Exploring Interactive Methods for Structural Analysis Optimization Problems

Surya Venkatram[1]
Georgia Institute of Technology, Atlanta, GA, 30332, United States of America

In this paper, the finite element method (FEM) is applied to a cantilever beam problem from Svanberg's 1987 publication on the Method of Moving Asymptotes to develop a visualization tool for the model. A custom MATLAB GUI is used to input the requisite model parameters, which are then used for the structural optimization problem corresponding to the system design variables. Given a valid set of inputs and constraints, the program returns a visual representation of the model as well as the optimized design variables – this process can also be repeated with different inputs to view how certain changes to the parameters directly affect the optimized results. The visualization tool is intended to be used in a learning environment, as it visually demonstrates both the relationship between the model and the inputs as well as the role of optimization methods in solving this type of structural analysis problem in a way that may be useful to those newly exploring these concepts.

I. Introduction

Svanberg [1] presents the Method of Moving Asymptotes (MMA) technique as a then-novel method for structural optimization. As a part of this work, Svanberg includes test problems that provide the design variables of interest and the nature of the constraints but with no specified general model for the optimization of the function of interest. As such, in the absence of a given model, if one wishes to observe the results of an optimization for various input parameters a model for the problem must be manually constructed. This is done using the Finite Element Method (FEM), wherein the structure in the problem is discretized into smaller finite elements to build a system of equations that can then be used in the creation of a representative function to be optimized. Optimization tools are utilized to optimize said function and obtain the values for the design variables corresponding to the requisite parameters.

While setting up such a model is not a task of great difficulty for the experienced, nor is performing the optimization through a regular MATLAB program, it is recognized that such concepts can be abstract and rely primarily on individuals creating the model and prescribing static parameters for a numerical result to show up in the command window. When these concepts are being taught to students and other interested parties in a novel context, there is no real method for visually demonstrating how a model response may change as the parameters are dynamically altered in a simple and concise manner, which leaves an interesting niche open as to what kind of solutions may be deployed to assist the learning process.

II. Problem Definition

Consider the problem shown in Fig. 1, wherein a cantilever beam consisting of five elements with a hollowed square cross section has a fixed support at node 1 and an applied vertical loading at node 6. We can designate our design variables to be the widths of the various beam elements, denoted as x_j in the figure, for each of the different beam elements with the given thickness of the solid section remaining constant.

[1] Undergraduate Student, Daniel Guggenheim School of Aerospace Engineering, AIAA Student Member (#1418909)

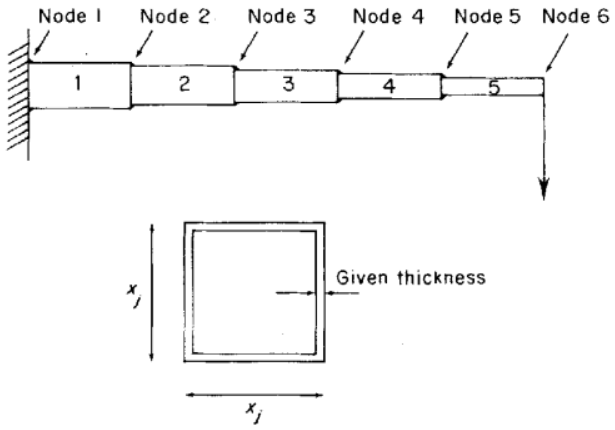

Fig. 1 Cantilever beam test problem from Svanberg [1]

Given this information we can also note that the requisite objective function to be optimized in this instance is the weight function of the beam, with the only constraint being the vertical displacement at node 6 where the applied load is acting. As such, the inputs necessary to construct a model for this problem consist of the maximum allowable tip displacement, applied load magnitude, segment lengths and the given thickness, all of which are used to optimize the weight function to obtain values for the segment widths.

III. Building the Model

Prior to employing FEM techniques, we must first discretize the beam into multiple elements, which has already been done for us as denoted by the labels in Fig. 1 showing Node 1 through Node 6. Therefore, we will consider each individual beam segment an element, and utilize the method given 5 elements and 6 nodes.

A. Defining Inputs

As mentioned previously, our design variables are the widths of the various beam elements, defined in Fig. 1 as x_j, which applies to each of our five segments. This results in five design variables for this problem, defined as x_1, x_2, x_3, x_4 and x_5.

Additionally, we must provide various physical parameters for the model corresponding to the requisite specifications of the optimization problem. In addition to the five distinct element lengths, defined as L_1 through L_5, we must also provide the uniform element thickness (defined as t), the applied load at node 6, the maximum allowable tip displacement, and a young's modulus of 69 GPa corresponding to the uniform aluminum material of the beam.

Using all these parameters, we can build the model and the weight function of the beam to be optimized.

B. Building the Global Stiffness and Force Matrices

To build the global stiffness and force matrices for this problem, we must first find our moments of inertia (MOI). Referencing Eq. (1), which shows the general MOI for the cross-section shown in Fig. 1, we are then able to multiply by our known young's modulus E (69 GPa) to obtain the EI_j term for each of the elements with I_j being the MOI for an element corresponding with section width x_j and uniform thickness t.

$$I_j = \frac{1}{12}\left(x_j^4 - \left(x_j - 2 * t\right)^4\right) \tag{1}$$

We can then calculate our element stiffness matrices using Eq. (2) for each element. As we have five elements in this case, the L_j and EI_j terms correspond to the j-th element sequentially across the beam.

$$K_j = \frac{EI_j}{L_j^3} * \begin{bmatrix} 12 & 6*L_j & -12 & 6*L_j \\ 6*L_j & 4*L_j^2 & -6*L_j & 2*L_j^2 \\ -12 & -6*L_j & 12 & -6*L_j \\ 6*L_j & 2*L_j^2 & -6*L_j & 4*L_j^2 \end{bmatrix} \tag{2}$$

To assemble the global stiffness matrix, we must essentially start with the first 4x4 element stiffness matrix and append the others in a diagonal fashion where the bottom right 2x2 corner of the first matrix overlaps with the top left 2x2 corner of the second matrix and so on for all five of our element stiffness matrices, which eventually results in a 12x12 matrix with all non-filled entries being replaced with zeros. The graphic on the left side of Fig. 2 demonstrates this idea, assuming each different color represents a 4x4 matrix corresponding to one of the elements.

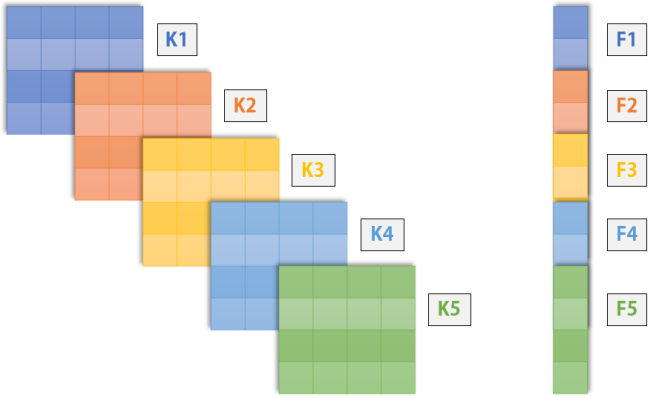

Fig. 2 Graphical representations of global stiffness and force matrix assembly

In the instance of a distributed or complex loading we would have to follow a similar procedure to our global stiffness matrix to construct our global force matrix. However, as the load is only applied at node 6, we can write out a simplified version of that matrix as a 12x1 matrix with load P being the eleventh value and all others as zero. This is reinforced through the graphic on the right side of Fig. 2, wherein the final four entries of the force matrix would correspond to the final element, in this case the one between nodes 5 and 6.

From here, the geometric boundary conditions (BCs) must be applied. We find our total degrees of freedom to be double that of the number of nodes, or twelve. However, as there is a rigid support at node 1, we can enforce that condition and remove two degrees of freedom from our model. Correspondingly, we can remove the first two rows and columns of our global stiffness matrix and the first two rows of the global force matrix to reflect the enforcement of the BCs.

C. Calculating the Weight Function

Considering the hollow sections of the various beam segments, we can express the weight function as shown in Eq. (3) where the upper bound corresponds to the number of beam segments in our problem. Also note that we multiply the summation by the density of aluminum, as the sum gives us the volume of material and the density of aluminum (2710 kg/m^3) is used to calculate the actual weight of the beam.

$$W = 2710 * \sum_{j=1}^{5} L_j * \left(x_j^2 - \left(x_j - 2 * t\right)^2\right) \tag{3}$$

D. Determining and Applying Optimization Conditions

To optimize the weight function, we must first solve the linear equations for displacement. Recalling our global stiffness matrix K_g and global force matrix F_g, Eq. (4) shows the system of equations that must be solved. In addition, Eq. (5) shows a breakdown of the solution vector $X_{v\theta}$ to the system in Eq. (4), where we are only really concerned with the nodal deflections v_i as opposed to nodal beam slope θ_i.

$$K_g * X_{v\theta} = F_g \tag{4}$$

$$X_{v\theta} = \begin{bmatrix} v_2 \\ \theta_2 \\ v_3 \\ \theta_3 \\ v_4 \\ \theta_4 \\ v_5 \\ \theta_5 \\ v_6 \\ \theta_6 \end{bmatrix} \tag{5}$$

At this step, we have all requisite information needed to run our optimization. MATLAB's `fmincon` function is used here, which allows us to find the minimum of a constrained nonlinear multivariable function. The function that is optimized here is our weight function, for which a guess needs to be provided for all initial values of x_j as well as upper and lower bounds of our choosing.

In this case, the lower bounds of x_j are defined as two times the uniform thickness and the upper bounds as the total length of the beam along with arbitrary initial x_j estimations. We can use our FEM calculations for the weight function constraints, as `fmincon` requires both an inequality and equality nonlinear constraint. Enforcing the maximum tip displacement requirement is done with the inequality constraint, wherein the calculated displacement v_6 from Eq. (5) cannot exceed the user-defined maximum. Since there is only one constraint given in the original problem, we do not have an equality constraint and instead provide `fmincon` an empty structure in lieu of a second expression.

It is certainly possible to do all of this within a standard text-based MATLAB program, but that is inconvenient to use in an educational context. Simply modifying variables within the code and showing the command line prints is a static experience with little to no application outside of obtaining numerical results, and there is certainly a more robust and dynamic way to present these sorts of problems to those learning about numerical optimization and the FEM in a way that is more visually appealing and interactive.

IV. MATLAB Visualization Methods and Optimization

Given that we are using MATLAB for model construction and optimization, the simplest method for building a visual interface is to use the inbuilt app designer tool. By utilizing the features included in the software we can create the interface shown in Fig. 3, which is the screen shown to the user upon initializing the program.

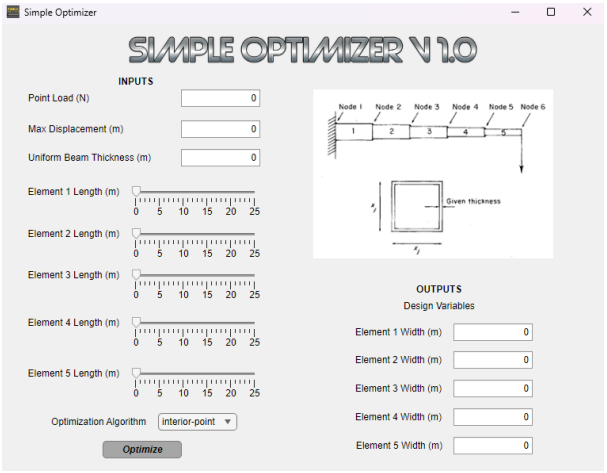

Fig. 3 MATLAB Visualization Interface at startup

To reflect our previously discussed inputs, the tool contains editable text boxes for the applied point load, the maximum displacement constraint, and the uniform beam thickness underneath the "INPUTS" label. Additionally, the element lengths are defined with slider inputs, having an arbitrary limit of 25 meters each. The "Optimization Algorithm" dropdown has three settings for `interior-point`, `sqp` and `active-set`, with `interior-point` being the default selection.

While it is possible to use all three algorithms to perform the optimization, from testing became obvious that `active-set` was not as functional as the others, especially when the inputs are relatively small. This is likely due to how that algorithm works, as `active-set` is a medium-scale algorithm that prioritizes speed [2] and takes larger steps that may conflict with the step tolerance [3] and cause the program to prematurely terminate.

In comparison, `interior-point` and `sqp` are large-scale and medium-scale algorithms respectively that satisfy the bounds at all iterations and can recover from `NaN` and `Inf` results with lower step sizes than `active-set`. As such, both algorithms generally require quite a few iterations but are much less prone to failure and generally produce results consistent with each other.

Given the inputs and the selected optimization method, pressing the “Optimize” button runs the code to optimize the weight function and inputs the given parameters into the SIMULINK model shown in Fig. 4 to produce a visual response for the problem in addition to numerical output values. The SIMULINK model is structured in the same way as the problem itself, with five distinct flexible beam elements connected to a rigid brick anchored to the world frame. By using this model, we can generate animations of the beam based on the physical parameters and actuation of the applied load.

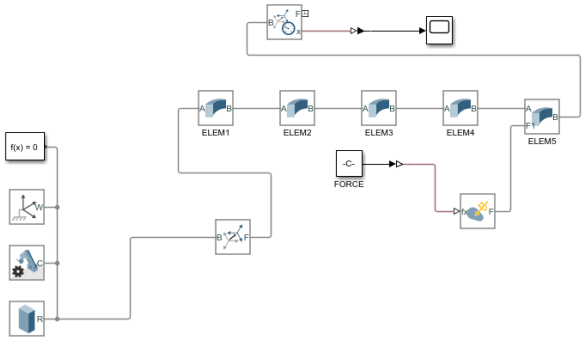

Fig. 4 SIMULINK model of the beam system

Inspection of Fig. 5 shows us that the image of the problem in the startup screen has been replaced by the Simscape Mechanics Explorer visualization of the beam, which is an animated GIF in the original software. This is done by saving the simulation obtained from the software as an AVI video file and then using a custom MATLAB function based on another AVI to GIF converter function written by Moshe Lindner in 2010 [4] to convert the recording to a GIF image. The Graphical User Interface (GUI) then replaces the placeholder image with the GIF of the model response, which can be repeated multiple times without closing the application window and will always update the GIF to that of the most recent optimized configuration.

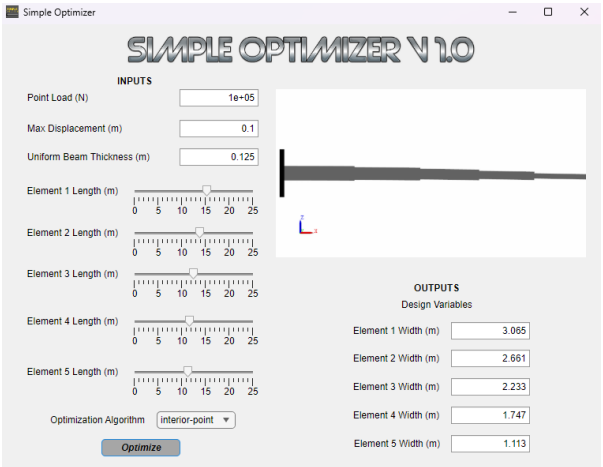

Fig. 5 MATLAB Visualization GUI with sample test parameters

Fig. 6 shows the response when none of the input parameters have been altered except the maximum allowed displacement, wherein the design variable outputs and model have changed significantly. This reflects the impact of the constraint on the problem and shows a visible difference from the original test case thus reinforcing the concept visually. Bolstering this idea is the knowledge that this test case was run without closing or refreshing the program, and as such we may confirm that all other input values are preserved from the run shown in Fig. 5.

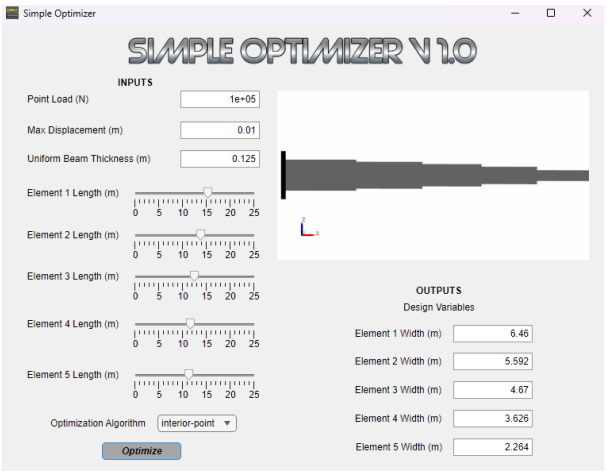

Fig. 6 MATLAB Visualization GUI with modified max displacement parameter

The time delay between runs is non-trivial, as the software must generate and convert the simulation of the model after performing the optimization and refresh the GUI accordingly, but that delay also remains consistent between instances of program execution and allows for successful execution without interruption or variation. During the GIF generation process a loading indicator is also shown in the top right corner of the interface until the model updates, which provides visual context for the end user that successive attempts to run the program may cause errors as the existing instance has not completed processing the simulation.

V. Conclusions

When addressing the question of how to visually demonstrate optimization concepts for structural analysis problems, the method explored in this paper provides a robust solution to that end. While this application is currently restricted to the specific problem explored in depth, the method by which the program and model were created are scalable to any number of problems. In the context of using this software as a tool for teaching and learning, the relatively accessible nature of both the end-user program and the tools used to develop it can provide significant value to educators and students seeking to explore the various parametric relationships and optimization techniques present in these types of problems.

Acknowledgments

I would like to thank Dr. Mayuresh Patil (Georgia Institute of Technology) for providing mentorship, guidance, and feedback on the manuscript. Additional thanks go to Dr. Neil Weston (Georgia Institute of Technology) for manuscript feedback.

References

[1] Svanberg, Krister. "The Method of Moving Asymptotes – a New Method for Structural Optimization." *International Journal for Numerical Methods in Engineering*, vol 24, no. 2, 1987, pp. 359-373. doi: 10.1002/nme.1620240207

[2] "Choosing the Algorithm." *MathWorks*, https://www.mathworks.com/help/optim/ug/choosing-the-algorithm.html#brppuoz

[3] "Tolerances and Stopping Criteria." *MathWorks*, https://www.mathworks.com/help/optim/ug/tolerances-and-stopping-criteria.html

[4] AVI to GIF convertor, MATLAB Add-On Software Package, Ver 1.2.0.1, Moshe Lindner, Bar-Ilan University, Israel, 2010-2018. https://www.mathworks.com/matlabcentral/fileexchange/28772-avi-to-gif-convertor?s_tid=prof_contriblnk