447	0.0941	0.0324	0.4354	0.3464
	28.125	51.556	34556	26765	24227	46778	0.1249	0.0077	0.3288	0.1592
3	38.250	66.667	26240	23735	27909	16024	0.0067	0.4479	0.3709	0.2952
	24.750	69.333	10796	38888	19318	1086	0.2688	0.1887	0.0705	0.4488
	14.625	67.556	46436	20704	2136	20417	0.2996	0.1641	0.4579	0.2616
	34.875	65.778	25052	44083	34045	39749	0.3304	0.1394	0.3512	0.0745

Table 6-3 - *Continued*

4	46.688	53.630	11984	22003	13182	40627	0.4795	0.1682	0.2024	0.2457
	47.531	60.741	48099	15076	28355	26568	0.3972	0.4314	0.2474	0.0762
	52.594	58.370	19587	32394	8719	1694	0.0273	0.3327	0.3147	0.3164
	53.016	55.210	27190	18539	43529	22783	0.3271	0.3656	0.4046	0.4718
5	55.125	64.889	4856	5551	10727	35355	0.0375	0.4232	0.2642	0.1080
	48.375	70.222	30992	28930	42636	5479	0.0684	0.3985	0.1575	0.4152
	60.188	69.630	23864	15942	40182	14266	0.1609	0.3245	0.3316	0.3482
	53.438	67.852	579	46248	18537	33598	0.1917	0.2998	0.2249	0.1610
6	64.617	68.181	5426	29549	4033	48062	0.0122	0.1244	0.3438	0.1263
	66.410	69.761	41541	7778	9388	2100	0.2588	0.4206	0.4570	0.1121
	65.988	68.971	27856	21632	15231	8589	0.2298	0.4934	0.2325	0.0839
	65.145	70.288	14550	48351	20586	15618	0.4474	0.2699	0.3672	0.0698

Table 6-4 Program parameters – parallel-beam optimization problem

Number of Start Points	24 (ref. Table 6-3)
Start Point Definition	Haltonset
Max Jumps (Steep Descent)	2
Max Jumps (SQP method)	250
Min 1D Polynomial order	3
Max 1D Polynomial order	4
Max no. of 1D points	15
Min R2 allowable	0.9995
1D error allowable	5%
Minimum Des. Var. step size	0.001
Penalty Method	'Squared Constraint Method' - Eq. (5-7)
Penalty multiplier value	0.250

As indicated, the minimum design variable step size is selected to be 0.001. This could be considered an 'overly coarse' value in light of findings regarding the SQP method on Schwefel's test function (Figure 5-31 vs. Figure 5-44) where a smaller step size was needed to find an accurate result. (Note that both the order of magnitude of the design space of Schwefel's function and that of the spring locations for the parallel beam are 'similar'.) However, the results for the Combined search method met accuracy goals with the larger step size. Therefore, the minimum step size of 0.001, together with the use of the Combined search method is acceptable for the parallel beam study.

Results and Discussion

The FEM models of the parallel beam were meshed with 1-D (beam) elements at a specified nodal density of 200 NPI, resulting in approximately 48,000 to 53,000 total nodes per model depending on the family variant. A representative illustration of a loaded and constrained model is given in Figure 4-11.

Among the 24 start points, a total of 811 function evaluations were made; 798 during the Steepest Feasible Descent phase and 13 during the follow-on Direct (SQP) search. Each of these function evaluations consisted of three (3) FEM models (for each of 3 family variants) for a total of 2433 FEM model solutions. (Note that each FEM model solution included both static and harmonic analysis components.) A more detailed summary of function count by start point and optimization phase is depicted in Table 6-5 below.

Table 6-5 Results data – summary of function evaluation count

Start Point	Super-Element Model (Figure 6-1)	Optimization Phase	Total Function Evaluations
1	2	Steepest Feasible Descent (Coarse Search)	40
2	1		15
3	1		15
4	2		41
5	1		40
6	3		40
7	2		50
8	1		32
9	3		30
10	5		30
11	2		50
12	3		30
13	5		30
14	3		40
15	4		40
16	5		30
17	5		30
18	4		30
19	4		30
20	4		30
21	6		35
22	6		30
23	6		30
24	6		30
SFD Result of #4	2	Direct Search (SQP)	13
			811

Each function solve lasted approximately 28 minutes (including all of the 3 family variants) for a total solution time of approximately 15 days, 18.5 hours. This, together with the use of FEM as a function solver, supports that the parallel beam problem under study is an example of an EBBF.

A maximum of two (2) 'jumps' were specified for each of the SFD start points and a maximum of 250 'jumps' for the 'follow-on' SQP effort (ref. Table 6-4 above). Among the SFD search paths, each starting point's search could potentially end in one (1) of three (3) ways:

1) Convergence achieved – design variable movement less than defined threshold
2) Convergence not achieved, max jumps completed
3) No feasible results found

A maximum number of 15 1-D line search points was also specified per jump (ref. Table 6-4 above). This number, plus one search point for each degree of freedom (10 variables) during the calculation of Sn, yields a maximum 25 potential function evaluations per jump. A summary of the SFD results for each start point is provided in Table D-1 in Appendix D. This summary includes design variable definitions, function values (penalized), maximum constraint values and stopping criteria. Results for the SQP portion of the search are also given in Table D-1.

An optimum configuration for the parallel beam structure was identified by the program as the result of the search from global starting point number 4. As depicted in Table D-1, the result of the first part of this search (SFD) was identification of the optimal 'region of possibility' as the lowest penalized function value (1.4511). The SQP search then began from the end of jump 2 and continued until convergence (result also shown in Table D-1). The resultant optimal configuration for the parallel-beam then is given in Table 6-6 below. The FRF for tip deflection of the optimal configuration (resulting from Start Point #4) is shown in Figure 6-2.

Table 6-6 Design variable values for optimal configuration of parallel beam

Design Variable	Optimal Value
Spring Location L_1	29.245 in
Spring Location L_2	50.417 in
Spring Stiffness K_1	44060 lb_f/in
Spring Stiffness K_2	2520 lb_f/in
Spring Stiffness K_3	14409 lb_f/in

Table 6-6 - *Continued*

Spring Stiffness K_4	4601 lb_f/in
Damping Coefficient C_1	0.4126
Damping Coefficient C_2	0.3344
Damping Coefficient C_3	0.3063
Damping Coefficient C_4	0.2403

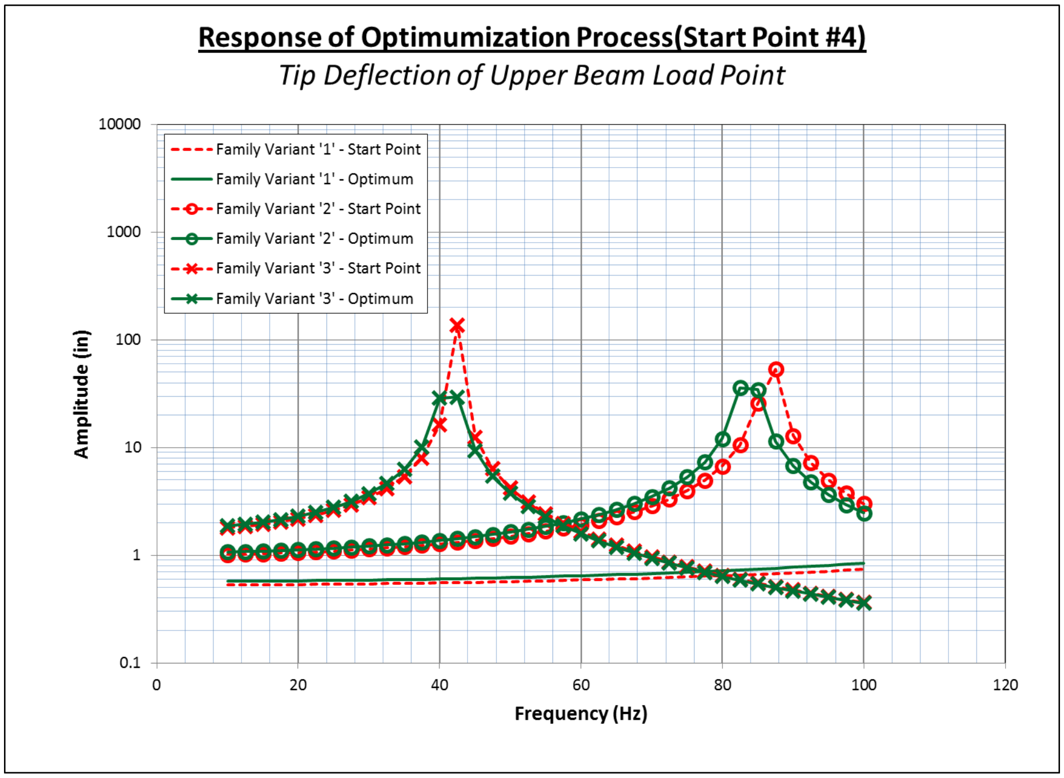


Figure 6-2 Harmonic tip response for optimal configuration of parallel beam

Figure 6-2 illustrates the ‘transition’ or ‘improvement’ of the FRF as a result of the optimization process for start point #4; ultimately identified as the optimal result. As shown, some improvement in response for family variants ‘2’ and ‘3’ was made, but the response for variant ‘1’ was made slightly worse in the process. This ‘trade-off’ or ‘compromise’ in the overall design was necessary and planned in order to reduce the selected objective function for consideration of the family as a whole, including the range of responses between the variants. As depicted in Table D-1, the overall (family) objective function corresponding to the changes of Figure 6-2 reduced from 2.4054 to 1.4511 (39.67% reduction). This resulting penalized function value (1.4511) is significantly lower than the local optimums resulting from some other start points. For example, start point 6 resulted in a function value of 14.0065 (a reduction of 69.46% from its initial value of 45.4097) and start point 14 in a value of 16.1061 (a reduction of 59.26% from its starting value of 16.1061). As a comparison to the optimal solution of Figure 6-2, the FRF 'improvement' path of Start Point #6 is provided in Figure 6-3 below. As shown, even though the natural frequency response for variant '2' of start point 6 (approx. 90 Hz) is reduced through the process by approximately 81%, this 'optimal' response is still significantly higher than the response (approx. 85 Hz) of the optimal configuration in Figure 6-2; improving by approximately 64% additionally. When viewed another way, the optimum result for start point 4 (Figure 6-2) at ~ 85 Hz is improved in comparison to the initial configuration for start point 6 (Figure 6-3) by approximately 99%.

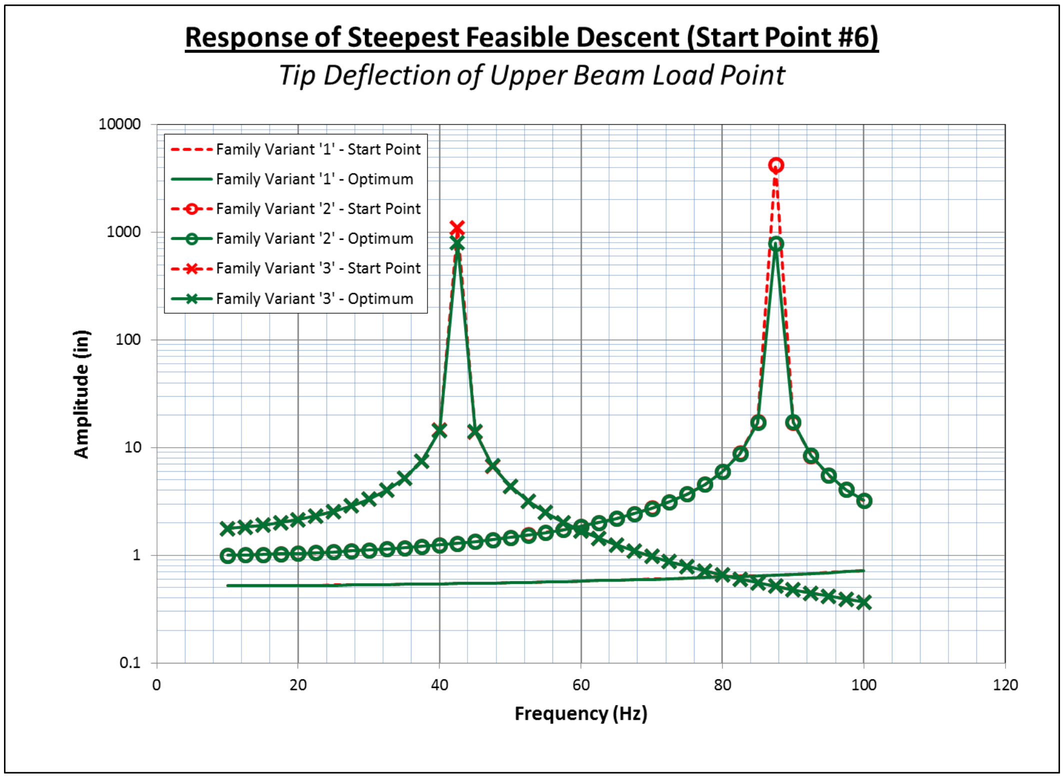


Figure 6-3 Harmonic tip response for start point #6

Even with these improvements, natural frequencies remain within the frequency range of interest for variants '2' and '3' of the resultant design. An original goal of this study was to reduce the flexural response for the parallel beam system given an assumed 'constraint' that the support elements for the upper beam could not vary in their location. This was based upon the premise that a platform-style design may include such rigidity in its requirements among the variants so as to preclude otherwise influential variables from being part of the optimization process. Figure 6-4 and Figure 6-5 illustrate the mode shapes of the natural frequencies for family variants ‘2’ and ‘3’ (of the optimal design from Figure 6-2) respectively and give insight as to why the optimization process did not eliminate those natural frequencies. In both cases, the main flexure is bending (mode 1) of the upper beam which is dominated by upper beam geometry and support location (discussed further later).

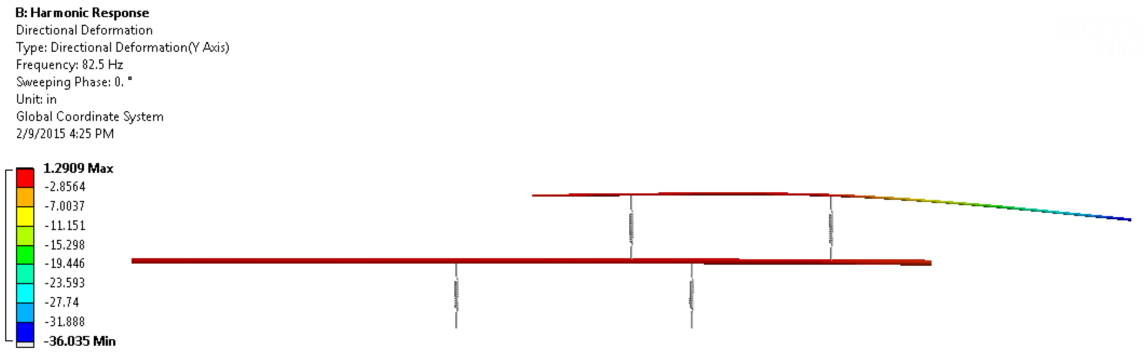


Figure 6-4 Mode shape of family variant '2', 82.5 Hz

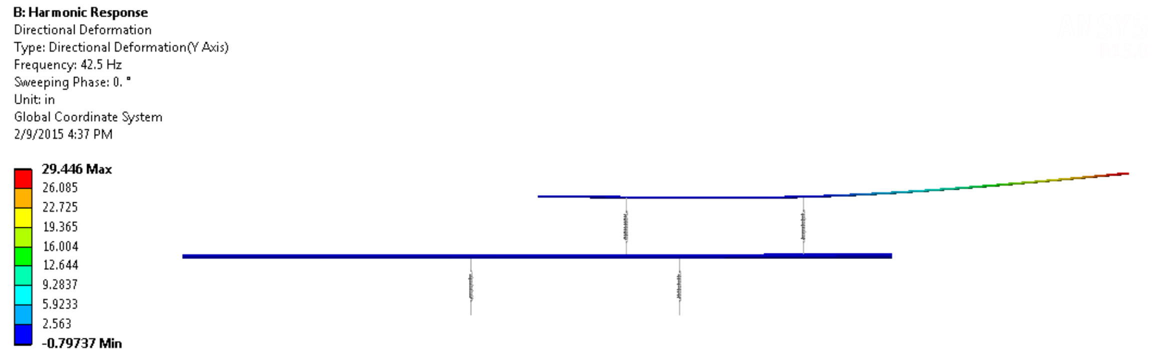


Figure 6-5 Mode shape of family variant '3', 42.5 Hz

This result illustrates a point made in the original motivation for this work; sometimes design constraints exist which ‘forces’ the system to ‘operate’ at or near a natural frequency(s).

In this example, the location of the upper beam's supports are fixed as are the variant beam's geometry and therefore not subject to the optimization effort. Without modification of these, any attempt to modify the upper beam's natural frequency is muted.

One potentially influential variable regarding the upper beam's response that was included in the optimization however are the spring stiffness of K_3 and K_4. Potentially, a significant change of stiffness there may have some influence to move the natural frequency. The data of Table D-1 indicates that these two spring stiffnesses were not modified during the optimization path of start point 4. As discussed later in this chapter, the search vector Sn provides insight as to the sensitivity of the various design variables. Since these stiffnesses did not change, it is logical to conclude that they had at most a minor effect upon the specified function value. This is discussed in more detail later in this chapter. Further investigation into the issue of the upper spring stiffnesses could yield fruit, either with modification of the parameters of the study, or even with an element of the algorithm that could improve effectiveness. For example, a benefit could potentially be achieved by 'splitting' the optimization into multiple runs so that the number of design variables is reduced for each run. This could aid in the sensitivity given to a particular variable, but is not pursued further within this work.

The result of the optimization effort discussed here is considered successful, in that the familial response was minimized given constraints of the problem and with consideration for the response of all family variants across the frequency range. This is demonstrated through comparison of two (2) representative results above (Figure 6-2 and Figure 6-3) as well as detailed data describing minimization of the objective function as discussed throughout the remainder of this chapter.

Table D-1 depicts that convergence was achieved during the SFD search for five (5) of the 24 start points and no feasible results were found for two (2) of them. The remainder (17) did result in feasible solutions but did not achieve convergence before completing the maximum specified number of jumps. As mentioned previously, the best solution from the SFD results was identified as being from starting point #4, which is one of those that did achieve convergence within two (2) SFD jumps. The SQP search began from this result, resulting in no appreciable change before achieving its convergence criteria. This is depicted in Figure 6-6 thru Figure 6-8 below.

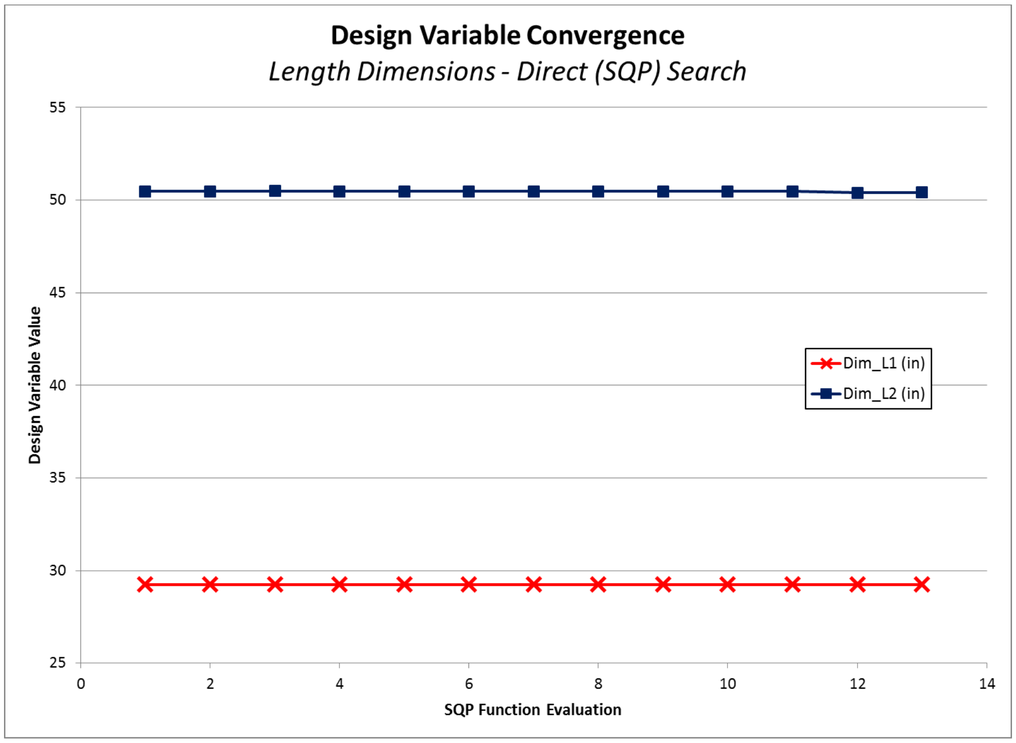


Figure 6-6 SQP Variable Convergence - Length Dimensions

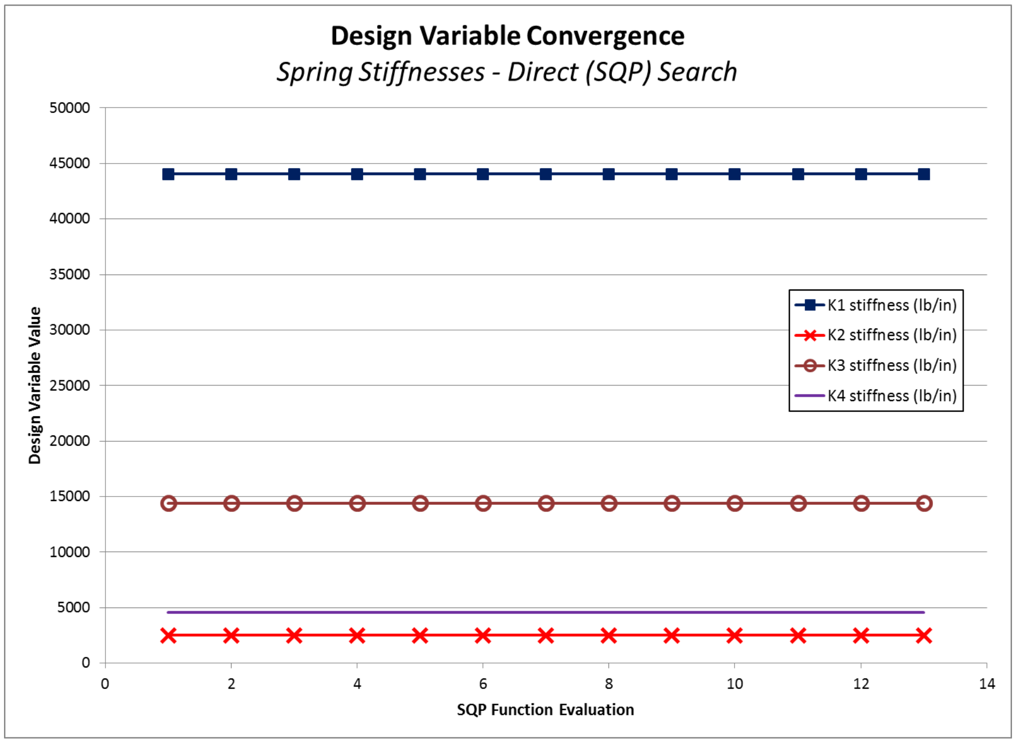


Figure 6-7 SQP Variable Convergence - Spring Stiffnesses

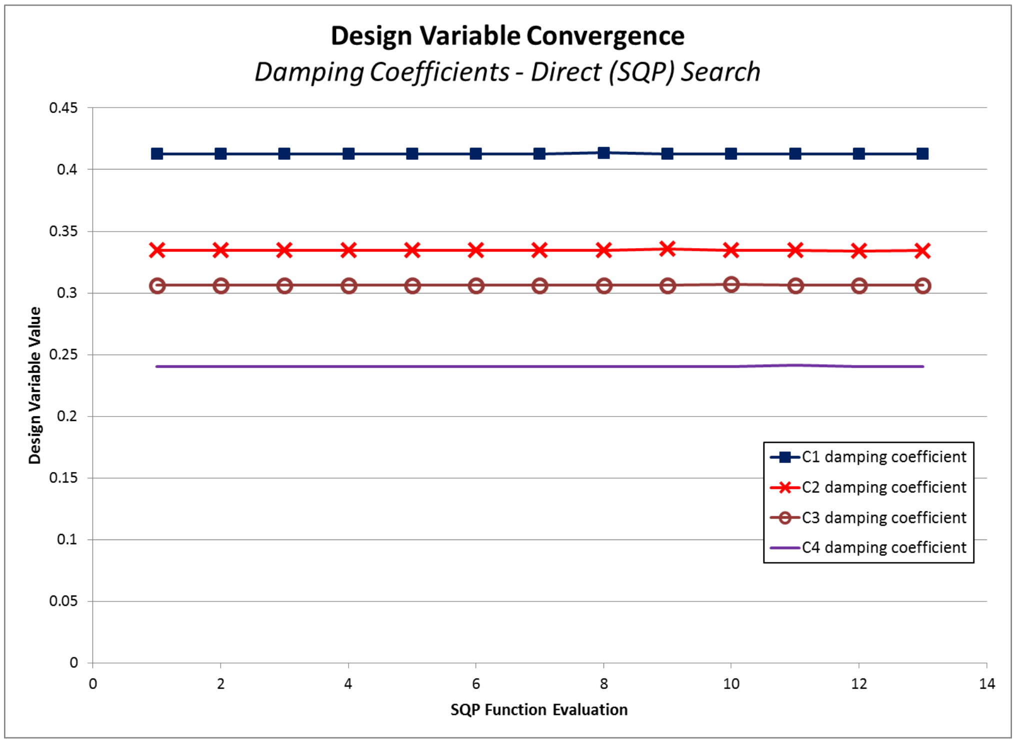


Figure 6-8 SQP Variable Convergence - Damping Coefficients

Table D-2 (Appendix D) depicts the Sn vector used during each jump of the SFD searches. This data can be interpreted as a form of 'sensitivity analysis', giving indication as to which variables were the most significant with respect to improvement in the function result at the point of the jump's X0 position. As shown, six (6) jumps had a Sn= [0].

By a comparison of the Sn data for these six (6) jumps (having Sn= [0]) against the data of Table D-1, it can be seen that starting points #2 and #3 resulted in $g_{max}>0$; indicating an infeasible solution. A penalized function value is recorded for the X0 point for each of these, but a function value of '0' is recorded as a result of the (infeasible) jump. This indicates that a descending vector (Sn) could not be identified which pointed to feasible space. The other four (4) jumps resulted in feasible solutions and the function value at the respective X0 positions was identified and considered as part of the overall search for a global minimum. In these four (4) cases, the reason for Sn= [0] is that the point was at a local minimum; as identified by the fact that both Sn= [0] and the solution met convergence criteria.

Considering Sn data from Table D-2 as a 'sensitivity metric', it can be shown that the position of the lower support springs (L_1 and L_2) were by far the most influential at reducing the objective function. In fact, these two (2) variables accounted for at least 80% of the Sn vector's magnitude for all but four (4) of the 46 total jumps (jumps with Sn=[0] excluded). This is computed by comparing the absolute magnitudes of the variables in question to the sum of the absolute magnitudes for all variables, as shown in Eq. (6-8) below.

$$Effectiveness_i = \frac{\sum_i |S_n(i)|}{\sum_j |S_n(j)|} \tag{6-8}$$

$$where\ i\ =\ \text{design variables in question}$$

$$j\ =\ \text{all design variables}$$

By comparison, for these 42 jumps where spring location dominated, spring stiffness design variables contributed less than 1.3% and damping coefficient up to 19.7% of the Sn vector's magnitude. For the four (4) jumps where spring location did not dominate, spring stiffness contributed between 3.8% and 49.4% while damping coefficient contributed between 50.6% and 96.2%.

These results indicate that, for the parallel-beam problem overall, the location of the support springs has the strongest effect on reduction of the objective function followed by damping coefficient and lastly by support spring stiffness.

At a high level, this is supported by observations from theory. As demonstrated with the damped, classical SDOF (Figure 3-2) the frequency ratio (r) has the most profound effect on the magnitude of the response, followed by the damping coefficient (when the frequency ratio selected is near to '1' indicating resonance).

In the parallel beam study, the location of the support springs showed the most significant effect on reduction of the objective function. This can be compared to the length of a cantilever beam with respect to its natural frequency response. As shown in Eq. (4-12), length can have a significantly stronger effect upon the natural frequency of a cantilever beam than other aspects of stiffness such as cross-sectional size or material strength. Because of this strongly influential role regarding natural frequency, the location of the spring supports also has a strong effect upon harmonic response magnitude, particularly when the natural frequency is near to (or within) the range of frequencies of interest. As shown in Figure 6-2, family variants '2' and '3' of the optimal solution both include a natural frequency within the scope of frequencies evaluated. Therefore it is logical to expect that support spring location would be highly influential for the parallel-beam problem under study.

Second to location of the support springs, theory shows that damping can have a profound effect upon the harmonic response where that response is near to a natural frequency of the system. (Ref. damped SDOF example of Figure 3-2. Since natural frequencies are shown to exist within the frequency range of interest for even the most optimum result, it is logical to conclude also that damping coefficient would have a pronounced effect upon the model.

These data also support that the decision to sensitize Sn to the scale of each of the design variables was a productive choice; as was predicted from the test results of Table 5-2.

Table D-3 depicts the results of the 1-D line searches for each of the SFD starting points and jumps. This data is significant because it gives insight into whether the line search algorithm was successful in its intended purpose.

Of the 46 jumps completed (across the 24 SFD start points), all but five (5) achieved acceptability criteria (R^2>0.9995 and error<±5.0%) per Table 6-4. Two (2) of these 'non-converged' searches (start point 7, jump 1 and start point 11, jump 1) reported R^2 values of 0.99631 and 0.99816 respectively but did achieve the error criteria (-0.3897% and -0.3671%). For each of these, the maximum number of evaluations (25) was conducted for the jump and the line search optimum was identified to be mid-span between X0 and the maximum length of the search vector.

Two (2) of the jumps (start points 2 and 3) found that all of the initial set of 1-D search points resulted in infeasible results. If, as in this case, the program determines with high confidence (R^2 > 0.98) along the full length of the Sn vector that none of the points will have a g_{max}<=0, then no further investigation takes place along that vector. It is recognized that potential exists for a feasible result to be found elsewhere along the vector, but this is not investigated given consideration for EBBF's and the predicted low-likelihood of a feasible result.

The remaining 'non-converged' search (start point 15, jump 2) reported a correlation coefficient of -∞. This indicates that there although some points along Sn were feasible, there were not enough from which to construct a polynomial after having evaluated the maximum allowable number of points. In this event, the algorithm selects the minimum solved value among the feasible points as the optimum. As shown in Table D-3 , this optimal point was located at the maximum length of the 1-D search vector. Given that this optimal point was a 'solved value', and that no polynomial existed, the error for this case is programmatically to be reported as '0', which was the case for this result.

Seven (7) additional searches also utilized the maximum number of function evaluations. In these cases however, the resulting polynomial met both R^2 and error acceptability criterion for polynomial accuracy. These results are consistent with behavior observed for some of the more complex test functions (ref. Figure 5-9 thru Figure 5-13) where ‘outlier’ points far from the observed minima were removed in order to improve the polynomial’s accuracy near the minima; indicating that the response along the search vector was likely multimodal (as with the test cases).

The specified minimum polynomial order for the line searches was set at three (3) (Table 6-4) which, together with 10 design variables, yields a minimum number of points allowable for the line searches of 14 (one (1) more than the polynomial order plus one (1) for each design variable). Of the 46 total jumps, all used more than this minimum number of points; indicating that none of the line searches met acceptance criteria (both R^2 and error) at the minimum polynomial order. Interestingly, most (32 of 46) of the line searches used 15 points per jump, indicating that both R^2 and error were satisfied with the addition of only one (1) more than the minimum number of points.

When the algorithm adds a point to the line search matrix, it also increments the polynomial order (to a prescribed maximum). Considering this, it is likely that the response for these 32 searches was more complex than a simple unimodal polynomial (as with DeJong’s and Rosenbrock’s search results in Figure 5-7 and Figure 5-8 respectively). The addition of an additional point, and more importantly the increase in polynomial order from ‘3’ to ‘4’ in these cases was sufficient to improve the polynomial’s accuracy to acceptable levels.

Six (6) of the 46 searches resulted in a 1-D line search optimum at the X0 position (x=0 along the search vector). As can be seen by a comparison of Table D-3 and Table D-2, those searches where the optimum was located at X0 are those where Sn= [0]. A total of 10 other line searches resulted in an optimum located ‘mid-span’ between the X0 and maximum length of the search path (nearest constraint bound). The remaining 30 line searches identified the optimum result at the maximum length of the search path. This indicates that the search algorithm did utilize the full length of the search vector during evaluations.

Considering all of these results, together with the algorithm’s performance on the various test functions (Figure 5-7 thru Figure 5-13), it appears that a variety of modalities existed with the polynomial approximations performed during the parallel beam optimization; requiring varying numbers of point evaluations and approximation methods.

Chapter 7

Summary and Conclusions

In the course of this work, a parallel beam structure with intermediate elastomeric supports was investigated to find the optimal placement and configuration of the supports (stiffness and damping coefficients) in order to minimize response to harmonic loading subject to multiple design variable and results-based constraints. A key objective of this effort was the development of an optimization strategy for use with such systems that is an effective compromise between finding an 'acceptable global optimum' and the overall design cost. The selection of the harmonically loaded parallel beam for this problem highlights a challenging consideration for a global optimization algorithm; that of finding a solution for a response that is likely multi-modal. The problem under study was further complicated by defining the parallel beam to be a 'platform-style' product with three (3) family variants to the geometry; each of which was considered as part of a 'compromise solution' to the overall optimization effort. Also, key variables defining the upper (variant) beam were 'fixed' as might be experienced in a practical application, preventing an ideal optimization of the upper beam's flexure from being accomplished. For these reasons, an optimization method was sought that would be appropriate for the practical solution of Expensive Black Box Functions (EBBF).

Using the proposed optimization method, an optimum set of design parameters for the platform-style, parallel-beam structure was identified which showed significant improvement in the desired characteristics. And, in the process, the algorithm was demonstrated to be useful for an EBBF with multi-modal response. Overall, this work is comprised of four (4) major elements. They are:

1. A review of theory and literature regarding structural dynamics as relevant to the problem under study, as well as optimization methodologies that may be appropriate for its solution as an EBBF.

2. Development and validation of an FEM model of the parallel beam system, including both static and dynamic aspects, as are needed in the solution of the constrained optimization problem.
3. Development and characterization of the proposed optimization algorithm, written in MATLAB ® language.
4. Optimization of the parallel beam problem using the proposed algorithm to coordinate a series of 'external function solutions' of the FEM model using ANSYS ® software.

The proposed optimization algorithm leverages benefits of two (2) methodologies common in the field of multi-disciplinary optimization; the first-order method of Steepest Descent and the direct search method of Sequential Quadratic Programming (SQP). While each has strengths and weaknesses regarding solution of an EBBF with multi-modal response, it is proposed that by combining and modifying them in a particular way that an effective tool can be created. In the process, the method of Steepest Descent is modified to be useful as a constrained search tool (termed the 'Method of Steepest Feasible Descent'), and objective functions are created that allow tools normally limited to application on single objective problems to be used for a multi-objective purpose. Prior to being used for solution of the parallel beam problem, the proposed optimization algorithm is challenged against four (4) standard test functions to evaluate its effectiveness on responses with a variety of modalities, and to support whether the theory of a 'combined' search methodology is productive as compared to use of the component methodologies independently.

Several conclusions are drawn as a result of this work. They are:

1. Results support that the proposed 'combined' search methodology does in fact leverage the best of each component methodology in order to find a global optimum that is both more effective (accuracy of result against theoretical solution) and more efficient (number of function evaluations required) against the standardized test functions where a high level of modality is present. In this way, the proposed method is demonstrated to be

more suitable for EBBF’s like the parallel-beam problem than either component method used independently.

2. Results support that the proposed ‘combined’ search methodology was not practically more effective or more efficient when used for simple, unimodal responses than either of the component methodologies used independently. However, the data also supports that use of the proposed method for such a response was no worse than either of the component methodologies.
3. The method of Steepest Feasible Descent, modified as part of this work from the classical unconstrained Steepest Descent methodology, is demonstrated to be useful for constrained searches. Key modifications are as follows:
    a. The line search was augmented to include both deterministic and programmatic penalty function effects.
    b. A sensitivity-based search vector was utilized to improve response in situations where the various design variables have significantly different orders of magnitude among their values. The effectiveness of this approach was demonstrated and confirmed on a Classical damped SDOF theoretical system.
    c. Programmatic identification of the search vector includes consideration for feasibility of the targeted design space, including modification as necessary to achieve feasible space.
4. Stopping criteria (step size) is demonstrated to be important to accuracy of the result, especially for multi-modal responses and particularly with the SQP methodology. A significance of this conclusion is that the selected minimum step size may need to be significantly smaller than what otherwise might be expected as important. For example, a tolerance of ±1% on a dimension may normally be considered significant. But in light of a multi-modal response, the stopping criteria may need to be significantly smaller to achieve an acceptable result; particularly for the SQP method. This of course carries the burden of longer solution times. However, use of the ‘coarse’ step size was

demonstrated to be acceptable with the proposed ‘Combined’ search methodology, further supporting its utility for EBBFs.

5. FEM validation data support that the practice of modeling beam structures with 1-D beam elements can be significantly more effective than modeling them with 3-D solid elements in terms of computing resources required and the potential for numerical problems with the larger models.
6. An optimal solution to minimize flexural response of the platform-style parallel beam problem was successfully identified using the proposed method, resulting in recommended values for each of the ten (10) design variables. Even though optimized, the solution still included natural frequencies within the frequency range of interest for two (2) of the three (3) family variants because of the assumed constraint that the upper beam’s geometry and support locations were fixed as part of the product family definition. Variation of the (other) allowed variables however, provided for an improvement in the response for the family product as intended. A review of the FRF plots of the optimal solution compared to other, less ideal solutions, show a reduction in the peak resonant response of between 33 and 99%, with a reduction in the objective function value of between 40 and 69% as a result of the method. This indicates success with respect to the goals of the study. Although significantly improved, natural frequencies do, however, remain within the operating range; highlighting an unfortunate aspect of practical design and a motivation for this work. Sometimes product constraints exist that preclude an ideal design. In this case, because the upper beam’s properties were fixed, natural frequencies could not be avoided for some of the family variants within the frequency range of interest. The proposed optimization method did, however, identify a set of design variables that minimized the responses significantly.
7. Results indicate that, for minimization of harmonic response across all of the family-variants for the parallel-beam problem, the location of the support springs demonstrate the strongest effect on reduction of the objective function followed by damping coefficient

and lastly by support spring stiffness. This aligns with theoretical predictions. Further, this alignment is attributed to the use of ‘sensitivity based’ search vectors in the proposed optimization method.

Chapter 8

Opportunities for Further Research

Following are areas where further research may yield further improvement to the current work.

1. When used with a problem requiring Super-Element configurations such as this parallel beam structure, a limitation exists where the solution cannot ‘cross’ between Super-Elements in search of an optimal solution. This requires that multiple start points be defined within each Super-Element configuration, adding to the overall time of the analysis. Research into methods of eliminating this constraint could yield significant benefit in reduced analysis time through a reduced number of start points.
2. The effects of the number of design variables upon the efficiency of the proposed method are not well understood. Research into these effects and predictive guidance to the user would be beneficial.
3. This proposed method is deterministic in nature and does not consider uncertainty of the design variables as part of the process. As illustrated in the parallel beam example, several minima were identified with similar objective function values; but with no guidance beyond the function values as to which is the better option. Research into incorporation of reliability based design optimization (RBDO) concepts could add important, additional capability to the process.

Appendix A

FEM Model Validation Data

Table A- 1 Mesh sensitivity and validation results - static deflection, cantilevered beam

Total Nodes	Theoretical Result Eq (4-17) (in)	FEM Result (in)		Error (%)	
		1D (beam)	3D (solid)	1D (beam)	3D (solid)
9	-3.315	-0.290	- Note 1 -	91.25%	- Note 1 -
17		-0.524	- Note 1 -	84.21%	- Note 1 -
33		-0.895	- Note 1 -	73.01%	- Note 1 -
97		-1.734	- Note 1 -	47.68%	- Note 1 -
193		-2.277	- Note 1 -	31.33%	- Note 1 -
261		- Note 1 -	-1.312	- Note 1 -	60.43%
385		-2.701	- Note 1 -	18.53%	- Note 1 -
501		- Note 1 -	-1.496	- Note 1 -	54.86%
769		-2.979	- Note 1 -	10.14%	- Note 1 -
1461		- Note 1 -	-1.796	- Note 1 -	45.82%
1921		-3.175	- Note 1 -	4.22%	- Note 1 -
2901		- Note 1 -	-1.982	- Note 1 -	40.21%
3841		-3.247	- Note 1 -	2.06%	- Note 1 -
5781		- Note 1 -	-2.160	- Note 1 -	34.85%
9601		-3.291	- Note 1 -	0.72%	- Note 1 -
11541		- Note 1 -	-2.326	- Note 1 -	29.85%
19201		-3.309	- Note 1 -	0.17%	- Note 1 -
38401		-3.318	- Note 1 -	-0.08%	- Note 1 -
76801		-3.337	- Note 1 -	-0.67%	- Note 1 -
147953		- Note 1 -	-2.867	- Note 1 -	13.50%
895065		- Note 1 -	-3.059	- Note 1 -	7.74%
5860976		- Note 1 -	-3.175	- Note 1 -	4.23%

Note 1: Total nodes differ between 1D and 3D models for given nodal density. Therefore, results not available for this combination of model and total nodes.

Table A- 2 Mesh sensitivity and validation results - static deflection, simply supported beam

Total Nodes	Theoretical Result Eq (4-17) (in)	FEM Result (in)		Error (%)	
		1D (beam)	3D (solid)	1D (beam)	3D (solid)
9	-0.207	-0.096	- Note 1 -	53.53%	- Note 1 -
17		-0.132	- Note 1 -	36.50%	- Note 1 -
33		-0.161	- Note 1 -	22.27%	- Note 1 -
97		-0.189	- Note 1 -	8.61%	- Note 1 -
193		-0.198	- Note 1 -	4.41%	- Note 1 -
261		- Note 1 -	-0.178	- Note 1 -	14.25%
385		-0.203	- Note 1 -	2.16%	- Note 1 -
501		- Note 1 -	-0.183	- Note 1 -	11.57%
769		-0.205	- Note 1 -	0.99%	- Note 1 -
1461		- Note 1 -	-0.190	- Note 1 -	8.38%
1921		-0.207	- Note 1 -	0.28%	- Note 1 -
2901		- Note 1 -	-0.193	- Note 1 -	6.82%
3841		-0.207	- Note 1 -	0.04%	- Note 1 -
5781		- Note 1 -	-0.196	- Note 1 -	5.54%
9601		-0.207	- Note 1 -	-0.10%	- Note 1 -
11541		- Note 1 -	-0.198	- Note 1 -	4.50%
19201		-0.208	- Note 1 -	-0.15%	- Note 1 -
38401		-0.208	- Note 1 -	-0.17%	- Note 1 -
76801		-0.208	- Note 1 -	-0.18%	- Note 1 -
147953		- Note 1 -	-0.204	- Note 1 -	1.61%
895065		- Note 1 -	-0.205	- Note 1 -	0.88%
5860976		- Note 1 -	-0.206	- Note 1 -	0.47%

Note 1: Total nodes differ between 1D and 3D models for given nodal density. Therefore, results not available for this combination of model and total nodes.

Table A- 3 Mesh sensitivity and validation results - modal frequencies, mode 1

Total Nodes	Theoretical Result Eq (4-13) (Hz)	FEM Result (Hz)		Error (%)	
		1D (beam)	3D (solid)	1D (beam)	3D (solid)
9	59.349	59.319	- Note 1 -	91.25%	- Note 1 -
17		59.313	- Note 1 -	84.21%	- Note 1 -
33		59.313	- Note 1 -	73.01%	- Note 1 -
97		59.313	- Note 1 -	47.68%	- Note 1 -
193		59.313	- Note 1 -	31.33%	- Note 1 -
261		- Note 1 -	59.481	- Note 1 -	-0.22%
385		59.313	- Note 1 -	18.53%	- Note 1 -
501		- Note 1 -	59.456	- Note 1 -	-0.18%
769		59.313	- Note 1 -	10.14%	- Note 1 -
1461		- Note 1 -	59.455	- Note 1 -	-0.18%
1921		59.313	- Note 1 -	4.22%	- Note 1 -
2901		- Note 1 -	59.455	- Note 1 -	-0.18%
3841		59.314	- Note 1 -	2.06%	- Note 1 -
5781		- Note 1 -	59.455	- Note 1 -	-0.18%
9601		59.313	- Note 1 -	0.72%	- Note 1 -
11541		- Note 1 -	59.454	- Note 1 -	-0.18%
19201		59.253	- Note 1 -	0.17%	- Note 1 -
38401		59.197	- Note 1 -	-0.08%	- Note 1 -
76801		59.438	- Note 1 -	-0.67%	- Note 1 -
147953		- Note 1 -	59.407	- Note 1 -	-0.10%
895065		- Note 1 -	54.324	- Note 1 -	8.47%
5860976		- Note 1 -	54.894	- Note 1 -	7.51%

Note 1: Total nodes differ between 1D and 3D models for given nodal density. Therefore, results not available for this combination of model and total nodes.

Table A- 4 Mesh sensitivity and validation results - modal frequencies, mode 2

Total Nodes	Theoretical Result Eq (4-13) (Hz)	FEM Result (Hz)		Error (%)	
		1D (beam)	3D (solid)	1D (beam)	3D (solid)
9	371.937	373.072	- Note 1 -	-0.31%	- Note 1 -
17		371.764	- Note 1 -	0.05%	- Note 1 -
33		371.679	- Note 1 -	0.07%	- Note 1 -
97		371.673	- Note 1 -	0.07%	- Note 1 -
193		371.673	- Note 1 -	0.07%	- Note 1 -
261		- Note 1 -	374.980	- Note 1 -	-0.82%
385		371.673	- Note 1 -	0.07%	- Note 1 -
501		- Note 1 -	372.675	- Note 1 -	-0.20%
769		371.673	- Note 1 -	0.07%	- Note 1 -
1461		- Note 1 -	372.562	- Note 1 -	-0.17%
1921		371.673	- Note 1 -	0.07%	- Note 1 -
2901		- Note 1 -	372.561	- Note 1 -	-0.17%
3841		371.674	- Note 1 -	0.07%	- Note 1 -
5781		- Note 1 -	372.561	- Note 1 -	-0.17%
9601		371.673	- Note 1 -	0.07%	- Note 1 -
11541		- Note 1 -	372.560	- Note 1 -	-0.17%
19201		371.664	- Note 1 -	0.07%	- Note 1 -
38401		371.653	- Note 1 -	0.08%	- Note 1 -
76801		371.675	- Note 1 -	0.07%	- Note 1 -
147953		- Note 1 -	371.916	- Note 1 -	0.01%
895065		- Note 1 -	368.503	- Note 1 -	0.92%
5860976		- Note 1 -	373.682	- Note 1 -	-0.47%

Note 1: Total nodes differ between 1D and 3D models for given nodal density. Therefore, results not available for this combination of model and total nodes.

Table A- 5 Mesh sensitivity and validation results - modal frequencies, mode 3

Total Nodes	Theoretical Result Eq (4-13) (Hz)	FEM Result (Hz)		Error (%)	
		1D (beam)	3D (solid)	1D (beam)	3D (solid)
9	1041.0	1069.440	- Note 1 -	-2.73%	- Note 1 -
17		1042.474	- Note 1 -	-0.14%	- Note 1 -
33		1040.662	- Note 1 -	0.03%	- Note 1 -
97		1040.539	- Note 1 -	0.04%	- Note 1 -
193		1040.538	- Note 1 -	0.04%	- Note 1 -
261		- Note 1 -	1067.999	- Note 1 -	-2.59%
385		1040.538	- Note 1 -	0.04%	- Note 1 -
501		- Note 1 -	1044.150	- Note 1 -	-0.30%
769		1040.538	- Note 1 -	0.04%	- Note 1 -
1461		- Note 1 -	1043.038	- Note 1 -	-0.20%
1921		1040.538	- Note 1 -	0.04%	- Note 1 -
2901		- Note 1 -	1043.031	- Note 1 -	-0.20%
3841		1040.538	- Note 1 -	0.04%	- Note 1 -
5781		- Note 1 -	1043.030	- Note 1 -	-0.20%
9601		1040.538	- Note 1 -	0.04%	- Note 1 -
11541		- Note 1 -	1043.029	- Note 1 -	-0.19%
19201		1040.535	- Note 1 -	0.04%	- Note 1 -
38401		1040.532	- Note 1 -	0.04%	- Note 1 -
76801		1040.539	- Note 1 -	0.04%	- Note 1 -
147953		- Note 1 -	1041.204	- Note 1 -	-0.02%
895065		- Note 1 -	1042.351	- Note 1 -	-0.13%
5860976		- Note 1 -	1038.798	- Note 1 -	0.21%

Note 1: Total nodes differ between 1D and 3D models for given nodal density. Therefore, results not available for this combination of model and total nodes.

Table A- 6 Mesh sensitivity and validation results - modal frequencies, mode 4

Total Nodes	Theoretical Result Eq (4-13) (Hz)	FEM Result (Hz)		Error (%)	
		1D (beam)	3D (solid)	1D (beam)	3D (solid)
9	2041.0	2265.202	- Note 1 -	-10.98%	- Note 1 -
17		2052.656	- Note 1 -	-0.57%	- Note 1 -
33		2039.511	- Note 1 -	0.07%	- Note 1 -
97		2038.597	- Note 1 -	0.12%	- Note 1 -
193		2038.586	- Note 1 -	0.12%	- Note 1 -
261		- Note 1 -	2167.207	- Note 1 -	-6.18%
385		2038.585	- Note 1 -	0.12%	- Note 1 -
501		- Note 1 -	2049.203	- Note 1 -	-0.40%
769		2038.585	- Note 1 -	0.12%	- Note 1 -
1461		- Note 1 -	2043.524	- Note 1 -	-0.12%
1921		2038.585	- Note 1 -	0.12%	- Note 1 -
2901		- Note 1 -	2043.493	- Note 1 -	-0.12%
3841		2038.585	- Note 1 -	0.12%	- Note 1 -
5781		- Note 1 -	2043.489	- Note 1 -	-0.12%
9601		2038.585	- Note 1 -	0.12%	- Note 1 -
11541		- Note 1 -	2043.486	- Note 1 -	-0.12%
19201		2038.584	- Note 1 -	0.12%	- Note 1 -
38401		2038.582	- Note 1 -	0.12%	- Note 1 -
76801		2038.586	- Note 1 -	0.12%	- Note 1 -
147953		- Note 1 -	2039.886	- Note 1 -	0.05%
895065		- Note 1 -	2039.504	- Note 1 -	0.07%
5860976		- Note 1 -	2038.149	- Note 1 -	0.14%

Note 1: Total nodes differ between 1D and 3D models for given nodal density. Therefore, results not available for this combination of model and total nodes.

Table A- 7 Harmonic mesh convergence error - 1D elements

Total Nodes	FEM Result						Convergence Error (%)					
	Load Point Deflection (vertical - in)		Sum of Forces (lbf)				Load Point Deflection (vertical)		Sum of Forces			
	Sum	Range	Spring 1	Spring 2	Spring 3	Spring 4	Sum	Range	Spring 1	Spring 2	Spring 3	Spring 4
48	21.716	0.165	23.891	45.283	91.705	105.376						
70	21.716	0.165	23.891	45.283	91.706	105.376	0.00%	-0.01%	0.00%	0.00%	0.00%	0.00%
126	21.716	0.165	23.891	45.283	91.706	105.376	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
242	21.716	0.165	23.891	45.283	91.706	105.376	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
482	21.716	0.165	23.891	45.283	91.706	105.376	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
962	21.716	0.165	23.891	45.283	91.706	105.376	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1922	21.716	0.165	23.891	45.283	91.706	105.376	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
4802	21.716	0.165	23.891	45.283	91.705	105.376	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
9602	21.716	0.165	23.891	45.283	91.706	105.377	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
24002	21.716	0.165	23.891	45.282	91.704	105.374	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
48002	21.717	0.165	23.892	45.284	91.707	105.378	0.00%	-0.03%	0.00%	0.00%	0.00%	0.00%
96002	21.719	0.165	23.894	45.290	91.732	105.410	-0.01%	-0.01%	-0.01%	-0.01%	-0.03%	-0.03%
120002	21.707	0.165	23.882	45.261	91.637	105.291	0.05%	0.28%	0.05%	0.06%	0.10%	0.11%
160004	21.700	0.166	23.871	45.240	91.632	105.293	0.04%	-0.62%	0.04%	0.05%	0.01%	0.00%
192002	21.719	0.166	23.895	45.292	91.691	105.354	-0.09%	-0.56%	-0.10%	-0.12%	-0.06%	-0.06%
240002	21.705	0.166	23.880	45.256	91.683	105.355	0.07%	0.51%	0.06%	0.08%	0.01%	0.00%

Table A- 8 Harmonic mesh convergence error - 3D elements

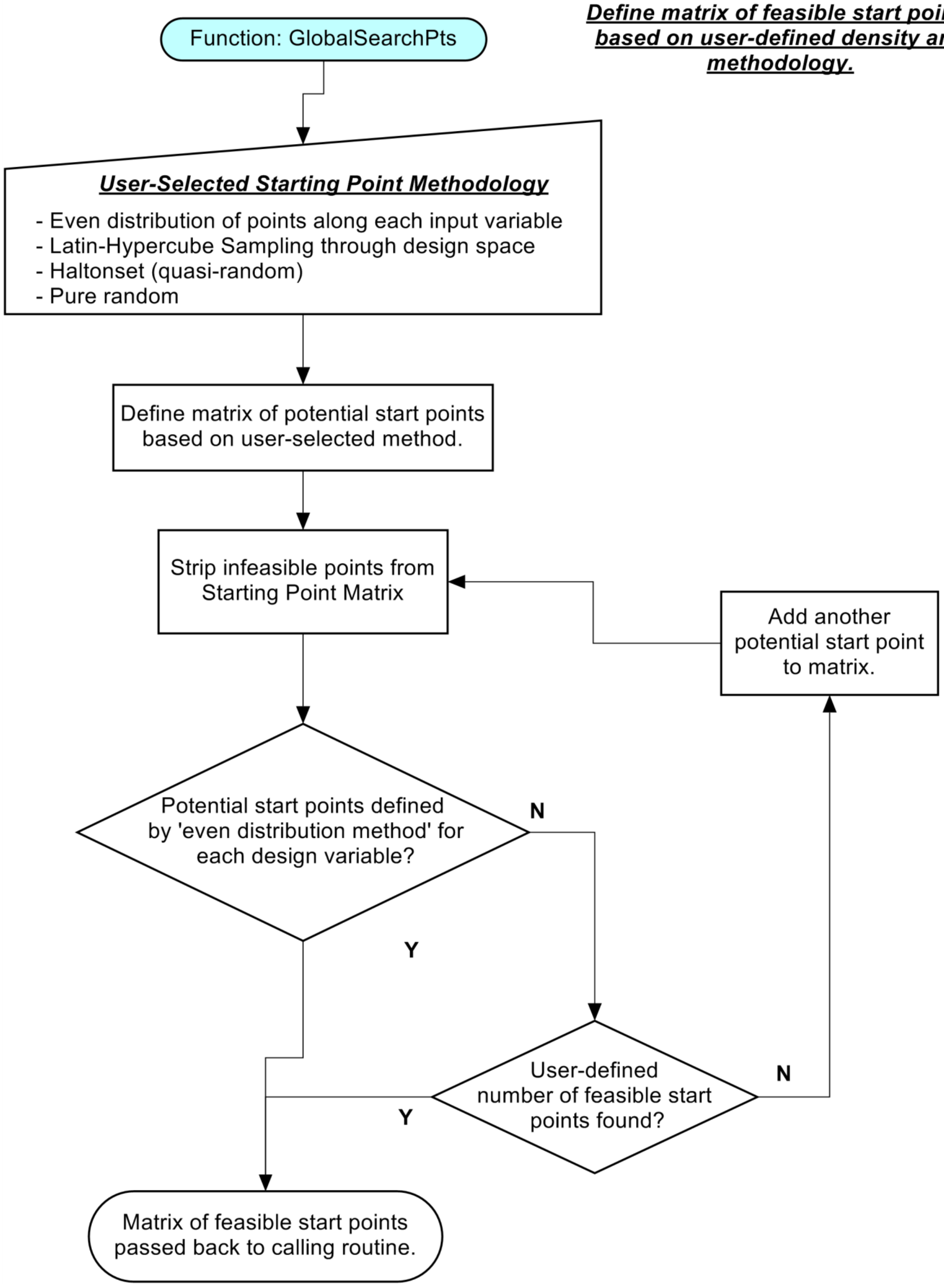
Total Nodes	FEM Result						Convergence Error (%)					
	Load Point Deflection (vertical - in)		Sum of Forces (lbf)				Load Point Deflection (vertical)		Sum of Forces			
	Sum	Range	Spring 1	Spring 2	Spring 3	Spring 4	Sum	Range	Spring 1	Spring 2	Spring 3	Spring 4
1932	28.676	0.591	25.006	49.775	125.592	143.651						
3642	30.918	0.718	25.344	50.777	128.900	148.564	-7.82%	-21.40%	-1.35%	-2.01%	-2.63%	-3.42%
7242	31.921	0.778	25.497	51.230	130.432	150.784	-3.24%	-8.46%	-0.61%	-0.89%	-1.19%	-1.49%
9672	32.097	0.789	25.524	51.308	130.684	151.151	-0.55%	-1.35%	-0.11%	-0.15%	-0.19%	-0.24%
14442	32.225	0.796	25.544	51.364	130.867	151.412	-0.40%	-0.95%	-0.08%	-0.11%	-0.14%	-0.17%
29226	32.361	0.808	25.567	51.441	131.255	151.914	-0.42%	-1.47%	-0.09%	-0.15%	-0.30%	-0.33%
47499	32.352	0.807	25.565	51.434	131.204	151.863	0.03%	0.13%	0.01%	0.01%	0.04%	0.03%
50825	32.360	0.807	25.566	51.437	131.214	151.876	-0.03%	-0.05%	0.00%	-0.01%	-0.01%	-0.01%
62244	32.363	0.807	25.566	51.437	131.208	151.867	-0.01%	0.02%	0.00%	0.00%	0.00%	0.01%
81902	32.363	0.807	25.566	51.436	131.202	151.861	0.00%	0.01%	0.00%	0.00%	0.00%	0.00%
140318	32.467	0.811	25.580	51.469	131.297	151.975	-0.32%	-0.48%	-0.05%	-0.06%	-0.07%	-0.08%

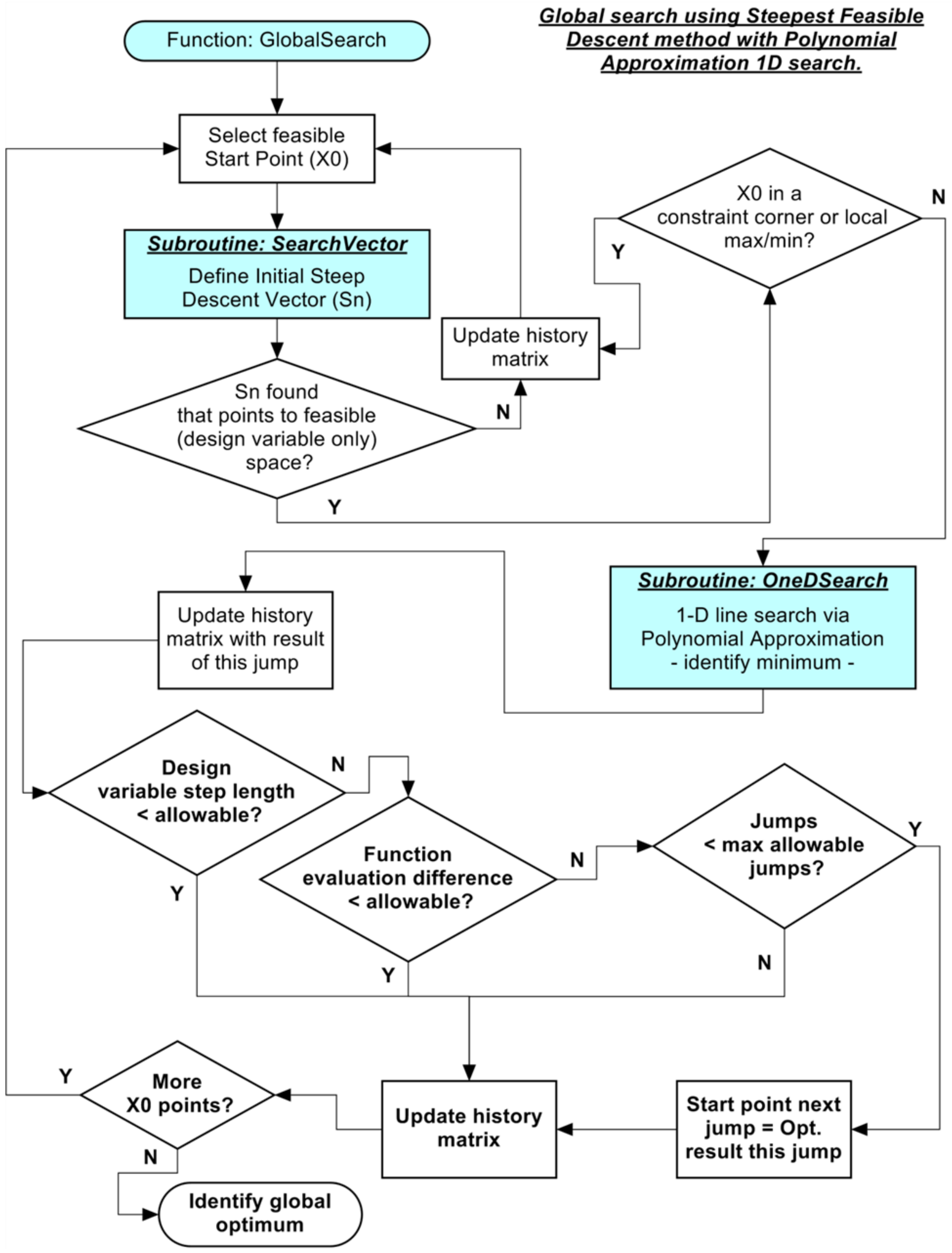
.

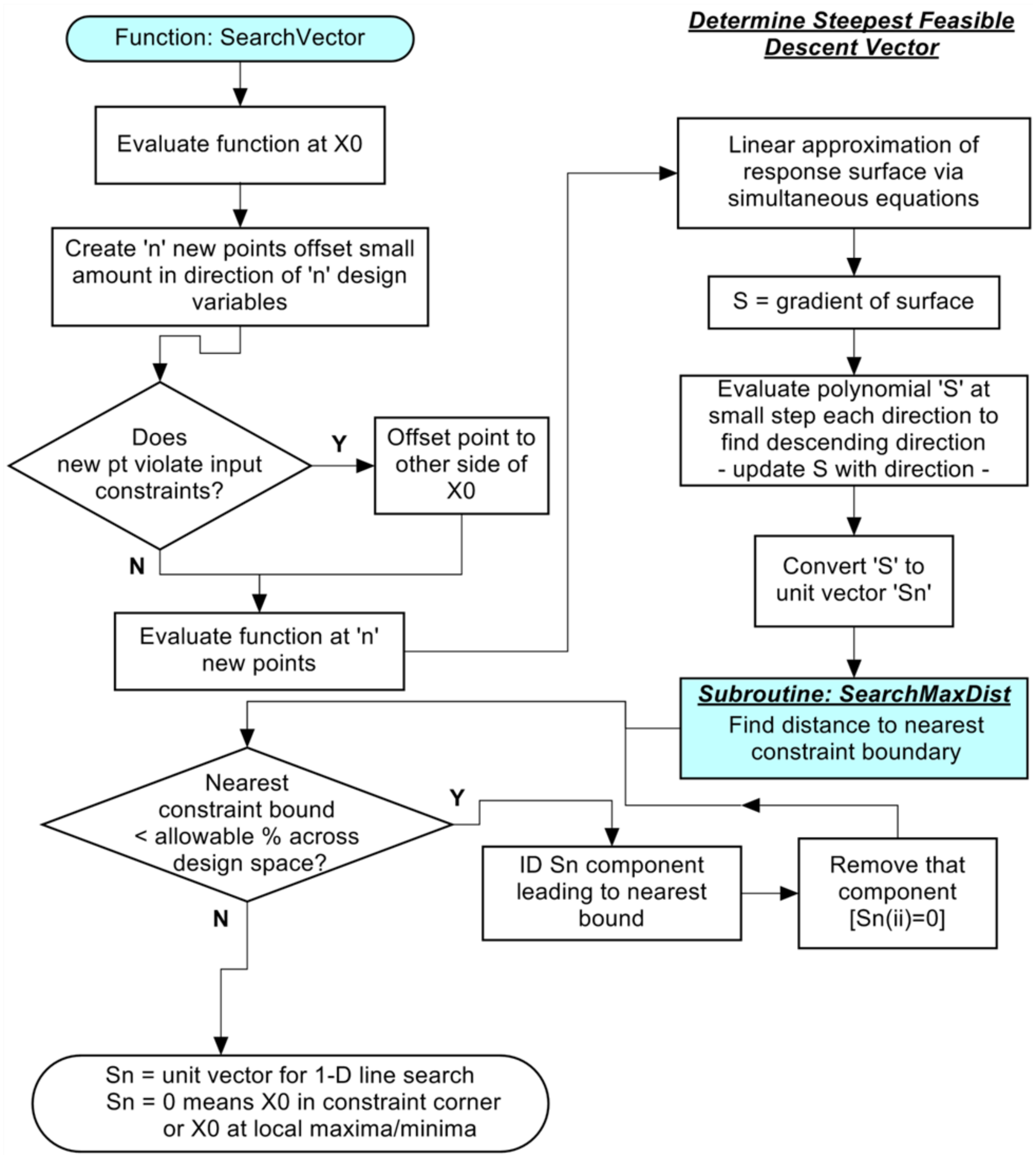
Appendix B

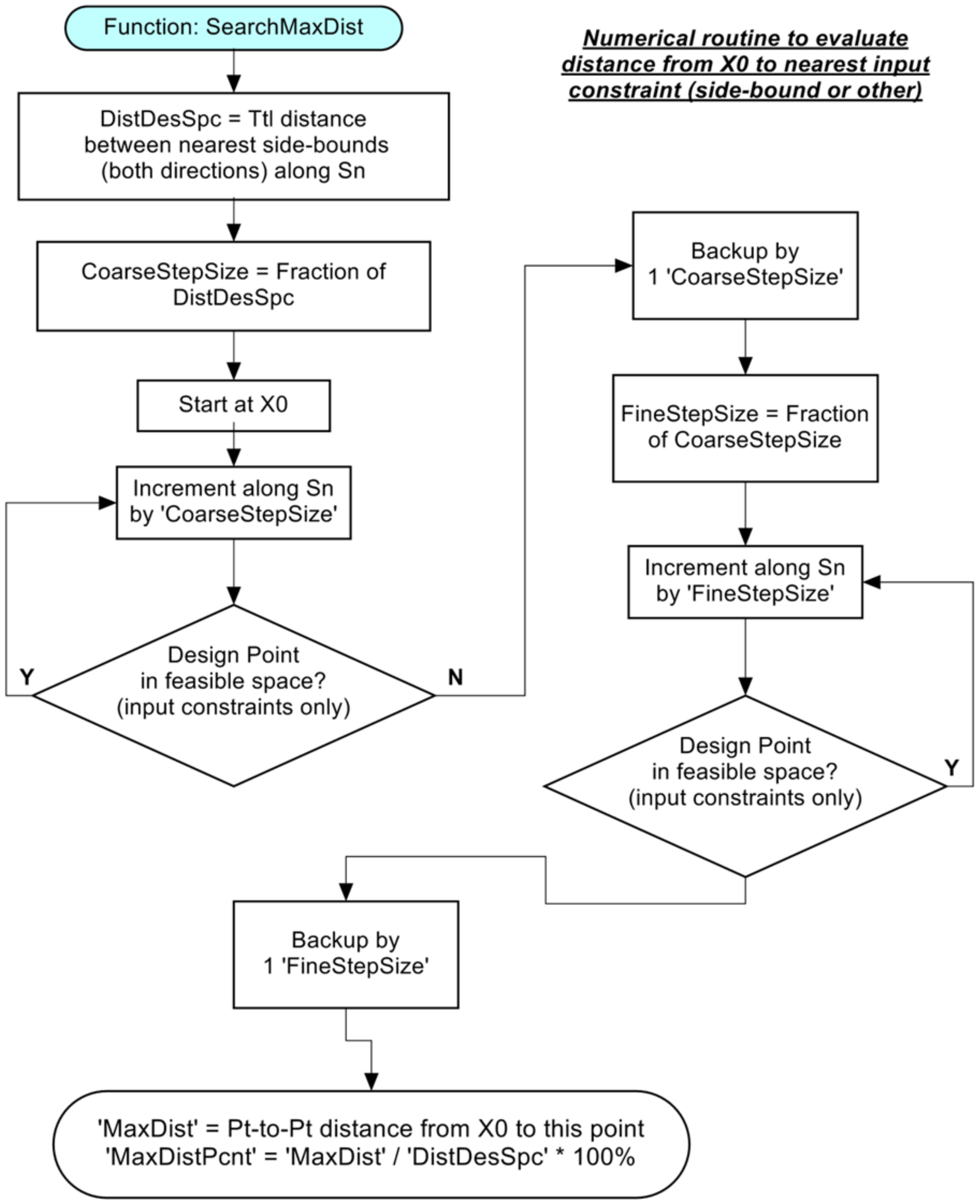
Flowcharts of Key Program Subroutines

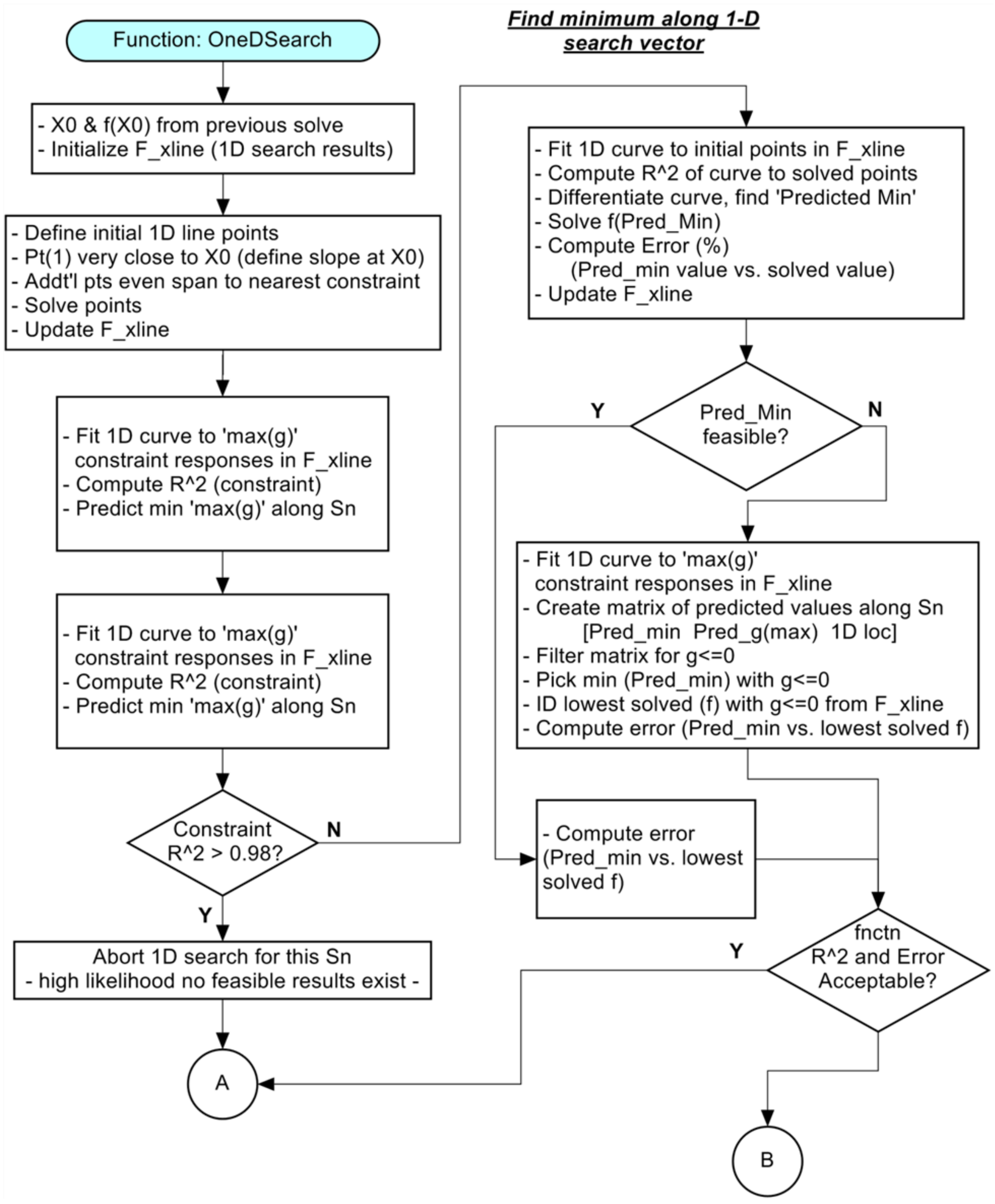
<u>Define matrix of feasible start points based on user-defined density and methodology.</u>

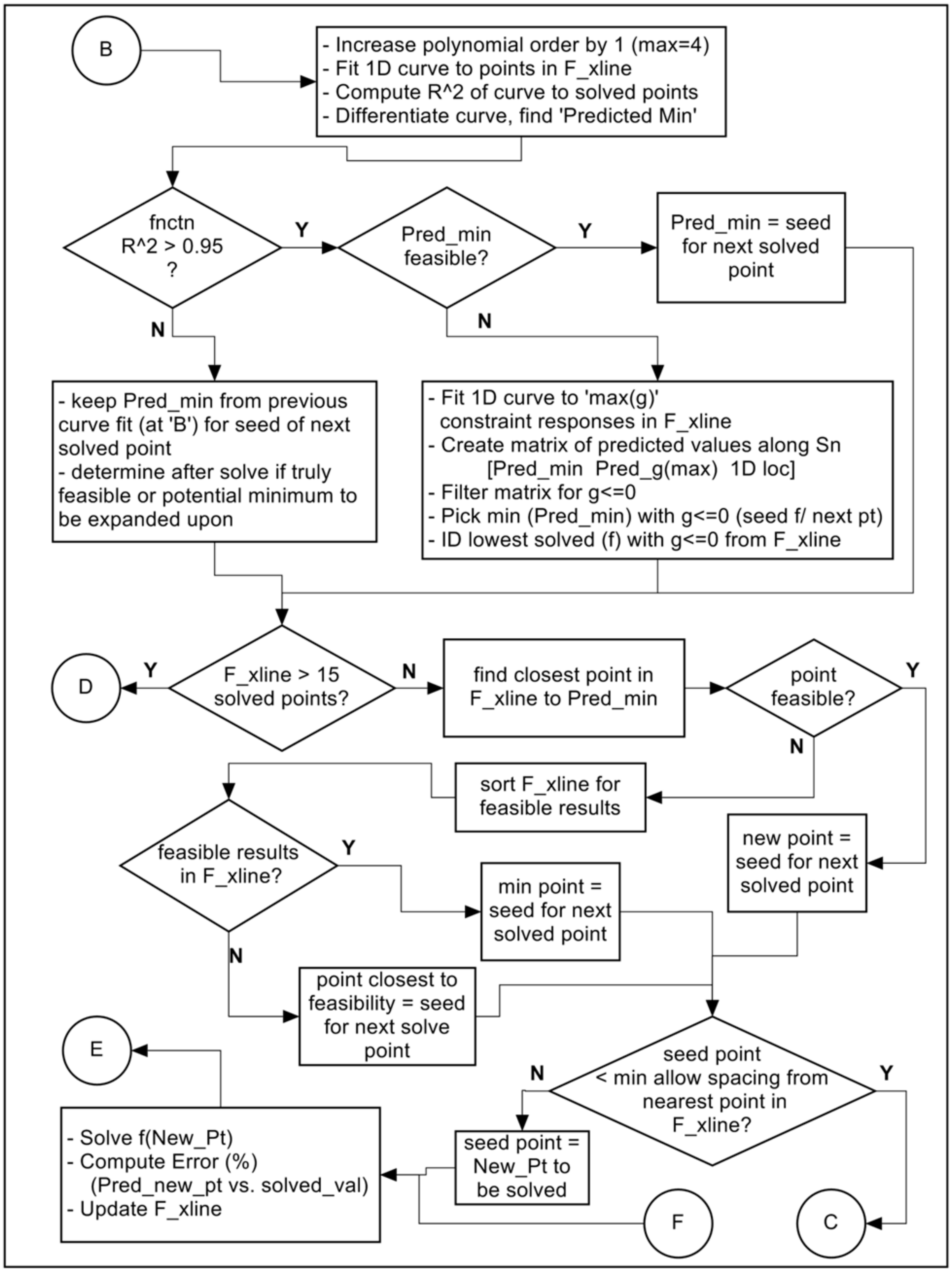

Function: GlobalSearchPts
User-Selected Starting Point Methodology
- Even distribution of points along each input variable
- Latin-Hypercube Sampling through design space
- Haltonset (quasi-random)
- Pure random
Define matrix of potential start points based on user-selected method.
Strip infeasible points from Starting Point Matrix
Add another potential start point to matrix.
Potential start points defined by 'even distribution method' for each design variable?
N
Y
User-defined number of feasible start points found?
N
Y
Matrix of feasible start points passed back to calling routine.

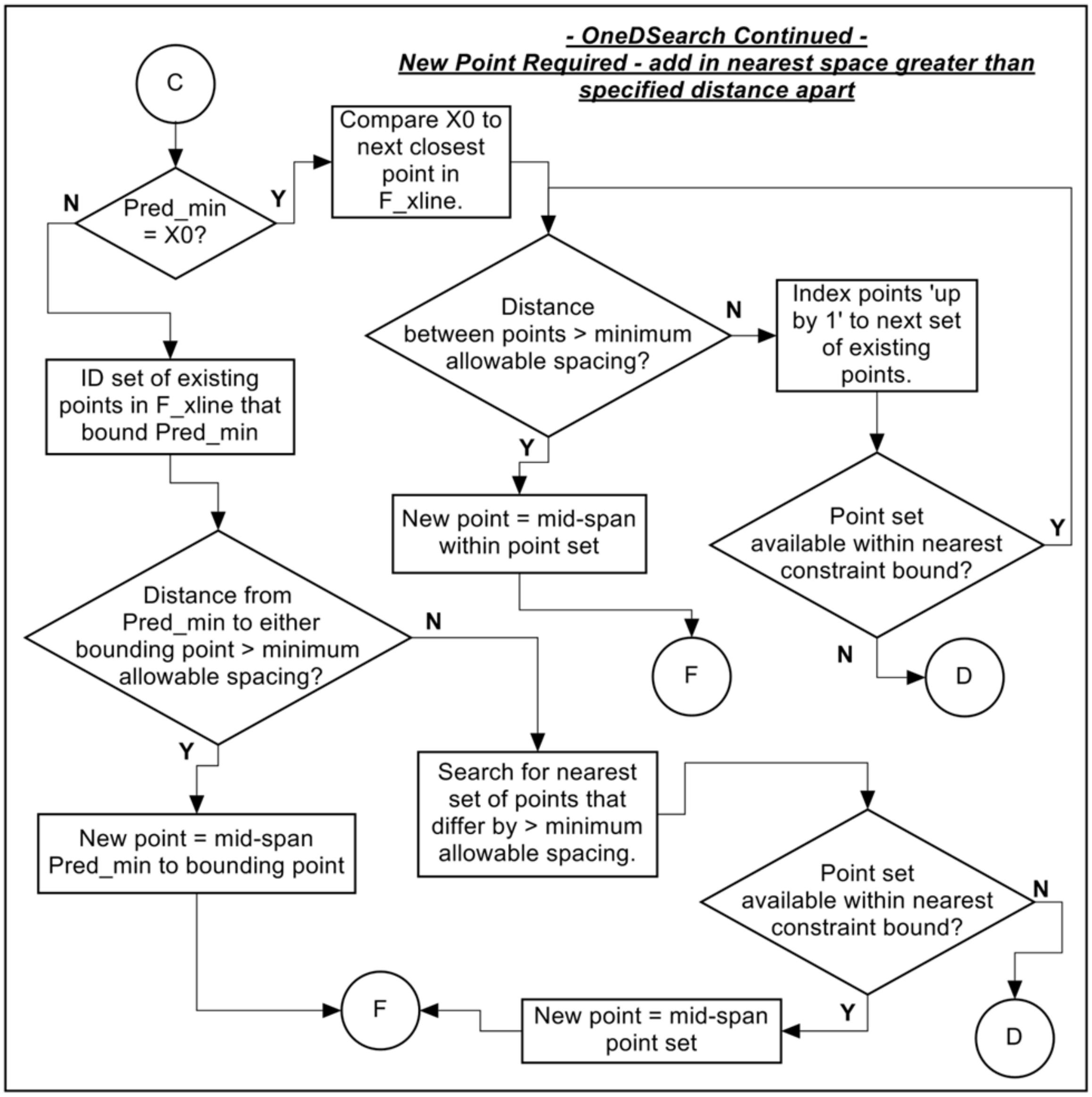

Global search using Steepest Feasible Descent method with Polynomial Approximation 1D search.
Function: GlobalSearch
Select feasible Start Point (X0)
Subroutine: SearchVector
Define Initial Steep Descent Vector (Sn)
Sn found that points to feasible (design variable only) space?
N
Y
Update history matrix
X0 in a constraint corner or local max/min?
Y
N
Subroutine: OneDSearch
1-D line search via Polynomial Approximation - identify minimum -
Update history matrix with result of this jump
Design variable step length < allowable?
N
Y
Function evaluation difference < allowable?
N
Y
Jumps < max allowable jumps?
Y
N
More X0 points?
Y
N
Update history matrix
Start point next jump = Opt. result this jump
Identify global optimum

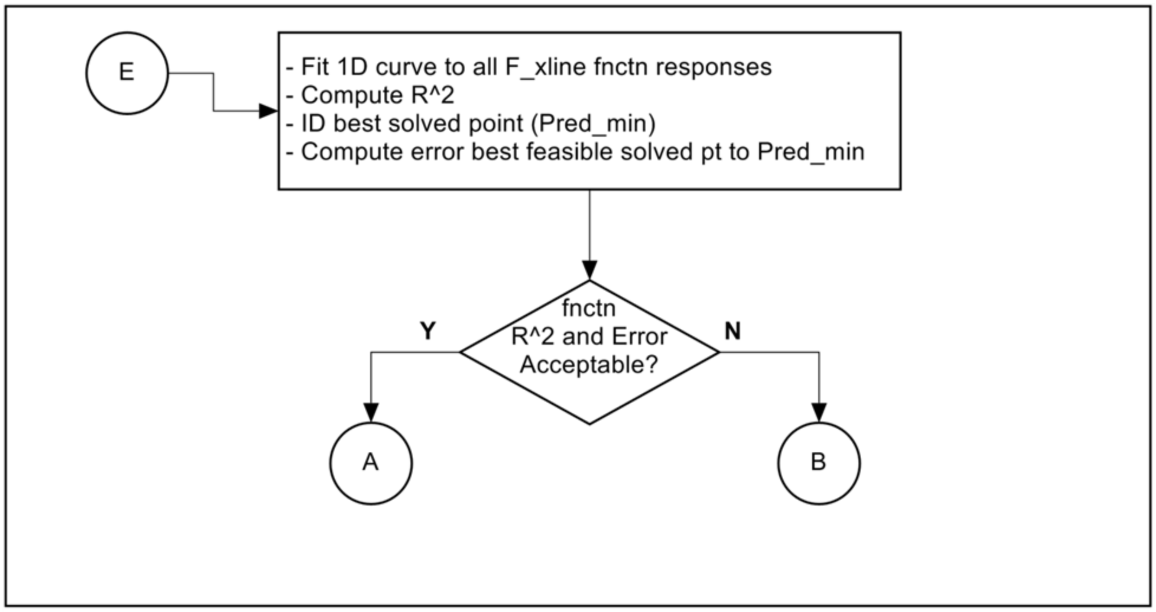

Function: SearchVector
Evaluate function at X0
Create 'n' new points offset small amount in direction of 'n' design variables
Does new pt violate input constraints?
Y
Offset point to other side of X0
N
Evaluate function at 'n' new points
Determine Steepest Feasible Descent Vector
Linear approximation of response surface via simultaneous equations
S = gradient of surface
Evaluate polynomial 'S' at small step each direction to find descending direction - update S with direction -
Convert 'S' to unit vector 'Sn'
Subroutine: SearchMaxDist
Find distance to nearest constraint boundary
Nearest constraint bound < allowable % across design space?
Y
ID Sn component leading to nearest bound
Remove that component [Sn(ii)=0]
N
Sn = unit vector for 1-D line search
Sn = 0 means X0 in constraint corner or X0 at local maxima/minima

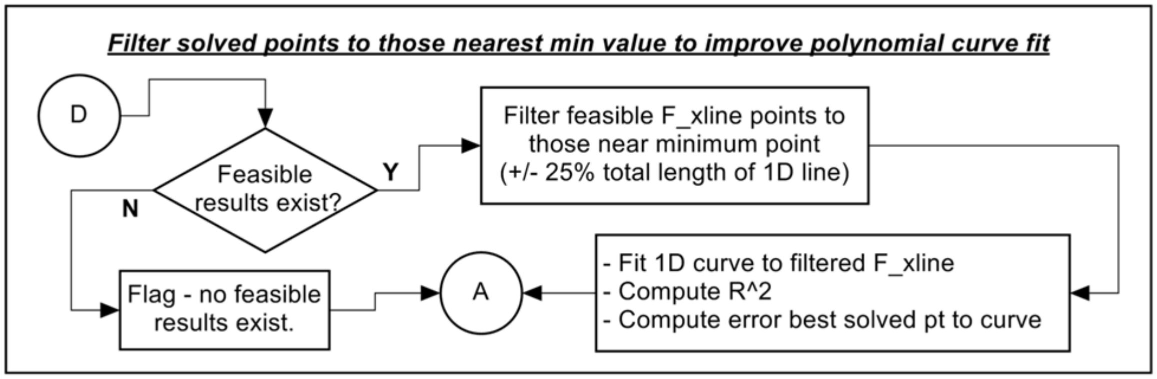

Function: SearchMaxDist
Numerical routine to evaluate distance from X0 to nearest input constraint (side-bound or other)
DistDesSpc = Ttl distance between nearest side-bounds (both directions) along Sn
CoarseStepSize = Fraction of DistDesSpc
Start at X0
Increment along Sn by 'CoarseStepSize'
Design Point in feasible space? (input constraints only)
Y
N
Backup by 1 'CoarseStepSize'
FineStepSize = Fraction of CoarseStepSize
Increment along Sn by 'FineStepSize'
Design Point in feasible space? (input constraints only)
Y
Backup by 1 'FineStepSize'
'MaxDist' = Pt-to-Pt distance from X0 to this point
'MaxDistPcnt' = 'MaxDist' / 'DistDesSpc' * 100%

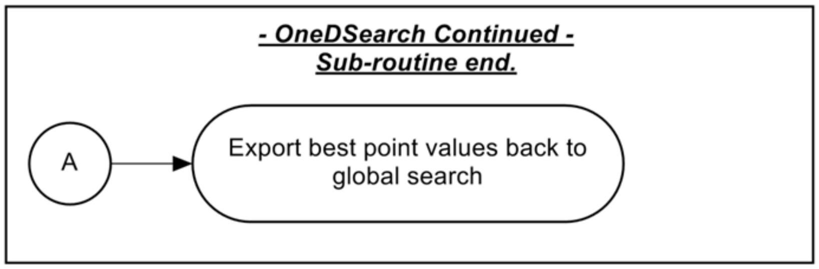

Find minimum along 1-D search vector
Function: OneDSearch
- X0 & f(X0) from previous solve
- Initialize F_xline (1D search results)
- Define initial 1D line points
- Pt(1) very close to X0 (define slope at X0)
- Addt'l pts even span to nearest constraint
- Solve points
- Update F_xline
- Fit 1D curve to 'max(g)' constraint responses in F_xline
- Compute R^2 (constraint)
- Predict min 'max(g)' along Sn
- Fit 1D curve to 'max(g)' constraint responses in F_xline
- Compute R^2 (constraint)
- Predict min 'max(g)' along Sn
Constraint R^2 > 0.98?
N
Y
Abort 1D search for this Sn
- high likelihood no feasible results exist -
A
- Fit 1D curve to initial points in F_xline
- Compute R^2 of curve to solved points
- Differentiate curve, find 'Predicted Min'
- Solve f(Pred_Min)
- Compute Error (%)
(Pred_min value vs. solved value)
- Update F_xline
Y
Pred_Min feasible?
N
- Fit 1D curve to 'max(g)' constraint responses in F_xline
- Create matrix of predicted values along Sn
[Pred_min Pred_g(max) 1D loc]
- Filter matrix for g<=0
- Pick min (Pred_min) with g<=0
- ID lowest solved (f) with g<=0 from F_xline
- Compute error (Pred_min vs. lowest solved f)
- Compute error (Pred_min vs. lowest solved f)
Y
fnctn R^2 and Error Acceptable?
B


B
- Increase polynomial order by 1 (max=4)
- Fit 1D curve to points in F_xline
- Compute R^2 of curve to solved points
- Differentiate curve, find 'Predicted Min'
fnctn R^2 > 0.95 ?
Y
Pred_min feasible?
Y
Pred_min = seed for next solved point
N
N
- keep Pred_min from previous curve fit (at 'B') for seed of next solved point
- determine after solve if truly feasible or potential minimum to be expanded upon
- Fit 1D curve to 'max(g)' constraint responses in F_xline
- Create matrix of predicted values along Sn [Pred_min Pred_g(max) 1D loc]
- Filter matrix for g<=0
- Pick min (Pred_min) with g<=0 (seed f/ next pt)
- ID lowest solved (f) with g<=0 from F_xline
D
Y
F_xline > 15 solved points?
N
find closest point in F_xline to Pred_min
point feasible?
Y
N
sort F_xline for feasible results
new point = seed for next solved point
feasible results in F_xline?
Y
min point = seed for next solved point
N
point closest to feasibility = seed for next solve point
E
seed point < min allow spacing from nearest point in F_xline?
N
Y
- Solve f(New_Pt)
- Compute Error (%) (Pred_new_pt vs. solved_val)
- Update F_xline
seed point = New_Pt to be solved
F
C


- OneDSearch Continued -
New Point Required - add in nearest space greater than specified distance apart
C
Pred_min = X0?
N
Y
Compare X0 to next closest point in F_xline.
Distance between points > minimum allowable spacing?
N
Index points 'up by 1' to next set of existing points.
Y
New point = mid-span within point set
Point set available within nearest constraint bound?
Y
N
F
D
ID set of existing points in F_xline that bound Pred_min
Distance from Pred_min to either bounding point > minimum allowable spacing?
N
Y
Search for nearest set of points that differ by > minimum allowable spacing.
New point = mid-span Pred_min to bounding point
Point set available within nearest constraint bound?
N
Y
F
New point = mid-span point set
D


E
- Fit 1D curve to all F_xline fnctn responses
- Compute R^2
- ID best solved point (Pred_min)
- Compute error best feasible solved pt to Pred_min
fnctn R^2 and Error Acceptable?
Y
N
A
B


Filter solved points to those nearest min value to improve polynomial curve fit
D
Feasible results exist?
Y
N
Filter feasible F_xline points to those near minimum point (+/- 25% total length of 1D line)
Flag - no feasible results exist.
A
- Fit 1D curve to filtered F_xline
- Compute R^2
- Compute error best solved pt to curve


- OneDSearch Continued -
Sub-routine end.
A
Export best point values back to global search

Appendix C

Full Program Listing

```
% ##########################################################################################
%
%         GLOBAL OPTIMIZATION ROUTINE
%
% ##########################################################################################
%  Written By:  Bret Hauser
%  Latest Revision Date:  Jan 16, 2015
% ##########################################################################################
% This code provides an optimization routine intended to find a 'global optimum' on a multimodal
% resoponse surface for use with an external solver.  Multiple optimization strategies are available
% to the user including:
%       1.)  'Coarse search' via Steepest Feasible Descent Method
%       2.)  'Refined search' via Direct Search using fmincon with SQP modifier
%       3.)  Combination of Coarse search to find region of best likelihood followed by SQP search
%            from single most promising result of coarse search.
% Although the program is intended for use with an ANSYS FEA external solver, it could be modified
% to call 'any' external solver.  Four (4) internal test functions exit within this code as a means
% of confirming code results as well as any future modifications.
%
% Note that for use of the 'Refined search' method, a MATLAB Optimization Toolbox license is
% required.  Also, use of the quasi-random tool 'haltonset' is used as one of multiple available
% methods to determine start points.  If haltonset is selected, a MATLAB Statistics Toolbox license
% is also required.
% ##########################################################################################
%  The project consists of 5 primary 'code' elements which are used as follows:
%     - This MATLAB program:  Is the 'top-level' code which creates a header text file for
%                           run-specific data, appends it to the main ANSYS input script file,
%                           launches the ANSYS run, reads and evaluates the results file,
%                           archives the script and results files, evaluates the convergence
%                           state of the solution and launches a subsequent iteration as
%                           needed.
%     - ANSYS Header file:    Is the code (written programmatically by this top-level MATLAB
%                           program) which includes the iteration-specific variables definition
%                           data for the given optimization loop.
%     - ANSYS Script file:    Is the code (written independently by the user in PYTHON) which
%                           interacts with the previously defined ANSYS project to modify values
%                           of established project parameters and write pre-defined results to
%                           a formatted output file.
%     - ANSYS Merged Script:  Is the script file programmatically generated as part of each
%                           iteration to append the ANSYS Header file to the front of the
%                           ANSYS Script file.  This ANSYS Merged Script is the input script
%                           called by this top-level MATLAB code for the ANSYS solve.
%     - ANSYS Project file:   The ANSYS project is independently established as a 'stand alone'
%                           project including geometry definition, pre- and post-processing
%                           elements.  Input and Output variables of interest are defined as
%                           parameters (conventionally named within ANSYS as "P1", "P2",
%                           etc.) and are available in the Parameters Data Container to be
%                           used as Design Points.  Multiple Design Points are not required
%                           for this process.  The Script file specifies the use of dP0 in
%                           this exercise for each loop of the optimization.  Input and
%                           output data for each iteration are archived programmatically in
%                           the form of the merged script file and the results file as well
%                           as stored in array form within the MATLAB Optimization routine
%                           and summarized in various output plots upon completion.
% ##########################################################################################
%     Key files needed to execute a job:
%
%        1.)  This MATLAB program
%        2.)  'Setup_Script_body.txt' = body of ANSYS Script file (above)
%        3.)  ANSYS *.wbpj project, predefined with all geometry, loads, BC's, input and output
%             parameters
% ##########################################################################################
% To Use This Code:
%    1.)  Ensure that path for ANSYS executable is set correctly.  To do so, obtain admin permission
%         and open Start\Control Panel\System\Advanced System Settings\Environment Variables\...
%        System Variables\Path - include desired path if not already there.  NOTE: This version of
%        code is set up for ANSYS 15.0.7, if ANSYS is updated then need to verify both executable
%        path and executable filename (specifie in function 'ANSYSsolve' below.
%    2.)  Set up ANSYS project including geometry definition, pre- and post processing
%        including solve.  Make sure all runs correctly without errors across full breadth of input
%        variable range.
%    3.)  Define parameters in ANSYS and identify input/results parameters by name (P1, P2,
%        etc.) for input into Input section below.
%    4.)  IMPORTANT ** Set 'Update Option' to 'Submit to RSM' and 'Solve Process Setting' to
%        selected Solver and cores for each Solution cell in the project and save project before
%        running this MATLAB file.
%    5.)  Populate Input section of this code (below).  Note that major sections of input are
%        identified in 'steps'.  Be sure that all of the steps are complete.
%    6.)  Set up objective function 'f' in function 'Objective_fun' below.  Note
%        that fmincon minimizes so to maximize a function need to minimize the negative.
%           - define objective function components and equation to compute function value
%           - identify results values that have a constraint bound(s) (to be included in output
%             matrix)
%    7.)  Set up constraint functions as needed in function Constraint_fun below.
%           - input (design variable) constraints
%           - output (results variable) constraints
%    8.)  Set the ANSYS project to solver choice as appropriate.
%    9.)  Best practice to import merged script file to ANSYS to verify solve for 1 iteration.
%        Can 'abort' job if long job using RSM to speed up process. Look for error messages within
%        ANSYS regarding script execution.
% ##################################################################################################


function main  % *** CHARTED ***
clc; clear all; close all
global pjctpath ANSYSpjctpath scriptname my_project my_resultsfile inputvar count_dv inputcols
global SpaceAllow his my_flagfile minstepsize count_outv outputvar arch_dir feval penalty result
global g_dv g_out globalX0 global_his rsqrdallow endpcnt linesearchsteps polyfitorder constrrad
global testfunction distdesspc funtol StartPt optsoltn HIS globalstart globalStartQty glX0min
global searchtype DVstepmin mu minspan max1Dpts maxpolyfitorder ErrorAllow1D consxcoef consycoef
global  solvertype my_solverfunction my_obj_fun my_cons_fun
  format long
  % Select the primary script file for the ANSYS model build and solve
  [scriptname,pjctpath] = uigetfile('*.txt','Select the *.txt ANSYS script file for the project');
  % --convert path syntax to ANSYS needs-ANSYS input script requires pathnames with '/', not '\'
  ANSYSpjctpath=pjctpath;
  for ii=1:length(ANSYSpjctpath)  % for loop to replace '\' with '/' in path
    if ANSYSpjctpath(ii)=='\'
      ANSYSpjctpath(ii)='/';
    end
  end
  % =====  check to see if project directory has spaces in it...if so alert user to redefine
  for ii=1:length(pjctpath)
    if pjctpath(ii)==char(32)   %char(32)= 'space'
      line1=' The project path name includes spaces which are problematic for Matlab. ';
      line2=['  ',pjctpath,' '];
      line3=' PLEASE RENAME THE FOLDER(S) SO THAT NO SPACES ARE INCLUDED AND RESTART ';
      % note:  all string lengths must be same to put in following composite vector
      msgstring=char(line1,' ',line2,' ',line3);
      msgtitle='Error - Spaces in pathname';
      msgbox(msgstring,msgtitle,'error');
      stop;  % end program execution within MATLAB - not clean but functional
    end
   end

 % ############################################################################################
  %%    ---  DEFINE INPUT DATA ---
  % -----------------------------------------------------------------------------------------
  %   Input data notes
  %   1.) FILE NAMES correspond to the pre-defined ANSYS file (*.wbpj), and names to be used
  %      for the results file and flag file.  Results file is written by the ANSYS input script
  %      and contains a record of Design Variables as well as output data to be read as the
  %      objective function later in this code.  The flag file is a 'marker' file that is
  %      written at the conclusion of the solve to indicate its completion.  Its presence is
  %      tracked later in this code as an indication that the solve is complete.
  %   2.) DEFINE TEST FUNCTION if applicable or choose to solve an external function.  Note that
  %      input variables must be set up for the test function as well as objective_fun and
  %      constraint_fun specifics desired for the test run.
  %   3.) DESIGN VARIABLES are entered as 'cell array of strings' data.  'ANSYS Parameter Name' is
  %      the named assigned to that parameter in ANSYS' 'Parameter Set' Data Component (eg: P1,
  %      P2, etc.) Note use of '{}' instead of '()' or '[]' to indicate cell array.  Note that
  %      test function variables are included by default, but only active if test function chosen.
  %   4.) MISC INPUT DATA should be confirmed prior to each run.  This includes various parameters
  %      including stopping criteria, allowable bounds for curve fits, etc.
  %   5.) RESULTS DATA sets up the parameters to be read by this code from ANSYS.  The format
  %      is similar to that of Design Variables and includes the ANSYS 'Parameter Set' name
  %      (P7,P8, etc.) and a description.  As with Design Variables, this is a cell array so
  %      note use of '{}' rather than '()' or '[]'. Note that 'Max Expected Value' is required for
  %      each parameter. This is used in the Objective_Fun and Constraint_Fun subroutines to
  %      normalize the results.
  %   6.) OPTIMIZATION SOLUTION PATH allows the user to select the optimization scheme. Note that a
  %      user-specified max number of Steepest Feasible Descent 'jumps' and SQP 'jumps' are
  %      available for definition.
  %   7.) STARTING POINTS may be defined by the user in a number of methods.  Note that in addition
  %      to a user-specified number of global start points, an optimization can be executed from a
  %      single user-specified startpoint in the format [DV1 DV2 DV3...] in the order of design
  %      variables previously specified.
  %   8.) TEST FUNCTION CONSTRAINTS may also be specified for use in confirming the algorithm's
  %      ability to negotiate around feasible/infeasible space.  Note that since the test
  %      functions are defined as '2D' space, the constraint bounds are defined as a circle of
  %      user-specified origin and radius, within which lies 'infeasible' space.
  %
===========================================================================================
  % -- STEP 1.)  Specify File Names (See Input Data Note 1) -------
      solvertype=2;  %[0]=ANSYS, [1]=External MATLAB Function, [2]=Test Function
      %  --- [0] = ANSYS Project ---
        if solvertype==0
           %name of ANSYS proj file to be executed
          my_project={'Parallel_Beam_Family_Config-1a_C-F.wbpj' 'Parallel_Beam_Family_Config-1b_C-F.wbpj'
'Parallel_Beam_Family_Config-1c_C-F.wbpj';
                  'Parallel_Beam_Family_Config-2a_C-F.wbpj' 'Parallel_Beam_Family_Config-2b_C-F.wbpj'
'Parallel_Beam_Family_Config-2c_C-F.wbpj';
                  'Parallel_Beam_Family_Config-3a_C-F.wbpj' 'Parallel_Beam_Family_Config-3b_C-F.wbpj'
'Parallel_Beam_Family_Config-3c_C-F.wbpj';
                  'Parallel_Beam_Family_Config-4a_C-F.wbpj' 'Parallel_Beam_Family_Config-4b_C-F.wbpj'
'Parallel_Beam_Family_Config-4c_C-F.wbpj';
                  'Parallel_Beam_Family_Config-5a_C-F.wbpj' 'Parallel_Beam_Family_Config-5b_C-F.wbpj'
'Parallel_Beam_Family_Config-5c_C-F.wbpj';
                  'Parallel_Beam_Family_Config-6a_C-F.wbpj' 'Parallel_Beam_Family_Config-6b_C-F.wbpj'
'Parallel_Beam_Family_Config-6c_C-F.wbpj'}
          my_resultsfile='logfile.dat'; %name of results file to be written by inputscript
          my_flagfile='flagfile.dat'; %name of temp flag file to show completion of ANSYS solve
        elseif solvertype==1
       % --- [1] = External MATLAB Function ---
          my_obj_fun=@Sn_Study_OF;
          my_cons_fun=@Sn_Study_CF;
          my_project=my_obj_fun;
          my_resultsfile='logfile.dat'; %name of results file to be written by inputscript
          my_flagfile='flagfile.dat'; %name of temp flag file to show completion of ANSYS solve
        elseif solvertype==2
       % --- [2] = Test Function (Internal - choose from below)
          testfunction=4;
            % 0=NO TEST FUNCTION IN USE - ANSYS or External Solve Used
            % 1=De Jong's function in 2D  - bounds in x & y = [0   5.12]
            % 2=Rosenbrock's Valley in 2D  - bounds in x & y = [0   2.048]
          % 3=Rastrigin's function in 2D - bounds in x & y = [0   5.12]
          % 4=Schwefel's function in 2D (modified for bounds) - bounds in x & y = [0   500]
        if testfunction==1
            my_project={'De Jongs Test Function'};
        elseif testfunction==2
            my_project={'Rosenbrocks Valley Test Function'};
        elseif testfunction==3
            my_project={'Rastrigins Test Function'};
        elseif testfunction==4
            my_project={'Schwefels Test Function'};
        end
        my_resultsfile='logfile.dat'; %name of results file to be written by inputscript
        my_flagfile='flagfile.dat'; %name of temp flag file to show completion of ANSYS solve
      end
% ------------------------------------------------------------------------------------------
% -- STEP 3.) Specify Design Variable Information (See Input Data Note 2)--------
          %  format = {  Param_ID, ValueMin,   ValueMax,   Description} ------
      if solvertype==0   % ANSYS function solve requested
      %  format = {ANSYS Parameter Name,   ValueMin,   ValueMax,   Description} ------
            inputvar={'P62',           '0',      '72',    'Dim_L1 (in)';
                      'P63',           '0',      '72',    'Dim_L2 (in)';
                      'P116',        '500',   '50000',    'K1 stiffness (lb/in)';
                      'P117',        '500',   '50000',    'K2 stiffness (lb/in)';
                      'P118',        '500',   '50000',    'K3 stiffness (lb/in)';
                      'P119',        '500',   '50000',    'K4 stiffness (lb/in)';
                      'P120',      '0.005',   '0.500',    'C1 damping coefficient';
                      'P121',      '0.005',   '0.500',    'C2 damping coefficient';
                      'P122',      '0.005',   '0.500',    'C3 damping coefficient';
                      'P123',      '0.005',   '0.500',    'C4 damping coefficient'}

      elseif solvertype==1  % External MATLAB function solve requested
          %  format = {  Param_ID, ValueMin,   ValueMax,   Description} ------
            inputvar={  'M',      '10',   '100',  'Mass (kg)';
                        'K',    '10000', '50000',  'Spring constant (N/m)';
                        'Zeta',   '0.01',  '0.50',  'Damping ratio'};
      elseif solvertype==2
         if testfunction==1   % De Jong's function
            inputvar={'x',        '-5.12',       '5.12',     'Test Dimension 1';
                      'y',        '-5.12',       '5.12',     'Test Dimension 2'}
         elseif testfunction==2   % Rosenbrock's Valley
            inputvar={'x',        '-2.048',      '2.048',    'Test Dimension 1';
                      'y',        '-2.048',      '2.048',    'Test Dimension 2'}
         elseif testfunction==3   % Rastrigin's function
            inputvar={'x',        '-5.12',       '5.12',     'Test Dimension 1';
                      'y',        '-5.12',       '5.12',     'Test Dimension 2'}
         elseif testfunction==4   % Schwefel's function
            inputvar={'x',        '-500',        '500',      'Test Dimension 1';
                      'y',        '-500',        '500',      'Test Dimension 2'}
         end
      end
% ------------------------------------------------------------------------------------------
% -- STEP 4.) Misc input data
   polyfitorder=3;  %order of 1-D polynomial (starting - will adjust automatically to meet R^2)
   rsqrdallow=0.9995; % min allowable R^2 - 1D curve fit
   minspan=2;  % no two points can be closer than minspan [pcnt] of max distance together
   max1Dpts=15; % maximum number of points allowed during 1-D line search
   maxpolyfitorder=4; % max allowable order of polynomial to prevent numerical errors
   ErrorAllow1D=5.0; % pcnt error ('5'=5%)allow between predicted min from 1D eqtn & actual solve
   DVstepmin=0.001;  % Minimum input variable step size indicating design convergence
   penalty=0.250;  % value of penalty function for output constraints in Objective_Fun
% ------------------------------------------------------------------------------------------
% -- STEP 5.) Results data setup (See Input Data Note 4) ----------------
    if solvertype==0   % no test function - these result values for ANSYS job
      % ------------
      % -- output variables for ANSYS function
      %   Note:  'Max_Expected_Value' is a rough approximation of the maximum absolute value of
      %   the given response.  This is used in the normalization of the output results at the
      %   bottom of Objective_Fun for the computation of the Objective Function Variable [f]
      %   only.  Results are NOT normalized for any other reason and are captured in the
      %   history file and log outputs in their non-normalized state.  Normalization of the
      %   Objective Function Variable is needed in order to aid in the application of penalty
      %   for constraint violations.
      % ------------
      % format={Parameter Name,  Max_Expected_Value    Description} ------
      outputvar={ 'P128',              '1',           'Static_Deflection (In)';
               'P129',              '300',          'my_Tip_UY_Sum (In)';
               'P130',              '100',          'my_Tip_UY_Range (In)'}

     % **** NOTE:  Only output parameters defined within Workbench can be passed out from
     % ANSYS to Matlab.  Parameters defined as output within the 'Parameters' cell based
     % on output values from Workbench do not have a 'value' property and therefore will
     % not pass out under PYTHON script.
     % ---------------------------------------------------------------------------------
    elseif solvertype==1  % output values for External MATLAB function
      % format={Parameter Name,  Max_Expected_Value    Description} ------
      outputvar={  'FRF',                  '1',       'Freq Response Fnctn at 40 Hz'}
     % ---------------------------------------------------------------------------------

    elseif testfunction==1   % De Jong's function
      % -- output variables for De Jong's test function
      % ------------
      outputvar={ 'scalarout',        '50',           'Scalar result from Test Function'};
    elseif testfunction==2   % Rosenbrock's Valley function
      % -- output variables for Rosenbrock's Valley test function
      % ------------
      outputvar={ 'scalarout',      '3000',           'Scalar result from Test Function'};
    elseif testfunction==3   % Rastrigin's function
      % -- output variables for Rastrigin's test function
      % ------------
      outputvar={ 'scalarout',        '80',           'Scalar result from Test Function'};
    elseif testfunction==4   % Schwefel's function
      % -- output variables for Schwefel's test function
      % ------------
      outputvar={ 'scalarout',      '1500',           'Scalar result from Test Function'};
    end
% -------------------------------------------------------------------------------------------
% -- STEP 6.) Define Optimization Solution Path ----------------
   optsoltn=1;
     % 0=Global Solution Only
     % 1=SQP Solution (fmincon) Only
     % 2=Combination - best Global Solution seeds Start Point for SQP Solution
   % variables for global search
     maxjumps=2; %max number of steepest descent 'jumps' during global optimization (ea point)
   % variables for SQP search
     SQPmaxjumps=250;   % number of maximum allowable jumps for SQP Solution
% -------------------------------------------------------------------------------------------
% -- STEP 7.) Define Starting Points ----------------
  globalstart=1;
     % 0=Evenly divided design space - use 'globalStartQty' below to specify
     % 1=Start points per Haltonset - use 'globalStartQty' below to specify
     % 2=Start points per Latin Hypercube Sampling - use 'globalStartQty' below to specify
     % 3=Start points per Random Generator (uniform distr) - DOES NOT REQUIRE STAT TOOLBOX
  % Number of FEASIBLE start points desired
     globalStartQty=75;
% -------------------------------------------------------------------------------------------
% -- STEP 8.) Define 'Output' Constraint Test Area for Test Functions ----------------
    consxcoef=1000.00;   % x-coordinate of center of circle
    consycoef=1000.00;   % y-coordinate of center of circle
    constrrad=2.0; % radius of circle
% ##########################################################################################
```

```
    if solvertype~=2
        testfunction=0;  % verifies that testfunction set correctly if actual solve in process
    end
   %%     --- FIND SIZE OF INPUT / OUTPUT ARRAYS & Minimum Stepsize ---
   [count_dv,inputcols]=size(inputvar);  %count number of design variables specified
   [count_outv,outputcols]=size(outputvar);  %count number of results variables specified
        distdesspc=0;  %initialize variable
%      % ----- find 'corner distance' across design space -----
%         distdesspc=0;
%         for distii=1:count_dv  %define design space 'corner distance' across all Des Var.
%             distdesspc=distdesspc+(str2double(inputvar(distii,3))-...
%                                         str2double(inputvar(distii,2)))^2;
%         end
%         distdesspc=sqrt(distdesspc);  % distance 'across' design space for parameterization
    % --- determine minimum step size for 1D search and Search Vector Offset -----
        minstepsize=0;  % initialize variable
        for ii=1:count_dv
            minstepsize=minstepsize+(str2double(inputvar(ii,3))-str2double(inputvar(ii,2)))^2;
%           if temp<minstepsize
%              minstepsize=temp;  % find minimum range in DV
%           end
        end
        minstepsize=sqrt(minstepsize)/1e5;  % minimum stepsize = fraction of minimum DV range
        %        minstepsize=(distdesspc)/200000;  % min step size
   % ####################################################################################################
    %% Set-up Archive Location
    % ---------------------------------------------------------------------------------------------
    % This section establishes an archive directory as a sub-level to the user-defined
    % project directory (the directory that the *.wbpj file was selected from).  Input script
    % files, output files (per iteration) and a summary results file are stored in the
    % archive directory.  In addition, this section checks to make sure that a previous
    % archive directory is not 'active' so as not to overwrite the previous run's data.  If a
    % previous archive directory is found, the program stops execution with a message to the
    % user that the previous directory was found and that it should be removed or renamed.
    % ---------------------------------------------------------------------------------------------
        arch_dir=strcat(pjctpath,'archive');
        direxists=0;
        direxists=exist(arch_dir,'dir'); % returns 0 if directory does not exist
        if direxists~=0
            line1='*************************************************************************';
            line2='*************************************************************************';
            line3='           AN ARCHIVE DIRECTORY ALREADY EXISTS UNDER THE NAME          ';
            line4=arch_dir;
            line5='   PLEASE EITHER REMOVE OR RENAME THE ARCHIVE DIRECTORY AND RESTART  ';
            line6='*************************************************************************';
            line7='*************************************************************************';
            msgstring=char(line1,line2,line3,line4,line5,line6,line7);
            msgtitle='Error - globalDVdiv<1';
            msgbox(msgstring,msgtitle,'error');
            stop;  % end program execution within MATLAB - not clean but functional
        end
        cmdline=['cd ' pjctpath];
        [status,cmdout]=dos(cmdline);   % make project folder active 'in dos window'
        cmdline=['mkdir '  'archive'];
        [status,cmdout]=dos(cmdline);  % make folder for archive as sub to project folder
     %% ******************************************************************************
     %   Optimization Sequence Calls
     %  ******************************************************************************
        if optsoltn==0  % Global Solution Only
%             [feaspts]=globalsearchpts(globalDVdiv,maxjumps);  % define global X0 start points
            [feaspts]=globalsearchpts(maxjumps);  % define global X0 start points
            % feaspts = number of global points that meet constraints
            [glX0min,glfevalmin,globalfevalcount]=globalsearch(feaspts,maxjumps);% global search(s)
            bestX0=glX0min;  %best solution from this global-only search
        end
        if optsoltn==1  % SQP Solution Only from global start points
%            feaspts=1;  % only 1 X0 start point for SQP search
%            % SQP X0 point defined in input data above
           [feaspts]=globalsearchpts(maxjumps);  % define global X0 start points
           % feaspts = number of global points that meet constraints
           [myXOPT,myFVAL,SQPfevalcount]=SQPsearch(feaspts,SQPmaxjumps);
           bestX0=myXOPT;  %best solution from this SQP-only search

        end
        if optsoltn==2; % Combination Global Search then refined SQP Search
%            [feaspts]=globalsearchpts(globalDVdiv,maxjumps);  % define global X0 start points
           [feaspts]=globalsearchpts(maxjumps);  % define global X0 start points
           % feaspts = number of global points that meet constraints
           [glX0min,glfevalmin,globalfevalcount]=globalsearch(feaspts,maxjumps);% global search(s)
           feaspts=1;  % only 1 X0 start point for SQP search
%            [XOPT,FVAL,SQPfevalcount]=SQPsearch(SQPX0,feaspts,SQPmaxjumps);
           [myXOPT,myFVAL,SQPfevalcount]=SQPsearch(feaspts,SQPmaxjumps);
           bestX0=myXOPT;  %best solution from this combination search
        end
        PlotResults(bestX0); %goto function PlotResults for final plotting of variable traces




%% ##########################################################################################
%    Sub-Functions
% ##########################################################################################

function [glX0min,glfevalmin,globalfevalcount]=globalsearch(feaspts,maxjumps) % *** CHARTED ***
% this function conducts the global search given a number of feasible points and specified max
% jumps.  The actual feasible points are in 'globalX0' as a global variable.
global pjctpath ANSYSpjctpath scriptname my_project my_resultsfile inputvar count_dv inputcols
global SpaceAllow his my_flagfile minstepsize count_outv outputvar arch_dir feval penalty result
global g_dv g_out globalX0 global_his rsqrdallow endpcnt linesearchsteps polyfitorder constrrad
global testfunction distdesspc funtol StartPt optsoltn HIS globalstart globalStartQty glX0min
global searchtype DVstepmin mu minspan max1Dpts maxpolyfitorder ErrorAllow1D consxcoef consycoef
global  solvertype my_solverfunction my_obj_fun my_cons_fun
% ==========================================================
% Steepest Descent Evaluation of Global Start Points with Polynomial Approximation 1-D search
 % ======  print header to screen output ================
    fprintf('%s\r',' ');
    global_his=[];
 % ====== plot test surface here only if 'global-only' solution, else plot in SQPsearch ======
    if optsoltn==0
        if testfunction==1
           nopts=50;
           PlotTestFunction(nopts)  %plot countour plot of function
        elseif testfunction==2
           nopts=50;
           PlotTestFunction(nopts)  %plot countour plot of function
        elseif testfunction==3
           nopts=150;
           PlotTestFunction(nopts)  %plot countour plot of function
        elseif testfunction==4
           nopts=150;
           PlotTestFunction(nopts)  %plot countour plot of function
        end
    end
 % ===== Open Global History File ==================================================
        summary_file=[pjctpath,'archive\GLOBAL OPT RESULTS SUMMARY.TXT'];
       fid20 = fopen(summary_file, 'w'); %open file for write permiss.-discard contents
       % note '\r\n' is carriage return with new line in Windows OS
       fprintf(fid20,'%s\r\n','# ##########################################################');
       fprintf(fid20,'%s\r\n','#   Results summary from Global Opt search (comma delimited)');
       fprintf(fid20,'%s\r\n','# ---------------------------------------------------------');
```

```
        for ii = 1:length(my_project)
            thisproject=char(my_project(ii,:));
            fprintf(fid20,'%s%s\r\n',' Project = ',thisproject);
        end
        fprintf(fid20,'%s\r\n','# -----------------------------------------------------');
        fprintf(fid20,'%s\r\n','#                  Convergence History');
        fprintf(fid20,'%s\r\n','# -----------------------------------------------------');
        hdr_text1=(',X0,Jump');
        for ii=1:count_dv
          hdr_text1=cat(2,hdr_text1,char(44),inputvar{ii,1});
        end
        for ii=1:count_outv
          hdr_text1=cat(2,hdr_text1,char(44),outputvar{ii,1});
        end
        hdr_text1=cat(2,hdr_text1,',X-position of 1-D result,Max 1-D Search Distance,R-Squared,'...
                 ,'Min-Pnt Error (%)',',No of Function Evals this Xo',',Max Constr Fun Value');
        fprintf(fid20,'%s\r\n',hdr_text1(:));
 % =====  'Outer loop' = Each Identified Global Start Point ========
     globalfevalcount=0;  % counter for total number of fevals during global search
     linesearchstepshold=linesearchsteps; %variable changes in jump, need to reset to original
     polyfitorderhold=polyfitorder;        %variable changes in jump, need to reset to original
     for X0point=1:feaspts  % step through the feasible globalX0 points
        searchtype=0;  % (0)=global search, (1)=fmincon search, (2)=globalsearchpts
        fprintf('%s%s%s\r','----- Global X0 Point #',num2str(X0point)',' -----');
        feval=0;  %counter for number of function evaluations conducted
        X0=globalX0(X0point,:);  % X0 = 'selected X0' from the globalX0 vector at row X0point
        % ===== Inner loop - step through steepest descent 'jumps' ===========
        convergedflag=0;  % indicates convergence for exit of loop
        tempconvergedflag=0; % indicates convergence temporarily until all output written, then exit
        while convergedflag==0
          for jumps=1:maxjumps
            linesearchsteps=linesearchstepshold;  %temp var - reset to linesearchsteps each jump
            polyfitorder=polyfitorderhold; % temp variable - reset to polyfitorder each jump
            F=[]; %initialize the objective function results matrix F
            % ===== Find initial search direction 'S' =====
              [Sn,maxdist,F,constrflag]=SearchVector(X0point,jumps,X0,F);%find desc unit vector
              % ---- write start point output to global history output file ----
                 if jumps==1
                   [globalhisrows,globalhiscols]=size(global_his);
                   for ii=1:globalhisrows
                       if global_his(ii,1)==X0point
                           startii=ii;
                       end
                   end
                   textstring=[];
                   for jj=1:globalhiscols
                     textstring=cat(2,textstring,char(44),num2str(global_his(...
                                                                  startii,jj)));
                   end
                   fprintf(fid20,'%s\r\n',textstring);
                 end
%                 % ---- end write start point output to global history output file ----
%                  if constrflag==1  % constrflag=1 means no Sn values point to feasible space
%                     F_X0=F(1,count_dv+2);
%                     global_his=[global_his; X0point jumps X0 F_X0 0 0 0 0 0];
%                     convergedflag=1;
%                     break
%                  end
            % ===== Fit polynomial approximation and find minimum along 1-D line search ===
            if ceil(max(max(abs(Sn))))==0
               convergedflag=1;  % X0 in a constraint corner or at point of zero gradient
               %===== log results to global history & screen =======
               precision=-min(floor(log10(minstepsize)));  % precision of minimum step size
               magnitude=ceil(log10(max(abs(X0)))); % order of mag of max value in DV 'inputvar'
               if magnitude < 1
```

```
      magnitude=1;  %artificially designate magnitude=1 for format purpose if 0<X0<1
    end
    % create format string with numerical formats matching no of input variables
    formatstring='%s%s%s';
    for jj=1:2  %format for both X0 and X0new
      for ii=1:count_dv
        formatstring=strcat(formatstring,'%',num2str(magnitude+precision+3),'.',...
                                            num2str(precision+1),'f%s');
      end
    end
    formatstring=strcat(formatstring,'\r');
    mymagnitude=magnitude+precision+4;  %formatting for X0 & X0new with spaces
    myformatstringsub=[];
    for formatii=1:count_dv
      myformatstringsub=strcat(myformatstringsub,'%',num2str(mymagnitude),...
        '.',num2str(precision+2),'f');
    end
    myformatstring=strcat('%s%s%s',myformatstringsub,'%s','%6.4f\r');
    F_X0=F(1,count_dv+2);
    % ---- write output to screen -----
    fprintf(myformatstring,'Jump ',num2str(jumps),' complete, X0=(',X0,...
                    '); in a corner or point of zero gradient, fmin=',F_X0);
    global_his=[global_his; X0point jumps X0 F_X0 0 0 0 0 0 0];
    % ---- write output to global history output file ----
    [globalhisrows,globalhiscols]=size(global_his);
    textstring=[];
    for jj=1:globalhiscols
     textstring=cat(2,textstring,char(44),num2str(global_his(globalhisrows,jj)));
    end
   fprintf(fid20,'%s\r\n',textstring);
else   % X0 NOT in a constraint corner or at point of zero gradient
    [linecoef,rsqrd,minfcalc,F_xline,constrflag,gmax]=OneDSearch(X0,Sn,F,maxdist,...
                                                     jumps,X0point);
    if constrflag==1  % constrflag=1 means no Sn values point to feasible space
      F_X0=F(1,count_dv+2);
      [rows,cols]=size(global_his);
      X0old=global_his(rows,3:count_dv+(3-1)); %resulting best pt from prev jump
      F_X0old=global_his(rows,count_dv+4);  %resulting best F_X0 from prev jump
      g_old=F_xline(1,count_dv+7);  % feasibility result of X0 this jump
      global_his=[global_his; X0point jumps X0old F_X0old 0 0 0 0 feval g_old];
      convergedflag=1;
      break
    end
    %===== compute new X0 and log results to global history & screen =======
    precision=-min(floor(log10(minstepsize)));  % precision of minimum step size
    magnitude=ceil(log10(max(abs(X0)))); % order of mag of max value in DV 'inputvar'
    if magnitude < 1
      magnitude=1;  %artificially designate magnitude=1 for format purpose if 0<X0<1
    end
    % create format string with numerical formats matching no of input variables
    formatstring='%s%s%s';
    for jj=1:2  %format for both X0 and X0new
      for ii=1:count_dv
        formatstring=strcat(formatstring,'%',num2str(magnitude+precision+3),'.',...
                                            num2str(precision+1),'f%s');
      end
    end
    formatstring=strcat(formatstring,'\r');
    X0new=X0+minfcalc(2)*Sn;  % new X0, start of next 'jump'
    mymagnitude=magnitude+precision+6;  %formatting for X0 & X0new with spaces
    myformatstringsub=[];
    for formatii=1:count_dv
      myformatstringsub=strcat(myformatstringsub,'%',num2str(mymagnitude),...
        '.',num2str(precision+2),'f');
    end
```

```
       myformatstring=strcat('%s%s%s',myformatstringsub,'%s',myformatstringsub,...
          '%s%6.4f%s%6.4f%s%6.4f%s%10.8f%s%6.3f%s%d%s%d\r');
       pcnterror=(minfcalc(3)-minfcalc(1))/minfcalc(3)*100;  %percent error of min point
       % ---- write output to screen -----
       fprintf(myformatstring,'Jump ',num2str(jumps),' complete, X0=(',X0,...
          '); ends at X1=(',X0new,'), feval =',minfcalc(3),' @ x=',minfcalc(2),...
          ', maxdist=',maxdist,', R^2=',rsqrd,', Min-Pnt Error(%)=',pcnterror,...
                   ', Ttl Fun Evals this X0 = ',feval,', Max Constraint Value =',gmax);
       global_his=[global_his; X0point jumps X0new ...
              minfcalc(3) minfcalc(2) maxdist rsqrd pcnterror feval gmax];
       % ---- write output to global history output file ----
       [globalhisrows,globalhiscols]=size(global_his);
       textstring=[];
       for jj=1:globalhiscols
         textstring=cat(2,textstring,char(44),num2str(global_his(globalhisrows,jj)));
       end
      fprintf(fid20,'%s\r\n',textstring);
   end
   % ----- check for convergence -----
     % -- convergence due to design variable stagnation
     if convergedflag==0  %jump not yet reached a convergence flag-check the following
        desvarstalled=0;
        for jj=1:count_dv
           if abs(X0new(jj)-X0(jj))<=DVstepmin
              desvarstalled=desvarstalled+1;  % this DV movement <= defined minimum
           end
        end
        if desvarstalled==count_dv
           tempconvergedflag=2;  % all des variables <= defined min therefore converged
        end
     % -- convergence due to function eval stagnation for this jump
        [globalhisrows,globalhiscols]=size(global_his);
        finalfeval=global_his(globalhisrows,count_dv+4);
        prevfeval=global_his(globalhisrows-1,count_dv+4);
        if abs(finalfeval-prevfeval)/finalfeval*100<=funtol;
           tempconvergedflag=3; %function eval <= defined min [%] therefore converged
        end
        if tempconvergedflag==0
           X0=X0new;  % prep for next loop
        end
        if isnan(rsqrd)==1||isinf(rsqrd)==1 %[1]='NaN' -or- +/-infinity
           fprintf('%s\r',...
                  'R^2=NaN or Inf maeans not enough feasible points to fit curve');
        end
        if tempconvergedflag~=0
           if tempconvergedflag==2
              fprintf('%s\r',...
                 'Design Variable Movement less than allowable tolerance - converged');
              convergedflag=2;
           end
           if tempconvergedflag==3
              fprintf('%s\r',...
                'Function Evaluation change less than allowable tolerance - converged');
              convergedflag=3;
           end
           break
        end
     end
end  % end of inner 'jumps' loop
if tempconvergedflag==0     %no convergence found
   if jumps==maxjumps       % max jumps completed
     fprintf('%s\r',...
        'Convergence not achieved, max jumps completed');
      convergedflag=4;
        % no convergence found, max jumps achieved, flag flipped to exit while loop
```

```
            end
          end
          if constrflag~=0
             fprintf('%s%s\r',...
                    'Feasible results not found along this Sn for Jump = ',num2str(jumps));
%                    'Sn not found that points to feasible space from this X0');
             convergedflag=5;
             % ---- write output to global history output file ----
               [globalhisrows,globalhiscols]=size(global_his);
               textstring=[];
               for jj=1:globalhiscols
                 textstring=cat(2,textstring,char(44),num2str(global_his(globalhisrows,jj)));
               end
              fprintf(fid20,'%s\r\n',textstring);
             %abort this X0, no Sn found that points to design space with feasible result constr.
          end
        end  % end of convergedflag==0 while loop
        globalfevalcount=globalfevalcount+feval;
     end  % end of outer 'X0point' loop
     fprintf(fid20,'%s\r\n','end of data');
     fclose(fid20);  %close global history log
     [globalhisrows,globalhiscols]=size(global_his);  % size of final global_his
%        % -----------------------------------------------------------------------------------------
        % --- print X0 radius values from origin as 'debug' confirmation of constraint routine ---
        Radius=[];
xcoef=consxcoef;
ycoef=consycoef;
        for ii=1:globalhisrows
           rad=0;
           if ii~=globalhisrows
              if global_his(ii+1,1)>global_his(ii,1)
                 Radius=[Radius; global_his(ii,1) sqrt((global_his(ii,3)-xcoef)^2+(global_his(ii,4)-ycoef)^2)];
%                   for jj=1:count_dv
%                      rad=rad+(global_his(ii,jj+2))^2;
%                   end
%                   rad=sqrt(rad);
%                   Radius=[Radius; global_his(ii,1) rad];
              end
           else
              Radius=[Radius; global_his(ii,1) sqrt((global_his(ii,3)-xcoef)^2+(global_his(ii,4)-ycoef)^2)];
%                for jj=1:count_dv
%                   rad=rad+(global_his(ii,jj+2))^2;
%                end
%                rad=sqrt(rad);
%                Radius=[Radius; global_his(ii,1) rad];
           end
        end
        Radius
        minradius=min(Radius(:,2))
        maxradius=max(Radius(:,2))
        % -----------------------------------------------------------------------------------------
     % ====== find minimum feasible feval from global search - input to SQP search ====
     fevalmin=1e12;
     fevalcol=2+count_dv+1;  % column for function eval value in global_his
     glX0no=1e12;  % initialize as flag for feasible result found or not
     for ii=1:globalhisrows
        if global_his(ii, globalhiscols)<0  %limit search for min feval to feasible results
           if global_his(ii,fevalcol)<fevalmin
              fevalmin=global_his(ii,fevalcol);
              glX0no=global_his(ii,1);                  %feasible minimum found
              glX0min=global_his(ii,3:3+(count_dv-1));
              glfevalmin=global_his(ii,fevalcol);
           end
        end
     end
```

```
    if glX0no==1e12
       fprintf('%s\r\n',' ');
       fprintf('%s\r\n','##########################################################');
       fprintf('%s\r\n','##             NO FEASIBLE RESULT FOUND !!              ##');
       fprintf('%s\r\n','##      VERIFY THAT CONSTRAINTS ARE APPLICABLE.         ##');
       fprintf('%s\r\n','##########################################################');
       fprintf('%s\r\n',' ');
    else
       fprintf('%s\r\n',' ');
       fprintf('%s\r\n','######################################');
       fprintf('%s\r\n','## Minimum Value from Global Search ##');
       fprintf('%s\r\n','--------------------------------------');
       my_stringsub=strcat('%s',myformatstringsub,'%s%3.0f\r\n');
       fprintf(my_stringsub,'    Identified optimum Design Point at (',glX0min,...
                                               ') - from Global X0 Point #',glX0no);
       my_stringsub=strcat('%s%',num2str(mymagnitude),'.',num2str(precision+2),'f\r\n');
       fprintf(my_stringsub,'    Estimated opt value based on 1-D line search = ',glfevalmin);
       fprintf('%s%5.0f\r\n','    No of (global) function evaluations = ',...
                                                       globalfevalcount);
       fprintf('%s\r\n','######################################');
    end

function [feaspts]=globalsearchpts(maxjumps) % *** CHARTED ***
% this function defines the potential X0 points for the global search and filters them for those
% that meet all constraint criteria.
global pjctpath ANSYSpjctpath scriptname my_project my_resultsfile inputvar count_dv inputcols
global SpaceAllow his my_flagfile minstepsize count_outv outputvar arch_dir feval penalty result
global g_dv g_out globalX0 global_his rsqrdallow endpcnt linesearchsteps polyfitorder constrrad
global testfunction distdesspc funtol StartPt optsoltn HIS globalstart globalStartQty glX0min
global searchtype DVstepmin mu minspan max1Dpts maxpolyfitorder ErrorAllow1D consxcoef consycoef
global  solvertype my_solverfunction my_obj_fun my_cons_fun
  %% Define Global Start Points (globalX0)
     % -----------------------------------------------------------------------------------------
     % This section identifies the feasible design space based on constraints appropropriate
     % to the Design Variables (as opposed to constraints appropriate to results) and then
     % identifies the specified start points (X0) matching the quantity specified in the input
     % data section.
     % -----------------------------------------------------------------------------------------
     searchtype=2;  % (0)=global search, (1)=fmincon search, (2)=globalsearchpts
     linesearchsteps=polyfitorder+1; %no of pts (add to X0) to use for 1-D poly approx.
     if polyfitorder<2  % Error check for number of divisions spec'd for linesearch
        line1='  The starting (minimum) order of the polynomial must be >= 2.    ';
        line2='                                                                  ';
        line3='  Please modify the value - polyfitorder - and restart. ';
        % note:  all string lengths must be same to put in following composite vector
        msgstring=char(line1,line2,line3);
        msgtitle='Error - 1-D LineSearchSteps<3';
        msgbox(msgstring,msgtitle,'error');
        stop;  % end program execution within MATLAB - not clean but functional
     end
     StartQty=globalStartQty;
     StartPt=0;  % flag to indicate that global start matrix meets defined number of points (rows)
     while StartPt==0
        if globalStartQty<2  % Error check for number of global start points
           line1=' The number of global start points must be greater than 1.';
           line2='                                                           ';
           line3='  Please modify the value - globalStartQty - and restart. ';
           % note:  all string lengths must be same to put in following composite vector
           msgstring=char(line1,line2,line3);
           msgtitle='Error - globalStartQty<1';
           msgbox(msgstring,msgtitle,'error');
           stop;  % end program execution within MATLAB - not clean but functional
        end
        if rem(globalStartQty,1)>0  % Error check for globalStartQty not integer
           line1=['The value globalStartQty = ',num2str(globalStartQty),' is not an integer.'];
```

```
   line2='                                                    ';
   line3='  Please modify the value to be an integer and restart. ';
   % note:  all string lengths must be same to put in following composite vector
   msgstring=char(line1,line2,line3);
   msgtitle='Error - globalStartQty not an integer';
   msgbox(msgstring,msgtitle,'error');
   stop;  % end program execution within MATLAB - not clean but functional
end
% ---------------------------------------------------------------------------
% populate potential global X0 matrix based on user selected method
% ---------------------------------------------------------------------------
if globalstart==0  % start points defined per evenly divided design space
   potglobalX0=[];  % initialize matrix for globalX0 values
   for ii=1:globalStartQty   % build by column, each column = design variable value
      value=0;
      for jj=1:globalStartQty^(ii)  %2^(ii)    % loop thru 'sets of rows' by globalDVdiv
         value=value+1;
         for kk=1:globalStartQty^(count_dv-(ii))  %2^(3-1)=4
            row=((jj-1)*globalStartQty^(count_dv-(ii)))+kk;
            minvalue=str2double(inputvar(ii,2));
            maxvalue=str2double(inputvar(ii,3));
            ratio=(value-1)/(globalStartQty-1);
            cellvalue=ratio*(maxvalue-minvalue)+minvalue;
            potglobalX0(row,ii)=cellvalue;
         end
         if value==globalStartQty
            value=0;
         end
      end
   end
end
if globalstart==1  % startpoints defined per haltonset
    PX0=haltonset(count_dv,'skip',1);       % haltonset for 'count_dv' columns
      % skip first haltonset point since always in extreme corner.  Don't want this point
      % for test functions since symmetric functions make it easier for point at origin
      % to point at the minimum value at the center. (eg. Rastrigins function)
    potglobalX0=net(PX0,StartQty); % length of matrix
    [potrows,potcols]=size(potglobalX0);
       % rows of potglobalX0 = each potential startpoint
       % columns represent each design variable (2 DV = 2 cols, etc.)
       % potglobalX0 always a 2-D matrix
    for ii=1:potrows   %scale potglobalX0 from haltonset 0~1 to full input variable range
        for jj=1:potcols
            minvalue=str2double(inputvar(jj,2));
            maxvalue=str2double(inputvar(jj,3));
            potglobalX0(ii,jj)=(maxvalue-minvalue)*potglobalX0(ii,jj)+minvalue;
        end
    end
end
if globalstart==2  % startpoints defined per latin hypercube sampling
    potglobalX0=lhsdesign(StartQty,count_dv,'iterations',5); % LHS for 'count_dv' columns
    [potrows,potcols]=size(potglobalX0);
       % rows of potglobalX0 = each potential startpoint
       % columns represent each design variable (2 DV = 2 cols, etc.)
       % potglobalX0 always a 2-D matrix
    for ii=1:potrows   %scale potglobalX0 from haltonset 0~1 to full input variable range
        for jj=1:potcols
            minvalue=str2double(inputvar(jj,2));
            maxvalue=str2double(inputvar(jj,3));
            potglobalX0(ii,jj)=(maxvalue-minvalue)*potglobalX0(ii,jj)+minvalue;
        end
    end
end
if globalstart==3  % startpoints defined per random number generator (uniform distr)
    potglobalX0=rand(StartQty,count_dv); % rand does not require Statistical Toolbox
    [potrows,potcols]=size(potglobalX0);
       % rows of potglobalX0 = each potential startpoint
       % columns represent each design variable (2 DV = 2 cols, etc.)
       % potglobalX0 always a 2-D matrix
    for ii=1:potrows   %scale potglobalX0 from rand 0~1 to full input variable range
        for jj=1:potcols
            minvalue=str2double(inputvar(jj,2));
            maxvalue=str2double(inputvar(jj,3));
            potglobalX0(ii,jj)=(maxvalue-minvalue)*potglobalX0(ii,jj)+minvalue;
        end
    end
end
% ----- end populate potential start points ----------
% Filter global X0 matrix for points that satisfy constraints for design variables.  Add
% a column to the globalX0 matrix that indicates result (1=feasible, 0=not-feasible).
%   Comments:
% Constraint function is polled by multiple sections of this code.  Here, the input
% variables are polled for satisfaction to the constraints established as opposed to
% results being checked against constraints governing defined bounds for output variables.
% Both checks (design variables and results) need to reside within the same constraint
% function.  When the global X0 variables are being checked, results
% do not yet exist.  To prevent an error since the results vector is empty, the flag
% 'resultflag' is set to '0' (off) to indicate no results exist and that section of the
% constraint function code is skipped.  When the objective function is called (FEA solve)
% resultflag' is set to 1' (on) indicating that all polls from there on out will include
% results and that section of the constraint function will be activated.'resultflag' is a
% global variable.  set it back to '0' if constraint function evaluated for DV only.
% ----- Filter and strip potential start points per input-variable feasibility constr. ---
[rows,cols]=size(potglobalX0);
feaspts=0;  % initialize counter for number of feasible global X0 values found
for ii=1:rows
   x=[];  %initialize temporary DV point for constraint eval
   for jj=1:count_dv
      x=[x potglobalX0(ii,jj)];
   end
   % --- test for constraints ---
       result=[]; % no results to pass to Constraint_Fun
%              resultflag=0; %  % [0]=no results exist, [1]=results to be passed
       [gout,~,gpos]=Constraint_Fun(x,1);
       %                [gout,~,gpos]=Constraint_Fun(x,result);
   % ----------------------------
   if gpos<=0
      potglobalX0(ii,count_dv+1)=1; %global X0 point satisfies constraints...feasible
      feaspts=feaspts+1;
   else
      potglobalX0(ii,count_dv+1)=0; %global X0 point not satisfy constraints, not-feasible
   end
end
% strip the non-feasible points from the globalX0 vector
globalX0=[];  %initialize feasible global X0 vector
for ii=1:rows
   if potglobalX0(ii,count_dv+1)==1  % X0 point feasible
      globalX0=[globalX0;potglobalX0(ii,1:count_dv)]; % put point in the feasible vector
   end
end
[rowglobal,colglobal]=size(globalX0);  % size of resulting feasible start points
if globalstart==0  % start points defined per evenly divided design space
    StartPt=1;  % do not add points to evenly divided start point routine, take only
              % what meets feasibility tests from original division of design variables
else  % for haltonset/LHS point sets, pointsets are >> needed, filter to find enough
    if rowglobal>=globalStartQty  % check if requested number of startpoints correct
        StartPt=1;   % flag to indicate that global start matrix meets defined rows
       if feaspts>globalStartQty
          globalX0=globalX0(1:globalStartQty,:);
        end
```

```
          [rowglobal,colglobal]=size(globalX0);
          feaspts=rowglobal;
       end
   end
   StartQty=StartQty+1;
end   %end of StartPt==0 'while' loop
  % --- plot feasible global X0 values ---
  if count_dv==3  % plot a 3-D plot if 3 input variables
     figure
     plot3(potglobalX0(:,1),potglobalX0(:,2),potglobalX0(:,3),'x','MarkerEdgeColor','k',...
                                            'MarkerSize',3);
     hold on
     plot3(globalX0(:,1),globalX0(:,2),globalX0(:,3),'o','MarkerFaceColor','r');
     grid on
     title({'\bf Global X0 Values Conforming to Input Variable Constraints';...
            ['\bfTotal ',num2str(feaspts),'\bf Points']});
     xlabel(inputvar(1,4));
     ylabel(inputvar(2,4));
     zlabel(inputvar(3,4));
     axis([str2double(inputvar(1,2)) str2double(inputvar(1,3)) str2double(inputvar(2,2))...
        str2double(inputvar(2,3)) str2double(inputvar(3,2)) str2double(inputvar(3,3))]);
     legend('Potential global X0 points','Feasible global X0 points','Location','Best');
  end
  if count_dv==2   % plot a 2-D plot if 2 input variables
     figure
     plot(potglobalX0(:,1),potglobalX0(:,2),'x','MarkerEdgeColor','k');
     hold on
     plot(globalX0(:,1),globalX0(:,2),'o','MarkerFaceColor','r');
     grid on
     title({'\bf Global X0 Values Conforming to Input Variable Constraints';...
            ['\bfTotal ',num2str(feaspts),'\bf Points']});
     xlabel(inputvar(1,4));
     ylabel(inputvar(2,4));
     axis([str2double(inputvar(1,2)) str2double(inputvar(1,3)) str2double(inputvar(2,2))...
        str2double(inputvar(2,3))]);
     legend('Potential global X0 points','Feasible global X0 points');
  end
  % --- construct a question dialog for globalX0 size approval
     line1='*** WARNING ***';
     line2=' Too many starting points during the global optimization phase may';
     line3=' take excessive compute time depending upon the model.';
     line4=' ';
     line5=['Are ',num2str(feaspts),' global X0 points acceptable?'];
     msgstring=char(line1,line2,line3,line4,line5);
     gX0choice=questdlg(msgstring,'Global X0 Vector Size','Yes','No','Cancel','Cancel');
     switch gX0choice  %evaluate selection from question dialog box
        case 'Yes'
          % size is selected to be ok...no action needed
        case 'No'
          line1='  Please modify the value - globalDVdiv - and restart. ';
          msgstring=char(line1);
          msgtitle='Size of globalX0 not acceptable';
          msgbox(msgstring,msgtitle,'help');
          stop;  % end program execution within MATLAB - not clean but functional
       case 'Cancel'
          stop;  % end program execution within MATLAB - not clean but functional
     end
  % ---
  fprintf('%s\r',' ');
  fprintf('%s\r','################################################################');
  fprintf('%s%s%s\r','A total of ',num2str(feaspts),...
     ' X0 points will be evaluated for global optimum.  Global optimizations conducted ');
  fprintf('%s%s\r','via steepest descent algorithm with polynomial approximation ',...
                                                        '1-D line search.');
  fprintf('%s"%s"%s%s\r','The max number of steepest descent ','jumps',...
                                                    ' = ',num2str(maxjumps));
    fprintf('%s\r','################################################################');
    fprintf('%s\r',' ');

function [myXOPT,myOptf,SQPfevalcount]=SQPsearch(feaspts,SQPmaxjumps)
% this function conducts a SQP optimization search given a starting 'SQPX0' point which may
% either be specified directly by the user or is an output of the global search routine.  A maximum
% number of jumps is also specified by the user.
global pjctpath ANSYSpjctpath scriptname my_project my_resultsfile inputvar count_dv inputcols
global SpaceAllow his my_flagfile minstepsize count_outv outputvar arch_dir feval penalty result
global g_dv g_out globalX0 global_his rsqrdallow endpcnt linesearchsteps polyfitorder constrrad
global testfunction distdesspc funtol StartPt optsoltn HIS globalstart globalStartQty glX0min
global searchtype DVstepmin mu minspan max1Dpts maxpolyfitorder ErrorAllow1D consxcoef consycoef
global  solvertype my_solverfunction my_obj_fun my_cons_fun
  % =====  'Outer loop' = Each Identified Global Start Point ========
 % ====== plot test surface here only if 'SQP' or 'combo' solution, else plot in global ======
    if optsoltn~=0
    if testfunction==1
      nopts=50;
      PlotTestFunction(nopts)  %plot countour plot of function
    elseif testfunction==2
      nopts=50;
      PlotTestFunction(nopts)  %plot countour plot of function
    elseif testfunction==3
      nopts=150;
      PlotTestFunction(nopts)  %plot countour plot of function
    elseif testfunction==4
      nopts=150;
      PlotTestFunction(nopts)  %plot countour plot of function
    end
    end
  globalfevalcount=0;  % counter for total number of fevals during global search
  HIS=[]; %initialize history record
  for X0point=1:feaspts  % step through the feasible start points
    searchtype=1;  % (0)=global search, (1)=fmincon search, (2)=globalsearchpts
    X0=globalX0(X0point,:);  % X0 = 'selected X0' from the globalX0 vector at row X0point
    if optsoltn==2 %use optimum result of global search as start point for single fmincon search
      X0=glX0min;
    end
    feval=0;  %initialize function eval tracking number
    % Define bounds, initial guess at start point
    %   set constraints A, B, Aeq, Beq = 0 since constraints handled separately
    Obj=@Objective_Fun;
    Con=@Constraint_Fun;
    A=[];B=[];Aeq=[];Beq=[];
    LB=[]; UB=[];  %initialize bounds variables
    his=[];  %initialize history record
    for ii=1:count_dv
      LB=[LB str2double(inputvar{ii,2})];
      UB=[UB str2double(inputvar{ii,3})];
    end
%           X0point=1;
    if X0point<100
      X0ptstr=strcat('00',num2str(X0point));
      if X0point<10
        X0ptstr=strcat('000',num2str(X0point));
      end
    end
    if searchtype==0
      StartPt=['GXo',X0ptstr];  % 'name' of start pt (X0) to be added to archive filenames
    else
      StartPt=['LXo',X0ptstr];  % SQP (fmincon) start point
    end
    OP=optimset('Display','iter','Algorithm','sqp','DiffMinChange',DVstepmin,...
                                  'TolX',DVstepmin,'maxfunevals',SQPmaxjumps);
        [XOPT,FVAL,EXITFLAG,OUTPUT,LAMBDA,GRAD] = fmincon(Obj,X0,A,B,Aeq,Beq,LB,UB,Con,OP);
        % compile convergence history - his = [ -XO locations- fval] from Objective_Fun
        [hisrows,hiscols]=size(his); % size of his for this fmincon start point
        [HISrows,HIScols]=size(HIS);  % size of overall history log (all fmincon start points)
        for hisii=1:hisrows
            HIS(hisii+HISrows,:)=[X0point his(hisii,:)];
        end
        [HISrows,HIScols]=size(HIS);  % size of overall history log (all fmincon start points)
    end % end this jump
%      [HISrows,HIScols]=size(HIS);
    SQPfevalcount=HISrows;
    % write optimization history to file in archive directory
    summary_file=[pjctpath,'archive\FOCUSED OPT RESULTS SUMMARY.TXT'];
    fid20 = fopen(summary_file, 'w'); %open file for write permiss.-discard contents
    % note '\r\n' is carriage return with new line in Windows OS
    fprintf(fid20,'%s\r\n','# ######################################################');
    fprintf(fid20,'%s\r\n','#   Results summary from Optimization (comma delimited)');
    fprintf(fid20,'%s\r\n','# -----------------------------------------------------');
        for ii = 1:length(my_project)
            thisproject=char(my_project(ii,:));
            fprintf(fid20,'%s%s\r\n','  Project = ',thisproject);
        end
    fprintf(fid20,'%s\r\n','# -----------------------------------------------------');
    if optsoltn==2
        fprintf(fid20,'%s\r\n','#  Optimum Design');
        for ii=1:count_dv
           fprintf(fid20,'%s%s%s%s%s%12.5f\r\n','#   ',inputvar{ii,4},', ',inputvar{ii,1},...
              ' = ',XOPT(ii));
        end
        fprintf(fid20,'%s\r\n','# -----------------------------------------------------');
    end
    fprintf(fid20,'%s\r\n','#  Convergence History');
    fprintf(fid20,'%s\r\n','# -----------------------------------------------------');
    hdr_text1=['Global X0'];
    hdr_text2=['St Pt #'];
    for ii=1:count_dv
        hdr_text1=cat(2,hdr_text1,char(44),inputvar{ii,1});
        hdr_text2=cat(2,hdr_text2,char(44),inputvar{ii,4});
    end
    for ii=1:count_outv
        hdr_text1=cat(2,hdr_text1,char(44),outputvar{ii,1});
        hdr_text2=cat(2,hdr_text2,char(44),outputvar{ii,3});
    end
    fprintf(fid20,'%s\r\n',hdr_text1(:));  % need to use {:} for cell array
    fprintf(fid20,'%s\r\n',hdr_text2(:));
    [rows,cols]=size(HIS);
    for ii=1:rows
        textline=[];
        for jj=1:cols
            textline=cat(2,textline,num2str(HIS(ii,jj)),char(44));
        end
        fprintf(fid20,'%s\r\n',textline);
    end
%      fprintf(fid20,'%s\r\n','# -----------------------------------------------------');
%      fprintf(fid20,'%s\r\n','#  Optimum Design');
%      for ii=1:count_dv
%         fprintf(fid20,'%s%s%s%s%s%12.5f\r\n','#   ',inputvar{ii,4},', ',inputvar{ii,1},...
%            ' = ',XOPT(ii));
%      end
%      fprintf(fid20,'%s\r\n','# -----------------------------------------------------');
    fprintf(fid20,'%s\r\n','end of data');
    fclose(fid20);
    % ------ find result of each jump to identify to find best of constrained solutions ----
    HISOPT=[];   %re-initialize values for search
    for ii=1:(feaspts-1)
```

```
        myOptf=1e12; flag=0;
        for jj=1:(HISrows-1)
            if HIS(jj+1,1)==ii+1
                if flag==0
                    HISOPT=[HISOPT;HIS(jj,:)];  %HISOPT = temp matrix of result of each X0
                    flag=1;
                end
            end
        end
    end
    HISOPT=[HISOPT;HIS(HISrows,:)];  %add result of last X0
    [HISOPTrows,HISOPTcols]=size(HISOPT);
    myOptf=1e12; myXOPT=[];
    for ii=1:HISOPTrows
        if HISOPT(ii,HISOPTcols)<myOptf
            myOptf=HISOPT(ii,HISOPTcols);
            myXOPT=HISOPT(ii,2:2+count_dv-1);
        end
    end
    fprintf('%s\r\n','######################################');
    fprintf('%s\r\n','## Minimum Value from SQP Search ##');
    fprintf('%s\r\n','--------------------------------------');
    myXOPT
    myOptf
    SQPfevalcount
    fprintf('%s\r\n','######################################');

function append (file1,file2,file3,solveloop)
% this function combines the text of two files into one file
global pjctpath ANSYSpjctpath scriptname my_project my_resultsfile inputvar count_dv inputcols
global SpaceAllow his my_flagfile minstepsize count_outv outputvar arch_dir feval penalty result
global g_dv g_out globalX0 global_his rsqrdallow endpcnt linesearchsteps polyfitorder constrrad
global testfunction distdesspc funtol StartPt optsoltn HIS globalstart globalStartQty glX0min
global searchtype DVstepmin mu minspan max1Dpts maxpolyfitorder ErrorAllow1D consxcoef consycoef
global  solvertype my_solverfunction my_obj_fun my_cons_fun
   min_file=[pjctpath,file3];
   fid1 = fopen(file1, 'r');
   fid2 = fopen(file2, 'r');
   A = fread(fid1);
   B = fread(fid2);
   fid11 = fopen(min_file, 'w');
   fwrite(fid11,[A;B]);
   fclose(fid11);
   fclose(fid1);
   fclose(fid2);
   % rename input script file to include feval tracking number & put in archive folder
      for ii=1:length(file3)     % identify only the primary part of the name (vs '.txt' extension)
         if file3(ii)==char(46)
            pointer=ii-1;
         end
      end
      fnamemain=file3(1:pointer);  % filename without extension
      fnameext=file3(pointer+2:length(file3));  % filename's extension
      if feval<1000
         fevalstr=strcat('0',num2str(feval));
         if feval<100
            fevalstr=strcat('00',num2str(feval));
            if feval<10
               fevalstr=strcat('000',num2str(feval));
            end
         end
      end
      if feval>=1000    %add '0's' to feval for function evals < 1000
         fevalstr=num2str(feval);
      end
```

```
     if solveloop<100
         solveloopstr=strcat('0',num2str(solveloop));
         if solveloop<10
             solveloopstr=strcat('00',num2str(solveloop));
         end
     end
     % append name and move input script file to archive directory
     cmdline=['copy ' file3 strcat(' archive','\',fnamemain,'_',StartPt,'_feval',...
                                         fevalstr,'_',solveloopstr,'.',fnameext)];
     [status,cmdout]=dos(cmdline);

function [f]=Objective_Fun(x)
global pjctpath ANSYSpjctpath scriptname my_project my_resultsfile inputvar count_dv inputcols
global SpaceAllow his my_flagfile minstepsize count_outv outputvar arch_dir feval penalty result
global g_dv g_out globalX0 global_his rsqrdallow endpcnt linesearchsteps polyfitorder constrrad
global testfunction distdesspc funtol StartPt optsoltn HIS globalstart globalStartQty glX0min
global searchtype DVstepmin mu minspan max1Dpts maxpolyfitorder ErrorAllow1D consxcoef consycoef
global  solvertype my_solverfunction my_obj_fun my_cons_fun
 % --- this loop set up for series execution of multiple ANSYS models ---
  [modelrows,modelcols]=size(my_project);
  if testfunction==0
      maxmodelcount=modelcols; % if not a test function, repeat Obj_Fun for multi-models
  else
      maxmodelcount=1; % if IS a test function, not multi-models therefore only 1 time thru
  end
  result=[];  %initialize variable
  feval=feval+1; %track index of fevals-primarily for naming of input/output files to archive
  format long
  for solveloop=1:maxmodelcount
       % --- select the correct family variant ---
       if x(1)<45 && x(2)<45
           family_variant=1;
       elseif x(1)<45 && x(2)>45 && x(2)<63
           family_variant=2;
       elseif x(1)<45 && x(2)>63
           family_variant=3;
       elseif x(1)>45 && x(1)<63 && x(2)>45 && x(2)<63
           family_variant=4;
       elseif x(1)>45 && x(1)<63 && x(2)>63
           family_variant=5;
       elseif x(1)>63 && x(2)>63
           family_variant=6;
       end
       % ---- correct family variant selected ---
       Results_file=[pjctpath,my_resultsfile];
       flagfile=[pjctpath,my_flagfile];  % file to flag that solve is complete
       % ---- delete flagfile from previous run if it exists -------
           fileexists=0;  %initialize variable
           fileexists=exist(flagfile,'file'); % returns 0 if file written by ANSYS does not exist
           if fileexists~=0
               delete(flagfile);  % del prev flag file indicating that ANSYS job is completed
           end
       % ------- end del flagfile -----------
       % ************************************************************************
       % Write header text to temporary file containing user defined input data
       % ************************************************************************
         header_file=[pjctpath,'header.txt'];
         fid10 = fopen(header_file, 'w');  % open file for write permission-discard contents
         % note '\r\n' is carriage return with new line in Windows OS
         fprintf(fid10,'%s\r\n','# ####################################################');
         fprintf(fid10,'%s\r\n','#   Define input data for use in script below');
         fprintf(fid10,'%s\r\n','# ----------------------------------------------------');
         fprintf(fid10,'%s%s%s\r\n','my_path = "',ANSYSpjctpath,'"');
         if modelcols~=1   % more than 1 geometry variant available per function evaluation
             thisproject=char(my_project(solveloop,geometry_variant,:));
```

```
        elseif modelcols==1 % only 1 geometry variant to choose from per functio evaluation
            thisproject=char(my_project(solveloop,:));
        end
        fprintf(fid10,'%s%s%s\r\n','my_project ='",thisproject,'"');
%           fprintf(fid10,'%s%s%s\r\n','my_project ='",my_project(solveloop),'"');
        fprintf(fid10,'%s%s%s\r\n','my_flagfile ='",my_flagfile,'"');
        fprintf(fid10,'%s%s%s\r\n','my_resultsfile ='",my_resultsfile,'"');
        fprintf(fid10,'%s%d\r\n','#No of Design Variables = ',count_dv);
        fprintf(fid10,'%s\r\n','# --- input design variables this iteration ---');
       % fprintf(fid10,'%s\t%s\t%s\t%s\r\n','#','Param No.','Value','Description');
        inputstrname=[];
        inputstrval=[];
        inputstrdescr=[];
        precision=-min(floor(log10(minstepsize)));  % precision of minimum step size
        magnitude=ceil(log10(max(abs(x)))); % order of magnitude of maximum value in DV 'x'
        for ii=1:count_dv    % loop to write input variables values to temp string array
          % adjust format string to include +1 order of magnitude to left of decimal point and +2
          % orders of magnitude to the right
          tempformat=strcat('%',num2str(magnitude+3),'.',num2str(precision+2),'f');
          tempformatzero=strcat('%',num2str(2),'.',num2str(4),'f');  %format if x(ii)='0'
          if ii<count_dv
            inputstrname=cat(2,inputstrname,char(39),inputvar{ii,1},char(39),',');  %ASCII(39)='
            if x(ii)==0
                inputstrval=cat(2,inputstrval,num2str(x(ii),tempformatzero),',');
            else
                inputstrval=cat(2,inputstrval,num2str(x(ii),tempformat),',');
            end
            inputstrdescr=cat(2,inputstrdescr,char(39),inputvar{ii,4},char(39),',');
          else
            inputstrname=cat(2,inputstrname,char(39),inputvar{ii,1},char(39));
            if x(ii)==0
                inputstrval=cat(2,inputstrval,num2str(x(ii),tempformatzero));
            else
                inputstrval=cat(2,inputstrval,num2str(x(ii),tempformat));
            end
            inputstrdesc=cat(2,inputstrdescr,char(39),inputvar{ii,4},char(39));
          end
        end
        % print values of design variable(s) to header text for this iteration
        fprintf(fid10,'%s%s%s\r\n','var_name=[',inputstrname,']');
        fprintf(fid10,'%s%s%s\r\n','var_val=[',inputstrval,']');
        fprintf(fid10,'%s%s%s\r\n','var_desc=[',inputstrdesc,']');
        fprintf(fid10,'%s\r\n','# --- output design variables this iteration ---');
        outputstrname=[];
        outputstrdescr=[];
        for ii=1:count_outv    % loop to write output variables to temp string array
          if ii<count_outv
            outputstrname=cat(2,outputstrname,char(39),outputvar{ii,1},char(39),',');
            outputstrdescr=cat(2,outputstrdescr,char(39),outputvar{ii,3},char(39),',');
          else
            outputstrname=cat(2,outputstrname,char(39),outputvar{ii,1},char(39));
            outputstrdesc=cat(2,outputstrdescr,char(39),outputvar{ii,3},char(39));
          end
        end
        % print description of output variable(s) to header text for this iteration
        fprintf(fid10,'%s%s%s\r\n','result_name=[',outputstrname,']');
        fprintf(fid10,'%s%s%s\r\n','result_desc=[',outputstrdesc,']');
        fprintf(fid10,'%s\r\n',' ');
        fclose(fid10);
     % ************************************************************************
     % Append header text to main body file (selected above) containing user defined input data
     % ************************************************************************
        append (header_file,scriptname,'inputscript.txt',solveloop) ;
     % ========== Evaluate Objective Function =================
        % -----------------------------------------------
```

```
    if solvertype==0   % ANSYS solve requested
       %  ------ Evaluate via direct ANSYS solve ---------
        [resultarray]=ANSYSsolve(flagfile,Results_file); %Call ANSYS & read results file
        result(:,solveloop)=resultarray; %convert 'temporary' pass variable to global variable
    elseif solvertype==1  % External MATLAB function solve requested
        [f,result]=my_obj_fun(x);
         % --- write results file ---
         fid50=fopen(Results_file,'w');  % write permission, discard contents
         fprintf(fid50,'%s\r\n','------- Input Data ---------');
         fprintf(fid50,'%s\r\n','[Param Name] = [Param Value] ;  [Description]');
         for ii=1:count_dv
             mystring=[char(inputvar(ii,1)),' = ',num2str(x(ii)),' ; ',char(inputvar(ii,4))];
             fprintf(fid50,'%s\r\n',mystring);
         end
         fprintf(fid50,'%s\r\n','Ready to solve:');  %match text as flag for later
         fprintf(fid50,'%s\r\n','Solve Complete:');  %match text as flag for later
         for ii=1:length(result)
             text1=cat(2,outputvar{ii,3},' = ');
             fprintf(fid50,'%s%s\r\n',text1,num2str(result(ii),9));
         end
         fclose(fid50);
    elseif solvertype==2  % Internal test function solve requested
    %  ---- Evaluate via specified test function ----
       [result]=EvalTestFunction(x); %'x' is var passed to this larger 'Objective' function
       % --- write results file ---
         fid50=fopen(Results_file,'w');  % write permission, discard contents
         fprintf(fid50,'%s\r\n','------- Input Data ---------');
         fprintf(fid50,'%s\r\n','[Param Name] = [Param Value] ;  [Description]');
         for ii=1:count_dv
             mystring=[char(inputvar(ii,1)),' = ',num2str(x(ii)),' ; ',char(inputvar(ii,4))];
             fprintf(fid50,'%s\r\n',mystring);
         end
         fprintf(fid50,'%s\r\n','Ready to solve:');  %match text as flag for later
         fprintf(fid50,'%s\r\n','Solve Complete:');  %match text as flag for later
         fprintf(fid50,'%s%s\r\n','scalarresult = ',num2str(result,9));
         fclose(fid50);
    end
% ------------------------------------------------------------------------------------------
  % ===============Rname and Copy Results File to Archive Directory =================
    for ii=1:length(my_resultsfile)
       if my_resultsfile(ii)==char(46)
          pointer=ii-1;                   % find length of filename without extension
       end
    end
    fnamemain=my_resultsfile(1:pointer);  % filename without extension
    fnameext=my_resultsfile(pointer+2:length(my_resultsfile));  %filename's extension
    if feval<1000
       fevalstr=strcat('0',num2str(feval));
       if feval<100
          fevalstr=strcat('00',num2str(feval));
          if feval<10
             fevalstr=strcat('000',num2str(feval));
          end
       end
    end
    if solveloop<100
        solveloopstr=strcat('0',num2str(solveloop));
        if solveloop<10
           solveloopstr=strcat('00',num2str(solveloop));
        end
    end
    if feval>=1000    %add '0's' to feval for function evals < 1000
       fevalstr=num2str(feval);
    end
    % append name and move results file to archive directory
```

```
        cmdline=['copy ' my_resultsfile strcat(' archive','\',fnamemain,'_',StartPt,'_feval',...
                    fevalstr,'_',solveloopstr,'.',fnameext)];
        [status,cmdout]=dos(cmdline);
  end  %end 1:maxmodelcount loop - all results stored in result(:,solveloop)
   %
==================================================================================================
   % ==   Specify Objective Function and Results Constraint Variables                        ==
   %
==================================================================================================
   %  ANSYS Solve Section -
   % ----------------------
   % ---- Objective function for ANSYS solve --
      % Note that results are stored in cell array 'result{ii}' in the order of outputvar[r,c].
      % The objective function 'f' below should be populated with the appropriate results as
      % required and referenced above in the data input section of function 'main'.
      %
      if solvertype==0   % ANSYS solve requested
        % derive key performance components
          % normalize results by expected output magnitude for objective function ONLY
          my_result=[];
          [rowsresult,colsresult]=size(result);
          for solveloop=1:maxmodelcount
            my_temp_result=[];
            for iiout=1:rowsresult
              my_temp_result=[my_temp_result result(iiout,solveloop)/...
                                              str2double(outputvar(iiout,2))];
            end
            my_result(:,solveloop)=my_temp_result;
          end
          % key performance components
             % ---------------------------------------------
             % *** NOTE: OBJECTIVE_FUN METRICS USE 'NORMALIZED' RESULT VALUES, NOT RAW ---
             %            raw results = [result]
             %      normalized results = [my_result]
             % ---------------------------------------------
             % --- Constants to be used in this section ---
               L1=x(1);
               L2=x(2);
               L3=45;
               L4=63;
               Lb=72;
               weight_sum=1.0;
               weight_range=0.5;
             sum_disp=0;  range_disp=0;
             for solveloop=1:3
                sum_disp=sum_disp+my_result(2,solveloop); % sum of integrated displ
                range_disp=range_disp+my_result(3,solveloop); % sum of range of displ
             end
             f=sum_disp*weight_sum+range_disp*weight_range;

   % ------------------------------------------------------------------------------------------
   %  External MATLAB Function Solve Section -
   % ------------------------------
      elseif solvertype==1   % External MATLAB solve requested
         % objective function defined externally in referenced file
   % ------------------------------------------------------------------------------------------
   %  Test Function Solve Section -
   % ------------------------------
      elseif solvertype==2   % Test Function solve requested

         f=result;  %scalar output of 2D test funciton
           f=f/str2double(outputvar{1,2});  % normalize result for use in Obj Fntn Variable
      end
   %
==================================================================================================
    %  Penalty Function -
    % -------------------
     for solveloop=1:maxmodelcount  %evaluate constraint for penalty separately for each model result
        [g,heq,gpos]=Constraint_Fun(x,solveloop);
        constr_sum=0;
        if length(g)>0
            for gii=1:length(g)
                constr_sum=constr_sum+(max(0,g(gii)))^2;
            end
            f=f+constr_sum*penalty;

        end
     end
    % -----  Check to verify that result not empty - must pass 'real' matrix -----
      [rowsRC,colsRC]=size(result);
      if rowsRC+colsRC<1
          result=[0];    %ensure that result<>[] even if no results constraints required
      end
    % write results to history matrix
    if rowsRC~=1||colsRC~=1
        hisresult=[]; hisf=[];
        for ii=1:colsRC
            hisresult=[hisresult result(:,ii).'];
        end
        hisf=[hisf f];
    elseif rowsRC==1&&colsRC==1
        hisresult=result;
        hisf=f;
    end
    his=[his;x hisresult hisf];   % tracking history

function [g,heq,gpos]=Constraint_Fun(x,solveloop)
global pjctpath ANSYSpjctpath scriptname my_project my_resultsfile inputvar count_dv inputcols
global SpaceAllow his my_flagfile minstepsize count_outv outputvar arch_dir feval penalty result
global g_dv g_out globalX0 global_his rsqrdallow endpcnt linesearchsteps polyfitorder constrrad
global testfunction distdesspc funtol StartPt optsoltn HIS globalstart globalStartQty glX0min
global searchtype DVstepmin mu minspan max1Dpts maxpolyfitorder ErrorAllow1D consxcoef consycoef
global  solvertype my_solverfunction my_obj_fun my_cons_fun
% --- definition of variables ---
%            x = input (design variable) coordinates of this point
%    result = result value for those variables corresponding to a results constraint (see end of
%                'Objective_Fun' subroutine).
%            g = inequality constraint matrix (g>0 is infeasible, g<=0 is feasible)
%          heq = equality constraint matrix
%     gpos = flag that infeasible constraint found
% --------------------------------
 gpos=0;  % [0] means no constraints violated - initialize variable with this assumption
%  if searchtype==1  %(0)=global search, (1)=fmincon search, (2)=globalsearchpts
%     result=[]; %[result] not defined in SQP search, define here for below logic
%  end
 emptytest=isempty(result); % [1] if result=[], [0] if result not empty
      % --- if results exist or operating from fmincon, evaluate against ouput constraints ---
 outputconstr=(emptytest==0||searchtype==1); % convert tests to logical scalar output
% ----- Scratch space to document constraint eqtns ------
    % --- Constants to be used in this section ---
       L1=x(1);
       L2=x(2);
       L3=45;
       L4=63;
       Lb=72;
       spaceallow=1.250;  % minimum spacing between bearings
    %  --- input constraints ---
    %    L1 & L2 not equal to L3 or L4 and L2>L1
    %      abs(L1-L3)>=spaceallow;  0>=spaceallow-abs(L1-L3);  g=spaceallow-abs(L1-L3);
    %                                                     g=1-abs(L1-L3)/spaceallow
```

```
   %     abs(L1-L4)>=spaceallow;  0>=spaceallow-abs(L1-L4):  g=spaceallow-abs(L1-L4)
   %                                                g=1-abs(L1-L4)/spaceallow
   %     abs(L2-L3)>=spaceallow;  0>=spaceallow-abs(L2-L3):  g=spaceallow-abs(L2-L3)
   %                                                g=1-abs(L2-L3)/spaceallow
   %     abs(L2-L4)>=spaceallow;  0>=spaceallow-abs(L2-L4):  g=spaceallow-abs(L2-L4)
   %                                                g=1-abs(L2-L4)/spaceallow
   %     L2 >= L1+spaceallow;  0>=L1+spaceallow-L2;  g=(L1-L2)+spaceallow; g=(L1-L2)/spaceallow+1
   %     L1>=0+spaceallow:  0>=spaceallow-L1;    g=spaceallow-L1;    g=1-L1/spaceallow
   %     L2<=Lb-spaceallow; 0>=L2-Lb+spaceallow; g=(L2-Lb)+spaceallow; g=(L2-Lb)/spaceallow+1
   %
   %  --- output (results) constraints ---
   %   abs(deflection_static)<=2;  0>=abs(defl_static)-2;  g=abs(defl_static)-2;
   %                                                  g=abs(result(1,ii))/2-1;
   %
   %
   %  ** NOTE: Do not use 'Length' as variable name since it is a Matlab command. **
% ===================================================================================================
% ==   Specify Objective Function and Results Constraint Variables                        ==
% ===================================================================================================
% ---------------------------------------------------------------------------------------------
%  ANSYS Solve Section -
% ----------------------
% ---- Constraints for ANSYS solve --
   % Constraint equations include both input and output constraints, grouped separately below.
   % Constraint equations are defined for more clarity in 'scratch space' above.
   %
   if solvertype==0   % ANSYS solve requested
      g_out=[];  g_dv=[];  % initialize variables
      % --- Design variable (input-based) constraints ---
          g_dv(1)=1-abs(L1-L3)/spaceallow;
          g_dv(2)=1-abs(L1-L4)/spaceallow;
          g_dv(3)=1-abs(L2-L3)/spaceallow;
          g_dv(4)=1-abs(L2-L4)/spaceallow;
          g_dv(5)=(L1-L2)/spaceallow+1;
          g_dv(6)=1-L1/spaceallow;
          g_dv(7)=(L2-Lb)/spaceallow+1;
      % --- Result variable (output-based) constraints ---
         if outputconstr==1  %logic var from top of this sub-routine - ok to evaluate results
           % derive key performance components (repeat of Objective_Fun)
           % ---------------------------------------------
           % *** NOTE: CONSTRAINT METRICS USE 'RAW' RESULT VALUES, NOT NORMALIZED ---
           %            raw results = [result]
           %      normalized results = [my_result]
           % ---------------------------------------------
           for solveloop_c=1:3  % each model must meet static defl constraint
              g_out(solveloop_c)=abs(result(1,solveloop_c))/2-1; % static defl for ea result
           end
         end
% ---------------------------------------------------------------------------------------------
%  External MATLAB Solve Section -
% ----------------------
   elseif solvertype==1  %External MATLAB function requested
     [g_dv,g_out,heq]=my_cons_fun(x, result, outputconstr);    %result,
   end



% ---------------------------------------------------------------------------------------------
%  Test Function Solve Section -
% ------------------------------
% ---- Test function objective function (COMMENT THIS SECTION OUT FOR TEST FUNCTION SOLVE)  --
   if testfunction~=0   % Test Function requested
      g_out=[];  g_dv=[];  % no constraints other than design variable side constraints
      % --- Design variable (input-based) constraints ---

         % put any design variable (input-based) constraints here
```

```
    % --- Result variable (output-based) constraints ---
       if outputconstr==1 %logic var from top of this sub-routine - ok to evaluate results
          % add output constraints within this if-end loop
            g_out(1)=constrrad-sqrt((x(1)-consxcoef)^2+(x(2)-consycoef)^2);
       end
       if searchtype==1
          if max(g_out)>0
             flag=1;  % fmincon in infeasible space
          end
       end
   end
% ==========================================================================================
% ---- global constraint matrix assembly and compliance check ---
   g=[g_dv g_out]; %NOTE order of individual '_dv' and '_out' constraints within global matrix
   noconstraint=isempty(g); % [1] if matrix=[], [0] if matrix not empty
   if noconstraint==1
      g=[-1];  % no constraint means unconstrained problem, set g<0 to signal feasible space
   end
   % loop to identify if any constraint is non-compliant - useful for quick evals in main code
   for gii=1:length(g)
     if g(gii)>0;
       gpos=1;    % non compliant constraint found
     end
   end
   heq=[];   % equality constraint
% ---- does Xpot violate any LB/UB side bounds defined in 'inputvar' ----
   for jj=1:length(x)
      if x(jj)< str2double(inputvar(jj,2))
         gpos=2;  % potential X0 point violates LB for this variable
      end
      if x(jj)> str2double(inputvar(jj,3))
         gpos=3;  % potential X0 point violates UB for this variable
      end
   end

function [tempdist]=Pt_Pt_Dist(x1,x2)
global pjctpath ANSYSpjctpath scriptname my_project my_resultsfile inputvar count_dv inputcols
global SpaceAllow his my_flagfile minstepsize count_outv outputvar arch_dir feval penalty
global g_dv g_out globalX0 global_his rsqrdallow endpcnt linesearchsteps polyfitorder constrrad
global testfunction distdesspc funtol StartPt optsoltn HIS globalstart globalStartQty glX0min
global searchtype DVstepmin mu minspan max1Dpts maxpolyfitorder ErrorAllow1D consxcoef consycoef
global  solvertype my_solverfunction my_obj_fun my_cons_fun
% Calculate the distance between 2 multi-variate points
tempdist=0;
for distii=1:length(x1)
   tempdist=tempdist+(x2(distii)-x1(distii))^2;
end
tempdist=sqrt(tempdist);

function [maxdist]=searchmaxdist(X0,Sn,X0point) % *** CHARTED ***
global pjctpath ANSYSpjctpath scriptname my_project my_resultsfile inputvar count_dv inputcols
global SpaceAllow his my_flagfile minstepsize count_outv outputvar arch_dir feval penalty result
global g_dv g_out globalX0 global_his rsqrdallow endpcnt linesearchsteps polyfitorder constrrad
global testfunction distdesspc funtol StartPt optsoltn HIS globalstart globalStartQty glX0min
global searchtype DVstepmin mu minspan max1Dpts maxpolyfitorder ErrorAllow1D consxcoef consycoef
global  solvertype my_solverfunction my_obj_fun my_cons_fun
% ######### Start of Numerical Search Technique ################################################
% --------------------------------------------
%  This subroutine uses numerical search technique along defined search vector Sn to find
%  the distance from X0 to the nearest constraing ('wall').  The numerical search is
%  accomplished by taking 'course' steps along Sn until crossing into infeasible range, then
%  backing up by '1 course step' and restepping 'out' in a much finer step(s) until the
%  'wall' or feasibility boundary is again reached (creates more accurate estimate of
%  boundary, but still in error by as much as length of small step.)  Method works, but
%  takes time due to numerous steps along Sn.  Consider as efficiency improvement replacing
%  numerical search technique below with direct mathematical calculation. (pgs 18-19)
  % ==========  Find max distance along 'Sn' to infeasible region =============
  %   (assumes starting within feasible space definition of starting point - X0 or other)
  feasflag=0;
  step=0;
  if distdesspc~=0
     deltastep=distdesspc/100;
  else
     deltastep=minstepsize;
  end
  while feasflag==0   % coarse search
     step=step+deltastep; % make coarse steps in only one direction
     Xpot=X0+step*Sn;
%          resultflag=0;  %only check for input constraints on subsequent ConstraintEval
%          [feasflag]=ConstraintEval(Xpot);
     % --- test for input variable only constraints ---
        result=[]; % no results to pass to Constraint_Fun
%          resultflag=0; %  % [0]=no results exist, [1]=results to be passed
        [gout,heqout,gpos]=Constraint_Fun(Xpot,1);
        if gpos~=0
           feasflag=1;  % indicates not feasible - exit loop
        end
        if feasflag==1
           break
        end
     % ----------------------------
     if distdesspc~=0
        if step*max(abs(Sn))>(distdesspc*1e4)  % Error check - no apparent constraints
           step
          distdesspc
          Xpot
          Sn
          line1='  The number of steps taken in the 1-D line search              ';
          line2='  are very high.  This may be because constraints are not       ';
          line3='  properly set.  The current search direction is printed in the ';
          line4='  Matlab command window  ';
          line5='   **** Do you want to continue or stop the process? ****     ';
          % note:  all string lengths must be same to put in following composite vector
          msgstring=char(line1,line2,line3,line4,line5);
            msgstring=char(line1,line2,line3);
            gX0choice=questdlg(msgstring,'Unbounding Constraints (Course)?'...
                                              ,'Continue','Stop','Cancel','Stop');
            switch gX0choice  %evaluate selection from question dialog box
              case 'Continue'
                % User elected to continue...no action needed
              case 'Stop'
                line1='  Please check the constraints and restart. ';
                msgstring=char(line1);
                msgtitle='Potential Unbounding Constraints';
                msgbox(msgstring,msgtitle,'help');
                stop;  % end program execution within MATLAB - not clean but functional
             case 'Cancel'
                stop;  % end program execution within MATLAB - not clean but functional
            end
        end
     end
  end   %exit loop when step crosses into infeasible region - backup one notch to feasible
  feasflag=0;
  Xpot;
  Xpot=Xpot-deltastep*Sn; %crossed into infeasible, backup one notch to feasible
  step=step-deltastep;  %reset stepsize for subsequent 'fine' search
%        lastcoursestep=step
  deltastep=deltastep/100;  % fine search = fraction of course step
  while feasflag==0
```

```
    step=step+deltastep;  % make fine steps in only one direction
    Xpot=X0+step*Sn;
%           resultflag=0;  %only check for input constraints on subsequent ConstraintEval
%           [feasflag]=ConstraintEval(Xpot);
    % --- test for input variable only constraints ---
       result=[]; % no results to pass to Constraint_Fun
%          resultflag=0; %  % [0]=no results exist, [1]=results to be passed
       [gout,heqout,gpos]=Constraint_Fun(Xpot,1);
       if gpos~=0
          feasflag=1;  % indicates not feasible - exit loop
       end
       if feasflag==1
          break
       end
    % ----------------------------
    if distdesspc~=0
       if step*max(abs(Sn))>(distdesspc*1e2) % Error check - no apparent constraints
          Sn
         line1='  The number of steps taken in the 1-D line search             ';
         line2='  are very high.  This may be because constraints are not       ';
         line3='  properly set.  The current search direction is printed in the ';
         line4='  Matlab command window  ';
         line5='   **** Do you want to continue or stop the process? ****     ';
         % note:  all string lengths must be same to put in following composite vector
         msgstring=char(line1,line2,line3,line4,line5);
           msgstring=char(line1,line2,line3);
           gX0choice=questdlg(msgstring,'Unbounding Constraints (Fine)?'...
                                             ,'Continue','Stop','Cancel','Stop');
           switch gX0choice  %evaluate selection from question dialog box
              case 'Continue'
                 % User elected to continue...no action needed
              case 'Stop'
                 line1='  Please check the constraints and restart. ';
                 msgstring=char(line1);
                 msgtitle='Potential Unbounding Constraints';
                 msgbox(msgstring,msgtitle,'help');
                 stop;  % end program execution within MATLAB - not clean but functional
             case 'Cancel'
                 stop;  % end program execution within MATLAB - not clean but functional
           end
       end
    end
  end   %exit loop when step crosses into infeasible region - backup one notch to feasible
  feasflag;
  Xpot=Xpot-deltastep*Sn; %exited above loop because crossed into infeasible, backup one notch
   %  Xpot=feasibility limit for refined search
%  [tempdist]=Pt_Pt_Dist(X0,Xpot);   % dist from X0 to Constraint wall via numerical technique
  tempdist=(Xpot-X0)/Sn;    % dist from X0 to Constraint wall via numerical technique
  % ######### End of Numerical search technique ############################################
  maxdist=tempdist;  % maxdist=max 1-D distance along 'S' to infeasible region

function [resultarray]=ANSYSsolve(flagfile,Results_file)
global pjctpath ANSYSpjctpath scriptname my_project my_resultsfile inputvar count_dv inputcols
global SpaceAllow his my_flagfile minstepsize count_outv outputvar arch_dir feval penalty result
global g_dv g_out globalX0 global_his rsqrdallow endpcnt linesearchsteps polyfitorder constrrad
global testfunction distdesspc funtol StartPt optsoltn HIS globalstart globalStartQty glX0min
global searchtype DVstepmin mu minspan max1Dpts maxpolyfitorder ErrorAllow1D consxcoef consycoef
global  solvertype my_solverfunction my_obj_fun my_cons_fun
   % ==============Call ANSYS & Solve==========================
   cmdline=('runwb2.exe -B -R inputscript.txt');  % ANSYS Workbench executable - verify and get
   %                   proper path from 'properties' of ANSYS icon if needed after new install.
   [status,cmdout]=dos(cmdline,'-echo');  % note, MATLAB program execution will pause until
%                                                         dos job is complete
   % status
   % cmdout
   % ============= Wait for Results File to Reappear following Solve ==========================
   % flagfile is a file that is written by the inputscript.txt following the solve as a flag to
   % this code that the solve is complete.  The results file could be used for this purpose,
   % but instead is opened (ie: 'appears' in the directory folder) prior to the solve so that
   % log file type data can be written as the solve executes.
   fileexists=0;  %initialize temp variable
   while fileexists==0
     pause(2) % check for existance of flagfile every 'x' seconds
     fileexists=exist(flagfile,'file'); % returns 0 if file does not exist
   end
   pause(10); %file found, pause few more seconds to verify all file write activity is complete
   % ==============Read Results File =================
   fid20=fopen(Results_file);
   foundit=0; %initialize temp variable
   while foundit==0  % loop to find start of output section of results file
     TLINE=fgetl(fid20);  % read a line of results file and ignore
     if TLINE(1:14)=='Solve Complete'
       foundit=1;
     end
   end
   for ii=1:count_outv    % read results data from output file
     TLINE=fgetl(fid20);  % read next line to start results output
     my_pos=0;  % temp variable for result string line length
     lineflag=0;  % temp flag variable
     while lineflag==0
       my_pos=my_pos+1;
       if TLINE(my_pos)==char(61)    %char(61) is '='
         my_pos=my_pos+1;  % advance pointer by 1 to get off of '=' sign
         lineflag=1;  % '=' found, exit loop
       end
     end
     resultstr=TLINE(my_pos:length(TLINE));  %read numerical value from pointer to end of line
     resultarray(ii)=str2double(resultstr);
   end
   fclose(fid20);

function [scalarresult]=EvalTestFunction(x)
global pjctpath ANSYSpjctpath scriptname my_project my_resultsfile inputvar count_dv inputcols
global SpaceAllow his my_flagfile minstepsize count_outv outputvar arch_dir feval penalty result
global g_dv g_out globalX0 global_his rsqrdallow endpcnt linesearchsteps polyfitorder constrrad
global testfunction distdesspc funtol StartPt optsoltn HIS globalstart globalStartQty glX0min
global searchtype DVstepmin mu minspan max1Dpts maxpolyfitorder ErrorAllow1D consxcoef consycoef
global  solvertype my_solverfunction my_obj_fun my_cons_fun
   X=x(1);
   Y=x(2);
   % ---- De Jong's Function ----
   if testfunction==1
     scalarresult=X^2+Y^2;
   end
   % ---- Rosenbrock's Valley -----
   if testfunction==2
     scalarresult=100*(Y-X^2)^2+(1-X)^2;  %Rosenbrock's Valley in 2D, result = scalar value
   end
   % ---- Rastrigin's Function ----
   if testfunction==3
     scalarresult=10*2+(X^2-10*cos(2*pi*X))+(Y^2-10*cos(2*pi*Y));
   end
   % ---- Schwefel's Function (modified for bounds to be continuous) ----
   if testfunction==4
     scalarresult=(418.9829*2)-(X*sin(sqrt(abs(X)))+Y*sin(sqrt(abs(Y))));
   end

function [linecoef,rsqrd]=myPolyFit(F_xline,my_x,my_y)
global pjctpath ANSYSpjctpath scriptname my_project my_resultsfile inputvar count_dv inputcols
global SpaceAllow his my_flagfile minstepsize count_outv outputvar arch_dir feval penalty result
global g_dv g_out globalX0 global_his rsqrdallow endpcnt linesearchsteps polyfitorder constrrad
global testfunction distdesspc funtol StartPt optsoltn HIS globalstart globalStartQty glX0min
global searchtype DVstepmin mu minspan max1Dpts maxpolyfitorder ErrorAllow1D consxcoef consycoef
global  solvertype my_solverfunction my_obj_fun my_cons_fun
% ========= Compute Polynomial Approximation along Sn for 1-D Search =========
warning('off');  % turn warning message for badly conditioned polynomial off
% ### NOTE:  warning for badly conditioned polynomial occurrs some times, particularly with
%     functions resembling Schwefel's or Rastrigin's with highly non-linear 1-D search paths.
%     Curve fit routine confirmed on these functions and curve fit from MATLAB's polyfit matches
%     a fit to the same data at same polynomial level with EXCEL.  Stepping through process line
%     by line with manual process (EXCEL) matches predicted location of 1-D minimum found by
%     MATLAB's process even though warning messages exist.  Tried using 'mu' element to polyfit
%     and polyval routines per MATLAB's help suggestion, but resulted in incorrect curve fits
%     with R^2 values less than zero.  Therefore, abandoned 'mu' element and just suppressed
%     warning messages during polyfit routine given that methodology is confirmed by comparison
%     to manual process.
   %       mu(1)=mean(F_xline(:,count_dv+2));
   %       mu(2)=std(F_xline(:,count_dv+3));
   %       [linecoef,S,mu]=polyfit(F_xline(:,count_dv+2),F_xline(:,count_dv+3),polyfitorder);
[linecoef,S]=polyfit(F_xline(:,my_x),F_xline(:,my_y),polyfitorder);
%note linecoef=[a^n a^n-1....a^0] where a^0 = constant term, n=max order of
%fit (polyfit(x,y,n))
warning('on');  %turn warning messages back on
% ===== calculate R^2 for curve fit =====
  residsqrd=0;
  sumsqrd=0;
  [F_xlinerows,F_xlinecols]=size(F_xline);
  for fitii=1:F_xlinerows  % sum of squares - residuals
    x=F_xline(fitii,my_x);
    [fcalc]=polyval(linecoef,x);
    factual=F_xline(fitii,my_y);
    residsqrd=residsqrd+(factual-fcalc)^2;    % sum of squares - residuals
    sumsqrd=sumsqrd+(F_xline(fitii,my_y)-mean(F_xline(:,my_y)))^2;
  end
  rsqrd=1-residsqrd/sumsqrd;  % R^2 for 1-D curve fit

function [minfcalc]=PolyMin(mindist,maxdist,linecoef)
global pjctpath ANSYSpjctpath scriptname my_project my_resultsfile inputvar count_dv inputcols
global SpaceAllow his my_flagfile minstepsize count_outv outputvar arch_dir feval penalty result
global g_dv g_out globalX0 global_his rsqrdallow endpcnt linesearchsteps polyfitorder constrrad
global testfunction distdesspc funtol StartPt optsoltn HIS globalstart globalStartQty glX0min
global searchtype DVstepmin mu minspan max1Dpts maxpolyfitorder ErrorAllow1D consxcoef consycoef
global  solvertype my_solverfunction my_obj_fun my_cons_fun
  minfcalc(1)=1e12;  % starting value for variable - minimum value of polynomial
  minfcalc(4)=-1e12; % starting value for variable - maximum value of polynomial
  % ###### Start polynommial minimum values via roots of diferentiated curve ######
  % ++++ Differentiate polynomial for 1-D line search and then find roots to determine min/max
  %      points, then search roots plus endpoints of differentiated polynomial to find min value.
  for min_ii=1:(length(linecoef)-1)   % differentiate polynomial (linecoef)
     termorder=length(linecoef)-min_ii;  % polynomial order of this particular term
     dlinecoef(min_ii)=termorder*linecoef(min_ii);  %diff for this term
  end
%    linecoef
%    dlinecoef
  myroots=roots(dlinecoef);    % roots of differentiated polynomial
  % evaluate polynomial at endpoints to find minimum value
  if polyval(linecoef,0)<minfcalc(1)   % value of basic polynomial at endpoint x=0
     minfcalc(1)=polyval(linecoef,mindist);
     minfcalc(2)=mindist;
  end
  if polyval(linecoef,maxdist)<minfcalc(1)  % value of basic polynomial at endpoint x=maxdist
     minfcalc(1)=polyval(linecoef,maxdist);
     minfcalc(2)=maxdist;
  end
  % evaluate differentiated polynomial between endpoints for value
```

```
   for root_ii=1:length(myroots)
      if isreal(myroots(root_ii))==1    % 1 means root is real, 0 means imag.
         if myroots(root_ii)>=mindist  %verify root between '0' and 'maxdist'
            if myroots(root_ii)<=maxdist
               rootval=polyval(linecoef,myroots(root_ii));  % value of poly at this root
               if rootval<minfcalc(1)
                  minfcalc(1)=rootval;  % new 'lowest' root value at dist = x
                  minfcalc(2)=myroots(root_ii); % new 'lowest' root location (x-value)
               end
               if rootval>minfcalc(4)
                  minfcalc(4)=rootval; % new 'maximum' value along Sn
                  minfcalc(5)=myroots(root_ii); % location of 'maximum' value along Sn
               end
            end
         end
      end
   end

function []=PlotTestFunction(nopts)
global pjctpath ANSYSpjctpath scriptname my_project my_resultsfile inputvar count_dv inputcols
global SpaceAllow his my_flagfile minstepsize count_outv outputvar arch_dir feval penalty result
global g_dv g_out globalX0 global_his rsqrdallow endpcnt linesearchsteps polyfitorder constrrad
global testfunction distdesspc funtol StartPt optsoltn HIS globalstart globalStartQty glX0min
global searchtype DVstepmin mu minspan max1Dpts maxpolyfitorder ErrorAllow1D consxcoef consycoef
global  solvertype my_solverfunction my_obj_fun my_cons_fun
   xlow=str2double(inputvar(1,2));
   xhigh=str2double(inputvar(1,3));
   ylow=str2double(inputvar(2,2));
   yhigh=str2double(inputvar(2,3));
  X1=linspace(xlow,xhigh,nopts);
  Y1=linspace(ylow,yhigh,nopts);
  [X2,Y2]=meshgrid(X1,Y1);
%    [X1,Y1]=meshgrid(xlow:nopts:xhigh,ylow:nopts:yhigh)
% evaluate gridpoints to find max Z over range of X and Y
   Z=[];
%    [Z]=EvalTestFunction(X2,Y2);
   for ii=1:length(X1)
      for jj=1:length(Y1)
         point=[X1(ii) Y1(jj)];
         [gridresult]=EvalTestFunction(point);
         Z(jj,ii)=gridresult;
      end
   end
   maxz=max(max(Z));  %scalar max value within [z]
   minz=min(min(Z));  %scalar min value within [z]
   deltaz=maxz-minz;  %spacing between max and min value within [z]
   % ----------------- plot surface countour ------------------------------------------
      figure
%       subplot(2,1,1);  %top plot of 2-Plot figure
   %    surf(X2,Y2,Z)%,'EdgeColor','none','facecolor','interp')
      surfc(X2,Y2,Z);  % surface contour plot
      view(-28,22);  % adjust the view angle
      axis([xlow xhigh ylow yhigh 0 ceil(maxz)]);
      shading faceted;  % interpolated shading
      xlabel('\bfx axis')
      ylabel('\bfy axis')
      if testfunction~=0
         if testfunction==1
            title('\bfTest Function: De Jongs Function in 2D');
         end
         if testfunction==2
            title('\bfTest Function: Rosenbrocks Valley in 2D');
         end
         if testfunction==3
            title('\bfTest Function: Rastrigins Function in 2D');
```

```
        end
        if testfunction==4
           title('\bfTest Function: Schwefels Function in 2D');
        end
      end
      grid on
      hold off
      pause(0.5) % pause to let plots execute before optimization process starts

function PlotResults(bestX0)
% this function conducts the global search given a number of feasible points and specified max
% jumps.  The actual feasible points are in 'globalX0' as a global variable.
global pjctpath ANSYSpjctpath scriptname my_project my_resultsfile inputvar count_dv inputcols
global SpaceAllow his my_flagfile minstepsize count_outv outputvar arch_dir feval penalty result
global g_dv g_out globalX0 global_his rsqrdallow endpcnt linesearchsteps polyfitorder constrrad
global testfunction distdesspc funtol StartPt optsoltn HIS globalstart globalStartQty glX0min
global searchtype DVstepmin mu minspan max1Dpts maxpolyfitorder ErrorAllow1D consxcoef consycoef
global  solvertype my_solverfunction my_obj_fun my_cons_fun
 % ****************************************************************************
 %   Plot Results
 %  ****************************************************************************
   % --- plot variable X0 starting points ---
    if solvertype==0||solvertype==1  %ANSYS or External MATLAB solution requested
       if count_dv==3  % plot a 3-D plot if 3 input variables
          figure
          plot3(globalX0(:,1),globalX0(:,2),globalX0(:,3),'o','MarkerFaceColor','r');
          hold on
          grid on
          title({'\bf Convergence Paths of Start Points'});
          xlabel(inputvar(1,4));
          ylabel(inputvar(2,4));
          zlabel(inputvar(3,4));
          axis([str2double(inputvar(1,2)) str2double(inputvar(1,3)) str2double(inputvar(2,2))...
             str2double(inputvar(2,3)) str2double(inputvar(3,2)) str2double(inputvar(3,3))]);
        end
        if count_dv==2   % plot a 2-D plot if 2 input variables
          figure
          plot(globalX0(:,1),globalX0(:,2),'o','MarkerFaceColor','r');
          hold on
          grid on
          title({'\bf Convergence Paths of Start Points'});
          xlabel(inputvar(1,4));
          ylabel(inputvar(2,4));
          axis([str2double(inputvar(1,2)) str2double(inputvar(1,3)) str2double(inputvar(2,2))...
             str2double(inputvar(2,3))]);
        end
    elseif solvertype==2  %test function requested
       if testfunction==1
         nopts=50;
       elseif testfunction==2
         nopts=50;
       elseif testfunction==3
         nopts=150;
       elseif testfunction==4
         nopts=150;
       end
       % ----------- plot contours ----------------------------------------------------
       % evaluate gridpoints to find max Z over range of X and Y
       xlow=str2double(inputvar(1,2));
       xhigh=str2double(inputvar(1,3));
       ylow=str2double(inputvar(2,2));
       yhigh=str2double(inputvar(2,3));
       X1=linspace(xlow,xhigh,nopts);
       Y1=linspace(ylow,yhigh,nopts);
       [X2,Y2]=meshgrid(X1,Y1);
       Z=[];
       for ii=1:length(X1)
         for jj=1:length(Y1)
            point=[X1(ii) Y1(jj)];
            [gridresult]=EvalTestFunction(point);
            Z(jj,ii)=gridresult;
         end
       end
       maxz=max(max(Z));  %scalar max value within [z]
       minz=min(min(Z));  %scalar min value within [z]
       deltaz=maxz-minz;  %spacing between max and min value within [z]
       figure
       CL=[];
       Spacing=[.007 .015 .032 .063 .125 .25 .5 1.0];
       CL=Spacing *(deltaz)+minz;
       C=contour(X1,Y1,Z,CL,'linestyle','-','linewidth',1);
       axis square;
       xlabel('\bfx axis')
       ylabel('\bfy axis')
       if testfunction~=0
         if testfunction==1
            title('\bfTest Function: De Jongs Function in 2D - Convergence Paths');
         end
         if testfunction==2
            title('\bfTest Function: Rosenbrocks Valley in 2D - Convergence Paths');
         end
         if testfunction==3
            title('\bfTest Function: Rastrigins Function in 2D - Convergence Paths');
         end
         if testfunction==4
            title('\bfTest Function: Schwefels Function in 2D - Convergence Paths');
         end
       end
       hold on
       % ----------  plot contraint area for test function --------------
          Xconstr=[]; Yconstr=[];
          if testfunction~=0
              for theta=0:1:359   % angle of 0 to 359 degrees, step of 1
                  thetarad=theta*pi/180;
                  Xconstr=[Xconstr;consxcoef+sin(thetarad)*constrrad];
                  Yconstr=[Yconstr;consycoef+cos(thetarad)*constrrad];
              end
          end
          fill(Xconstr,Yconstr,[1 0.75 0.75],'facealpha',0.75);  %filled constraint area
          hold on
       % ---------- end plot contraint area for test function --------------
       grid on
    end
   % --- end plot 'initial' figures ---
   % ---- plot jumps for each starting X0 (Global) point ----
   [globalhisrows,globalhiscols]=size(global_his);
   [rowsHIS,colsHIS]=size(HIS);
   globalplotparams=[];  %initialize temp plot matrix
   if optsoltn~=1  % global solutions exist to be plotted

       if count_dv==2||count_dv==3   % plot a 2-D or 3-D plot of results
         X0point=0;
         for ii=1:globalhisrows
            if X0point~=global_his(ii,1)
                X0point=global_his(ii,1);
            end
            maxX0jumps=0;  % initialize temp variable to count max jumps for this X0 point
               if global_his(ii,2)==0
                  globalplotparams(X0point,1)=ii;  %row loctn of jump 0 for this X0 point
               end
```

```
                if global_his(ii,2)>maxX0jumps
                   globalplotparams(X0point,2)=ii; %row loctn of max jump for this X0 point
                   maxX0jumps=global_his(ii,2);
                end
          end
          [glplotparamrows,glplotparamcols]=size(globalplotparams);
          global_his;
          if count_dv==2  % --- 2D plot of Steepest Feasible Descent Results ---
              for ii=1:glplotparamrows
                 rowmin=globalplotparams(ii,1);
                 rowmax=globalplotparams(ii,2);
                     h1=plot(global_his(rowmin,3),global_his(rowmin,4),'ro','linewidth',3);
                     h2=plot(global_his(rowmin:rowmax,3),global_his(rowmin:rowmax,4),'b--',...
                                                                'linewidth',1);
                     h3=plot(global_his(rowmax,3),global_his(rowmax,4),'bo','linewidth',2);
                     h4=plot(global_his(rowmin:rowmax,3),global_his(rowmin:rowmax,4),'x',...
                           'MarkerEdgeColor','k','MarkerSize',5); %markers for each corner pt
                     h5=plot(bestX0(1),bestX0(2),'bo','MarkerSize',15,'linewidth',3);
              end
          end
          if count_dv==3   % --- 3D plot of Steepest Feasible Descent Results ---
              for ii=1:glplotparamrows
                 rowmin=globalplotparams(ii,1);
                 rowmax=globalplotparams(ii,2);
                 h1=plot3(global_his(rowmin,3),global_his(rowmin,4),global_his(rowmin,5),...
                                                          'ro','linewidth',3);
                 h2=plot3(global_his(rowmin:rowmax,3),global_his(rowmin:rowmax,4),...
                                    global_his(rowmin:rowmax,5),'b--','linewidth',1);
                 h3=plot3(global_his(rowmax,3),global_his(rowmax,4),...
                                          global_his(rowmax,5),'bo','linewidth',2);
                 h4=plot3(global_his(rowmin:rowmax,3),global_his(rowmin:rowmax,4),...
                        global_his(rowmin:rowmax,5),'x','MarkerEdgeColor','k','MarkerSize',5);
              end
              h5=plot3(bestX0(1),bestX0(2),bestX0(3),'bo','MarkerSize',15,'linewidth',3);
          end
           legend([h1 h2 h3 h4 h5],{'Starting Point (X0)','Search Path','Path End Point',...
                               'Corner Marker','Absolute Optimum'},'Location','southwest')
%             colorbar('location','eastoutside');
           legend BOXON;  % turn 'on' box around legend
       end  % end of plot 2-D loop
    end
    if optsoltn~=0   %fmincon solutions exist to be plotted
       globalplotparams=[]; %initialize temp plot matrix
       if count_dv==2||count_dv==3   % plot a 2-D or 3-D plot of results
          X0point=0;
          maxjumps=0;
          for ii=1:rowsHIS
             if X0point~=HIS(ii,1)
                 X0point=HIS(ii,1);
                 globalplotparams(X0point,1)=ii;  %row loctn of jump 0 for this X0 point
             end
             maxjumps=maxjumps+1;
             globalplotparams(X0point,2)=maxjumps;
          end
          [glplotparamrows,glplotparamcols]=size(globalplotparams);
          if count_dv==2   % --- 2D plot of Steepest Feasible Descent Results ---
              for ii=1:glplotparamrows
                 rowmin=globalplotparams(ii,1);
                 rowmax=globalplotparams(ii,2);
                 h1=plot(HIS(rowmin,2),HIS(rowmin,3),'ro','linewidth',3);
                 h2=plot(HIS(rowmin:rowmax,2),HIS(rowmin:rowmax,3),'b--','linewidth',1);
                 h3=plot(HIS(rowmax,2),HIS(rowmax,3),'bo','linewidth',2);
                 h4=plot(HIS(rowmin:rowmax,2),HIS(rowmin:rowmax,3),'x',...
                   'MarkerEdgeColor','k','MarkerSize',5);   %plot markers for each corner pt
                 h5=plot(bestX0(1),bestX0(2),'bo','MarkerSize',15,'linewidth',3);
```

```
            end
         end
         if count_dv==3   % --- 3D plot of Steepest Feasible Descent Results ---
            for ii=1:glplotparamrows
               rowmin=globalplotparams(ii,1);
               rowmax=globalplotparams(ii,2);
               h1=plot3(HIS(rowmin,2),HIS(rowmin,3),HIS(rowmin,4),'ro','linewidth',3);
               h2=plot3(HIS(rowmin:rowmax,2),HIS(rowmin:rowmax,3),HIS(rowmin:rowmax,4),...
                                                   'b--','linewidth',1);
               h3=plot3(HIS(rowmax,2),HIS(rowmax,3),HIS(rowmax,4),'bo','linewidth',2);
               h4=plot3(HIS(rowmin:rowmax,2),HIS(rowmin:rowmax,3),HIS(rowmin:rowmax,4),...
                     'x','MarkerEdgeColor','k','MarkerSize',5); %markers for each corner pt
            end
               h5=plot3(bestX0(1),bestX0(2),bestX0(3),'bo','MarkerSize',15,'linewidth',3);
         end
         legend([h1 h2 h3 h4 h5],{'Starting Point (X0)','Search Path','Path End Point',...
                           'Corner Marker','Absolute Optimum'},'Location','southwest')
         legend BOXON;  % turn 'on' box around legend
%           colorbar('location','eastoutside');
         for ii=1:count_dv   % plot input design variables as function of iterations
           figure
           for jj=1:glplotparamrows
               rowmin=globalplotparams(jj,1);
               rowmax=globalplotparams(jj,2);
               plot(HIS(rowmin:rowmax,ii+1),'b--','linewidth',2);
               hold on;
               titlestr=strcat('\bfConvergence of Design Variable: ',inputvar(ii,4));
               title(titlestr);
               xlabel('\bfIterations');
               ylabelstr=strcat('\bf',inputvar(ii,4),',  ',inputvar(ii,1));
               ylabel(ylabelstr);
               grid on;
           end
          end
         for ii=1:count_outv   % plot input design variables as function of iterations
           figure
           for jj=1:glplotparamrows
               rowmin=globalplotparams(jj,1);
               rowmax=globalplotparams(jj,2);
               plot(HIS(rowmin:rowmax,ii+1+count_dv),'b--','linewidth',2);
               hold on;
               titlestr=strcat('\bfConvergence of Result: ',outputvar(ii,3));
               title(titlestr);
               xlabel('\bfIterations');
               ylabelstr=strcat('\bf',outputvar(ii,3),',  ',outputvar(ii,1));
               ylabel(ylabelstr);
               grid on;
           end
         end
      end  % end of SQP-only plot loop
   end

function [Sn,maxdist,F,constrflag]=SearchVector(X0point,jumps,X0,F)  % *** CHARTED ***
global pjctpath ANSYSpjctpath scriptname my_project my_resultsfile inputvar count_dv inputcols
global SpaceAllow his my_flagfile minstepsize count_outv  arch_dir feval penalty result
global g_dv g_out globalX0 global_his rsqrdallow endpcnt linesearchsteps polyfitorder constrrad
global testfunction distdesspc funtol StartPt optsoltn HIS globalstart globalStartQty glX0min
global searchtype DVstepmin mu minspan max1Dpts maxpolyfitorder ErrorAllow1D consxcoef consycoef
global  solvertype my_solverfunction my_obj_fun my_cons_fun
constrflag=0;  % initialize flag to indicate that no Sn found that points to feasible space
% ===== Find initial search direction 'S' =====
% --- evaluate function at X0 ---
  % --- create string 'name' to be used in output filename ---
  if X0point<1000
      X0ptstr=strcat('0',num2str(X0point));
```

```
    if X0point<100
       X0ptstr=strcat('00',num2str(X0point));
       if X0point<10
          X0ptstr=strcat('000',num2str(X0point));
       end
    end
 end
 if searchtype==0
    StartPt=['GXo',X0ptstr];  % 'name' of start pt (X0) to be added to archive filenames
 else
    StartPt=['LXo',X0ptstr];  % SQP (fmincon) start point
 end
 [f]=Objective_Fun(X0); % evaluate f(X0)
 [g,heq,gpos]=Constraint_Fun(X0,1);
 % --- convert 2D result matrix (potential) to 1D array ---
   [rowsRC,colsRC]=size(result);
   if rowsRC~=1||colsRC~=1
      hisresult=[];
      for ii=1:colsRC
         hisresult=[hisresult result(:,ii).'];
      end
   elseif rowsRC==1&&colsRC==1
      hisresult=result;
   end
 F=[F;0 X0 f hisresult max(g)];
 if jumps==1
    global_his=[global_his;X0point 0 X0 f 0 0 0 0 0 max(g)]; %log X0 to glbl his array
    % ===== Print X0 and F_X0 to screen ======
    precision=-min(floor(log10(minstepsize)));  % precision of minimum step size
    magnitude=ceil(log10(max(abs(X0)))); % order of magnitude of max value in DV 'inputvar'
    if magnitude < 1
       magnitude=1;  %artificially designate magnitude=1 for format purpose if 0<X0<1
    end
    % create format string with numerical formats matching no of input variables
       formatstring='%s';
       for ii=1:count_dv
          % adj format string to include +1 order of mag to left of decimal point and +2
          formatstring=strcat(formatstring,'%',num2str(magnitude+precision+3),'.',...
             num2str(precision+1),'f'); %includes one space between numbers
       end
    formatstring=strcat(formatstring,'%s%6.4f\r');
    fprintf(formatstring,'Start of Jump 1, X0=(',X0,'), feval = ',...
                                     F(1,count_dv+2));  %print X0 to screen
 end
 for ii=1:count_dv   % evaluate offset-X0's to find 'S'
    % set 'minstepsize' for each DV as x% of avg magnitude for that DV (minstepsize diff each DV)
    % rationale is that some DV magnitudes >> others & pcntge of total space (envelope)
    % disproportionatly different for some DV's vs. others.  (eg. K=500~50000, c=.005~.500)
    Snpcntstep=0.5;  % step size (%) chosen for pcnt of ttl span for each DV
    avgMagnitude=(str2double(inputvar(ii,3))-str2double(inputvar(ii,2)))/2;
    Snstepsize=(Snpcntstep/100)*avgMagnitude;  %'x' pcnt of avg magnitude for this input variable
    % --- evaluate offset point as part of array for initial calc of Sn ---
    X0_offset=X0;
    X0_offset(ii)=X0_offset(ii)+Snstepsize;  %offset X0 in specific DV direction
%      X0_offset(ii)=X0_offset(ii)+minstepsize;  %offset X0 in specific DV direction
%          resultflag=0;  %only check for input constraints on subsequent ConstraintEval
%          [feasflag]=ConstraintEval(X0_offset);  %does X0_offset violate any input constraints?
    % --- test for input variable only constraints ---
       result=[]; % no results to pass to Constraint_Fun
%             resultflag=0; %  % [0]=no results exist, [1]=results to be passed
       [gout,heqout,gpos]=Constraint_Fun(X0_offset,1);
    % ----------------------------
    if gpos~=0
       X0_offset(ii)=X0(ii)-minstepsize;  %pick point on the 'other side' of X0
    end
    [f]=Objective_Fun(X0_offset);
    [g,heq,gpos]=Constraint_Fun(X0_offset,1);
     % --- convert 2D result matrix (potential) to 1D array ---
       [rowsRC,colsRC]=size(result);
       if rowsRC~=1||colsRC~=1
           hisresult=[];
           for ii=1:colsRC
               hisresult=[hisresult result(:,ii).'];
           end
       elseif rowsRC==1&&colsRC==1
           hisresult=result;
       end
    F=[F;0 X0_offset f hisresult max(g)];
  end
% ---- solve 'F' simult to get the coefficients for the equation of the response variables----
  % linear approximation of response surface computed via simultaneous equations
  %  surface = sum(ai*xi)+C... e.g. in 3 variables => f=ax+by+cz+(1)C where 'C' = offset
  %  compute via matrix math, coeffs[a;b;c...]=A\B
  %  A=[a(ii) b(ii) c(ii) 1];  B=[f(ii)] (column vector)! where ii=presolved pts
  A=[F(:,2:count_dv+1),ones(count_dv+1,1)]; %Des Variables
  B=F(:,count_dv+2);  % obj function result vector from objective function 'f'
  coef=A\B; % coefs for the equation of plane (obj fnctn) in 'order of Design Variables'
  delF=coef.'; % transpose obj fnctn equation to row vector
% ---- Find gradient of surface ----
  % since surface = linear approximation, gradient = coefficients of design
  % variables.  For 3 variable example, f=ax+by+cz+C; f'=ai+bj+ck; therefore
  % gradient becomes all but the last term of 'delF'.
  gradF=delF(1:count_dv); %gradient of surface = 'S' = search path
% ------- Identify direction of 'S' for descending direction ----------
  % find the descending direction of 'S' using very small step in direction of 'S'
  %   (gives SQP result of 'S' direction at X0 that is 'free' of complications
  %   if surface is highly non-linear...might not get same results if use large
  %   step)
  difflag=0;  diffloop=0; loopout=[];
  while difflag==0  % X0 may be at SQP min/max - expand search until diff found
    diffloop=diffloop+1;  %'counter' to increase step size
    thisstep=(gradF*(diffloop*(minstepsize*500)));
    fa=(X0+thisstep)*coef(1:count_dv)+coef(count_dv+1);  % eval fun with positive offset
    fb=(X0-thisstep)*coef(1:count_dv)+coef(count_dv+1);  % eval fun with negative offset
    thisdiff=abs((fa-fb)/fa*100);  %pcnt difference between fa and fb
    loopout=[loopout;diffloop fa fb thisdiff];
    myloopdata=[diffloop fa fb thisdiff];   % for debugging purposes
    if thisdiff>0.1 %greater than 0.1% difference noted in slopes=indicates slope
       if fa<fb  % assuming positive slope 'S', fa<fb
          S=gradF;  % want descending direction so choose opposite sign for S
       else
          S=-gradF; % slope of S is actually descending so keep the sign
       end
       difflag=1;  %difference in slopes noted & decisions made, exit loop
    end
    if diffloop>1e5
       %fprintf('%s%','No difference in slope detected...pls confirm');
       %loopout   %if loop 'hangs' to here, no diff in slope perceived - abort
       %stop
       S=[];
       for ii=1:count_dv
          S=[S 0];
       end
       difflag=1;
    end
  end
% ----- convert S to a unit vector and redirect if heading into infeasible range -----
  Lngth_S=0;
  for ii=1:count_dv
    Lngth_S=Lngth_S+S(ii)^2;
  end
  Lngth_S=sqrt(Lngth_S);
  if max(S)==0   % X0 at SQP maxima/minima
     Sn=S;      % Sn=S=[0]
     maxdist=0;
  else
     Sn=S/Lngth_S;  % resulting unit vector in search direction 'S'
    % ######### Start of 'Direct Distance' Evaluation Method ##################################
       % find distance from X0 to Side Constraint Boundaries along Sn (max DV envelope)
       Constr_Dist=[];  % vector for constraint distance to each DV boundary in dir of Sn
       for dist_ii=1:count_dv   % step thru loop for each DV
          LB_dist=(str2double(inputvar(dist_ii,2))-X0(dist_ii))/Sn(dist_ii);
          UB_dist=(str2double(inputvar(dist_ii,3))-X0(dist_ii))/Sn(dist_ii);
          Constr_Dist=[Constr_Dist;abs(LB_dist) abs(UB_dist)]; %distance to LB/UB for DV(ii)
       end
       distdesspc=min(Constr_Dist(:,1))+min(Constr_Dist(:,2)); %min dist across between all ii's
         % distdesspc = distance from X0 along Sn to nearest LB + distance along Sn to nearest UB.
         % Note that the nearest LB and UB may not be the same input variable side bound depending
         % upon the slope of Sn and the placement of X0.
% ######### End of 'Direct Distance' Evaluation Method ####################################
     % ===== Detmn if 'Sn' is headed to infeasible (input var) range, redirect if needed =====
     % --- find distance along S from X0 to nearest input constraint
       if distdesspc~=0
          [maxdist]=searchmaxdist(X0,Sn,X0point);  % find dist along Sn from X0 to nearest 'wall'
          maxdistpcnt=maxdist/distdesspc*100;  %pcnt of maxdist as function of des space
       else
          maxdist=DVstepmin;  % avoid division by '0' later on
          maxdistpcnt=0;
       end
     % --- redirect Sn if too close to input constraint ---
       % strategy is that if length along Sn to nearest constraint is < 0.1% of
       % distance across design space, then declare point X0 to 'against a wall' and Sn
       % needs to be adjusted to move along that wall.  This is done by removing the
       % vector component of Sn that points to the wall.  Process is repeated to
       % identify if X0 is 'against 2 or more walls'.  If X0 is in a 'corner' of all
       % constraints where the only descending direction is into infeasible space, then
       % the loop for this X0 point is aborted and the program moves on to evaluation
       % of the next X0 point.
       distpcntthreshold=0.1;  %[%]
       while maxdistpcnt < distpcntthreshold %[percent]
         % find which Sn vector component direction is shortest
         tempdist=[1 -1e12]; %[dummy_ID  starting_min_value]
         for ii=1:count_dv
           tempSn=Sn;   % Sn = 0 indicates either maxima/minima or constraint 'corner'
           tempSn(ii)=0;
           if tempSn==0
             Sn=tempSn;
             maxdistpcnt=1000;  % X0 in a maxima/minima or constraint 'corner'
             break
           end
           Snlngth=sqrt(sum(tempSn.^2)); % find current length of Sn
           tempSn=tempSn/Snlngth;  % normalize Sn to unit vector
%             maxcomponent=max(abs(tempSn));  %find max abs value component in tempSn
%             tempSn=tempSn/maxcomponent;  % normalize tempSn against maxcomponent
           [constrdist]=searchmaxdist(X0,tempSn,X0point);
           if constrdist>tempdist(2)
             tempdist=[ii constrdist]; %closest constraint in direction ii
           end
         end
         if maxdistpcnt>=distpcntthreshold
           break
         end
         % --- remove vector component 'ii'
         ii=tempdist(1);
         Sn(ii)=0;  % remove vector component
```

```
        Snlngth=sqrt(sum(Sn.^2)); % find current length of Sn
       %  maxcomponent=max(abs(Sn));  %find max abs value component in tempSn
        Sn=Sn/Snlngth;  % normalize Sn to unit vector
        [maxdist]=searchmaxdist(X0,Sn,X0point);     % evaluate with modified Sn
        maxdistpcnt=maxdist/distdesspc*100;
        % if that change to Sn successful then exits loop, else repeats
        if maxdistpcnt>=distpcntthreshold
          break
        end
      end
      Snlngth=sqrt(sum(Sn.^2)); % find current length of Sn
      if Snlngth~=0
         Sn=Sn/Snlngth;  % normalize Sn to unit vector
      elseif Snlngth==0
         Sn=Sn;  %Sn=[0 0...] therefore in constraint corner, etc.
      end
      Sn;  % resulting Sn is unit vector 'clear' of close constraints
      maxdist; %max distance in resulting Sn from X0 to closest constraint
  end  % Sn into feasible space found w.r.t. input constraints

function [linecoef,rsqrd,minfcalc,F_xline,constrflag,gmax]=OneDSearch(X0,Sn,F,maxdist,jumps,X0point) % *** CHARTED ***
global pjctpath ANSYSpjctpath scriptname my_project my_resultsfile inputvar count_dv inputcols
global SpaceAllow his my_flagfile minstepsize count_outv outputvar arch_dir feval penalty result
global g_dv g_out globalX0 global_his rsqrdallow endpcnt linesearchsteps polyfitorder constrrad
global testfunction distdesspc funtol StartPt optsoltn HIS globalstart globalStartQty glX0min
global searchtype DVstepmin mu minspan max1Dpts maxpolyfitorder ErrorAllow1D consxcoef consycoef
global  solvertype my_solverfunction my_obj_fun my_cons_fun
  minfcalc=[0 0 0 0 0];  %initialize in case no feasible results found at mingcalc and abort
  constrflag=0;  %initialize variable-[0]=feasible solutions found, [1]=no feasible solutions found
  gmax=0;  % initialize variable
  F_xline=[];  %reinitialize matrix to hold function solves within 1-D line search routine
  % ===== 1D Curve Fit along 'S' =====
  %   to save a function solve, use previously evaluated f(X0) as first point
  [rows,cols]=size(F);
  F_X0=F(1,count_dv+2);   %value of f at X0
  g_X0=F(1,cols);  % constraint value (max) at X0 in last column of F
  F_xline=[0 X0 0 F_X0 0 0 0 g_X0];   % temp matrix to hold function solves within this subroutine
   % F_xline=[row_number Physical_Loc_this_Pt  Pt_loc_on_search_line  Fval_this_Pt ...
   %                       R^2 Pred_Min_this_curve_fit Error_this_Pt_to_curve_fit max_g_this_pt]
  % --- generate initial points ---
  xloc(linesearchsteps)=maxdist;    %n'th point at maxdist, others spaced in between
  xloc(1)=(minspan/100)*maxdist; %pt very near to X0 (xloc(0)) to establish grad<0 per Sn descent
  %distribute remainder 'xloc' points evenly between xloc(1) and maxdist, note xloc=0 is X0 is '0'
  for lineii=2:linesearchsteps
     xloc(lineii)=(maxdist-xloc(1))/(linesearchsteps-1)*(lineii-1)+xloc(1);
  end
  % --- evaluate initial points and populate F_xline ---
  for lineii=1:linesearchsteps %compute f for 1-D search pts per step matrix xloc[]
    % note that f(0) already computed above = 'xloc(0)' = F_xline(1)
    X0line=X0+xloc(lineii)*Sn;
    [f]=Objective_Fun(X0line);
    [g,heq,gpos]=Constraint_Fun(X0line,1);
    gmax=max(g);
    F_xline=[F_xline; lineii X0line xloc(lineii) f 0 0 0 max(g)];
  end
  goodfit=1;
  % === evaluate fit of initial points
     % ==== Check to see if likelihood that range meets feasibility criteria
       my_x=count_dv+2;   % column of F_xline with x-variable - 1D dist along Sn
       my_y=count_dv+7;   % column of F_xline with y-variable - max(g)
       [linecoef,rsqrd]=myPolyFit(F_xline,my_x,my_y);%,temporder,n);
       [mingcalc]=PolyMin(0,maxdist,linecoef); % find estimated min along 1-D search path
         % mingcalc has same form as minfcalc - mingcalc(4) is estimated maximum
       if mingcalc(1)>0  %.001 vs. '0' as tolerance to rough calculations
          if rsqrd>0.98
```

```
            constrflag=1;  % high confidence of no feasible points along Sn
            goodfit=1;  % flag indicating time to exit this subroutine
            gmax=max(F_xline(:,count_dv+7)); % maximum [g] value of 1D points evaluated
          end
        end
      if constrflag==0   % skip these evaluations of points if no confidence of feasible points
        % ==== Compute Polynomial Approximation & R^2 along Sn for 1-D Search =========
          my_x=count_dv+2;   % column of F_xline with x-variable
          my_y=count_dv+3;   % column of F_xline with y-variable
          [linecoef,rsqrd]=myPolyFit(F_xline,my_x,my_y);%,temporder,n);
        % ======  use polynomial approximation to find 1-D minimum along Sn
          [minfcalc]=PolyMin(0,maxdist,linecoef); % find estimated min along 1-D search path
          % -- is proposed solution feasible?  if not, find best solution that is feasible
           testX0=X0+minfcalc(2)*Sn;  % proposed best result
           [g,heq,gpos]=Constraint_Fun(testX0,1);
           gmax=max(g);
           if gmax>0  %find next best solution if optimum is infeasible
               my_x=count_dv+2;   % column of F_xline with x-variable - 1D dist along Sn
               my_y=count_dv+7;   % column of F_xline with y-variable - max(g)
               [glinecoef,grsqrd]=myPolyFit(F_xline,my_x,my_y);%curvefit of 'g';
               fitmatrix=[]; fitstep=maxdist/1000; fitloc=0; tempstep=0; tempfitmatrix=[];
               for fitii=1:1000
                   fitloc=fitstep*fitii;
                   pred_gmax=polyval(glinecoef,fitloc); % pred gmax for this point
                       % temp storage of all responses for plotting purposes
                       tempfitmatrix(fitii,1)=polyval(linecoef,fitloc); %pred f for this point
                       tempfitmatrix(fitii,2)=pred_gmax; %pred g for this point
                       tempfitmatrix(fitii,3)=fitloc; %1D location of predicted minimum
                   if pred_gmax<0
                       tempstep=tempstep+1;
                       fitmatrix(tempstep,1)=polyval(linecoef,fitloc); %pred f for this point
                       fitmatrix(tempstep,2)=pred_gmax; %pred g for this point
                       fitmatrix(tempstep,3)=fitloc; %1D location of predicted minimum
                   end
               end
               %% added rows 12-23-14
               [rowstemp,colstemp]=size(fitmatrix);
               if rowstemp>0
                   [x,i]=min(fitmatrix(:,2));  %i=row index of minimum gmax value
                   minfcalc(2)=fitmatrix(i,3); % distance along S for identified minimum [in]
                   minfcalc(1)=polyval(linecoef,minfcalc(2)); % predicted value of (f)
               else   % no points along 1D line are feasible
          %         constrflag=1;  % no feasible points along Sn
                   minfcalc(2)=0;
                   minfcalc(1)=polyval(linecoef,minfcalc(2));
               end
               %% end added rows
           end
          % minfcalc(1)=fcalc; % polynomial value at dist = x (estimated value of min FVAL)
          % minfcalc(2)=x;  % distance along S for identified minimum [in]
          % minfcalc(3)=actual FVAL from optimum ID'd by 'PolyMin'
          % minfcalc(4)=maximum value along Sn
          % minfcalc(5)= location (1D) of maximum value (similar to minfcalc(2)
         [F_xlinerows,F_xlinecols]=size(F_xline);
         tempmin=1e12;
         for ii=1:F_xlinerows %find min solved (f) that is feasible
             if F_xline(ii,count_dv+3)<tempmin&&F_xline(ii,F_xlinecols)<=0
                 tempmin=F_xline(ii,count_dv+3);
             end
         end
         minfcalc(3)=tempmin;  %min solved f that is also feasible
        % -------- best feasible minfcalc(1),(2) and (3) identified --------
%            % ----- find minimum response within F_xline that is also feasible ----
%                tempf=1e12; gmax=1;
%                for ii=1:F_xlinerows
%                     if F_xline(ii,count_dv+3)<tempf
%                         if F_xline(ii,F_xlinecols)<0 % g<0 means feasible
%                             tempf=F_xline(ii,count_dv+3); % feasible minimum found
%                             constrflag=0;  % no constraint conflicts
%                             gmax=F_xline(ii,F_xlinecols);  % max(g) for this solved point
%                               feasii=ii;  % pointer for feasible point
%                         else
%                 %           gmax=0;  %don't want to loose ID of feasible solution above...
%                 %           constrflag=1;  % point is not feasible
%                         end
%                     end
%                 end
                if gmax>0
                    constrflag=1;  %ID constrflag=1 iff ALL F_xline results infeasible
                end
            % --- check for acceptability criteria ----
                if gmax<0  % feasible min found - check for R^2 and error
                    Error1Dmin=abs((minfcalc(1)-minfcalc(3))/minfcalc(3));
                    if rsqrd < rsqrdallow
                       goodfit=0;  % R^2 value not acceptable, repeat fit with addt'l point
                    end
                    if abs(Error1Dmin)*100>ErrorAllow1D %
                        goodfit=0;  % error between predicted / actual FVAL at min point too large
                    end
                else
                    goodfit=0;  % no feasible result found - need to add another point
                end
      end
    outofbounds=0;  % flag to indicate that search for acceptable pair failed
    while goodfit==0;   %flag goodfit=0 means r^2 target / error not yet achieved
      constrflag=0; %reset constraint flag indicating infeas results for this section
      polyfitorder=polyfitorder+1;  %increment polyfit order by 1
      if polyfitorder>maxpolyfitorder
          polyfitorder=maxpolyfitorder;  % limit polynomial order to avoid numerical problems
      end
      linesearchsteps=linesearchsteps+1; %maintain 1 more point than order of eqtn
      newdist=1e12;   % initialize variable
      % ==== Compute Polynomial Approximation & R^2 along Sn for 1-D Search ==========
       my_x=count_dv+2;   % column of F_xline with x-variable
       my_y=count_dv+3;   % column of F_xline with y-variable
       [linecoef,rsqrd]=myPolyFit(F_xline,my_x,my_y);%,temporder,n);
      % ======  use polynomial approximation to find 1-D minimum along S
      [minfcalc]=PolyMin(0,maxdist,linecoef); % find estimated min along 1-D search path
      if rsqrd>0.95  %diff logic schemes if 'f' curvefit good or not
          % -- R^2 good, proposed solution feasible?  if not, find best solution that is feasible
            testX0=X0+minfcalc(2)*Sn;  % proposed best result
            [g,heq,gpos]=Constraint_Fun(testX0,1);
            gmax=max(g);
            if gmax>0  %find next best solution if optimum is infeasible
                my_x=count_dv+2;   % column of F_xline with x-variable - 1D dist along Sn
                my_y=count_dv+7;   % column of F_xline with y-variable - max(g)
                [glinecoef,grsqrd]=myPolyFit(F_xline,my_x,my_y);%curvefit of 'g';
                fitmatrix=[]; fitstep=maxdist/1000; fitloc=0; tempstep=0;
                for fitii=1:1000
                    fitloc=fitstep*fitii;
                    pred_gmax=polyval(glinecoef,fitloc); % pred gmax for this point
                    if pred_gmax<0
                        tempstep=tempstep+1;
                        fitmatrix(tempstep,1)=polyval(linecoef,fitloc); %pred f for this point
                        fitmatrix(tempstep,2)=pred_gmax; %pred g for this point
                        fitmatrix(tempstep,3)=fitloc; %1D location of predicted minimum
                    end
                end
                    %% added rows 12-23-14
                    [rowstemp,colstemp]=size(fitmatrix);
                    if rowstemp>0
                        [x,i]=min(fitmatrix(:,2));  %i=row index of minimum gmax value
                        minfcalc(2)=fitmatrix(i,3); % distance along S for identified minimum [in]
                        minfcalc(1)=polyval(linecoef,minfcalc(2)); % predicted value of (f)
                    else   % no points along 1D line are feasible
        %               constrflag=1;  % no feasible points along Sn
                        minfcalc(2)=0;
                        minfcalc(1)=polyval(linecoef,minfcalc(2));
                    end
                    %% end added rows
            end
            % minfcalc(1)=fcalc; % polynomial value at dist = x (estimated value of min FVAL)
            % minfcalc(2)=x;  % distance along S for identified minimum [in]
            % minfcalc(3)=actual FVAL from optimum ID'd by 'PolyMin'
            % minfcalc(4)=maximum value along Sn
            % minfcalc(5)= location (1D) of maximum value (similar to minfcalc(2)
            [F_xlinerows,F_xlinecols]=size(F_xline);
            tempmin=1e12;
            for ii=1:F_xlinerows %find min solved (f) that is feasible
                if F_xline(ii,count_dv+3)<tempmin&&F_xline(ii,F_xlinecols)<=0
                    tempmin=F_xline(ii,count_dv+3);
                end
            end
            minfcalc(3)=tempmin;  %min solved f that is also feasible
      else  %poor fit for curve fit of f to data - sort for best of F_xline for seed point
          % keep predicted minimum point from minfcalc(1),(2),(3) for next solved point
          % determine then if it is feasible and/or potential minimum to be expanded upon
      end
       % -------- best feasible minfcalc(1),(2) and (3) identified --------
       % --- remove outlier pts if reached max number of pts along line ---
       if outofbounds==1
          F_xlinerows=max1Dpts+1;  % use F_xlinerows as flag to enter end adding pts and find min.
       end
       if F_xlinerows>=max1Dpts %max number of points evaluated...filter points to improve curve fit
          qtrspan=maxdist/4;  %keep points<+/-qtrspan from tempdist (potential min)
          if bestpt~=0  % best feasible point found within F_xline
                bestval; bestpt; bestdist;% bestg; % best feasible results from solved data in F_xline
                tempF_xline=[];
                for iitemp=1:F_xlinerows  %filter points to < qtrspan
                    if abs(F_xline(iitemp,count_dv+2)-F_xline(bestpt,count_dv+2))<=qtrspan
                        if F_xline(iitemp,F_xlinecols)<0  % only keep pts with feasible results
                            tempF_xline=[tempF_xline; F_xline(iitemp,:)];
                        end
                    end
                end
               F_xline=tempF_xline;  % points filterd to reduced span around best feasible point
                my_x=count_dv+2;   % column of F_xline with x-variable
                my_y=count_dv+3;   % column of F_xline with y-variable
                [linecoef,rsqrd]=myPolyFit(F_xline,my_x,my_y);%,temporder,n);
               % ======  use polynomial approximation to find 1-D minimum along S
               [F_xlinerows,F_xlinecols]=size(F_xline);
               minend=1e12;  maxend=-1e12;
               for ii=1:F_xlinerows  %find min and max endpoints of new F_xline for curve fit
                if F_xline(ii,count_dv+2)<minend
                    minend=F_xline(ii,count_dv+2);
                end
                if F_xline(ii,count_dv+2)>maxend
                    maxend=F_xline(ii,count_dv+2);
                end
               end
               % === find estimated min val and location for best feas point from solved data
               minfcalc(1)=polyval(linecoef,bestdist);  % evaluate polynomial from best solved point
               minfcalc(2)=bestdist;  % 1D location of new point
               minfcalc(3)=bestval; % best solved value in F_xline
%                  gmax=bestg;  % max constraint value for best solved value in F_xline
```

```
        goodfit=1;  % process complete - exit loop
     else
         constrflag=1;  %constrflag=1 means no values found within feasible space this Sn
         goodfit=1;  %process complete - exit loop
     end
end
if goodfit==1
     testX0=X0+minfcalc(2)*Sn;  % proposed best result
     [g,heq,gpos]=Constraint_Fun(testX0,1);
     gmax=max(g);
     break
end
% ==== New Point Needed - Don't duplicate new point if too close to existing in F_xline
   % ---- [minfcalc] to new, higher order curve fit feasible result?
   tempspacing=1e12;  %initialize temp variable to find closest F_xline result to minfcal(2)
   for ii=1:F_xlinerows
       if abs(F_xline(ii,count_dv+2)-minfcalc(2))<tempspacing
           tempspacing=abs(F_xline(ii,count_dv+2)-minfcalc(2));
           tempii=ii; % closest F_xline result found
       end
   end
   if F_xline(tempii,F_xlinecols)<0  %predicted result is feasible - use as seed for next pt.
       gmax=F_xline(tempii,F_xlinecols);  % predicted min = feasible
   else  %pred result not feas - find F_xline point with 1.)feasibility and 2.)lowest feval
       tempF_xline=[];thisline=0;
       for ii=1:F_xlinerows
           if F_xline(ii,F_xlinecols)<=0 % find feasible results in F_xline
               thisline=thisline+1;
               tempF_xline(thisline,:)=F_xline(ii,:);
           end
       end
       [rows,cols]=size(tempF_xline);
       if rows>0
           [x,feasii]=min(tempF_xline(:,count_dv+3)); %find min f within feasible F_xline
           minfcalc(1)=tempF_xline(feasii,count_dv+5);
           minfcalc(2)=tempF_xline(feasii,count_dv+2);
           minfcalc(3)=tempF_xline(feasii,count_dv+3);
       elseif rows==0 %no feasible results in F_xline, find closest to feasible result
           [x,feasii]=min(F_xline(:,F_xlinecols)); %find minimum g within F_xline
           minfcalc(1)=F_xline(feasii,count_dv+5);
           minfcalc(2)=F_xline(feasii,count_dv+2);
           minfcalc(3)=F_xline(feasii,count_dv+3);
       end
   end
   % --- minfcalc(2) now holds seed for next point ---
   outofbounds=0;  % initialize flag to indicate that search for acceptable pair failed
   [F_xlinerows,F_xlinecols]=size(F_xline);
   counter=max(F_xline(:,1)); % counter = max value = 1 less than actual number of rows
   duplicatepoint=0;  % flag to identify new point duplicates existing within F_xline
   thisminspan=abs(minfcalc(2)-F_xline(1,count_dv+2));%dist pred min to X0 along Sn
   for iicheck=2:counter+1  % find dist pred min to nearest neighboring point
       if abs(minfcalc(2)-F_xline(iicheck,count_dv+2))<thisminspan
           thisminspan=abs(minfcalc(2)-F_xline(iicheck,count_dv+2));
       end
   end
   if thisminspan/maxdist*100<minspan
       duplicatepoint=1;  %pot point < minspan to existing point
   end
   if duplicatepoint==0 % no nearby point found - ok to make new point at this loc
       newdist=minfcalc(2);
   else  % find nearest gap to proposed point and place new point midspan
     % === add new point midway between neighboring existing pt in area of potential minimum
         tempdist=minfcalc(2);
         xloc=F_xline(:,count_dv+2);  % evaluate point positions in 1-D along Sn so far
         sortxloc=sort(xloc); % sort xloc for temp use in following sub-section of routine
```

```
if rsqrd<0  % debugging check
   rsqrd
   stop
end
[F_xlinerows,F_xlinecols]=size(F_xline);
% --- add point routine dependent upon location of predicted min (tempdist) ---
if tempdist>=max(sortxloc)
   tempdist=max(sortxloc);
end
if tempdist~=0
   myflag=0;
   for iicheck=1:(length(sortxloc)) %look for nearest open gap to put new point
      if tempdist~=sortxloc(iicheck)
         if tempdist<sortxloc(iicheck)  %0<tempdist<sortxloc(1)
            step=0;
            minbound=0; maxbound=1;
            if (tempdist-0)/max(sortxloc)>=minspan/100
               newdist=(tempdist+0)/2;
               myflag=1;
            end
            if myflag==0
               if (sortxloc(iicheck)-tempdist)/max(sortxloc)>=minspan/100
                  newdist=(tempdist+sortxloc(iicheck))/2;
                  myflag=1;
               end
            end
            while myflag==0   % look for first gap > 0 that is wide enough
               newmax=maxbound+step;
               newmin=minbound+step;
               if step==0
                  if(sortxloc(newmax)-0)/max(sortxloc)>=minspan/100
                     newdist=(0+sortxloc(newmax))/2;
                     myflag=1;
                  end
                  if myflag~=0
                     break
                  end
               else
                  if sortxloc(newmax)>max(sortxloc)
                     outofbounds=1;  % exceeded maxdist range
                  end
                  if(sortxloc(newmax)-sortxloc(newmin))/max(sortxloc)>=...
                                                   minspan/100
                     newdist=(sortxloc(newmin)+sortxloc(newmax))/2;
                     myflag=2;
                  end
                  if myflag~=0
                     break
                  end
               end
               step=step+1;
            end
         end  %end if tempdist<sortxloc(iicheck)
         if tempdist>sortxloc(iicheck)
            if tempdist<sortxloc(iicheck+1)
               minval=sortxloc(iicheck);
               maxval=sortxloc(iicheck+1);
               % check if maxbound-minbound > 'minspan' of overall maxdist
               % if not, put new point in first region where span > minspan
               if (maxval-tempdist)/max(sortxloc)>=minspan/100
                  newdist=(tempdist+maxval)/2;
                  myflag=3;
               end
               if myflag==0
                  if (tempdist-minval)/max(sortxloc)>=minspan/100
               newdist=(minval+tempdist)/2;
               myflag=3;
            end
         end
         if myflag==0
            maxendflag=0; minendflag=0;
            step=1; minbound=iicheck;  maxbound=iicheck+1;
            while myflag==0  % check to see if next step 'up' > minspan
               newmax=maxbound+step;  % upper step identifier
               newmin=minbound+step;  % lower step identifier
               if newmax<=length(sortxloc)
                  if(sortxloc(newmax)-sortxloc(newmin))/...
                                      max(sortxloc)>=minspan/100
                     newdist=(sortxloc(newmin)+sortxloc(newmax))/2;
                     myflag=4;
                  end
                  if myflag~=0
                     break
                  end
               else
                  maxendflag=1;     % exceeded upper limit of curve
               end
               newmax=maxbound-step;
               newmin=minbound-step;
               if newmin<0
                  minendflag=1;  %exceeded min limit of curve
               else
                  if newmin==0
                    if(sortxloc(newmax)-0)/max(sortxloc)>=minspan/100
                       newdist=(0+sortxloc(newmax))/2;
                       myflag=5;
                    end
                    if myflag~=0
                       break
                    end
                  else
                     if(sortxloc(newmax)-sortxloc(newmin))/...
                                     max(sortxloc)>=minspan/100
                        newdist=(sortxloc(newmin)+sortxloc(newmax))/2;
                        myflag=6;
                     end
                  end
               end
               if myflag~=0
                  break
               end
               if maxendflag==1
                  if minendflag==1
                     outofbounds=1;
                     break  %exceeded both max and min ends of curve
                  end
               end
               step=step+1;
            end
         end
         if myflag~=0
            break
         end
      end
   end   % end tempdist>sortxloc(iicheck)
else  %tempdist same as existing point
   if iicheck<length(sortxloc)
      newdist=(sortxloc(iicheck)+sortxloc(iicheck+1))/2;
      myflag=7;
   else
```

```
                        newdist=(sortxloc(iicheck-1)+sortxloc(iicheck))/2;
                        myflag=8;
                     end
                     if myflag~=0
                        break
                     end
                  end
               end
            else   % tempdist=0
               if sortxloc(1)/max(sortxloc)>=minspan/100
                  newdist=(0+sortxloc(1))/2;
                  myflag=9;
                  if myflag~=0
                     break
                  end
               else
                  myflag=0;
                  step=1;
                  minbound=0; maxbound=1;
                  while myflag==0
                     newmax=maxbound+step;
                     newmin=minbound+step;
                     if sortxloc(newmax)>max(sortxloc)
                        stop
                     end
                     if(sortxloc(newmax)-sortxloc(newmin))/max(sortxloc)>=minspan/100
                        newdist=(sortxloc(newmin)+sortxloc(newmax))/2;
                        myflag=9;
                     end
                     if myflag~=0
                        break
                     end
                     step=step+1;
                  end
               end
            end  % end of loop to add new point based on predicted location
            [F_xlinerows,F_xlinecols]=size(F_xline);
            counter=max(F_xline(:,1));
%              xloc(counter)=newdist;  % distance along 1D curve for added point
         end  %end add new point in nearby midspan location
      % === evaluate function at new point
      if outofbounds~=1
         X0new=X0+newdist*Sn;  % X0 resulting from newly added point
              %  note...don't put XOnew into minfcalc since minfcalc = 1 col & X0 = diff size
         [f]=Objective_Fun(X0new);   % actual FVAL at location of estimated minimum
         [g,heq,gpos]=Constraint_Fun(X0new,1);
         gmax=max(g);
         minfcalc(1)=polyval(linecoef,newdist);  % evaluate polynomial from new point
         minfcalc(2)=newdist;  % 1D location of new point
         minfcalc(3)=f;  % result of actual solve for location along Sn of predicted min
         Error1Dmin=abs((minfcalc(1)-minfcalc(3))/minfcalc(3));
         % ===== add evaluated potential minimum to F_xline 1-D search sub-database
         F_xline=[F_xline; counter+1 X0new minfcalc(2) minfcalc(3) rsqrd minfcalc(1) ...
                                                            Error1Dmin max(g)];
      end
       % ===== assess acceptability of curve fit =====
         % ==== Compute Polynomial Approximation & R^2 along Sn for 1-D Search ==========
          my_x=count_dv+2;   % column of F_xline with x-variable
          my_y=count_dv+3;   % column of F_xline with y-variable
          [linecoef,rsqrd]=myPolyFit(F_xline,my_x,my_y);%,temporder,n);
         % ======  use polynomial approximation to find 1-D minimum along S
         [minfcalc]=PolyMin(0,maxdist,linecoef); % find estimated min along 1-D search path
         [F_xlinerows,F_xlinecols]=size(F_xline);
         bestpt=0; bestval=1e12;  % flag for row of F_xline with minimum value
         for ii=1:F_xlinerows
```

```
            if F_xline(ii,count_dv+3)<bestval
                if F_xline(ii,F_xlinecols)<0  % only accept results that are in feasible space
                    bestval=F_xline(ii,count_dv+3);  % best feasible solved value in F_xline
                    bestpt=ii;                        % index of best solved value in F_xline
                    bestdist=F_xline(ii,count_dv+2); % 1D distance of best solved value in F_xline
         %          bestg=F_xline(ii, F_xlinecols); % max_g for this best solved value in F_xline
                    gmax=F_xline(ii, F_xlinecols); % max_g for this best solved value in F_xline
                end
            end
        end
        if bestpt~=0    % solved minimum found that is within feasible space
            if F_xline(bestpt,count_dv+5)==0
                minfcalc(1)=polyval(linecoef,F_xline(bestpt,count_dv+2)); %evaluate from polynomial
            else
                minfcalc(1)=F_xline(bestpt,count_dv+5);
            end
            minfcalc(2)=F_xline(bestpt,count_dv+2);
            minfcalc(3)=F_xline(bestpt,count_dv+3);
            ErrorMinPt=abs((minfcalc(1)-minfcalc(3))/minfcalc(3));
            if rsqrd > rsqrdallow
                if abs(ErrorMinPt)*100<ErrorAllow1D %
                    goodfit=1;  % R^2 value and error acceptable with new fit including new point
                end
            end
        end
  end
%    penalty=minfcalc(3)*10; % penalty = midspan 1D range
  linecoef;  % coefficients of 1D response curve along Sn
  rsqrd;     % R^2 value for 1D curve fit
```

Appendix D

Tabulated Results

Table D-1 Summary of results by start point

Start Point	Point Definition	Design Variable Definition										Function Value (penalized)	g_{max}	Stop Criteria
		L_1 (in)	L_2 (in)	K_1 (lb_f/in)	K_2 (lb_f/in)	K_3 (lb_f/in)	K_4 (lb_f/in)	C_1	C_2	C_3	C_4			
Steepest Feasible Descent Results														
	X0	18.000	48.000	20300	14643	9500	8115	0.0632	0.0571	0.0480	0.0391	1.8211	-9.0E-02	
1	Jump 1 end	19.151	52.205	20300	14643	9500	8115	0.0632	0.0571	0.0480	0.0390	1.4763	-1.6E-01	(2)
	Jump 2 end	19.191	52.390	20300	14643	9500	8115	0.0632	0.0571	0.0480	0.0390	1.4526	-1.2E-01	
2	X0	9.000	32.000	40100	28786	18500	15731	0.1215	0.1092	0.0911	0.0733	2.3119	6.9E-02	(3)
	Jump 1 end	9.000	32.000	40100	28786	18500	15731	0.1215	0.1092	0.0911	0.0733	0.0000	6.9E-02	
3	X0	22.500	26.667	4460	22724	45500	38577	0.2962	0.2655	0.2202	0.1757	2.2445	9.9E-02	(3)
	Jump 1 end	22.500	26.667	4460	22724	45500	38577	0.2962	0.2655	0.2202	0.1757	0.0000	9.9E-02	
	X0	31.500	58.667	44060	2520	14409	4601	0.4127	0.3697	0.3063	0.2440	2.4054	-1.2E-01	
4	Jump 1 end	29.245	50.478	44060	2520	14409	4601	0.4126	0.3347	0.3063	0.2403	1.4511	-9.5E-02	(1)
	Jump 2 end	29.245	50.478	44060	2520	14409	4601	0.4126	0.3347	0.3063	0.2403	1.4511	-1.2E-01	
	X0	2.250	42.667	16340	16663	23409	12216	0.4709	0.4218	0.3494	0.2781	2.0746	-2.2E-02	
5	Jump 1 end	2.425	43.499	16340	16663	23409	12216	0.4709	0.4219	0.3494	0.2779	1.5840	-3.7E-04	(2)
	Jump 2 end	3.217	43.499	16340	16663	23409	12216	0.4674	0.4182	0.3456	0.2748	1.5704	-3.7E-04	
	X0	38.250	66.667	26240	23735	27909	16024	0.0067	0.4479	0.3709	0.2952	45.4097	-7.4E-02	
6	Jump 1 end	38.321	64.507	26240	23735	27909	16024	0.0067	0.4480	0.3708	0.2954	14.2590	-4.4E-03	(2)
	Jump 2 end	38.429	64.500	26240	23735	27909	16024	0.0067	0.4480	0.3708	0.2957	14.0065	-2.1E-04	
	X0	11.250	53.333	8420	44949	41409	27447	0.0941	0.0324	0.4354	0.3464	1.5147	-1.0E-01	
7	Jump 1 end	11.338	53.791	8420	44949	41409	27447	0.0941	0.0324	0.4354	0.3464	1.4548	-1.0E-01	(1)
	Jump 2 end	11.338	53.791	8420	44949	41409	27447	0.0941	0.0324	0.4354	0.3464	1.4548	-7.9E-02	
	X0	29.250	37.333	28220	10602	1318	35062	0.1523	0.0845	0.4785	0.3805	1.6676	-3.2E-02	
8	Jump 1 end	29.324	37.574	28220	10602	1318	35062	0.1523	0.0846	0.4797	0.3808	1.6119	-3.4E-02	(2)
	Jump 2 end	29.394	37.788	28220	10602	1318	35062	0.1523	0.0846	0.4811	0.3808	1.5891	-3.6E-02	
	X0	24.750	69.333	10796	38888	19318	1086	0.2688	0.1887	0.0705	0.4488	5.1330	-3.7E-02	
9	Jump 1 end	24.310	70.500	10796	38888	19318	1086	0.2689	0.1888	0.0707	0.0546	4.4817	-7.2E-05	(2)
	Jump 2 end	23.999	70.500	10796	38888	19318	1086	0.2689	0.1888	0.0707	0.0050	4.4524	-7.2E-05	
	X0	55.125	64.889	4856	5551	10727	35355	0.0375	0.4232	0.2642	0.1080	9.6272	-9.9E-02	
10	Jump 1 end	61.497	69.498	4856	5551	10727	35355	0.0380	0.4059	0.2649	0.1080	3.6211	-1.9E-03	(2)
	Jump 2 end	61.497	70.494	4856	5551	10727	35355	0.0379	0.4024	0.2645	0.1081	3.5188	-1.9E-03	
	X0	28.125	51.556	34556	26765	24227	46778	0.1249	0.0077	0.3288	0.1592	1.6408	-6.4E-02	
11	Jump 1 end	27.877	50.431	34556	26765	24227	46778	0.1249	0.0077	0.3288	0.1592	1.4546	-6.4E-02	(1)
	Jump 2 end	27.877	50.431	34556	26765	24227	46778	0.1249	0.0077	0.3288	0.1592	1.4546	-6.4E-02	
	X0	14.625	67.556	46436	20704	2136	20417	0.2996	0.1641	0.4579	0.2616	4.5127	-6.2E-02	
12	Jump 1 end	12.765	64.501	46436	20704	2136	20417	0.2996	0.1641	0.3900	0.2616	2.8336	-8.8E-04	(2)
	Jump 2 end	1.503	64.501	46436	20704	2136	20417	0.2996	0.1639	0.1636	0.2587	2.0563	-8.8E-04	
	X0	48.375	70.222	30992	28930	42636	5479	0.0684	0.3985	0.1575	0.4152	5.9100	-2.5E-02	
13	Jump 1 end	48.448	70.496	30992	28930	42636	5479	0.0684	0.3985	0.1575	0.4149	5.4937	-2.8E-03	(2)
	Jump 2 end	61.496	70.496	30992	28930	42636	5479	0.0685	0.3985	0.1575	0.3472	3.4575	-2.4E-03	

Table D-1 - *Continued*

Start Point	Point Definition	Design Variable Definition										Function Value (penalized)	g_{max}	Stop Criteria
		L_1 (in)	L_2 (in)	K_1 (lb_f/in)	K_2 (lb_f/in)	K_3 (lb_f/in)	K_4 (lb_f/in)	C_1	C_2	C_3	C_4			
14	X0	34.875	65.778	25052	44083	34045	39749	0.3304	0.1394	0.3512	0.0745	39.5374	-8.6E-02	(2)
	Jump 1 end	35.810	64.502	25052	44083	34045	39749	0.3305	0.1394	0.3514	0.0745	24.2341	-1.3E-03	
	Jump 2 end	37.586	65.611	25052	44083	34045	39749	0.3307	0.1394	0.3512	0.0772	16.1061	-8.9E-02	
15	X0	46.688	53.630	11984	22003	13182	40627	0.4795	0.1682	0.2024	0.2457	2.3867	-4.1E-02	(2)
	Jump 1 end	46.500	52.226	11984	22003	13182	40627	0.4792	0.1681	0.2024	0.2457	1.9251	-1.9E-04	
	Jump 2 end	46.500	48.001	11984	22003	13182	40627	0.4791	0.1682	0.2024	0.2457	1.6110	-1.9E-04	
16	X0	60.188	69.630	23864	15942	40182	14266	0.1609	0.3245	0.3316	0.3482	3.7861	-3.3E-02	(2)
	Jump 1 end	60.661	70.494	23864	15942	40182	14266	0.1609	0.3245	0.3316	0.3479	3.6010	-4.3E-03	
	Jump 2 end	61.497	70.494	23864	15942	40182	14266	0.1608	0.3242	0.3316	0.3477	3.5543	-1.9E-03	
17	X0	53.438	67.852	579	46248	18537	33598	0.1917	0.2998	0.2249	0.1610	6.6213	-5.8E-02	(2)
	Jump 1 end	55.080	70.351	579	46248	18537	33598	0.0050	0.2998	0.2251	0.1610	4.2305	-7.1E-03	
	Jump 2 end	55.103	70.414	579	46248	18537	33598	0.0050	0.2998	0.2251	0.1610	4.2004	-1.5E-05	
18	X0	47.531	60.741	48099	15076	28355	26568	0.3972	0.4314	0.2474	0.0762	13.5771	-3.8E-02	(2)
	Jump 1 end	47.441	61.497	48099	15076	28355	26568	0.3972	0.4319	0.2474	0.0763	10.6653	-2.2E-03	
	Jump 2 end	46.501	61.497	48099	15076	28355	26568	0.3973	0.4332	0.2474	0.0763	10.3250	-6.0E-04	
19	X0	52.594	58.370	19587	32394	8719	1694	0.0273	0.3327	0.3147	0.3164	8.4744	-1.6E-01	(2)
	Jump 1 end	53.761	55.262	19587	32394	8719	1694	0.0280	0.3330	0.3149	0.2536	1.9539	-8.0E-04	
	Jump 2 end	53.761	55.262	19587	32394	8719	1694	0.0337	0.3330	0.3141	0.0060	1.9537	-8.0E-04	
20	X0	53.016	55.210	27190	18539	43529	22783	0.3271	0.3656	0.4046	0.4718	2.4888	-5.7E-02	(1)
	Jump 1 end	53.288	54.800	27190	18539	43529	22783	0.3271	0.3656	0.4046	0.4718	1.8908	-7.7E-03	
	Jump 2 end	53.288	54.800	27190	18539	43529	22783	0.3125	0.0124	0.4101	0.4597	1.8908	-7.7E-03	
21	X0	64.617	68.181	5426	29549	4033	48062	0.0122	0.1244	0.3438	0.1263	3.4690	-3.9E-02	(2)
	Jump 1 end	66.311	67.815	5426	29549	4033	48062	0.0121	0.1244	0.3349	0.1263	3.1904	-3.0E-03	
	Jump 2 end	66.311	67.815	5426	29549	4033	48062	0.0121	0.1234	0.0128	0.1258	3.1903	-3.0E-03	
22	X0	66.410	69.761	41541	7778	9388	2100	0.2588	0.4206	0.4570	0.1121	3.0474	-3.1E-02	(2)
	Jump 1 end	67.564	69.065	41541	7778	9388	2100	0.2587	0.4163	0.4557	0.0080	2.9586	-3.1E-04	
	Jump 2 end	67.564	69.065	41541	7778	9388	2099	0.2699	0.0055	0.3551	0.0080	2.9585	-3.1E-04	
23	X0	65.988	68.971	27856	21632	15231	8589	0.2298	0.4934	0.2325	0.0839	3.3869	-1.3E-02	(1)
	Jump 1 end	67.320	68.827	27856	21632	15231	8589	0.2298	0.4931	0.2325	0.0830	3.2970	-4.3E-03	
	Jump 2 end	67.320	68.827	27856	21632	15231	8589	0.2298	0.4931	0.2325	0.0830	3.2970	-4.3E-03	
24	X0	65.145	70.288	14550	48351	20586	15618	0.4474	0.2699	0.3672	0.0698	3.4558	-2.4E-02	(2)
	Jump 1 end	65.265	70.497	14550	48351	20586	15618	0.4474	0.2699	0.3673	0.0697	3.4457	-2.1E-03	
	Jump 2 end	68.995	70.497	14550	48351	20586	15618	0.4472	0.2697	0.3692	0.0686	3.3988	-1.4E-03	
Direct Search (SQP) Results														
Start Point		29.245	50.478	44060	2520	14409	4601	0.4126	0.3347	0.3063	0.2403	1.4511	(1) Constraints Satisfied	
Iteration 0		29.245	50.478	44060	2520	14409	4601	0.4126	0.3347	0.3063	0.2413	1.4511		
Iteration 1 (end point)		29.245	50.417	44060	2520	14409	4601	0.4126	0.3344	0.3063	0.2403	1.4511		
Stopping Criteria: (1) Convergence achieved, (2) Convergence not achieved, max jumps completed, (3) No feasible results found														

Table D-2 Sn Vectors by starting point and jump

Start Point	Jump	Sn Vector Component									
		L_1	L_2	K_1	K_2	K_3	K_4	C_1	C_2	C_3	C_4
1	1	2.64E-01	9.65E-01	0.00E+00	2.29E-05	0.00E+00	0.00E+00	-9.18E-07	-2.75E-06	-3.90E-06	-2.25E-05
	2	2.11E-01	9.77E-01	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.05E-05	0.00E+00	-1.05E-05
2	1	Sn=[0]									
3	1	Sn=[0]									
4	1	-2.66E-01	-9.64E-01	0.00E+00	-1.06E-04	0.00E+00	-3.53E-05	-1.18E-06	-4.12E-03	-4.71E-06	-4.33E-04
	2	Sn=[0]									
5	1	2.06E-01	9.79E-01	0.00E+00	0.00E+00	0.00E+00	0.00E+00	4.70E-05	1.18E-05	0.00E+00	-2.59E-04
	2	1.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	-4.51E-03	-4.68E-03	-4.76E-03	-3.85E-03
6	1	3.28E-02	-9.99E-01	0.00E+00	0.00E+00	0.00E+00	4.63E-05	1.90E-06	2.78E-05	-1.39E-05	1.06E-04
	2	9.98E-01	-5.92E-02	0.00E+00	0.00E+00	0.00E+00	0.00E+00	6.47E-06	9.24E-05	0.00E+00	2.59E-03
7	1	1.88E-01	9.82E-01	0.00E+00	0.00E+00	0.00E+00	0.00E+00	4.08E-05	0.00E+00	0.00E+00	-2.15E-05
	2	Sn=[0]									
8	1	2.93E-01	9.56E-01	0.00E+00	0.00E+00	0.00E+00	0.00E+00	3.98E-05	2.07E-04	4.97E-03	9.15E-04
	2	3.10E-01	9.51E-01	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	2.53E-04	5.95E-03	8.88E-05
9	1	-3.37E-01	8.92E-01	0.00E+00	0.00E+00	0.00E+00	-3.82E-03	7.65E-05	3.06E-05	1.29E-04	-3.01E-01
	2	-9.87E-01	0.00E+00	0.00E+00	0.00E+00	0.00E+00	-1.53E-02	-3.18E-05	3.18E-05	1.72E-04	-1.58E-01
10	1	8.10E-01	5.86E-01	-6.36E-05	-1.40E-04	-1.27E-05	0.00E+00	6.31E-05	-2.20E-03	8.90E-05	5.09E-06
	2	0.00E+00	1.00E+00	0.00E+00	-2.01E-04	0.00E+00	0.00E+00	-1.66E-04	-3.52E-03	-4.22E-04	1.00E-05
11	1	-2.15E-01	-9.77E-01	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.74E-07	0.00E+00	8.68E-06
	2	Sn=[0]									
12	1	-5.20E-01	-8.54E-01	0.00E+00	0.00E+00	-3.08E-04	0.00E+00	0.00E+00	2.80E-06	-1.90E-02	-1.12E-05
	2	-1.00E+00	0.00E+00	0.00E+00	-8.88E-06	-3.91E-04	-1.78E-05	8.88E-07	-1.86E-05	-2.01E-02	-2.55E-04
13	1	2.57E-01	9.66E-01	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	-1.27E-03
	2	1.00E+00	0.00E+00	0.00E+00	-7.66E-06	0.00E+00	-3.07E-04	6.05E-06	-3.83E-06	-2.30E-06	-5.19E-03
14	1	5.91E-01	-8.07E-01	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.90E-05	6.32E-06	1.01E-04	1.26E-05
	2	8.48E-01	5.29E-01	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.19E-04	0.00E+00	-8.59E-05	1.28E-03
15	1	-1.32E-01	-9.91E-01	0.00E+00	0.00E+00	0.00E+00	0.00E+00	-1.98E-04	-2.12E-05	-7.06E-06	0.00E+00
	2	0.00E+00	-1.00E+00	-2.37E-05	0.00E+00	0.00E+00	0.00E+00	-2.13E-05	4.73E-06	-7.10E-06	-4.73E-06
16	1	4.81E-01	8.77E-01	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.02E-05	-4.06E-05	0.00E+00	-2.54E-04
	2	1.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	-1.20E-04	-1.32E-04	-3.35E-04	2.39E-05	-2.51E-04
17	1	5.48E-01	8.34E-01	-8.68E-04	0.00E+00	0.00E+00	0.00E+00	-6.23E-02	3.34E-06	5.01E-05	1.00E-05
	2	3.39E-01	9.41E-01	0.00E+00	0.00E+00	0.00E+00	0.00E+00	-5.33E-04	0.00E+00	1.51E-04	0.00E+00

Table D-2- *Continued*

Start Point	Jump	Sn Vector Component									
		L_1	L_2	K_1	K_2	K_3	K_4	C_1	C_2	C_3	C_4
18	1	-1.18E-01	9.93E-01	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.31E-05	5.78E-04	1.31E-05	4.47E-05
	2	-1.00E+00	0.00E+00	0.00E+00	2.13E-04	0.00E+00	0.00E+00	1.06E-05	1.42E-03	4.25E-05	4.25E-05
19	1	3.51E-01	-9.36E-01	-3.01E-05	0.00E+00	0.00E+00	-6.62E-04	2.13E-04	9.94E-05	4.22E-05	-1.89E-02
	2	0.00E+00	0.00E+00	-9.28E-03	-4.04E-03	-4.04E-04	-2.70E-02	2.32E-02	-8.07E-05	-2.99E-03	-9.99E-01
20	1	5.53E-01	-8.33E-01	0.00E+00	0.00E+00	0.00E+00	0.00E+00	-2.03E-05	-8.12E-05	0.00E+00	0.00E+00
	2	0.00E+00	0.00E+00	-1.75E-01	-4.35E-01	-4.99E-04	-5.74E-03	-3.64E-02	-8.82E-01	1.39E-02	-3.01E-02
21	1	9.77E-01	-2.11E-01	-1.15E-04	0.00E+00	-2.31E-04	0.00E+00	-1.50E-05	-1.15E-05	-5.14E-03	0.00E+00
	2	0.00E+00	0.00E+00	-6.04E-02	-5.26E-03	-4.52E-02	-9.29E-04	-1.55E-04	-3.13E-03	-9.97E-01	-1.39E-03
22	1	8.54E-01	-5.15E-01	0.00E+00	-2.22E-04	-7.40E-05	-5.18E-03	-5.18E-05	-3.19E-03	-1.01E-03	-7.70E-02
	2	0.00E+00	0.00E+00	-6.40E-04	-6.74E-02	-1.41E-02	-7.33E-01	1.78E-02	-6.57E-01	-1.61E-01	0.00E+00
23	1	9.94E-01	-1.08E-01	0.00E+00	-7.46E-05	0.00E+00	-2.24E-04	-7.46E-06	-2.31E-04	2.99E-05	-6.79E-04
	2	Sn=[0]									
24	1	4.99E-01	8.67E-01	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	4.15E-04	-1.12E-04
	2	1.00E+00	0.00E+00	0.00E+00	0.00E+00	-5.36E-05	-2.14E-04	-6.70E-05	-5.36E-05	4.93E-04	-3.16E-04

Table D-3 Line search results by starting point and jump

Start Point	Jump	Location of minimum along 1D search vector	Maximum length of 1D search vector	Polynomial correlation (R^2)	Polynomial error at identified minimum (%)	Function evaluations per jump
1	1	4.359	13.989	0.99965	0.1349	15
	2	0.190	9.510	0.99984	-0.1377	25
2	1	0.000	0.000	0.00000	0.0000	15
3	1	0.000	0.000	0.00000	0.0000	15
4	1	8.493	12.614	1.00000	0.0000	25
	2	0.000	4.061	1.00000	0.0162	16
5	1	0.851	0.851	0.99997	0.0022	15
	2	0.791	39.572	0.99996	-0.0821	25
6	1	2.161	2.161	1.00000	0.0000	25
	2	0.108	0.108	1.00000	0.0000	15
7	1	0.466	8.313	0.99631	-0.3897	25
	2	0.000	7.382	1.00000	0.0000	25
8	1	0.251	4.306	0.99999	-0.9621	16
	2	0.225	3.414	1.00000	-0.2857	16
9	1	1.308	1.308	1.00000	-0.0001	15
	2	0.315	0.315	1.00000	0.0000	15
10	1	7.865	7.865	0.99988	0.0257	15
	2	0.996	0.996	1.00000	0.0000	15
11	1	1.152	5.176	0.99816	-0.3671	25
	2	0.000	4.312	0.99998	0.0240	25
12	1	3.577	3.577	1.00000	-0.0005	15
	2	11.264	11.264	1.00000	0.0001	15
13	1	0.283	0.283	1.00000	-0.0001	15
	2	13.049	13.049	1.00000	0.0001	15
14	1	1.582	1.582	0.99984	0.0113	15
	2	2.095	9.065	0.99980	-0.1138	25
15	1	1.416	1.416	1.00000	0.0001	15
	2	4.225	4.225	$-\infty$	0.0000	25
16	1	0.985	0.985	1.00000	0.0000	15
	2	0.836	0.836	1.00000	0.0000	15
17	1	2.997	2.997	1.00000	0.0004	15
	2	0.066	0.066	1.00000	0.0000	15
18	1	0.761	0.761	1.00000	0.0001	15
	2	0.940	0.940	0.99987	-0.0005	15
19	1	3.321	3.321	1.00000	0.0075	15
	2	0.248	0.248	1.00000	0.0000	15
20	1	0.493	0.493	1.00000	0.0000	15
	2	0.401	0.404	1.00000	0.0000	15
21	1	1.733	1.733	1.00000	-0.0001	15
	2	0.323	0.323	1.00000	0.0000	20

Table D-3- *Continued*

Start Point	Jump	Location of minimum along 1D search vector	Maximum length of 1D search vector	Polynomial correlation (R^2)	Polynomial error at identified minimum (%)	Function evaluations per jump
22	1	1.352	1.352	1.00000	0.0000	15
	2	0.625	0.625	1.00000	0.0000	15
23	1	1.340	1.340	1.00000	0.0000	15
	2	0.000	2.818	1.00000	-0.0056	15
24	1	0.241	0.241	0.99986	0.0000	15
	2	3.730	3.730	1.00000	0.0000	15

References

[1] R. Norton, *Machine Design, An Integrated Approach*, 4th Edition, Prentice Hall, 2011

[2] R. Craig, A. Kurdila, *Fundamentals of Structural Dynamics*, 2nd Edition, John Wiley & Sons, Inc., 2006

[3] J. Shigley, L. Mitchell, *Mechanical Engineering Design*, 4th Edition, McGraw-Hill, 1983

[4] J.P. Den Hartog, *Mechanical Vibrations*,4th Edition, Dover Publication, 1985

[5] A. Piersol, T. Paez, *Harris' Shock and Vibration Handbook*, 6th Edition, McGraw-Hill, 2010

[6] K. Billah, R. Scanlan, Resonance, Tacoma Narrows bridge failure, and undergraduate physics textbooks, *American Journal of Physics* 59 (1991) 118-124

[7] B. Akesson, N. Olhoff, Minimum stiffness of optimally located supports for maximum value of beam eigenfrequencies, *Journal of Sound and Vibration* 120 (1988) 457-463

[8] R. Courant, D. Hilbert, *Methods of Mathematical Physics*, vol. 1, Interscience Publishers, 1953

[9] D. Wang, J. Jiang, W. Zhang, Optimization of support positions to maximize the fundamental frequency of structures, *International Journal for Numerical Methods in Engineering* 61, (2004) 1584-1602

[10] D. Wang, Optimization of support positions to minimize the maximal deflection of structures, *International Journal of Solids and Structures* 41, (2004) 7445-7458

[11] D. Wang, Optimal design of structural support positions for minimizing maximal bending moment, *Finite Elements in Analysis and Design* 43, (2006) 95-102

[12] D. Wang, M. Friswell, Y. Lei, Maximizing the natural frequency of a beam with an intermediate elastic support, *Journal of Sound and Vibration* 291, (2006) 1229-1238

[13] J. Zhu, W. Zhang, Maximization of structural natural frequency with optimal support layout, *Structural and Multidisciplinary Optimization* 31, (2006) 462-469

[14] H. Frahm (1911) *U.S. Patent No. 989,958*, Washington DC: US Patent Office

[15] H. Ozguven, B. Candir, Suppressing the first and second resonances of beams by dynamic vibration absorbers, *Journal of Sound and Vibration* 111, (1986) 377-390

[16] C. Yang, D. Li, L. Cheng, Dynamic vibration absorbers for vibration control within a frequency band, *Journal of Sound and Vibration* 330 (2011) 1582-1598

[17] D. Russell, Bending modes, Damping and the sensation of sting in baseball bats, *Engineering of Sport 6*, 6th International Sports Engineering Conference (2006) ISEA2006

[18] K. Hao, L. Mei, Z. Ripin, Tuned vibration absorber for suppression of hand-arm vibration in electric grass trimmer, *International Journal of Industrial Ergonomics* 41 (2011) 494-508

[19] H. Pierson, Tunable dynamic support for resonance avoidance in bar feeders, *Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Graduate School of the Ohio State University*, Industrial and Systems Engineering, 2012

[20] G. Vanderplaats, *Multidiscipline Design Optimization*, 1st Edition, Vanderplaats Research and Development, Inc., 2007

[21] J. Fliege, B. Svaiter, Steepest Descent Methods for Multicriteria Optimization, *Mathematical Methods of Operations Research*, Vol. 51, (2000), No. 3, 479-494

[22] J. Nocedal, S. Wright, *Numerical Optimization*, 2nd Edition, Springer Science & Business Media, 2006

[23] Fletcher, R. and C.M. Reeves, Function Minimization by Conjugate Gradients, *Br. Computer J.*, vol. 7, no. 2, (1964) 149-154

[24] Huang, H.Y., Unified Approach to Quadratically Convergent Algorithms for Function Minimization, *J. Optim. Theory Appl.*, vol. 5 (1970) 405-423

[25] T. Kolda, R. Lewis, V. Torczon, Optimization by direct search: New perspectives on some classical and modern methods, *Society for Industrial and Applied Mathematics* Vol. 45, No. 3 (2003) 385-482

[26] Hajela, P., Genetic Search – An Approach to the Nonconvex Optimization Problem, *AIAA Journal*, Vol. 26, No 7 (1990) 1205-1210

[27] B. Wang, J. Chen, Application of genetic algorithm for the support location optimization of beams, *Computers and Structures* Vol. 58, No. 4 (1996) 797-800

[28] Venter, G. and Sobieszczanski-Sobieski, J., Particle Swarm Optimization, *AIAA Journal*, Vol. 41, No. 8 (2003) 1583-1589

[29] A. Banks, J. Vincent, C. Anyakoha, A review of particle swarm optimization. Part I: background and development, *Natural Computing* Vol. 6, Issue 4 (2007) 467-484

[30] J. Vesterstrøm, R. Thomsen, A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems, *Evolutionary Computation CEC2004*, Vol. 2 (2004) 1980-1987

[31] E. Laskari, K. Parsopoulos, M. Vrahatis, Particle swarm optimization for minimax problems, *Evolutionary Computing CEC2002*, Vol. 2 (2002) 1576-1581

[32] Powell, M.J.D., *A Fast Algorithm for Nonlinearly Constrained Optimization Calculations*, no. DAMPTP77/NA 2, University of Cambridge, England, 1977

[33] MATLAB Help Documentation, www.mathworks.com/help/optim/ug/constrained-nonlinear-optimization-algorithms.html, *MATLAB R2014a documentation* (2014)

[34] J. Knowles, ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems, *IEEE Transactions on Evolutionary Computation*, Vol. 10, No. 1 (2005) 50-66

[35] D. Jones, M. Schonlau, W. Welch, Efficient global optimization of expensive black-box functions, *Journal of Global Optimization 13* (1998) 455-492

[36] A. Booker, J. Dennis Jr., P. Frank, D. Serafini, V. Torczon, M. Trosset, A rigorous framework for optimization of expensive functions by surrogates, *NASA/CR-1998-208735, ICASE Report No. 98-47* (1998)

[37] L. Wang, S. Shan, G. Wang, Mode-pursuing sampling method for global optimization on expensive black-box functions, *Engineering Optimization*, Vol. 36,No. 4, (2004) 419-438

[38] R. Blevins, *Formulas for Natural Frequency and Mode Shape*, Van Nostrand Reinhold Company, 1979

[39] J.P. Den Hartog, *Strength of Materials*, Dover Edition, Dover Publication, 1977

[40] M. Molga, C. Smutnicki, *Test Functions for Optimization Needs*, 2005

[41] T. Chen, T. Chi, On the improvements of the particle swarm optimization algorithm, *Advances in Engineering Software* 41 (2010) 229-239

[42] *JMP® 10 Design of Experiments Guide*. SAS Institute Inc., Cary, NC (2012)

[43] Y. Kay, E. Zahara, A hybrid genetic algorithm and particle swarm optimization for multimodal functions, Applied Soft Computing 8 (2008) 849-857

[44] ANSYS ® Help Documentation v 14.5.7, SASIP, Inc. (2012)

Biographical Information

Mr. Hauser earned his B.S. in Mechanical Engineering at Texas A&M University in 1983. Since that time, he has worked in industry with focus on the design of complex machinery including gas compressors, engines and medical devices. Key areas of interest relate to optimization of the design process and the development of methods to reduce both time and cost required to bring products to market. His part-time return to academia to pursue a M.S. in Mechanical Engineering is intended to add additional ‘tools’ to aid with that effort.