Hindawi
Mathematical Problems in Engineering
Volume 2018, Article ID 1058732, 14 pages
https://doi.org/10.1155/2018/1058732

Research Article

Optimizing the Geometry of Flexure System Topologies Using the Boundary Learning Optimization Tool

Ali Hatamizadeh, Yuanping Song, and Jonathan B. Hopkins

Mechanical and Aerospace Engineering Department, University of California, Los Angeles, 420 Westwood Plaza, Eng. IV 46-147F, Los Angeles, CA, USA

Correspondence should be addressed to Jonathan B. Hopkins; hopkins@seas.ucla.edu

Received 17 November 2017; Revised 7 February 2018; Accepted 27 February 2018; Published 29 March 2018

Academic Editor: Delfim Soares Jr.

Copyright © 2018 Ali Hatamizadeh et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

We introduce a new computational tool called the Boundary Learning Optimization Tool (BLOT) that identifies the boundaries of the performance capabilities achieved by general flexure system topologies if their geometric parameters are allowed to vary from their smallest allowable feature sizes to their largest geometrically compatible feature sizes for given constituent materials. The boundaries generated by the BLOT fully define the design spaces of flexure systems and allow designers to visually identify which geometric versions of their synthesized topologies best achieve desired combinations of performance capabilities. The BLOT was created as a complementary tool to the freedom and constraint topologies (FACT) synthesis approach in that the BLOT is intended to optimize the geometry of the flexure topologies synthesized using the FACT approach. The BLOT trains artificial neural networks to create models of parameterized flexure topologies using numerically generated performance solutions from different design instantiations of those topologies. These models are then used by an optimization algorithm to plot the desired topology's performance boundary. The model-training and boundary-plotting processes iterate using additional numerically generated solutions from each updated boundary generated until the final boundary is guaranteed to be accurate within any average error set by the user. A FACT-synthesized flexure topology is optimized using the BLOT as a simple case study.

1. Introduction

The freedom and constraint topologies (FACT) synthesis approach [1–3] was created to help designers rapidly consider and compare the flexure topology solutions that achieve a desired set of degrees of freedom (DOFs). FACT utilizes a complete library of spaces, which guide designers in arranging the best number and kind of flexible elements (e.g., wire or blade elements) within the geometry of the spaces for synthesizing topologies that stiffly constrain the system's bodies to not move in certain directions while permitting them to move in other directions with high compliance (i.e., the DOFs). The mathematics underlying FACT are such that only the number, kind, location, and orientation of the flexible elements that constitute the resulting topologies are considered during the synthesis process. The material properties and geometric parameters (e.g., the lengths, widths, or thicknesses of the flexible elements or the shape and size of the rigid bodies that the flexible elements join together within the system) are not considered. Thus, once the FACT approach has synthesized topologies that achieve desired DOFs, there is a need for a complementary follow-on tool that can optimize the geometric parameters of the topology solutions so final designs can be produced that best achieve desired performance requirements in addition to achieving the desired DOFs for given constituent materials.

The aim of this paper is to introduce such a tool for enabling the geometry optimization of FACT-synthesized topologies. This fully automated tool, called the Boundary Learning Optimization Tool (BLOT), utilizes the results generated from a limited number of numerical simulations of a FACT-synthesized topology using different geometric parameter values to construct a preliminary mathematical model of the topology by training an artificial neural network [4]. This model is then used in conjunction with a modified multiobjective optimization algorithm to identify a first-pass boundary that circumscribes the combination of desired capabilities that can be achieved by the geometric

instantiations of the topology. New numerical simulations of the various design instantiations along the initially identified boundary are then used in conjunction with the original simulation data to retrain a more accurate neural network model. This model is then used again to identify an improved boundary and the entire process is iterated until the topology's actual performance boundary is rigorously obtained with an accuracy specified by the user (e.g., less than 10% average error). Optimal design instantiations can then be selected from portions of the final boundary.

Although the BLOT was originally created to plot the performance capability regions of FACT-designed architectured materials [5] on Ashby plots [6] to compare their engineered properties with the properties achieved by natural materials, a simple FACT-designed case study is optimized in this paper to most clearly demonstrate how the BLOT works with the FACT approach to generate general flexure system designs from start to finish. Suppose, for instance, a designer wished first to synthesize a flexure system topology that achieves three specific DOFs: two orthogonal translations shown by the black arrows in Figure 1(a) and one orthogonal rotation shown by the red line with a circular arrow about its axis. Upon inspection of FACT's complete library of spaces, the designer would identify that this desired set of DOFs lies within the freedom space [1–3] shown in Figure 1(b). This space consists of an infinite number of rotation lines that are parallel to the desired rotation line, shown as a box of red lines in the figure, and an infinite number of translation arrows that point in the directions that are perpendicular to the axes of these lines, shown as the disk of black arrows in the same figure. This freedom space represents all the permissible motions that would result from linearly combining the three DOFs shown in Figure 1(a). According to the FACT library, this freedom space uniquely links to a complementary constraint space [1–3], shown as the box of blue lines in Figure 1(b) that consists of an infinite number of constraint lines [1–3] that are parallel to the desired red rotation line. This constraint space represents the region of space within which flexible elements should be selected to achieve the desired DOFs using a parallel topology configuration (i.e., a configuration that consists of a single rigid stage joined directly to a fixed ground by flexible elements). Instructions about how to select the appropriate number of flexible element constraints from within the constraint space so that the resulting topology is exactly constrained or overconstrained are provided in previous publications [1–3]. Suppose for this example that the designer synthesized a symmetric overconstrained topology by selecting four wire elements within the constraint space such that the axes of the wires are collinear with four of the space's parallel constraint lines as shown in Figure 1(c). At this point in the design process the designer has completed the FACT approach by synthesizing a topology that successfully achieves the three desired DOFs regardless of the design's constituent material properties or its geometry (e.g., how thick or long its wire elements are). A modal analysis of one of the topology's design instantiations shows that the most compliant directions of motion (i.e., the mode shapes corresponding to the lowest natural frequency values) are the desired DOFs from Figure 1(a). One of the first two identical translational mode shapes is shown in Figure 1(d), and the third rotational mode shape is shown in Figure 1(e).

Suppose that the designer wished then to identify what geometric instantiation of the FACT-synthesized topology in Figure 1(c) simultaneously achieves the largest range of motion and the highest natural frequency along the translational DOF shown in Figure 1(d) for a given constituent material. The more a flexure system can deform in a particular direction before it yields (i.e., its range of motion) with respect to its overall size and the higher the natural frequency is that corresponds with the mode shape along that direction, the easier it is to control the system at high speeds with precision over large ranges. To identify a geometric instantiation that achieves this combination of capabilities, the designer should label the topology's features with independent geometric parameters that fully define its geometry. Suppose for this example that the designer set the rigid ground and rigid stage geometry to be the same rectangular prism constrained by the parameter W as shown in Figure 1(f). Then suppose that the four wire elements that join the corners of this rigid stage are each set to possess the same length, L, and the same square cross-section with a side length of T (i.e., the width and thickness of each wire element is T). The designer would then provide the BLOT with the smallest achievable feature sizes for each of these three independent parameters as well as the largest geometrically compatible values for the same parameters. Properties of the constituent materials should also be provided as well as the tolerance capabilities of the available fabrication process to identify the smallest resolution increment by which each geometric parameter could be changed to sweep between these smallest and largest values. Using this information, the BLOT would then plot the boundary that circumscribes the combination of desired capabilities (i.e., the ranges of motion, d, per system characteristic length, $\sqrt{L^2 + W^2}$, and the natural frequencies, ω_n) that can be achieved along the direction of the translational DOF by the topology's geometric instantiations for the given range of parameters. If the smallest parameter values are set to $T_{\text{min}} = 1\,\text{mm}$, $W_{\text{min}} = 10\,\text{mm}$, and $L_{\text{min}} = 1\,\text{mm}$ and the largest parameter values are set to $T_{\text{max}} = 50\,\text{mm}$, $W_{\text{max}} = 100\,\text{mm}$, and $L_{\text{max}} = 100\,\text{mm}$, the boundary resulting from BLOT is shown in Figure 1(g) for the example topology of Figure 1(c) using a resolution increment value of 0.5 mm and using aluminum as the constituent material with a Young's modulus of 69 GPa, a shear modulus of 26 GPa, a density of 2,705 kg/m^3, and a yield strength of 105 MPa. Once this boundary is known, the designer can select optimal design instantiations that simultaneously achieve the largest values of each of the desired capabilities from along the top-right portion of the boundary as shown in Figure 1(g). Two extreme designs are shown as large black dots that lie on either end of the optimal portion of the red boundary plotted.

Note that although the BLOT introduced in this paper utilizes numerically generated data taken from a variety of corresponding design instantiations to train neural networks for creating an accurate model of FACT-synthesized topologies, the BLOT is also capable of using closed-form

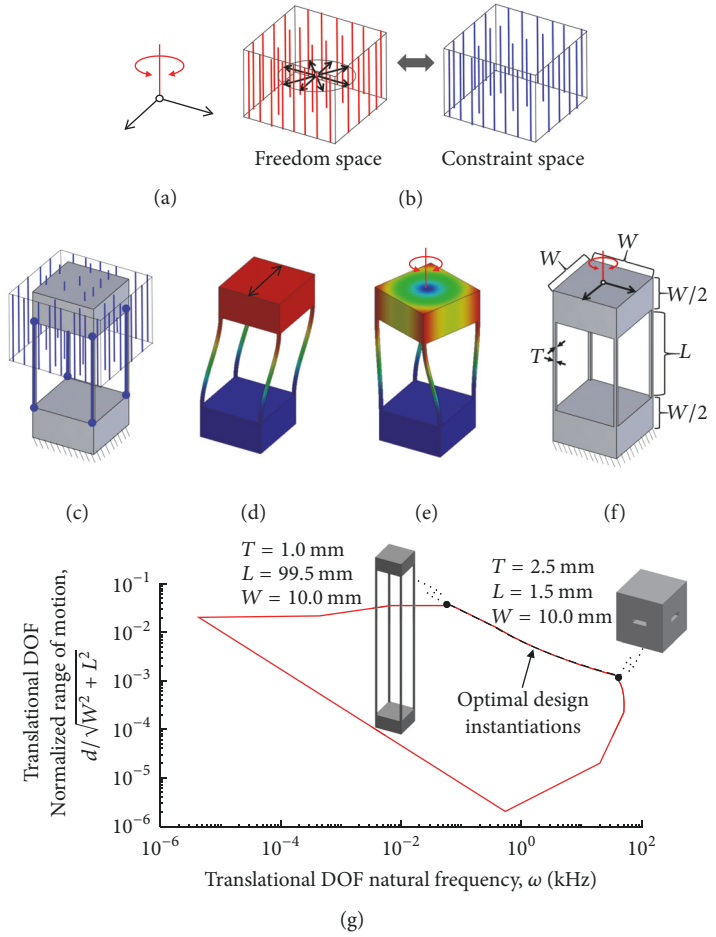


Figure 1: Desired DOFs (a); freedom and constraint spaces (b); synthesized topology (c); validated DOFs (d) and (e); topology's geometric parameters (f); boundary of the topology's achievable ranges of motion and natural frequencies corresponding to the translational DOF generated by the BLOT (g).

analytical models of the parameterized topologies if such models are available. Closed-form analytical models are, however, difficult to construct and their assumptions usually only produce valid results for specific topologies or for limited ranges of geometric parameters that define other topologies. Thus, this paper focuses on the theory necessary to train neural networks for creating models based on the results of numerically generated data because this approach is more general than closed-form analytical approaches. Note also that although the BLOT is introduced in this paper as a tool for optimizing the geometry of flexure system topologies, it could also be applied to a host of other diverse applications.

Prior to this paper, artificial neural networks have been used extensively to model and control a variety of mechanical systems including compliant mechanisms, robots, manipulators, and rigid linkages. Cheng and Patel [7] used a neural network-based strategy for achieving stable tracking control of a flexible macro-micro manipulator. He et al. [8] studied the tracking control of an uncertain n-link robot with full-state constraints. Kobayashi and Ozawa [9] presented an adaptive neural network control for tendon-driven robotic mechanisms with elastic tendons. Takeuchi and Kosugi [10] proposed a finite element method based on applications of neural networks for boundary value problems. Levin and Lieven [11] presented a neural network-based approach for dynamic finite element model updating. Boillat et al. [12] combined finite element and neural networks in order to find the optimal parameters used in a selective laser sintering process. Lefik [13] studied composite materials using a hybrid finite element and artificial neural network model. Hashash et al. [14] proposed the numerical implementation of a neural network-based material model in finite element analysis. Manevitz et al. [15] presented a finite element mesh-generation approach using self-organizing neural networks. Li and Cao [16] utilized neural networks to select two dimensional compliant mechanism design schemes. Qin and Feng [17] modified the structural parameters of flexure hinges by employing trained neural networks. Kong et al. [18] presented a method for identifying mechanism kinematic chain isomorphism by applying neural networks.

Although few researchers have directly attempted system-performance boundary identification, its goal is similar to the goal of multiobjective optimization, which has been

studied extensively. A multiobjective optimization problem (MOOP) deals with more than one objective function and aims at finding a set of solutions that optimizes all the objective functions simultaneously. Several methods have been proposed to solve the local or global Pareto-optimal solution set [19]. One of the most widely used methods is the weighting method [19]. Haimes et al. [20] introduced the ε-Constraint method in 1971 and Yu [21] introduced the method of global criterion in 1973. Wierzbicki [22, 23] introduced the achievement scalarizing function approach in 1981. Other solving methods include normal boundary intersection [24], evolutionary algorithm [25], lexicographic ordering [26], and goal programming [27]. More recently, MOOP methods have focused on stochastic algorithms, including a variety of evolutionary algorithms [28, 29]. Such algorithms generate more reliable global Pareto-optimal solution sets but require significantly more function evaluations than deterministic algorithms and are thus generally better suited for complex black-box-model optimizations. The boundary identification approach proposed in this paper has in part been adapted from various deterministic multiobjective optimization methods such that the complete continuous boundary (including concave portions) that circumscribe the performance capabilities achieved by general flexure topologies can be identified and refined with a desired accuracy. Therefore, a topology's full design space can be identified using the BLOT.

By combining the utility of the BLOT with the current FACT approach, a new advantageous approach emerges, which is unique from other existing design-optimization approaches. Whereas other approaches (e.g., topology optimization [30–32] or module optimization [33]) simultaneously combine the tasks of optimizing a design's topology with its geometry, the proposed approach decouples those tasks in such a way that the time-consuming computations are reserved solely for the simpler task of geometry optimization only. This optimization occurs after the FACT approach has directly generated and finalized the most promising topologies without performing expensive iterative calculations. Thus, by decoupling the tasks of topology synthesis and geometry optimization in this way, the speed that optimal designs can be generated from start to finish as well as the likelihood that the global-optimum solutions are identified increases.

The specific contributions of this paper include the following: (i) a new fully automated tool is created and demonstrated (i.e., BLOT) that identifies the rigorous performance capability boundaries achieved by flexure topologies generated via FACT; (ii) a new algorithm is proposed within this tool for optimizing and iteratively training the architectures of artificial neural networks using data generated from specially selected numerical simulations of design instantiations along previously generated boundaries to create accurate models of general flexure topologies; (iii) a new approach is also introduced within the BLOT that combines existing multiobjective optimization methods to use these models for iteratively refining the convex and concave portions of a general topology's performance capability boundary with an accuracy determined by the user.

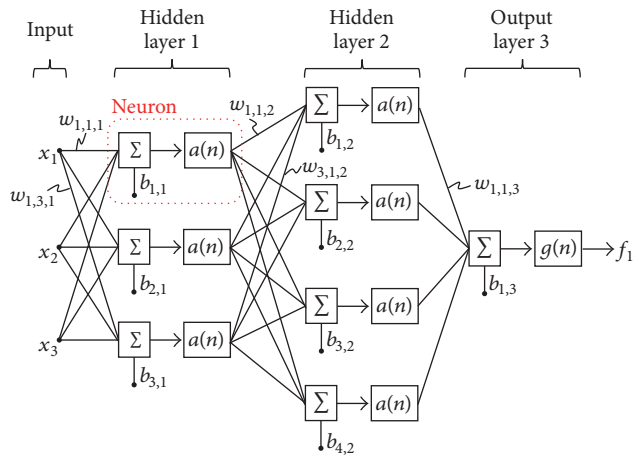


Figure 2: Neural network example with three inputs, one output, and two hidden layers of neurons.

2. Model Generation via Neural Networks

This section summarizes how the BLOT generates models of FACT-synthesized topologies by training artificial neural networks using numerical simulations performed on various geometric instantiations of these topologies. Artificial neural networks, like the kind shown in Figure 2, consist of several processing layers, which can be tuned to successfully map any inputs, x_i, to any corresponding outputs, f_j, when given a sufficient number of diversified input and corresponding target output values to learn from. For the application of this paper, inputs are the geometric parameters that define a flexure system's topology (e.g., $x_1 = W$, $x_2 = T$, and $x_3 = L$ from the topology of Figure 1(f)) and outputs are the performance capabilities (e.g., range of motion or natural frequency) achieved by the design instantiations that are defined by these corresponding input parameters. The known input and target output values are obtained using finite element analysis (FEA) on sample design instantiations with geometric parameters that span the topology's full design space from their smallest achievable feature sizes to their largest geometrically compatible values. The theory of this section provides a systematic way to train neural networks for creating sufficiently accurate flexure system topology models using a minimal number of known input and target output values so minimal computational effort and time are required by the BLOT.

Each layer within a neural network consists of interconnected computational nodes called neurons as labeled in Figure 2. The lines that interconnect the neurons are associated with scalar weight values, $w_{e,t,l}$, where e is a subscript that corresponds to the labeled neuron number (or input) from which the line extends, t is a subscript that corresponds to the labeled neuron number at which the line terminates, and l is a subscript that corresponds to the labeled layer number in which that neuron belongs. Other lines enter the neurons with associated scalar bias values, $b_{t,l}$, that utilize similar subscript conventions. Each neuron within the network's hidden layers (i.e., the layers in between the input

values and the output layer) utilizes an activation function [4], $a(n)$. In this paper we employ the hyperbolic tangent sigmoid function [34] defined by

$$a(n) = \frac{e^n - e^{-n}}{e^n + e^{-n}}. \quad (1)$$

The output from each activation function passes from one hidden layer of neurons to the next until the output layer is reached where this paper employs the linear activation function [34], $g(n) = n$, to produce the final output, f_j. This process is called forward propagation because of the direction in which the mathematics propagates or flows. To demonstrate how the mathematics flows within the network diagramed in Figure 2, consider the neuron highlighted by the dotted red box. The n value that is plugged into the function of (1) corresponding to this highlighted neuron is

$$n = (x_1 \cdot w_{1,1,1}) + (x_2 \cdot w_{2,1,1}) + (x_3 \cdot w_{3,1,1}) + b_{1,1}. \quad (2)$$

Once a sufficient number of known input and target output values have been numerically calculated, various methods can be employed to use these values to iteratively tune the weights, $w_{e,t,l}$, and biases, $b_{t,l}$, within a neural network such that it can link any set of input values within the design space to accurate output values (i.e., the network can be trained to create an accurate predictive model). A popular way to tune the weights and biases is called back propagation [35, 36]. The input values are normalized and fed into the network using initial-guess weights and biases. The resulting outputs are compared against the target outputs that are known to be correct from a priori numerical simulation. A gradient vector of the resulting error with respect to each weight and bias is then used to tune these values in the opposite direction of the gradient vector. By repeating this process numerous times, the network can be trained to accurately predict the correct output values. A variation of the back propagation training method (i.e., Bayesian Regularization [37, 38]) was used to train the neural networks that model the flexure system topologies of this paper.

The specific iterative process used in the BLOT to generate accurate models of general FACT-synthesized topologies is as follows. A user provides a MATLAB script with the independent geometric parameters that define the desired flexure topology. The smallest and largest values for each of these parameters are also provided to the script to specify the full design space over which the resulting model will accurately predict. The script then begins by generating all the possible design instantiations that result from applying every combination of these smallest and largest values with three other evenly spaced values in between them (e.g., $T_{\min}$, $(3T_{\min} + T_{\max})/4$, $(T_{\min} + T_{\max})/2$, $(T_{\min} + 3T_{\max})/4$, and $T_{\max}$) for every independent parameter to the topology's geometry. Each resulting design instantiation is checked for geometric compatibility against various geometric constraint functions, which are specific to their topology. The geometric constraint function imposed on the design instantiations of the example topology labeled in Figure 1(f) is $W - 2T > 0$. Then the MATLAB script passes the resulting design instantiations that are geometrically compatible to a PYTHON script that automatically creates computer-aided-design (CAD) models of these instantiations, defines their constitutive materials, meshes their models, and applies the desired boundary and loading conditions. This script then performs the FEA simulation in parallel batches using ABAQUS to produce the target output values that correspond with each design instantiation defined by the given input values. Thus, large amounts of data can be acquired within the design space of general flexure topologies using the automated scripts.

The numerically generated data is then randomly divided among two different sets. 80% of the data is assigned to a training set and the remaining 20% is assigned to a testing set. The training set of data is used to train a neural network that possesses an initial-guess architecture with two hidden layers and an output layer with only one neuron. The number of neurons in the first hidden layer is initially set to 20 neurons and the number of neurons in the second hidden layer is initially set to zero. The modified mean squared error [39], E_{training}, is calculated for the training set of data using this initial-guess network architecture with initial-guess weights and biases. The Bayesian Regularization algorithm then trains the architecture by updating the weights and biases within the network using the gradient of E_{training} for each iteration as described previously until one of the following five conditions is satisfied.

(1) Training is stopped once an excessively large number of iterations have occurred. For this paper, 1,000 iterations were used as the cutoff.

(2) Training is stopped once an excessive amount of time has lapsed. For this paper, the cutoff time was set to 10 hours.

(3) Training is stopped once the modified mean squared error of the training set of data, E_{training}, reaches a small enough number. For this paper, the number was set to 10−12.

(4) Training is stopped once the gradient of the modified mean squared error of the training set of data, E_{training}, becomes very small. For this paper, 10−7 is used as the cutoff.

(5) Training is stopped once the damping factor [40, 41] used within the Bayesian Regularization algorithm exceeds a certain value. For this paper, the damping factor cutoff was set to 1010.

After training has stopped, the mean absolute percentage error [42], $\text{MAPE}_{\text{total}}$, for the total set of training and testing data combined is calculated using

$$\text{MAPE}_{\text{total}} = \frac{100}{N_{\text{total}}} \sum_{k=1}^{N_{\text{total}}} \left| \frac{Q_{k,\text{total}} - H_{k,\text{total}}}{Q_{k,\text{total}}} \right|, \quad (3)$$

where N_{total} is the total number of target output values from both the training and testing sets combined, $Q_{k,\text{total}}$ are the target output values from both the training and testing sets, and $H_{k,\text{total}}$ are all the corresponding predicted output values calculated using the current trained network. In general, the smaller a trained network's $\text{MAPE}_{\text{total}}$ is, the better the trained neural network can model the behavior of the flexure system topology.

Thus, the $\text{MAPE}_{\text{total}}$ of the trained initial-guess network architecture (i.e., 20 neurons in the first hidden layer and 0 neurons in the second hidden layer) is used as the starting point for optimizing the network's architecture. Pattern search [43], which is described in Section 3 in detail, is used to optimize the number of neurons in the network's architecture in two different phases. In the first phase, pattern search is used to optimize an architecture with no neurons in the second hidden layer. For this phase, the lower bound of neurons in the first hidden layer is set to 5 and the upper bound is set to 40. The input tolerance (The MathWorks Inc.) is set to 1, the initial mesh size (The MathWorks Inc.) is set to 8, and the expansion factor (The MathWorks Inc.) is set to 2. The cost function that is minimized for this optimization is $\text{MAPE}_{\text{total}}$ defined in (3). Once the best trained network architecture is found using this first hidden layer optimization (i.e., the trained network architecture with the smallest $\text{MAPE}_{\text{total}}$), the mean absolute percentage error of the training, $\text{MAPE}_{\text{training}}$, and testing, $\text{MAPE}_{\text{testing}}$, data sets is each calculated using (3) for this best trained network architecture but with the subscripts currently labeled *"total"* replaced by the words *"training"* and *"testing,"* respectively. If both of these values (i.e., $\text{MAPE}_{\text{training}}$ and $\text{MAPE}_{\text{testing}}$) are equal to or less than 35, the neural network is used as the initial model of the FACT-synthesized flexure topology. If, however, either $\text{MAPE}_{\text{training}}$ or $\text{MAPE}_{\text{testing}}$ is greater than 35, the second phase of the network architecture optimization is initiated. In the second phase, pattern search is again used to optimize a network architecture but with 40 neurons in the first hidden layer and an initial-guess of 4 neurons in the second hidden layer. For this phase, the lower bound of neurons in the second hidden layer is set to 1 and the upper bound is set to 15. The input tolerance is again set to 1 and the expansion factor is again set to 2. The best number of neurons in the second hidden layer of the network with 40 neurons in the first hidden layer is identified by minimizing the same cost function in (3), $\text{MAPE}_{\text{total}}$. Once the best trained network architecture is found using this second hidden layer optimization, the mean absolute percentage error of the training, $\text{MAPE}_{\text{training}}$, and testing, $\text{MAPE}_{\text{testing}}$, data sets is each calculated using (3) for this best trained network architecture but with the subscripts currently labeled *"total"* replaced by the words *"training"* and *"testing,"* respectively. If both of these values (i.e., $\text{MAPE}_{\text{training}}$ and $\text{MAPE}_{\text{testing}}$) are equal to or less than 35, the neural network is used as the initial model of the FACT-synthesized flexure topology. If, however, either $\text{MAPE}_{\text{training}}$ or $\text{MAPE}_{\text{testing}}$ is greater than 35, the trained network architecture with the lowest $\text{MAPE}_{\text{total}}$ found from the network optimization of both the first and second phase is used as the initial model of the FACT-synthesized flexure topology.

Thus, using minimal computation, an initial model of general FACT-synthesized flexure system topologies can be identified. Note that many of the hard numbers chosen in this section were selected for rapidly generating accurate models of flexure system topologies with less than ~10 independent geometric parameters. The numbers in this section may need to be adjusted for scenarios with larger numbers of input parameters.

3. Plotting Performance Boundaries

This section discusses the portion of the BLOT that utilizes the models of flexure system topologies generated from Section 2 to plot their performance capability boundaries. Recall that the geometric parameters of a flexure system topology (e.g., W, T, and L for the topology of Figure 1(f)) are its model's inputs, x_i, and the performance capabilities (e.g., range of motion and natural frequency for the example of Figure 1(f)) achieved by the design instantiations that are defined by these corresponding input parameters are the model's outputs, f_j. For this section, a theoretical example of a system with only two inputs, x_1 and x_2, will be used to conceptually explain the algorithm that plots the boundary that circumscribes the system's full design space for two of the system's achievable outputs, $f_1(x_1, x_2)$ and $f_2(x_1, x_2)$.

The boundary-plotting algorithm first requires the smallest and largest values of each input parameter, $x_{i,\min}$ and $x_{i,\max}$, respectively, as well as resolution increment values, Δx_i, for each input parameter. Constraint functions should also be provided to define what combination of input values are permissible (e.g., $W - 2T > 0$ for the example of Figure 1(f)). For the theoretical example of this section, the constraint function is shown as the red spline boundary line in Figure 3(a).

The algorithm begins by supplying a randomly selected combination of permissible inputs, shown as the blue dot labeled $O_{1,1}$ in Figure 3(a), to the Sequential Quadratic Programming (SQP) [44, 45] optimization algorithm. This algorithm first identifies a set of other permissible input combinations that result from adding and subtracting the resolution increment of each input, Δx_i, to and from the first randomly selected combination of inputs along each input's axis. For the example of Figure 3(a), this first set of input combinations are shown as the four blue dots immediately surrounding the blue dot labeled $O_{1,1}$. Note that although Δx_1 is shown as being equal to Δx_2 in the example, these resolution increments do not have to be equal for other scenarios. The system's model is then used to map all of the combinations of these inputs to their corresponding combinations of outputs, which are represented by the five blue dots shown in Figure 3(b). Note that the original input dot, $O_{1,1}$, maps to the output dot, $Z_{1,1}$. The SQP algorithm then approximates the gradient (i.e., first derivatives) and Hessian matrix (i.e., the symmetric matrix of second derivatives) of the objective function defined by

$$J(x_1, x_2) = \cos(\theta) f_2(x_1, x_2) + \sin(\theta) f_1(x_1, x_2) \quad (4)$$

using the input and output combinations (i.e., the adjacent blue dots in Figures 3(a) and 3(b)), where θ in (4) is initially set to 0 so that the largest f_2 output can be pursued first. This gradient and Hessian matrix are then used to construct a Quadratic Programming (QP) subproblem [46], which is solved to determine another combination of inputs that will map to combinations of outputs that produce a larger objective function value. In the example of Figure 3, suppose the new combination of inputs determined is shown as the red dot labeled $O_{1,2}$ in Figure 3(a). Note that this dot's corresponding combination of outputs, which is shown as the

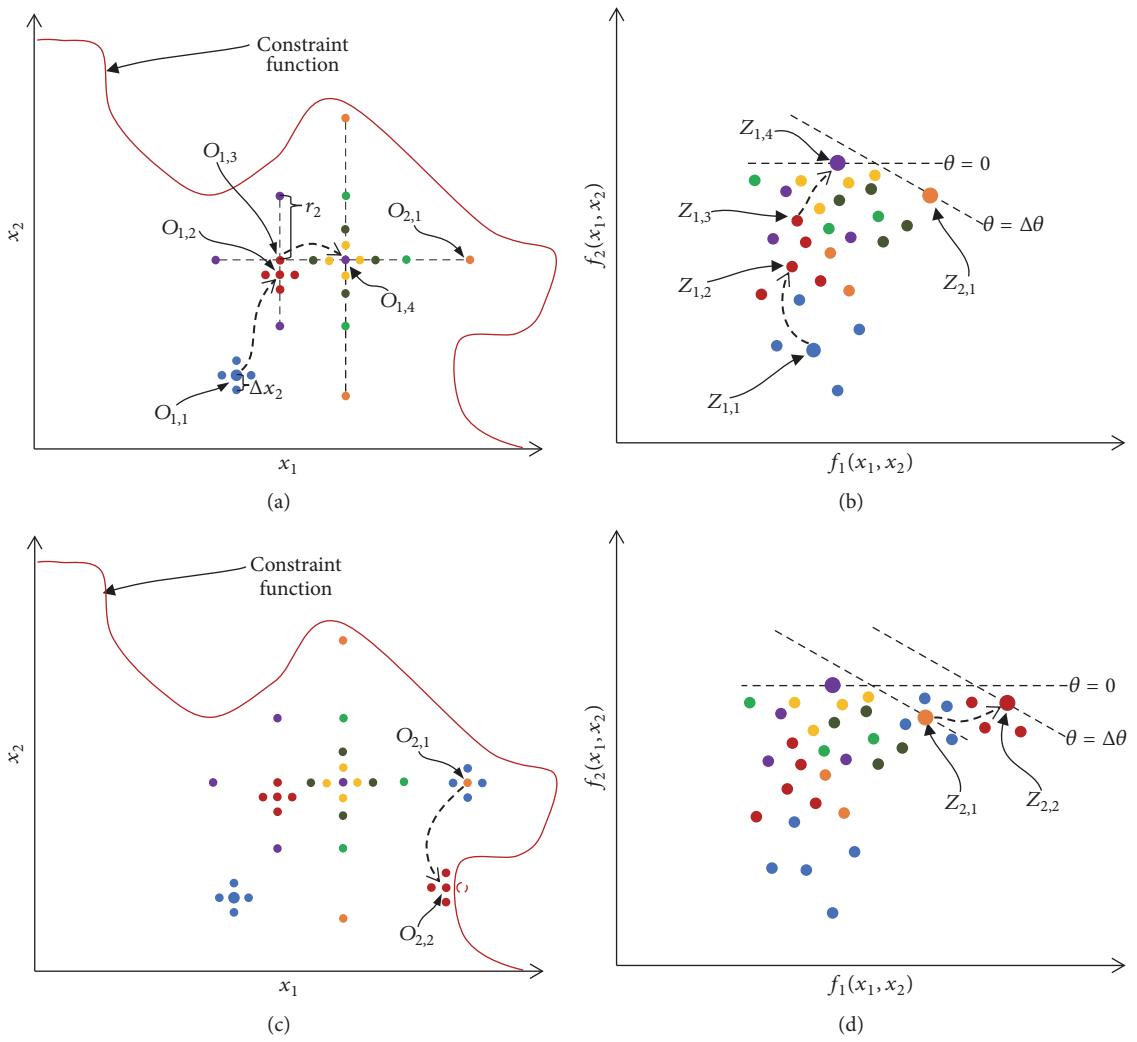


FIGURE 3: Progression of the SQP and then ALPS optimization algorithms for $\theta = 0$ initialized in the objective function in the input space (a) and in the corresponding output space (b); continued progression of the same optimization algorithms for $\theta = \Delta\theta$ incremented in the objective function in the same input space (c) and in the same output space (d).

red dot labeled $Z_{1,2}$ in Figure 3(b), possesses f_2 value that is larger than any of the other previous blue dots shown in the same figure. The SQP algorithm then repeats this process by finding another set of permissible input combinations that result from adding and subtracting the resolution increment of each input, Δx_i, to and from the new combination of inputs (i.e., $O_{1,2}$ from Figure 3(a)) along each input's axis. For the example of Figure 3(a) this new set of input combinations are shown as the four red dots immediately surrounding the red dot labeled $O_{1,2}$. Note that each of the five red dots in Figure 3(a) maps to a corresponding red dot in Figure 3(b). Thus, the SQP algorithm rapidly finds an efficient path toward a local maximum of the objective function by iteratively stepping from one cluster of dots to the next in this manner. The SQP algorithm terminates when either (i) the step distance from one central dot to the next (e.g., the distance between $O_{1,1}$ and $O_{1,2}$ in Figure 3(a)) is smaller than the prescribed resolution of the input parameters, Δx_i, or (ii) the process fails to generate the next step, which typically occurs because the first or second derivatives of the objective function cannot be calculated. The latter reason typically occurs when the algorithm encounters discontinuities within the objective function.

After the SQP algorithm terminates, the boundary-plotting algorithm of this paper continues the optimization process by supplying the combination of permissible inputs that map to the combination of outputs that produce the largest objective function value identified by the SQP algorithm to the Augmented Lagrangian Pattern Search (ALPS) [47–49] optimization algorithm. For the example of Figure 3(a), suppose the SQP process was unable to step beyond the cluster of dots surrounding $O_{1,2}$ and thus terminated at that location. The combination of inputs that would be supplied to the ALPS algorithm would be the red dot labeled $O_{1,3}$ in Figure 3(a) since this combination of inputs maps to the combination of outputs that achieve the largest f_2 value found using the SQP algorithm. This combination of outputs is shown as the red dot labeled $Z_{1,3}$ in Figure 3(b).

The ALPS algorithm first identifies a set of other permissible input combinations that result from adding and subtracting an initial mesh size, r_i, to and from the combination of inputs supplied to the algorithm along each input's axis. The initial mesh size, labeled r_2 in Figure 3(a), is typically set to 20% of the range of its corresponding input parameter (i.e., $r_i = 0.2|x_{i,\max} - x_{i,\min}|$). For the example of Figure 3(a), the first set of ALPS-generated input combinations are shown as the four purple dots surrounding the red dot labeled, $O_{1,3}$. Note that although r_1 is shown as being equal to r_2 in the example, these mesh sizes are not typically equal for other scenarios. The system's model is then used to map the combinations of these inputs to their corresponding combinations of outputs, which are represented by the four purple dots shown in Figure 3(b). The ALPS algorithm then identifies if any of these combinations of inputs map to a combination of outputs that produce an objective function value that is larger than any produced previously in the optimization process. Suppose, for instance, that the input combination of the example in Figure 3(a), labeled $O_{1,4}$, maps to the output combination, labeled $Z_{1,4}$ in Figure 3(b), which achieves the largest f_2 value previously identified. The ALPS algorithm would then step to the dot representing that input combination (e.g., $O_{1,4}$). The algorithm would then identify a set of other permissible input combinations that result from adding and subtracting the previous mesh size (e.g., r_i in this case) multiplied by an expansion factor to and from this combination of inputs along each input's axis. For this paper, the expansion factor is set to 2. Thus, for the example of Figure 3(a), the next set of ALPS-generated input combinations are shown as the three orange dots surrounding the purple dot labeled, $O_{1,4}$. These orange input dots shown in Figure 3(a) map to the three orange output dots shown in Figure 3(b). The algorithm then identifies if any of these output combinations produce an objective function value that is larger than any produced previously in the optimization process. Since none of the orange dots in Figure 3(b) possess an f_2 value that is larger than $Z_{1,4}$, the ALPS algorithm would then identify a set of other permissible input combinations that result from adding and subtracting the previous mesh size (e.g., $2r_i$ in this case) divided by the same expansion factor to and from the combination of inputs labeled $O_{1,4}$ in Figure 3(a) along each input's axis. Thus, for the example of Figure 3(a), the next set of ALPS-generated input combinations are shown as the three light-green dots surrounding the same purple dot labeled $O_{1,4}$. These light-green input dots shown in Figure 3(a) map to the three light-green output dots shown in Figure 3(b). Again, since none of the light-green dots in Figure 3(b) possess an f_2 value that is larger than $Z_{1,4}$, the ALPS algorithm would then identify another set of other permissible input combinations that result from adding and subtracting the previous mesh size (e.g., r_i in this case) divided by the same expansion factor to and from the combination of inputs labeled $O_{1,4}$ in Figure 3(a) along each input's axis. Thus, for the example of Figure 3(a), the next set of ALPS-generated input combinations are shown as the four dark-green dots surrounding the same purple dot labeled $O_{1,4}$. This process repeats until either (i) one of the new input combinations maps to an output combination with an objective function value that is larger than any produced previously or (ii) the mesh size becomes equal to or less than a specified input tolerance, which for this paper is set to the resolution of the input parameters, Δx_i. If the first option (i) occurs, the algorithm will step to the improved input combination and the ALPS process will continue iterating. If the second option (ii) occurs, the ALPS algorithm will terminate. For the example of Figure 3(a), the second option occurred because the mesh size of the four yellow dots shown immediately surrounding the purple dot, labeled $O_{1,4}$, is equal to Δx_2, labeled in the same figure, and none of the new output dots generated ever surpassed the f_2 value of $Z_{1,4}$ as shown in Figure 3(b).

Once the SQP and then ALPS algorithms have both run their full course for determining the maximum value of $J(x_1, x_2)$ from (4) for $\theta = 0$, the θ parameter is then incrementally increased by $\Delta\theta$, which is typically set to a value between $\pi/10$ to $\pi/20$. Using this new θ parameter, the algorithm then computes the value of the objective function in (4) for all existing input combinations, which for the example of this section are shown as the dots in Figure 3(a). From these input combinations, the one that produces the largest objective function value for $\theta = \Delta\theta$ corresponds to the input combination represented by the dot labeled $O_{2,1}$ in Figure 3(a), which maps to the output combination represented by the orange dot labeled $Z_{2,1}$ in Figure 3(b). The algorithm would then supply this input combination to the SQP algorithm to generate more input combinations that produce larger objective function values as described previously. The four blue dots immediately surrounding the dot labeled $O_{2,1}$ in Figure 3(c) would be identified first for the example of this section using this approach. Those four blue dots map to the four new blue dots shown in Figure 3(d) that surround the dot labeled $Z_{2,1}$. The SQP algorithm would then identify the next input combination (e.g., $O_{2,2}$ shown in Figure 3(c)) that produces a larger objective function value. Note that the input combination dot, $O_{2,2}$, maps to an output combination dot, labeled $Z_{2,2}$ in Figure 3(d), that is farther away along the direction prescribed by the new θ parameter value (i.e., $\Delta\theta$). As the SQP algorithm continues, four other input combinations would be identified that immediately surround the input combination dot labeled $O_{2,2}$ in Figure 3(c). These dots are colored red in Figure 3(c). Note, however, that one of the new dots lies outside the red spline line and thus represents a design instantiation that violates the geometric compatibility constraints imposed. Thus, only three of the new surrounding dots map to permissible output combinations as shown in Figure 3(d). The SQP algorithm would continue in this way until it terminates. The ALPS algorithm would then take over where the SQP algorithm left off as described previously until the ALPS algorithm also terminates. Once the ALPS algorithm terminates, the algorithm will have found the input combination that achieves the largest objective function value identified from among those previously tested for $\theta = \Delta\theta$.

The algorithm iterates this pattern of steps to identify the combinations of output values that lie farthest away along their prescribed directions defined by their corresponding θ value in a clockwise fashion until $\theta \geq 2\pi$ (i.e., all the

directions have been swept). It is important to note that any time before the θ parameter is incrementally advanced, the objective function of (4) is rechecked for every combination of input values that have been evaluated up to that point to make sure that each of the previously identified output dots that are used to be the farthest away along their prescribed directions is still the farthest away. If a new dot is ever identified that is farther away than a previous dot along a specific direction, θ (i.e., if a new dot exceeds the dashed lines in Figure 3(b)), the iterative process is reset to that direction and the process continues using that improved output dot.

Once this process is complete, the MATLAB boundary function (The MathWorks Inc.) can be used to identify all the combinations of output values that lie along the boundary of a convex shape that circumscribes the output dots calculated and plotted using the system's model during the iterative process. Many systems, however, produce a cloud of output dots that form a concave—not convex—region like the one shown in Figure 4(a). If the output dots found using the MATLAB boundary function are used to define the boundary of the example in Figure 4(a), the result would be the red boundary shown in Figure 4(b). This boundary would grossly overestimate the actual system's achievable performance space since it is convex instead of concave like the cloud of output dots. Thus, to address this issue, the BLOT identifies all the vectors that point from each output dot along the existing boundary to their neighboring dots on the same boundary. The magnitude, $h_{\max}$, of the longest vector is identified because this vector points between the two output dots (e.g., Z_{h1} and Z_{h2} shown in Figure 4(b)) that usually correspond to the opening of a previously unknown concave boundary. The BLOT then computes a new objective function, $J_g(x_1, x_2)$, defined by

$$\begin{aligned} J_g(x_1, x_2) = {} & \left(\frac{\cos^2\alpha}{R^2} + \sin^2\alpha\right) \\ & \cdot \left(f_1(x_1, x_2) - \frac{f_{1,Zh1} + f_{1,Zh2}}{2}\right)^2 \\ & + \left(\frac{\sin^2\alpha}{R^2} + \cos^2\alpha\right) \\ & \cdot \left(f_2(x_1, x_2) - \frac{f_{2,Zh1} + f_{2,Zh2}}{2}\right)^2 + 2 \\ & \cdot \sin\alpha\cos\alpha\left(\frac{1}{R^2} - 1\right) \\ & \cdot \left(f_1(x_1, x_2) - \frac{f_{1,Zh1} + f_{1,Zh2}}{2}\right) \\ & \cdot \left(f_2(x_1, x_2) - \frac{f_{2,Zh1} + f_{2,Zh2}}{2}\right) \end{aligned} \quad (5)$$

for all the input combinations that have been previously evaluated for $R = 1$. This objective function is minimized to identify output combinations that lie within circular or elliptical regions like those shown in Figure 4(c). If the R variable in (5) is 1, the region is a circle. If, however, this variable increases, it becomes an increasingly elongated ellipse as shown in Figure 4(c). The angle, α, in (5) is defined by

$$\alpha = \arctan\left(\frac{f_{1,Zh2} - f_{1,Zh1}}{f_{2,Zh1} - f_{2,Zh2}}\right), \quad (6)$$

where $f_{1,Zh1}$ and $f_{2,Zh1}$ from (5) and (6) are the horizontal and vertical components of the Z_{h1} dot labeled in Figure 4(c) and $f_{1,Zh2}$ and $f_{2,Zh2}$ are the horizontal and vertical components of the Z_{h2} dot labeled in the same figure. After the BLOT computes the objective function of (5) for all the input combinations that have been previously evaluated for $R = 1$, it identifies the existing input combination that produces the smallest objective function value. This input combination is the combination that maps to the output combination that is closest to the center of the circle shown in Figure 3(c). If no output dots are found within the circle, the input combination corresponding to either the output dot Z_{h1} or Z_{h2} will be chosen as the closest to the center of the circle and is thus supplied to the SQP and then ALPS algorithm described previously to evaluate new input combinations. For this SQP-ALPS optimization, however, the objective function in (5) is minimized for $R = 1$ instead of maximizing the objective function of (4) for different values of θ as was done previously. Suppose, for the example of Figure 4, that when this optimization is performed, the new group of blue output dots shown in Figure 4(c) is generated. Since none of these output dots lie within the center of the circle shown in the figure, the previous R value in the objective function of (5) is multiplied by a factor of 2 and the search region is expanded to an ellipse shown in Figure 4(c). Figure 4(e) shows the elliptical contour diagram of $J_g(x_1, x_2)$ for this R value (i.e., $R = 2$). If no output dots are found within the new ellipse, the process continues to iterate by multiplying the previous R value by the same factor of 2 to increase the elliptical search region further. For the example of Figure 4(c), however, there are output dots that lie within the elliptical search region corresponding to R value of 2. Thus, the input combination that maps to the output dot that lies within this region and possesses the smallest objective function value for $R = 2$ is supplied to the SQP-ALPS optimization algorithm to identify an even better output dot that achieves an even smaller objective function value. This process will produce new output dots (e.g., the new set of orange dots shown in Figure 4(e)). Whether these new output dots achieve a smaller objective function value or not, the output dot that achieves the smallest objective function value is identified and considered part of the system's performance boundary. It is thus redefined as either Z_{h1} or Z_{h2}. In the example of Figure 4(e), the Z_{h2} output dot is the one that is redefined. Note also that $h_{\max}$ is also updated. This boundary learning process is repeated until both (i) the horizontal component of the boundary vector with the largest magnitude (i.e., $h_{\max}$) is less than a set percentage of the horizontal distance across the full cloud of output values and (ii) the vertical component of the same vector is also less than the same percentage of the vertical distance across the same cloud. This percentage threshold is typically set between 5% and

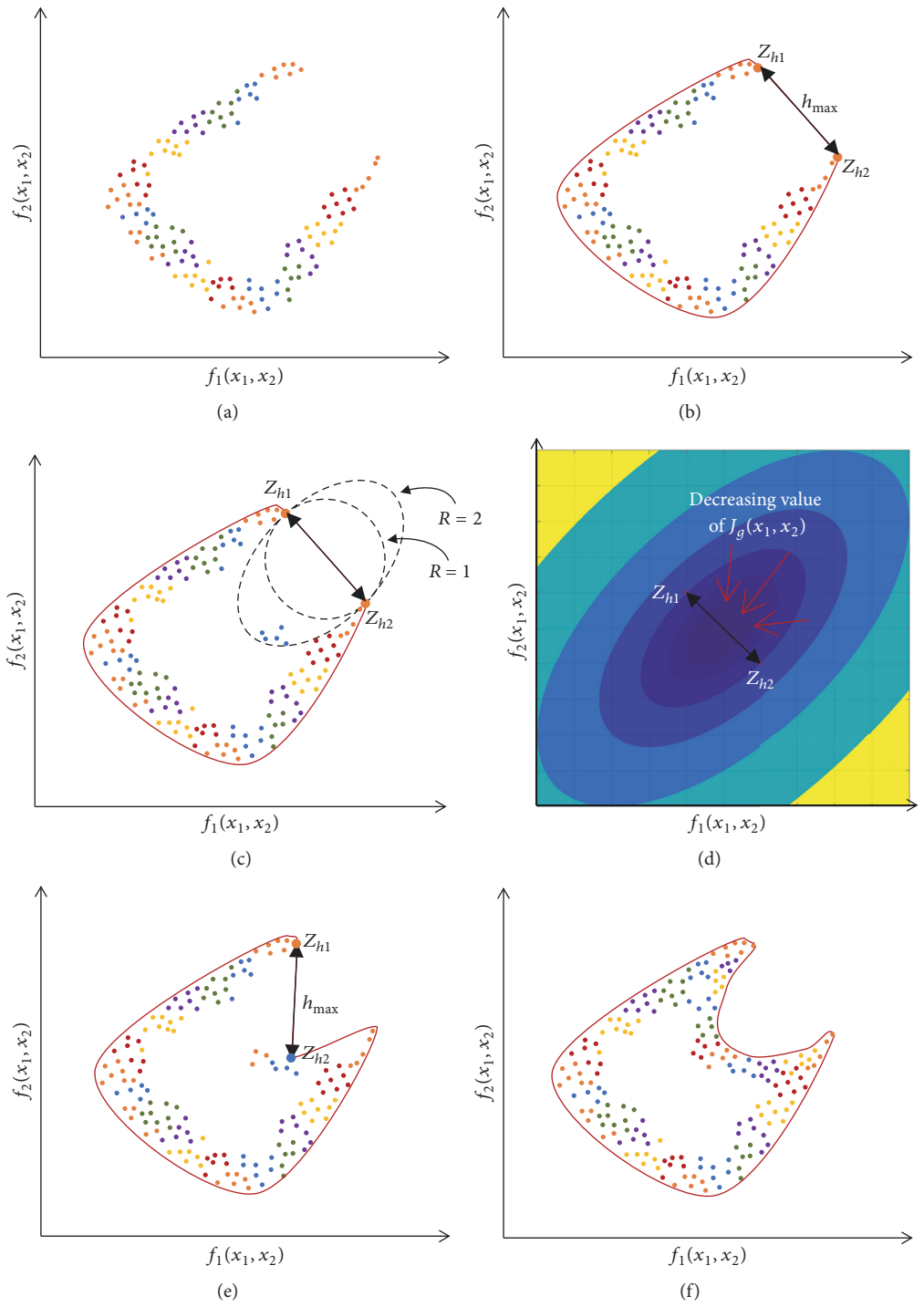


FIGURE 4: Generated cloud of output dots (a); convex boundary first identified (b); new objective function minimized to identify other output dots within circular or elliptical regions (c); elliptical contour diagram of the new objective function with an R value of 2 (d); boundary is updated with a new $h_{\max}$ value (e); process successfully identifies boundaries that are concave (f).

10%. Additionally, before the largest boundary vector is updated with a new magnitude (i.e., $h_{\max}$), the entire optimization process of this section is repeated using the previous objective function in (4) for all θ values to ensure that any new dots evaluated since using that objective function are allowed to improve the boundary's accuracy if possible. In this way both convex and concave boundaries, like the concave boundary shown in Figure 4(f), can be identified that accurately define the system's achievable performance space.

A simple case study is performed to validate the performance of the boundary tracing algorithm of this section.

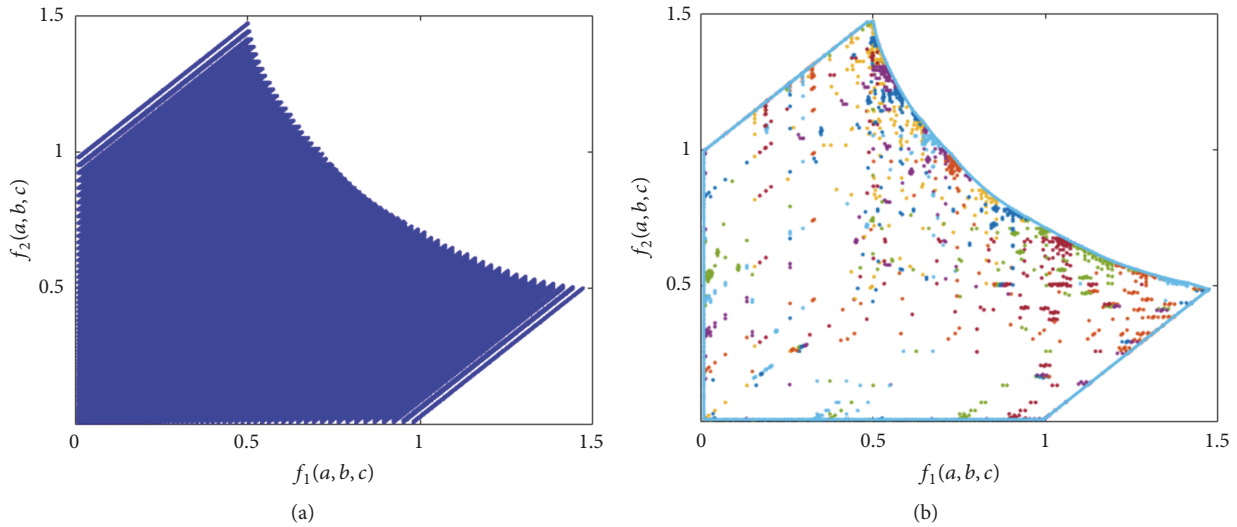


Figure 5: Output values of a full parameter sweep (a); boundary identified for the same example using BLOT (b).

Suppose that three normalized input parameters, p_1, p_2, and p_3, map to two output functions, f_1 and f_2, according to

$$f_1(p_1, p_2, p_3) = p_1^3 + \frac{1}{2}p_2,$$
$$f_2(p_1, p_2, p_3) = p_3^3 + \frac{1}{2}p_2. \quad (7)$$

If the input parameters are constrained according to

$$10p_1 + p_2 + 10p_3 \le 15,$$
$$p_1^2 + \frac{1}{100}p_2^2 + p_3^2 \le 1 \quad (8)$$

and each parameter's smallest and largest values range from 0 to 1 with a resolution increment of 0.01, the boundary of the region achieved by the functions of (7) can be found by plotting the output values for every permissible input value. Figure 5(a) shows all of these output values plotted as blue dots. Another way to find the desired boundary, but with much fewer calculations, is to apply the algorithm of this section as described. The results are shown in Figure 5(b). Note that the algorithm identified a boundary (Figure 5(b)) that is very close to the true boundary that contains the output values (Figure 5(a)) after calculating and plotting only 2,612 dots instead of the 745,298 dots that were calculated and plotted for the full parameter sweep of Figure 5(a).

4. Iterative Model and Boundary Refinement

This section provides the iterative process used to refine the initial neural network-system models generated using the theory of Section 2 as well as their corresponding performance capability boundary identified using the theory of Section 3. Note that a single neural network is trained for each desired output performance capability plotted (e.g., one neural network is trained to predict the flexure system's range of motion for the example of Figure 1(f) and another is trained to predict its natural frequency). The process begins by identifying all the design instantiations that lie along the first boundary plotted using the initial system models. The automated FEM approach discussed in Section 2 is then used to calculate their desired performance capabilities and the results are saved as new sets of data, called boundary sets. The combined $\text{MAPE}_{\text{boundary}}$ of both sets of output performance capabilities is calculated using (3) but with all the subscripts labeled "total" replaced by the word "boundary" and where N_{boundary} is the total number of target output values from both sets of output performance capabilities summed together. If the resulting $\text{MAPE}_{\text{boundary}}$ is equal to or less than 10, the neural network models pertaining to each output performance capability are used to generate the final boundary. If, however, this $\text{MAPE}_{\text{boundary}}$ is greater than 10, 80% of the new data is randomly added to the former training set of data described in Section 2 for each of the corresponding neural network models. The remaining 20% of the boundary data set is added to the former testing set of data also described in Section 2. The entire process of Sections 2 through 4 is then repeated as new boundary data is added to the full body of previously acquired FEM data until the $\text{MAPE}_{\text{boundary}}$ is equal to or less than 10. Once this condition is satisfied, the final neural network models used to predict the output performance capabilities are used to plot the final boundary of the system's performance capabilities using the theory in Section 3. In this way, all the points that constitute the final performance boundary plotted using BLOT for any flexure system are guaranteed to be accurate within an average error of 10% or less.

5. Flexure System Case Study Using the BLOT

This section uses the FACT-synthesized flexure system topology of Figure 1(f) as a detailed case study to demonstrate the capabilities of the BLOT. Suppose that the smallest and

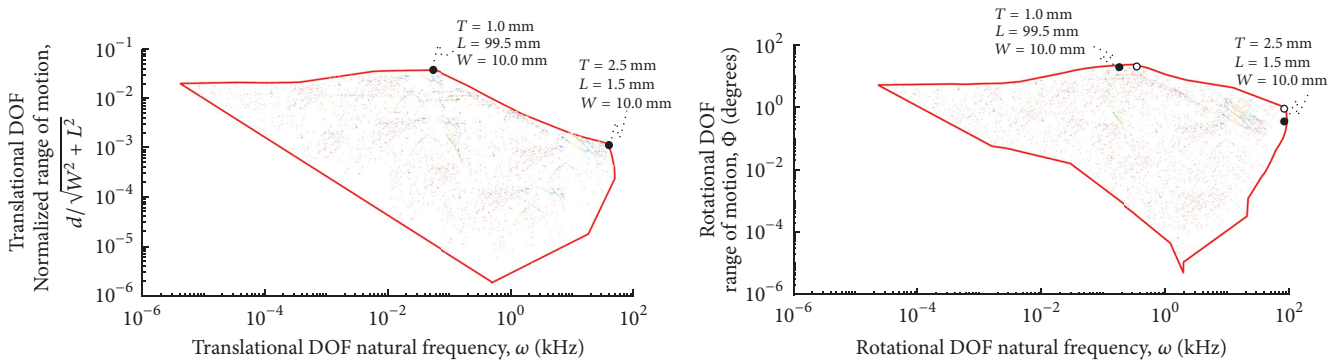


Figure 6: Boundaries of the topology's achievable ranges of motion and natural frequencies corresponding to the translational (a) and rotational (b) DOF generated by the BLOT.

largest values for each of the case study's three independent parameters (i.e., W, T, and L), its resolution increment, and the properties of its constituent material are all set to be the same values as those given in Section 1. Using the $W - 2T > 0$ constraint function provided earlier, the BLOT can be used to train various neural networks to create models that map any desired geometric parameter values within the design space to design instantiations that achieve desired performance capabilities. The performance capabilities of interest to this case study included (i) the topology's range of motion along its translational DOF direction as shown in Figure 1(d), (ii) the natural frequency of the topology that corresponds with the mode shape of its translational DOF, (iii) the topology's angular range of motion about its rotational DOF as shown in Figure 1(e), and (iv) the natural frequency of the topology that corresponds with the mode shape of its rotational DOF. The PYTHON script that automates the collection of all the target data was set to ground the entire top surface of the fixed body shown with thatched lines in Figure 1(f) on the side where the four wire elements are attached. The range of motion of each design instantiation along the direction of the translational DOF was calculated using FEA simulations by applying a shear force on the entire bottom face of the system's stage on the side where the four wire elements are attached. The magnitude of this shear force was gradually increased until any portion within the system began to yield according to Mises yield criterion. The largest amount the stage translated before reaching this yield criterion was defined to be the range of motion, d, of the corresponding design instantiation. This range of motion was normalized by dividing it by its characteristic length, $\sqrt{L^2 + W^2}$. The angular range of motion of each design instantiation about the rotational DOF was also calculated using FEA simulations by applying two opposing shear forces on either side of the system's stage to create a pure moment along the direction of the axes of the wire elements. The magnitude of this moment was gradually increased until any portion within the system began to yield according to Mises yield criterion. The largest amount the stage rotated before reaching this yield criterion was defined to be the angular range of motion, Φ, of the corresponding design instantiation. The natural frequencies, ω_n, corresponding to both of the translational and rotational DOF mode shapes for each design instantiation were also calculated using finite element modal analysis simulations. A unique neural network with a single neuron output layer was trained for each of the four performance capabilities of interest.

Once the BLOT successfully created four accurate models of the flexure system topology of Figure 1(f) by training four neural networks to each map arbitrary design instantiations of the topology to any of the four desired performance capabilities of interest, the BLOT then plotted final boundaries that circumscribe the achievable combinations of these performance capabilities. The boundary corresponding to the normalized range of motion and natural frequency of the mode shape along the direction of the translational DOF is show in Figure 6(a). Note that this was the same boundary provided in Figure 1(g), but the plot in Figure 6(a) includes the colored output dots used to identify the final boundary. The geometric parameters for two optimized design instantiations that span between the optimal portion of the boundary are provided in Figures 1(g) and 6(a). Other optimal design instantiations can be selected from this portion of the boundary to achieve different maximum combinations of the performance capabilities. The boundary corresponding to the angular range of motion and natural frequency of the mode shape about the rotational DOF is provided in Figure 6(b). Note that the two optimal design instantiations shown as large black dots within the plots of Figures 1(g) and 6(a) are also plotted in their corresponding locations within the boundary provided in Figure 6(b). Although these design instantiations are not the two optimal design instantiations within the plot of Figure 6(b), they are close to the optimal instantiations, which are shown as large white dots in Figure 6(b). Finally, it is worth noting that the plots of Figure 6 are presented using a log-log scale. Since such plots can never achieve zero or negative values, the BLOT establishes a cutoff threshold when identifying the boundary to more rapidly and clearly define a practical region of achievable performance capabilities. This threshold was set to 10^{-6} for the plots of Figure 6.

6. Conclusions and Future Work

This paper introduces an optimization tool called the Boundary Learning Optimization Tool (BLOT), which utilizes an automated method for training neural networks to create accurate models of general flexure system topologies synthesized using the FACT approach. The BLOT uses these models to plot the boundaries of regions containing the combinations of desired performance capabilities that are achievable by different design instantiations of the FACT-synthesized topologies. Both the neural network training and boundary identification portions of the BLOT are iterated using additional boundary data until the final boundary is guaranteed to be accurate within a value specified by the user (e.g., less than 10% average error). A flexure system is provided and optimized as a case study.

The BLOT will be used in future works to generate Ashby-like material plots but for different architectured material topologies instead of for natural materials to compare the achievable properties of such microarchitectured structures composed of the same material. These plots will enable engineers to identify trends among architectured materials that result from microstructure instead of composition.

Disclosure

A preliminary version of this paper's theory was first presented at ASME's IDETC as paper no. DETC2017-67465.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Authors' Contributions

Ali Hatamizadeh and Yuanping Song are co-first authors as their contributions to this paper are equal.

Acknowledgments

This work was supported by the Air Force Office of Science Research under Award no. FA9550-15-1-0321. Program officer Byung "Les" Lee is gratefully acknowledged. This work used computational and storage services associated with the Hoffman2 Shared Cluster provided by UCLA Institute for Digital Research and Education's Research Technology Group.

References

[1] J. B. Hopkins, *Design of Parallel Flexure Systems via Freedom and Constraint Topologies (FACT)*, Massachusetts Institute of Technology, 2007.

[2] J. B. Hopkins, *Design of Flexure-Based Motion Stages for Mechatronic Systems via Freedom, Actuation and Constraint Topologies (FACT)*, Massachusetts Institute of Technology, 2010.

[3] J. B. Hopkins, "Designing hybrid flexure systems and elements using freedom and constraint topologies," *Mechanical Sciences*, vol. 4, no. 2, pp. 319–331, 2013.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[5] J. B. Hopkins, K. J. Lange, and C. M. Spadaccini, "Designing microstructural architectures with thermally actuated properties using freedom, actuation, and constraint topologies," *Journal of Mechanical Design*, vol. 135, no. 6, Article ID 061004, 2013.

[6] M. F. Ashby, *Materials Selection in Mechanical Design*, Butterworth-Heinemann, 2005.

[7] X. P. Cheng and R. V. Patel, "Neural network based tracking control of a flexible macro-micro manipulator system," *Neural Networks*, vol. 16, no. 2, pp. 271–286, 2003.

[8] W. He, Y. Chen, and Z. Yin, "Adaptive neural network control of an uncertain robot with full-state constraints," *IEEE Transactions on Cybernetics*, vol. 46, no. 3, pp. 620–629, 2016.

[9] H. Kobayashi and R. Ozawa, "Adaptive neural network control of tendon-driven mechanisms with elastic tendons," *Automatica*, vol. 39, no. 9, pp. 1509–1519, 2003.

[10] J. Takeuchi and Y. Kosugi, "Neural network representation of finite element method," *Neural Networks*, vol. 7, no. 2, pp. 389–395, 1994.

[11] R. I. Levin and N. A. J. Lieven, "Dynamic finite element model updating using neural networks," *Journal of Sound and Vibration*, vol. 210, no. 5, pp. 593–607, 1998.

[12] E. Boillat, S. Kolossov, R. Glardon, M. Loher, D. Saladin, and G. Levy, "Finite element and neural network models for process optimization in selective laser sintering," *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol. 218, no. 6, pp. 607–614, 2004.

[13] M. Lefik, "Hybrid, finite element-artificial neural network model for composite materials," *Journal of Theoretical and Applied Mechanics*, vol. 42, pp. 539–563, 2004.

[14] Y. M. A. Hashash, S. Jung, and J. Ghaboussi, "Numerical implementation of a neural network based material model in finite element analysis," *International Journal for Numerical Methods in Engineering*, vol. 59, no. 7, pp. 989–1005, 2004.

[15] L. Manevitz, M. Yousef, and D. Givoli, "Finite-element mesh generation using self-organizing neural networks," *Computer-Aided Civil and Infrastructure Engineering*, vol. 12, no. 4, pp. 233–250, 1997.

[16] Z. Li and D. Cao, "Conceptual Design of Compliant Mechanism Based on Port Ontology," *Advances in Mechanical Engineering*, vol. 5, p. 401492, 2015.

[17] Y. Qin and Z. Feng, "Structural dynamic modification of flexure hinge structure based on artificial neural network," *Aviation Precision Manufacturing Technology*, vol. 5, no. 11, 2006.

[18] F. G. Kong, Q. Li, and W. J. Zhang, "An artificial neural network approach to mechanism kinematic chain isomorphism identification," *Mechanism and Machine Theory*, vol. 34, no. 2, pp. 271–283, 1999.

[19] E. K. Burke and G. Kendall, Eds., *Search Methodologies*, Springer US, Boston, MA, USA, 2005.

[20] Y. V. Haimes, L. S. Lasdon, and D. A. Wismer, "On a bicriterion formulation of the problems of integrated system identification and system optimization," *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-1, pp. 296-297, 1971.

[21] P. L. Yu, "A class of solutions for group decision problems," *Management Science*, vol. 19, no. 8, pp. 936–946, 1973.

[22] A. P. Wierzbicki, "On the completeness and constructiveness of parametric characterizations to vector optimization problems," *OR Spectrum*, vol. 8, no. 2, pp. 73–87, 1986.

[23] A. P. Wierzbicki, "A mathematical basis for satisficing decision making," in *Organizations: Multiple Agents with Multiple Criteria*, vol. 190 of *Lecture Notes in Economics and Mathematical Systems*, pp. 465–486, Springer Berlin Heidelberg, Berlin, Heidelberg, 1981.

[24] I. Das and J. E. Dennis, "Normal-boundary intersection: a new method for generating the Pareto surface in nonlinear multicriteria optimization problems," *SIAM Journal on Optimization*, vol. 8, no. 3, pp. 631–657, 1998.

[25] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Longman Publishing, Boston, MA, USA, 1st edition, 1989.

[26] "Exceptional paper—lexicographic orders, utilities and decision rules: a survey," *Management Science*, vol. 20, pp. 1442–1471, 1974.

[27] A. Charnes and W. W. Cooper, "Goal programming and multiple objective optimizations," *European Journal of Operational Research*, vol. 1, no. 1, pp. 39–54, 1977.

[28] K. A. de Jong, *Evolutionary Computation: a Unified Approach*, MIT Press, Cambridge, Mass, USA, 2006.

[29] N. Padhye, P. Bhardawaj, and K. Deb, "Improving differential evolution through a unified approach," *Journal of Global Optimization*, vol. 55, no. 4, pp. 771–799, 2013.

[30] O. Sigmund, "On the design of compliant mechanisms using topology optimization," *Mechanics of Structures and Machines*, vol. 25, no. 4, pp. 493–524, 1997.

[31] M. P. Bendsøe and O. Sigmund, *Topology Optimization: Theory, Method and Application*, Springer Berlin Heidelberg, Berlin, Germany, 2003.

[32] Z. Luo, L. Chen, J. Yang, Y. Zhang, and K. Abdel-Malek, "Compliant mechanism design using multi-objective topology optimization scheme of continuum structures," *Structural and Multidisciplinary Optimization*, vol. 30, no. 2, pp. 142–154, 2005.

[33] L. Cao, A. T. Dolovich, A. L. Schwab, J. L. Herder, and W. Zhang, "Toward a unified design approach for both compliant mechanisms and rigid-body mechanisms: module optimization," *Journal of Mechanical Design*, vol. 137, no. 12, Article ID 122301, 2015.

[34] H. B. Demuth, M. H. Beale, O. Jess De, and M. T. Hagan, *Neural Network Design*, 2nd edition, 2014.

[35] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[36] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '89)*, vol. 1, pp. 593–605, Washington, DC, USA, June 1989.

[37] F. Dan Foresee and M. T. Hagan, "Gauss-Newton approximation to Bayesian learning," in *Proceedings of the International Conference on Neural Networks (IJCNN '97)*, vol. 3, pp. 1930–1935, IEEE, Houston, Tex, USA, June 1997.

[38] L. E. Scales, *Introduction to Non-Linear Optimization*, Macmillan Education UK, London, 1985.

[39] M. H. Beale, M. T. Hagan, and H. B. Demuth, *Neural Network Toolbox^{TM} User's Guide for MATLAB R2012a*, Natick, MA, USA, 2012.

[40] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly of Applied Mathematics*, vol. 2, pp. 164–168, 1944.

[41] D. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *SIAM Journal on Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[42] J. S. Armstrong and F. Collopy, "Error measures for generalizing about forecasting methods: empirical comparisons," *International Journal of Forecasting*, vol. 8, no. 1, pp. 69–80, 1992.

[43] C. Audet and J. Dennis, "Analysis of generalized pattern searches," *SIAM Journal on Optimization*, vol. 13, no. 3, pp. 889–903, 2002.

[44] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*, Academic Press, 1981.

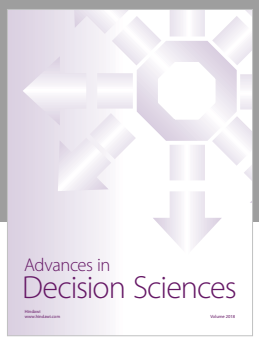
[45] R. Fletcher, *Practical Methods of Optimization*, John Wiley & Sons, New York, NY, USA, 2nd edition, 2000.

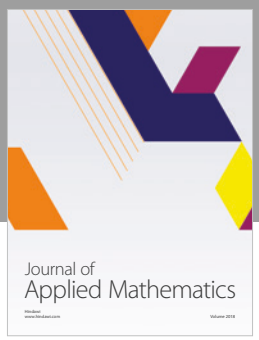
[46] P. E. Gill, W. Murray, and M. H. Wright, *Numerical Linear Algebra and Optimization*, Addison-Wesley, 1991.

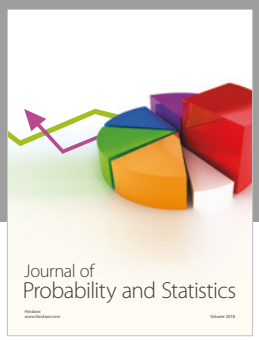
[47] A. R. Conn, N. I. Gould, and P. Toint, "A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds," *SIAM Journal on Numerical Analysis*, vol. 28, no. 2, pp. 545–572, 1991.

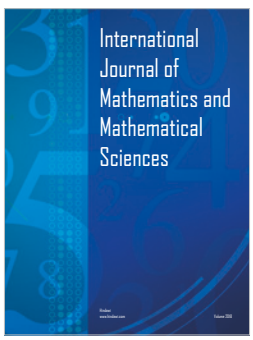
[48] A. R. Conn, N. Gould, and P. L. Toint, "A globally convergent Lagrangian barrier algorithm for optimization with general inequality constraints and simple bounds," *Mathematics of Computation*, vol. 66, no. 217, pp. 261–288, S1–S11, 1997.

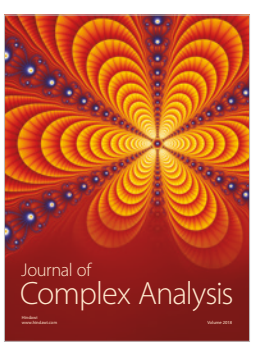
[49] T. G. Kolda, R. M. Lewis, and V. J. Torczon, "A generating set direct search augmented Lagrangian algorithm for optimization with a combination of general and linear constraints," Tech. Rep. SAND2006-5315, Sandia National Laboratories, USA, 2006.

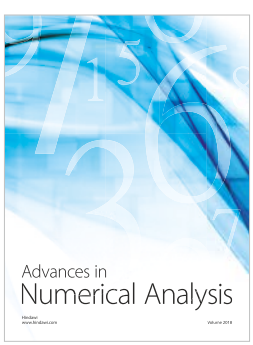


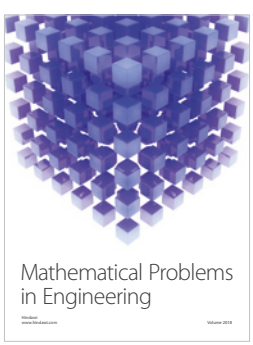


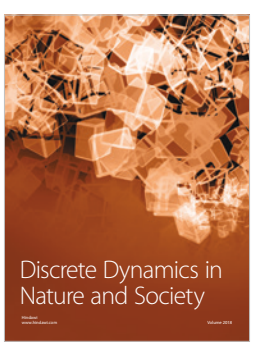


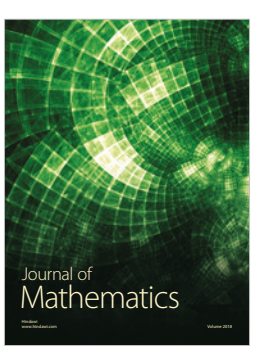


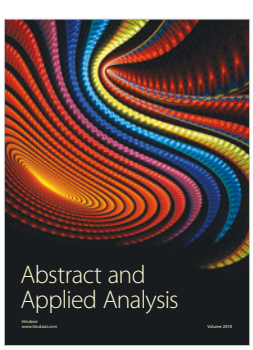


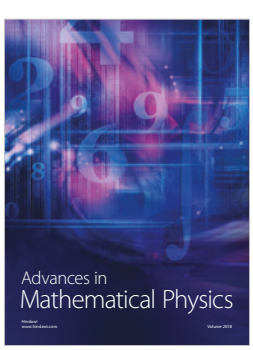












Hindawi

Submit your manuscripts at
www.hindawi.com