ABSTRACT

Compliant mechanisms use and store a part of the input energy due to the elastic deformation of its members. Therefore, reduced energy is available to act upon a workpiece or against a load. Thus, the mechanical advantage, which is defined as the ratio of the output force to the input force, is affected. On the basis of a prior work, here, we analyze the mechanical advantage of a compliant mechanism by dividing it into two parts: kinematic and compliant. Based on this analysis, it is inferred that the total mechanical advantage (MA) of a compliant mechanism is always less than the kinematic component of MA, implying that the compliant component of MA is always negative. We observe that, in order for it to be positive, the rate of strain energy of the mechanism with respect to deformation should be negative, which is possible by release of a pre-load. An example is presented to demonstrate that the compliant component of MA can be a positive contributor to the total MA and thus the total MA can be greater than the kinematic component of MA within a range of motion of the deformation variable.

We also present a compliant mechanism whose MA has minimal dependence on the workpiece stiffness. The sensitivity index, which defines the sensitivity of MA of a compliant mechanism to the workpiece stiffness, is analyzed using the spring-lever model introduced in earlier works. An optimization problem is presented to obtain a topology for compliant mechanism that exhibits low sensitivity towards workpiece stiffness. Furthermore, MA is plotted against a non-dimensional parameter that captures the topology, shape, overall size, and cross-section size of the beam segments, material, and forces. The resulting MA maps provide insights into the transmission capability of a compliant mechanism of given topology and shape for fixed proportions irrespective of size, material, and forces.

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to my guide, Prof. G.K. Ananthasuresh for providing me an opportunity to work with him. His constant motivation and support have always encouraged me for my work. It is also my privilege to learn from faculty members here at IISc who will always be an inspiration for me.

I thank M2D2 lab for providing excellent facilities and a warm and friendly environment that helped me during my course and my project work. I acknowledge my lab-mates Santosh, Sreenath, Nandhini, Shantanu, Rinku, Bharath, Ravi, Ramu, Anju, Saurabh, Ananya, Jagdish, Ramnath, Navaneet, Sandeep for their constant help and support.

I am grateful to my organization Defence Research and Development Organization (DRDO) to allow me to pursue my post-graduation here at IISc. I particularly thank my senior colleague Kiran Akella who always motivated me towards my work.

I thank my colleagues, friends, the staff members at ME office who have always helped me in my studies and IISc for providing gymkhana facilities that made my stay in Bangalore comfortable.

Finally, I thank my parents, my sister who whole-heartedly supported me in my academic endeavors.

Mohit Mathur

Department of Mechanical Engineering

Indian Institute of Science

Bangalore.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1. INTRODUCTION

Mechanical advantage (MA) is an important metric for mechanisms as it indicates how good the mechanism is in transmitting a force from an input to an output. It is well- studied for rigid-body mechanisms [1]. MA is defined as the ratio of the output force to the input force. For a rigid-body mechanism, it is equal to the ratio of the first-order changes in input and output displacements, i.e., $MA = \Delta u_{in} / \Delta u_{out}$. On the other hand, as was observed by Salamon and Midha [2], the mechanical advantage of a compliant mechanism can be split into rigid-body and compliant components, MA_r and MA_c as

$$MA = MA_r + MA_c = \frac{\Delta u_{in}}{\Delta u_{out}} + \left(-\frac{\Delta SE}{F_{in} \Delta u_{out}} \right) \tag{1}$$

where ΔSE is the incremental change in the strain energy and F_{in} is the input force. Based on their analysis, Salamon and Midha [2] had made three important observations regarding MA of compliant mechanisms.

1) Some of the input energy is used for elastically deforming a compliant mechanism, which affects MA, usually unfavorably.
2) Unlike in rigid-body mechanisms, the mechanical advantage of compliant mechanisms depends on the workpiece stiffness, k_{ext} and Δu_{out}. Hence, it needs to be defined differently when the workpiece is rigid and when it is not, and for different values of the input force and output displacement.
3) The mechanical advantage of a compliant mechanism for a given input force as a function of the workpiece stiffness, k_{ext}, given by

$$MA = MA_s \left(\frac{k_{ext}}{s_k + k_{ext}} \right) \tag{2}$$

where MA_s is the bounding value for MA and s_k is the sensitivity index.

We revisit these findings and extend those using newer models of compliant mechanisms so that a method can be developed to design by maximizing MA_s while keeping s_k sufficiently small.

1.1. Motivation

The motivation for this work is derived from [3] which reported that both the compliant and rigid components of MA can be positive and therefore, the total MA can exceed that of the rigid-body counterpart. In this work, we demonstrate this with the help of analysis and an actual prototype. Also, a compliant mechanism is designed with large MA_s and small s_k using topology optimization and spring-lever model [4].

1.2. Related work and scope of this work

Few concepts and methods that form the basis of the work presented in this report are explained first.

Pseudo rigid-body Model

Pseudo rigid-body (PRB) models [5] are used to model the deflection of flexible members using spring-constrained rigid-body linkages that have the same force-deflection characteristics. They can then be used for

analysis and synthesis [6]. In this work, we use the PRB concept to analyze and synthesize compliant mechanisms.

Spring-lever (SL) Model

The SL model is a lumped two degree-of-freedom (2-dof) representation that comprises three independent parameters that can be interpreted as intrinsic properties of a compliant mechanism. The three parameters in SL modeling can also be understood by referring to input-output model presented by Wang [10].

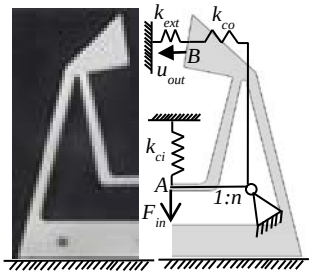


Figure 1 A symmetric-half of a compliant mechanism and equivalent spring-lever model

Figure 1 depicts a compliant mechanism that consists of input-port at A and output-port at B. The input-output stiffness matrix $\mathbf{K}$ in static and linear finite element analysis (FEA) can be represented in the following form:

$$\begin{bmatrix} K_{ii} & K_{io} \\ K_{io} & K_{oo} \end{bmatrix} \begin{Bmatrix} u_{in} \\ u_{out} \end{Bmatrix} = \begin{Bmatrix} F_{in} \\ F_{out} \end{Bmatrix} \tag{3}$$

Since $\mathbf{K}$ is symmetric, any single-input and single-output system can be represented by three constants that are intrinsic properties of a system and are function of the geometry and material properties of the mechanism under consideration. The SL model, introduced in [4] and [7], consists of three parameters (n, k_{ci} and k_{co}), seen in Figure 1, which can be related to those in Eq. (3) as follows:

$$\begin{aligned} K_{ii} &= n^2 k_{co} + k_{ci} \\ K_{io} &= -n k_{co} \\ K_{oo} &= k_{co} \end{aligned} \tag{4}$$

Although, the SL model was introduced for small-displacement of the compliant mechanisms in [4] and [7], it is valid for large- displacement, as shown in appendix A. In this work, we use the SL model to validate the form of Eq. (2) and express MA_s and s_k in terms of the SL-parameters.

Optimization

An ideal compliant mechanism, from the viewpoint of force transmission, is one that has high MA_s and low s_k. We observed, with the help of SL-parameters, that maximizing MA_s is in conflict with minimizing s_k. Hence, we posed this as an optimization problem. Since topology is often the critical factor, we formulated as a topology optimization using beam-element ground structure method [8].

Non-dimensionalization

Mechanical advantage is a non-dimensional quantity and it depends on the topology, shape and proportion of sizes of a compliant mechanism. Following [9], we present plots of *MA* against a non-dimensional quantity that include forces, Young's modulus, overall size, and cross-section dimensions of the beam segments. The use of

this non-dimensional portrayal of *MA* and a way to depict *MA* as a function of non-dimensionalized workpiece stiffness are presented in this work.

1.3. Objectives of the work

i. Demonstration of a practical example where MA of compliant mechanism is enhanced beyond the kinematic part.
ii. Designing compliant mechanisms using SL-parameters and topology optimization with high MA and low sensitivity with respect to the stiffness of the workpiece.
iii. Generating MA maps with non-dimensional parameters involving input force, output force, geometric and material properties, and workpiece stiffness.

1.4. Organization of the report

A compliant mechanism is synthesized whose MA is greater than its kinematic (or rigid-body) component and the methods involved are presented in Section 2. Analysis of MA using the SL model is presented in Section 3. Furthermore, based on the SL-parameters, topology optimization problems posed and the optimized compliant mechanisms are discussed in Section 4. Non-dimensional maps of MA for different compliant mechanisms are presented in Section 5. Concluding remarks and contributions of this work are in Section 6.

2. COMPLIANT COMPONENT OF MECHANICAL ADVANTAGE

Consider the first-order approximation of the input work, output work, and strain energy in a small incremental step from a given configuration of a compliant mechanism. The incremental work ΔW done by input and output forces is given by

$$\Delta W = (F_{in} + \frac{1}{2}\Delta F_{in})\Delta u_{in} - (F_{out} + \frac{1}{2}\Delta F_{out})\Delta u_{out} \tag{5}$$

where F_{in} and F_{out} are input and output forces and Δu_{in} and Δu_{out} are the corresponding incremental displacements. By neglecting second order terms in Eq. (5), we get

$$\Delta W \approx F_{in}\Delta u_{in} - F_{out}\Delta u_{out} \tag{6}$$

According to the principle of virtual work, ΔW equals the first order change in the strain energy ΔSE . Thus,

$$\begin{aligned} &\Delta SE = \Delta W = F_{in}\Delta u_{in} - F_{out}\Delta u_{out} \\ &\Rightarrow \frac{F_{out}}{F_{in}} = \frac{\Delta u_{in}}{\Delta u_{out}} - \frac{\Delta SE}{F_{in}\Delta u_{out}} \Rightarrow MA = MA_r + MA_c \end{aligned} \tag{7}$$

$$MA = \frac{F_{out}}{F_{in}} = \text{total mechanical advantage} \tag{8}$$

$$MA_r = \frac{\Delta u_{in}}{\Delta u_{out}} = \text{rigid-body (kinematic) component of the mechanical advantage, which is the reciprocal of the geometric advantage; and} \tag{9}$$

$$MA_c = -\frac{\Delta SE}{F_{in}\Delta u_{out}} = \text{compliant component of the mechanical advantage.} \tag{10}$$

This analysis was reported by Salamon and Midha [2]. Eq. (7) and Eq. (10) may imply that the total mechanical advantage (MA) is always less than its rigid-body component (MA_r), as MA_c is negative. However, MA_c can be

positive if ΔSE is negative, i.e., if the strain energy decreases with the incremental elastic deformation of the compliant mechanism. This is the motivation for designing a compliant mechanism that has negative ΔSE with increasing elastic deformation. A practical example is presented for a compliant mechanism that is analyzed using PRB modeling wherein flexure dimensions are obtained using small-length flexural pivot approximation [5].

2.1. Pseudo-rigid-body (PRB) model

Consider a PRB model of a compliant mechanism shown in Figure 2 with A and B as the input and output ports respectively. Torsional springs with torsional stiffness κ_1 and κ_2 are attached at the two ends of a rigid bar of length *l*. The two sliders on the rollers depict symmetry boundary conditions of the full mechanism and translational springs k_{in} and k_{ext} represent input-side stiffness and workpiece stiffness respectively. The angle made by the rigid bar is θ and in the initial configuration it is represented by θ_0. The numerical values for Figure 2 are given in Table 1.

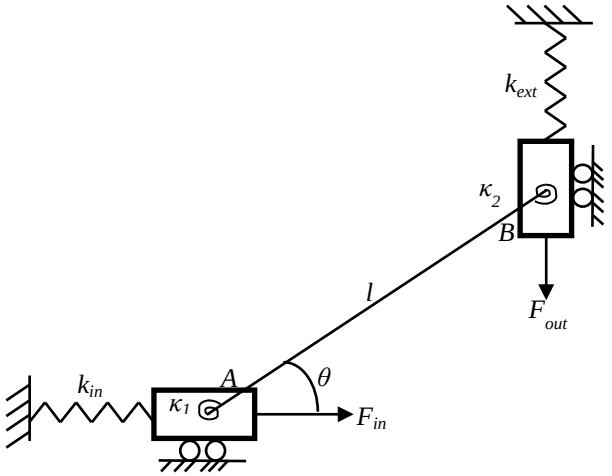


Figure 2 A pseudo-rigid-body model

Table 1 Numerical data for Figure 2

Quantity	**Numerical Value**	**Quantity**	**Numerical Value**
l	0.25 m	θ_0	55°
k_{in}	0 N/m	F_{out}	27 N
k_{ext}	0 N/m	F_{in}	2 to 20 N
κ_1	5 Nm/rad		
κ_2	8 Nm/rad		

As discussed in [3], the expression of MA can be written as

$$MA = MA_r + MA_c \tag{11}$$

where MA_r is the rigid-body component of MA is given by $MA_r = \tan\theta$ (12)

and MA_c is the compliant component of MA given by

$$MA_c = -\frac{1}{F_{in} l \cos\theta}\begin{Bmatrix} k_{in} l^2 (\cos\theta_0 - \cos\theta)\sin\theta \\ +k_{ext} l^2 (\sin\theta - \sin\theta_0)\cos\theta \\ +(\kappa_1 + \kappa_2)(\theta - \theta_0) \end{Bmatrix} \tag{13}$$

As can be observed in Figures 3 and 4, MA is greater than the rigid-body component (MA_r) for equilibrium angle range of approximately 30^o to 55^o. It is observed from Figure 5 that the strain energy decreases for the same range of equilibrium angle (i.e. of approximately 30^o to 55^o). Thus, Figures 3 to 5 prompt us to design a physical form for the PRB model using a compliant mechanism whose strain energy decreases with the increasing deformation of its compliant members. We use flexures for the physical realization of the torsional springs.

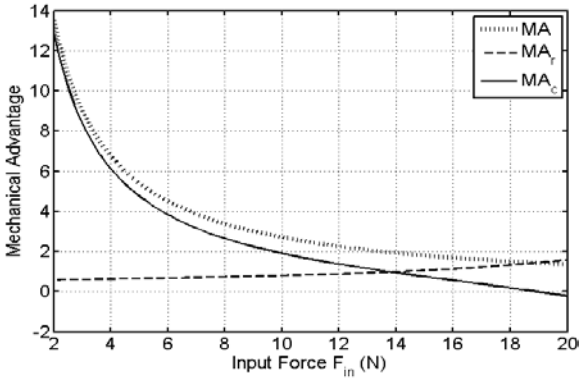


Figure 3 Mechanical advantage vs input force

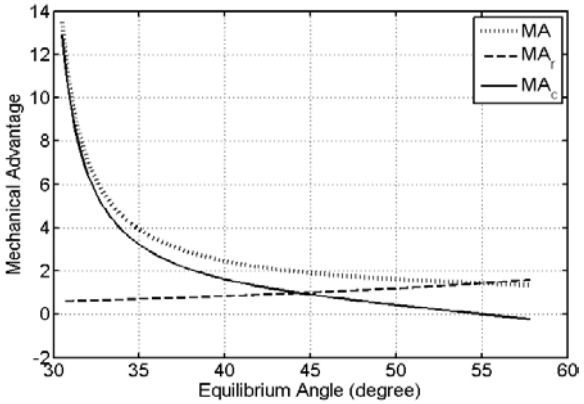


Figure 4 Mechanical advantage vs equilibrium angle

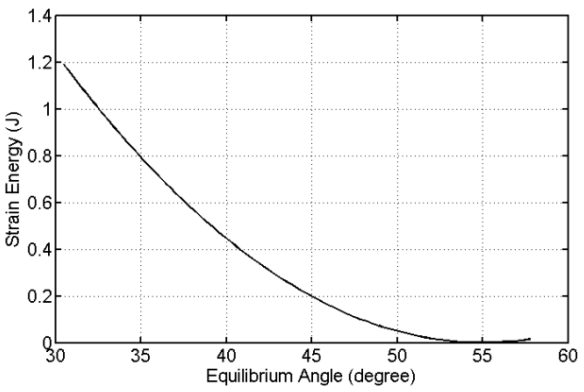


Figure 5 Strain energy vs equilibrium angle

2.2. Small-length flexural pivots and equivalent beam compliant model

In order to determine the accuracy of the PRB model, a physical form consisting of small-length flexural pivots is constructed. As discussed by Howell in [5], the torsional stiffness of a PRB model for a small-length flexure is given by $K = \frac{EI}{l}$. Therefore, an equivalent beam compliant model consisting of small-length flexure can be synthesized based on this PRB model.

By referring to Figures 2 and 6, and Table 2, we note that the torsional stiffness of the spring is given by $\kappa_1 = \frac{EI_1}{l_1} = 5\,Nm/rad$ where $I_1 = \frac{bh_1^3}{12}$ =*1.04e-10 m*^{4}. Upon solving, we get l_1 = *0.027 m*. Similarly, l_2 = *0.027 m* and $l = \frac{l_1}{2} + \frac{l_2}{2} + l_3 \Rightarrow l_3$ = *0.228 m*. The out-of-plane thickness b is chosen to be 0.01 m.

The beam compliant model in Figure 6 depicts only a quarter of the compliant mechanism. Therefore, a 2D continuum model of the compliant mechanism, shown in Figure 7, is analyzed in Abaqus (www.simulia.com).

The numerical values for Figure 6 and Figure 7 are given in Table 2 and Table 3 respectively, in which *b* is the out-of-plane thickness and *E* is the Young's modulus.

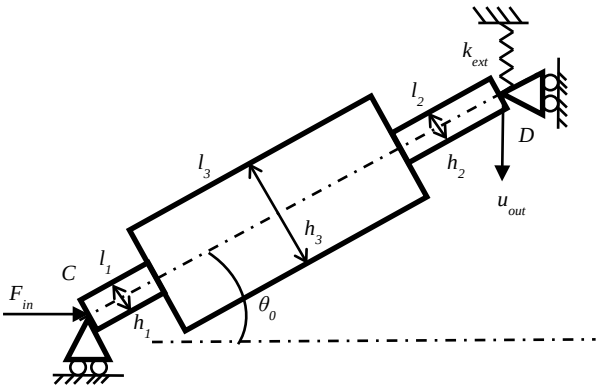


Figure 6 Equivalent beam compliant model

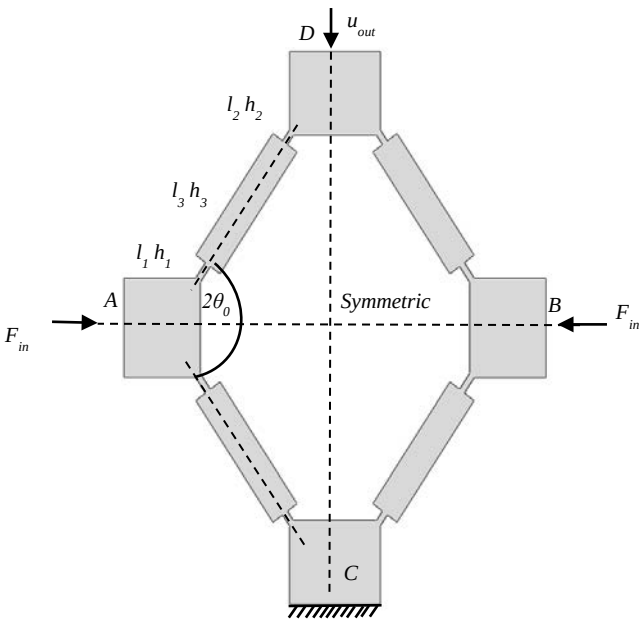


Figure 7 2D continuum model of the compliant mechanism

Table 2 Numerical data for Figure 6

Quantity	Numerical Value	Quantity	Numerical Value
E	1.3 GPa	h_2	0.005 m
b	0.01 m	h_3	0.05 m
l_1	0.027 m	θ_0	55°
l_2	0.017 m	k_{ext}	0 N/m
l_3	0.228 m	F_{in}	2-20 N
h_1	0.005 m		

Table 3 Numerical data for Figure 7

Quantity	Numerical Value	Quantity	Numerical Value
E	1.3 GPa	h_1	0.005 m
b	0.01 m	h_2	0.005 m
l_1	0.027 m	h_3	0.05 m
l_2	0.017 m	θ_0	55°
l_3	0.228 m		

For purpose of comparison, the prescribed displacement, u_{out} in Figure 7 is taken to be the same as that of the PRB model in Section 2.1 and the input force is varied from 2 N to 20 N as in the PRB model. The comparison of results of the PRB model and the 2D continuum model is shown in Figures 8 to 10.

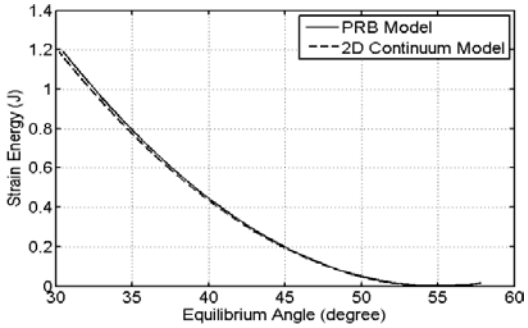


Figure 8 Comparison of strain energy with equilibrium angle using the PRB and the continuum model

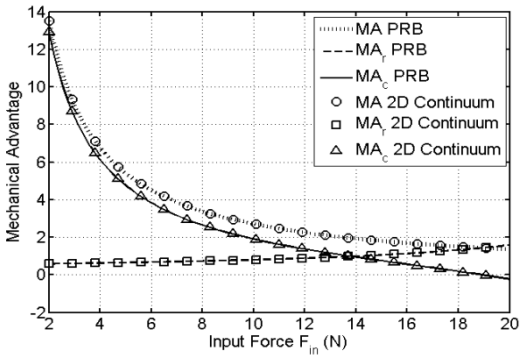


Figure 9 Comparison of MA with input force using the PRB and the continuum model

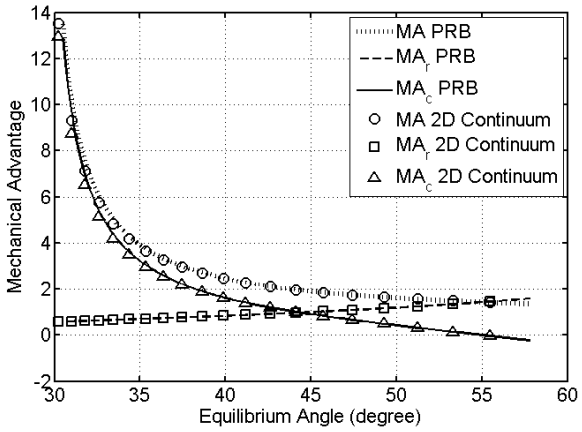


Figure 10 Comparison of MA with equilibrium angle using the PRB and the continuum model

As can be observed from Figures 8-10, there is a very close match (less than 2% discrepancy) between the results of PRB model and the 2D continuum model. Thus, the PRB model is an accurate representation of the small-length flexure.

As shown in Figure 11, a scaled-down prototype (1:10) is fabricated by 3D printing technique to validate the design. For the sake of demonstration, the prototype can be operated by hand. This prototype is tested for measuring the mechanical advantage by using Dynamic Mechanical Analyzer (ElectroForce 3100, Bose Corp.). This experimentation is deemed important to validate the concept of compliant part of MA being a positive contributor.

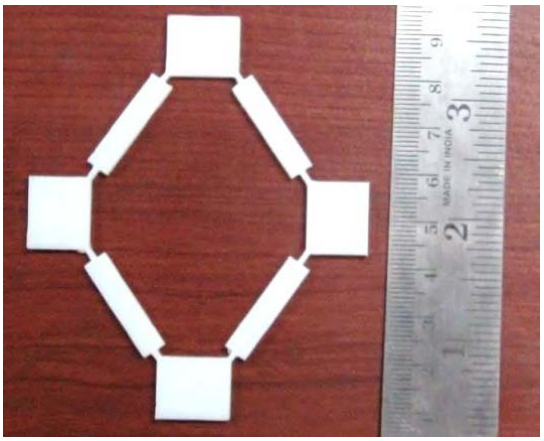


Figure 11 A scaled-down prototype of the compliant mechanism

2.3. Dynamic Mechanical Analyzer (DMA) experiment

As shown in Figure 12, the compliant mechanism is rotated by 90^0 from the schematic shown in Figure 7 with one of the input port fixed to DMA. The ports *A, B, C* and *D* in Figure 12 represent the respective ports in Figure 7. For the purpose of uniform output force, F_{out}, a helical spring is attached at output ports *C* and *D*. The displacement is prescribed at input port *A* and the input force, F_{in}, is measured by DMA at port *B*.

The spring stiffness is measured on DMA and is found to be 0.17 N/mm. The experimental graph is shown in Figure 13.

As can be observed from Figure 13 the force-displacement curve is linear and the stiffness (slope) is found to be 0.17 N/mm by curve-fitting. Also, the intercept on y-axis is not zero. This is because the zero displacement considered for calculation is the stretched configuration of the spring that gave non-zero value on the y-intercept. The initial stretch at zero displacement is about 50 mm. However, the slope/stiffness remains constant for the displacement considered.

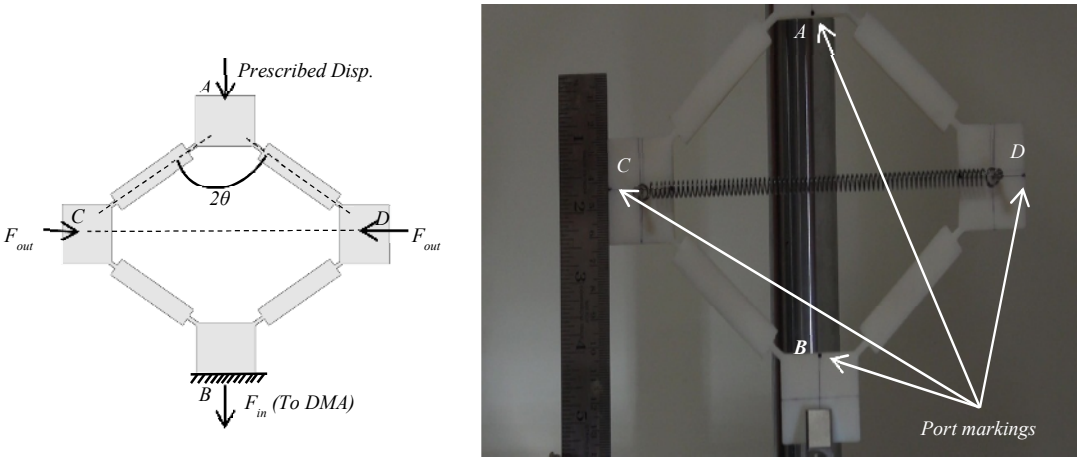


Figure 12 Schematic and experimental setup for DMA experiment

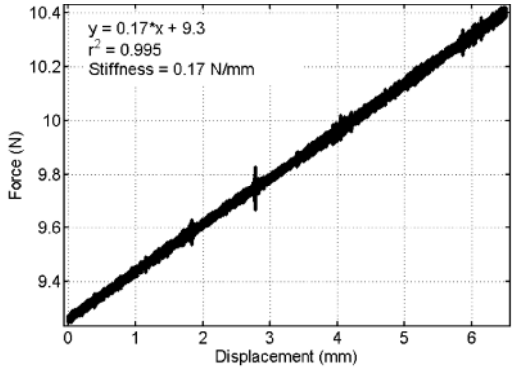


Figure 13 Force vs displacement graph for spring used for DMA experiment

As in Figure 12, the relative displacement between the ports is measured using image-processing that are correlated with the (input) displacement and (input) force readings from DMA. The output force is then calculated by multiplying the stiffness of the spring to the displacement obtained using image-processing. The mechanical advantage (MA) is calculated using Eq. (8) and the rigid-body component (MA_r) is calculated by Eq. (12) as $MA_r = \tan\theta$. Image-processing is used to find θ. The compliant component (MA_c) is given by $MA_c = MA - MA_r$.

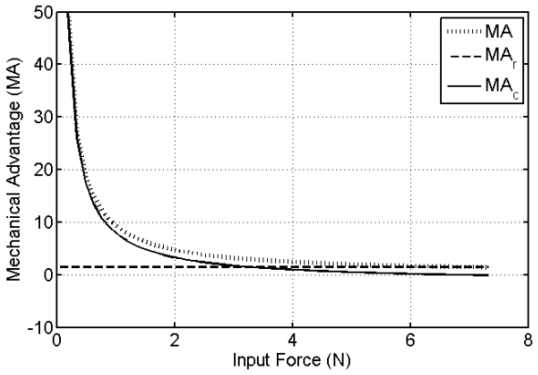


Figure 14 Mechanical advantage plots from the DMA experiment

As can be observed from Figure 14, compliant component of MA, i.e., MA_c is positive for substantial range of the input force, thus indicating larger MA than the rigid-body counterpart. Since MA_c is positive, we can conclude from Eq. (10) that the incremental change in the strain energy (ΔSE) is negative. This phenomenon of decrease in strain energy (SE) may be understood as follows. Since the system is a single-input and single-output system, an input force causes the output displacement in the known direction. In the case presented in Sections 2.1 to 2.3, the output force is constant and the input force is increased gradually. For smaller value of input force (as compared to the output force), the system has some definite non-zero SE. As input force increases, it helps the system to return to its initial configuration, and hence SE decreases till it reaches a minimum. When SE is at its minimum, the system is very close to its initial undeformed configuration. As we further increase the input force, the system goes past its original configuration and hence, SE increases. Although the phenomenon described here involves the release of strain energy that is stored prior to the usage of the compliant mechanism, this method can be used for a compliant mechanism to obtain higher MA than the rigid-body counterpart.

Thus, we demonstrated that compliant component of MA (MA_c) can be positive thereby improving MA beyond rigid-body counterpart (MA_r). However, MA can also be affected by workpiece or external stiffness as in Eq. (2) Therefore, designing a compliant mechanism that exhibits little or no sensitivity to the external stiffness is discussed next. This is done with the help of a spring-lever model introduced in [4].

3. MECHANICAL ADVANTAGE AND SPRING-LEVER MODEL

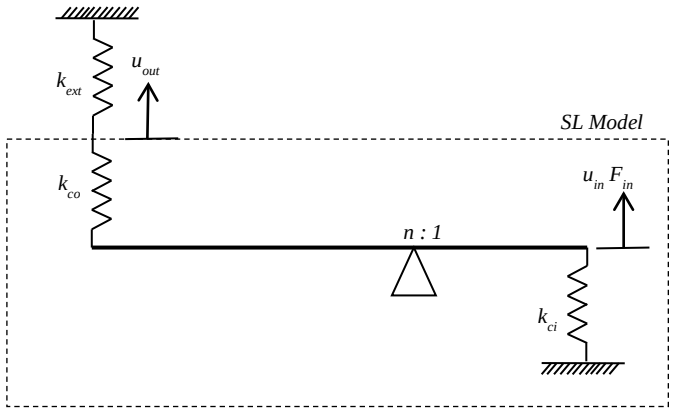


Figure 15 A schematic of spring-lever model for a compliant mechanism

When a compliant mechanism operates on a workpiece, the output load comes from the workpiece itself. If we assume that output stiffness is linear, i.e. $k_{ext}u_{out}$ is the output force, then, there is no additional output force (F_{out}). Hence, we take $F_{out} = 0$ as shown in Figure 15.

The potential energy of the SL model in Figure 15 is given by

$$PE = \frac{1}{2}k_{ci}u_{in}^2 + \frac{1}{2}k_{co}\left(u_{out} - nu_{in}\right)^2 + \frac{1}{2}k_{ext}u_{out}^2 - F_{in}u_{out} \tag{14}$$

The static force equilibrium equations of the SL model are given by

$$\left\{\begin{array}{l} \dfrac{\partial PE}{\partial u_{in}} = k_{ci}u_{in} - nk_{co}\left(u_{out} - nu_{in}\right) - F_{in} = 0 \\ \dfrac{\partial PE}{\partial u_{out}} = k_{co}\left(u_{out} - nu_{in}\right) + k_{ext}u_{out} = 0 \end{array}\right\} \tag{15}$$

The mechanical advantage (MA) of the SL model is given by

$$MA = \frac{k_{ext}u_{out}}{F_{in}} \tag{16}$$

Solving for u_{out} and F_{in} by using Eq. (15), we get

$$\left\{\begin{array}{l} u_{out} = \dfrac{nk_{co}u_{in}}{k_{co} + k_{ext}} \\ F_{in} = k_{ci} + \dfrac{n^2k_{co}k_{ext}u_{in}}{k_{co} + k_{ext}} \end{array}\right\} \tag{17}$$

Substituting the expressions from Eq. (16) in Eq. (17) and solving, we get MA as

$$MA = \frac{nk_{co}k_{ext}}{k_{ci}k_{co} + \left(k_{ci} + n^2k_{co}\right)k_{ext}} \tag{18}$$

According to Salamon and Midha [2], the MA of a compliant mechanism can be expressed in the form

$$MA = MA_s\left(\frac{k_{ext}}{s_k + k_{ext}}\right) \tag{19}$$

where MA_s is the bounding value for MA and s_k represents the sensitivity (index) of the mechanism with respect to external stiffness.

Comparing the two forms of MA in Eq. (18) and Eq. (19), we get the expressions for MA_s and s_k as follows:

$$\left\{\begin{array}{l} MA_s = \dfrac{nk_{co}}{k_{ci} + n^2k_{co}} \\ s_k = \dfrac{k_{ci}k_{co}}{k_{ci} + n^2k_{co}} \end{array}\right\} \tag{20}$$

It can be observed that MA_s and s_k are related as

$$MA_s = \frac{n}{k_{ci}}s_k \tag{21}$$

Figure 16 shows the comparison of MA obtained from expressions in Eq. (16) (curve MA) and Eq. (18) (curve MA_2) for the input force value of 10 N and as can be observed, the two curves coincide. It is because the spring-lever model accurately describes the displacements at the input and the output ports as discussed in [4], and therefore, the expressions for MA in Eq. (16) and Eq. (18) are equivalent.

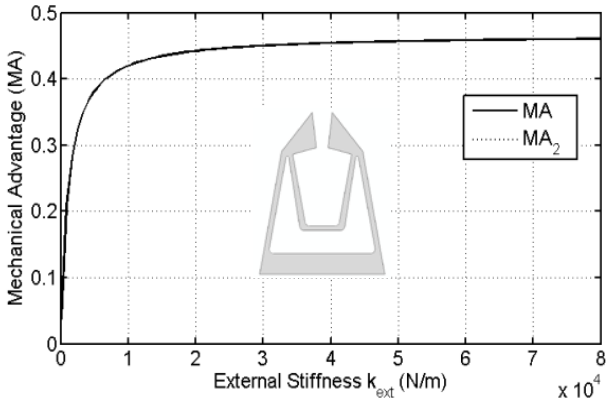
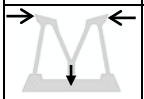
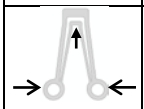
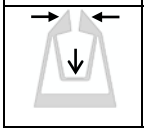


Figure 16 Mechanical advantage vs external stiffness

Table 4 represents the SL-parameter values for the given compliant mechanisms. We can observe from Table 4 that although the second mechanism has a high value for MA_s but also has a high value for s_k which means that the mechanism is more sensitive to external stiffness. This juxtaposition leads to trade-off between MA and the sensitivity of the mechanism, which can be posed as an optimization problem.

Table 4 SL-parameter values for different compliant mechanisms

Mechanism	n	k_{ci} **(N/m)**	k_{co} **(N/m)**	MA_s	s_k **(N/m)**
	0.61	344.1	43922.0	1.59	898.4
	0.13	68.09	16580.0	6.17	3224.3
	1.91	4600.4	10462.0	0.47	1124.7

4. OPTIMIZATION

As can be gleaned from general expression of MA in Eq. (2), for rendering a compliant mechanism insensitive to the external stiffness, the value for sensitivity index, s_k, must be minimum. At the same time, it is desirable to have higher MA which in turn, implies that the value of MA_s should be sufficiently high.

Since, both quantities MA_s and s_k are inherently depended on SL-parameters, a simple and direct method for designing a compliant mechanism which exhibits insensitivity to the external stiffness and at the same time has sufficiently high MA may not be possible. However, an optimization problem may be posed to obtain a suitable topology of the mechanism. The optimization problem is posed as

$$\underset{\mathbf{d}}{Min}\left[-MA_s\right]$$

$$\text{Subject to } \begin{Bmatrix} \Lambda_1 : V - V^* \le 0 \\ \Lambda_2 : s_k - s_k^* \le 0 \end{Bmatrix}$$

Data: E V^* s_k^* , $\mathbf{d_l} \le \mathbf{d} \le \mathbf{d_u}$

The optimization problem can also be posed as

$$\underset{\mathbf{d}}{Min}\left[s_k\right]$$

Subject to $\left\{\begin{matrix}\Lambda_1 :\ V - V^* \le 0 \\ \Lambda_2 :\ MA_s^* - MA_s \le 0\end{matrix}\right\}$

Data: $E\ \ V^*\ MA_s^*$, $\mathbf{d_l} \le \mathbf{d} \le \mathbf{d_u}$

Here, $\mathbf{d}$ is the design variable, $\mathbf{d_l}$ and $\mathbf{d_u}$ are the lower and upper bounds respectively for the design variable, E the Young's Modulus, V the volume and the starred quantities are the limiting values. In case of design space consisting of beams, $\mathbf{d}$ is the in-plane width of the beams.

For any optimization problem, the gradients of the objective function and the constraints with respect to the design variable are always important. The derivation of these gradients is described in appendix B.

Some results from *fmincon* optimization routine in Matlab (www.mathworks.com) are shown in Figures 17 to 20.

Please note that in Figures 17 and 19,

i. *A* and *B* represents the input and output port respectively.
ii. All dimensions are in mm.
iii. Arrows indicate the direction of the force at the port.
iv. Initial in-plane width is 2 mm at thick members and 1 mm at thin members.
v. Out-of-plane thickness is 1 mm for all members.
vi. Plot on right in Figures 17 and 19 is the deflection plot while an input force is applied.

It can be observed from Figure 18 that MA_s value for mechanism 1 is high at 2.67 but s_k is also high at 26.4 N/mm. It is because constraint for s_k is not considered for the first optimization problem. However, when constraint for s_k is introduced in second optimization problem in Figure 20, the optimized mechanism 2 has very low s_k value of 0.79 N/mm with MA_s value of 1.85 that satisfies the constraint for MA being greater than 1.4.

Optimization results

Problem 1:

$$\underset{\mathbf{d}}{Min}\left[-MA_s\right]$$

Subject to $V \le V^*$

Data: $E = 1300\, N/mm^2$, $V^* = 1500\ mm^3$

The design space, initial conditions, and the deflection plot of the optimized topology are shown in Figure 17 and the prototype of the full mechanism is shown in Figure 18 along with the iterative history of the objective function.

Problem 2:

$$\underset{\mathbf{d}}{Min}\left[s_k\right]$$

Subject to $\left\{\begin{matrix}-MA_s \le -MA_s^* \\ V \le V^*\end{matrix}\right\}$

Data: $E = 1300\, N/mm^2$, $MA_s^* = 1.4$, $V^* = 1500\ mm^3$

The design space, initial conditions, and the deflection plot of the optimized topology are shown in Figure 19 and the prototype of the full mechanism is shown in Figure 20 along with the iterative history of the objective function.

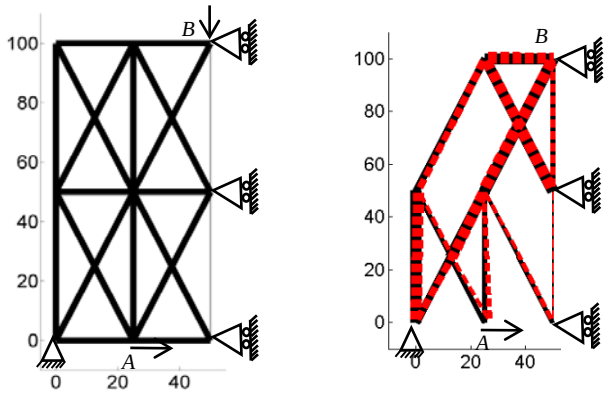


Figure 17 Initial conditions and deflection plot for symmetric half of mechanism 1

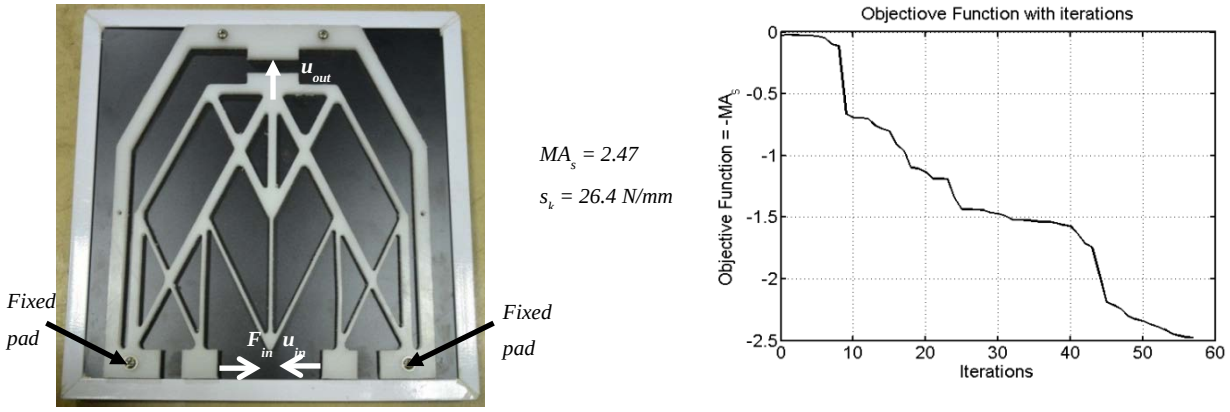


Figure 18 Optimized topology and objective function for mechanism 1

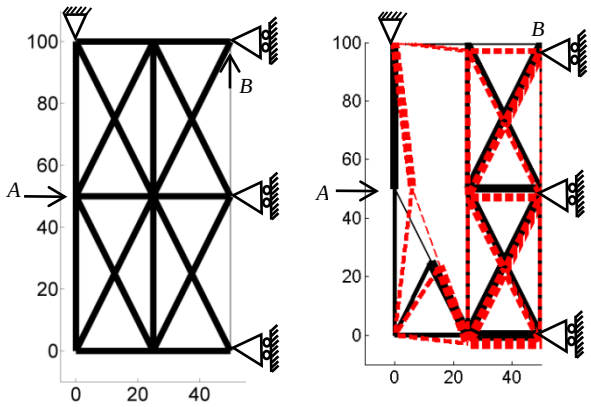


Figure 19 Initial conditions and deflection plot for symmetric half of mechanism 2

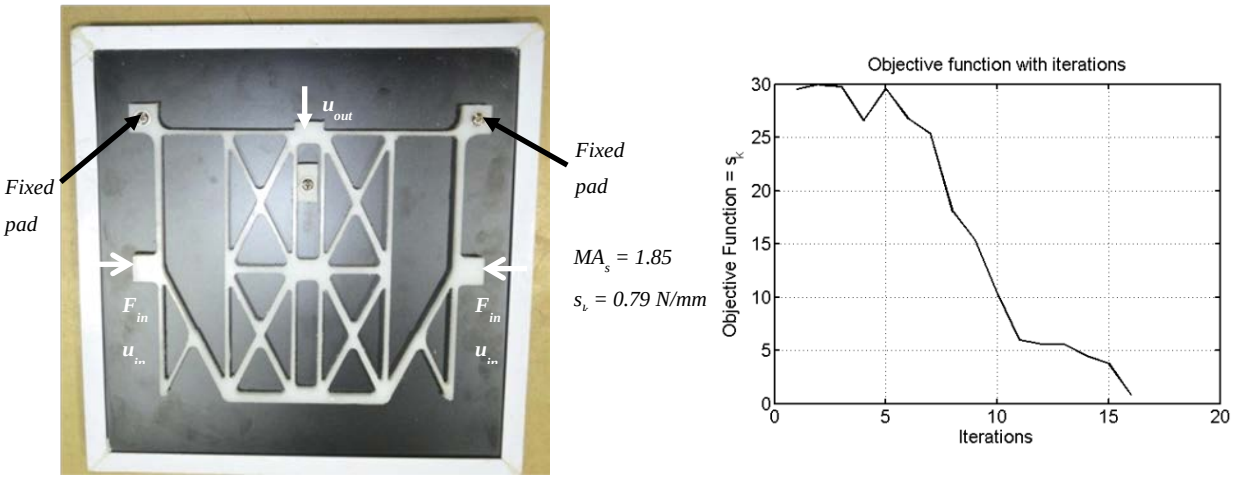


Figure 20 Optimized topology and objective function for mechanism 2

5. NON-DIMENSIONAL MAPS

These maps capture the behavior of MA for a compliant mechanism against non-dimensional parameters as discussed in [9]. These parameters involve forces, material properties, geometric variables and workpiece/external stiffness. The non-dimensional parameters are η and $\widehat{k_{ext}}$ (external stiffness) and are defined as

$$s = \frac{\bar{l}}{\bar{d}},\ \eta_{in} = \frac{F_{in}s^2}{E\bar{b}\bar{d}}, \eta_{out} = \frac{F_{out}s^2}{E\bar{b}\bar{d}},\ \widehat{k_{ext}} = \frac{k_{ext}\bar{l}}{E\bar{b}\bar{d}}$$

where, *s* is the slenderness ratio, *E* the Young's Modulus, *b* the out-of-plane thickness of beam, *d* the in-plane width of beam.

The non-dimensional maps are obtained using large- displacement finite element analysis in Abaqus. Figure 21 shows the plot of MA with non-dimensional parameters η_{in} and η_{out} for constant non-dimensional stiffness $\widehat{k_{ext}}$ and slenderness ratio *s* for different geometric scales of the given compliant mechanism. As can be observed, the curves overlap. This indicates an important property that all non-dimensional curves for MA for a given non-dimensional stiffness ($\widehat{k_{ext}}$) and for a given slenderness ratio (*s*) of a given compliant mechanism are the same. Using this property, the non-dimensional maps for MA of different compliant mechanisms for different *s* values are presented in Figure 22 (a-j).

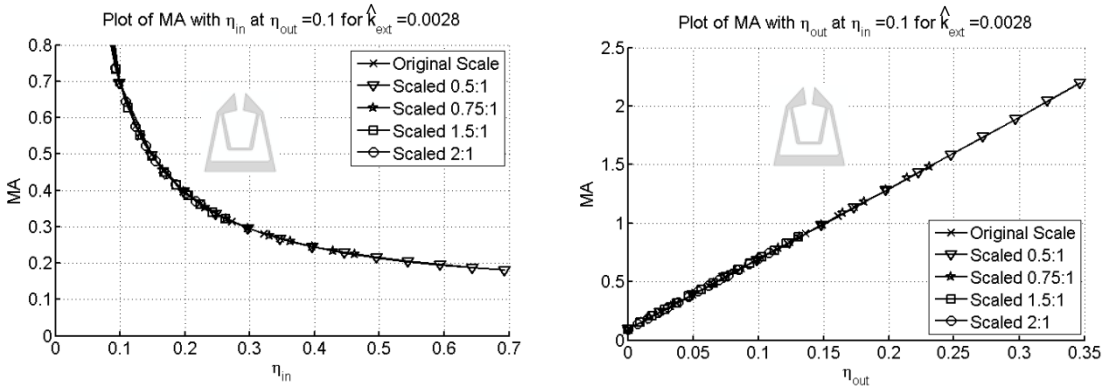
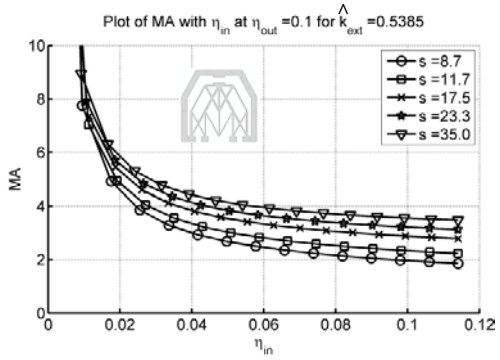


Figure 21 Non-dimensional map of MA for *s* = 14.6

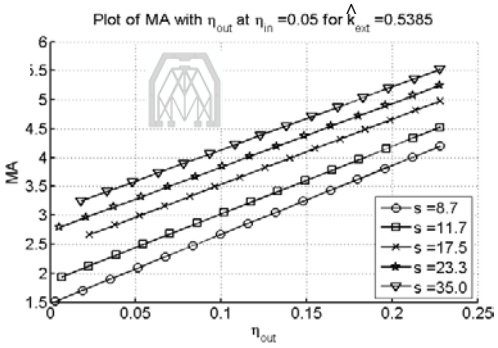
Non-dimensional maps in Figure 22 can be used to compare compliant mechanism of different topologies. One may select a compliant mechanism based on desired MA, input, output force and then choose a suitable slenderness ratio and decide on the geometric dimensions. One can also select a mechanism based on the slope of the curves in Figure 22 in such a way that a small change in the input and output force (or η_{in} and η_{out}) may not affect the MA considerably. The value of MA for those slenderness ratios not included in Figure 22 may be appropriately interpolated from the given slenderness ratio. Therefore, if the curves for different slenderness ratios are close to each other such that the MA is not affected greatly (as in Figure 22.i), one may infer that the MA of compliant mechanism will not be affected noticeably regardless of the geometric dimensions.

It can be observed in Figure 22 that all the plots for MA with η_{in} have a lower bound. This lower bound represents the minimum MA obtainable from the compliant mechanism and may serve as a useful criterion for comparison and selection of the compliant mechanism.

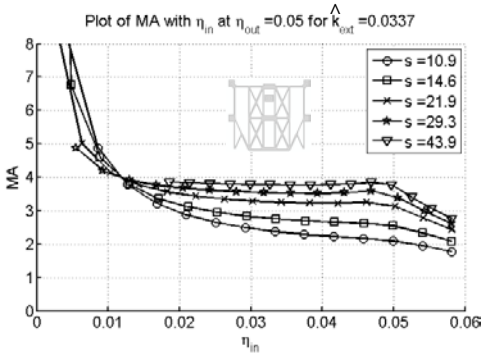
a



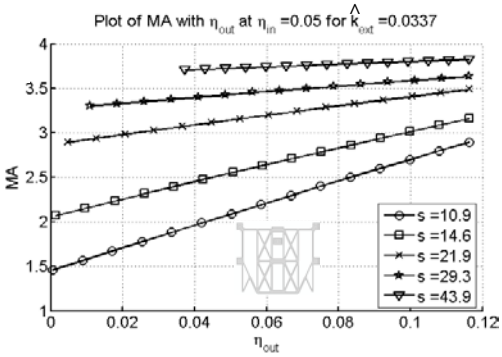
b



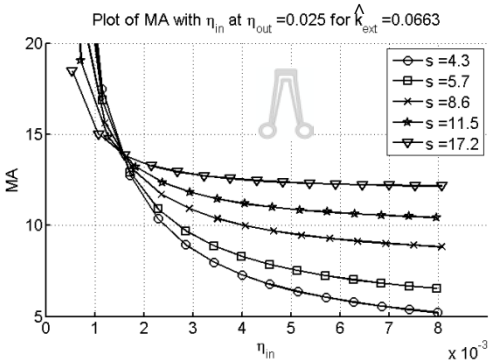
c



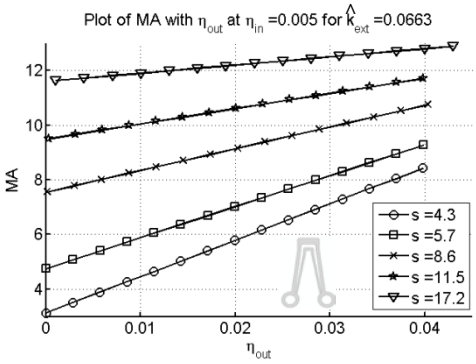
d



e



f



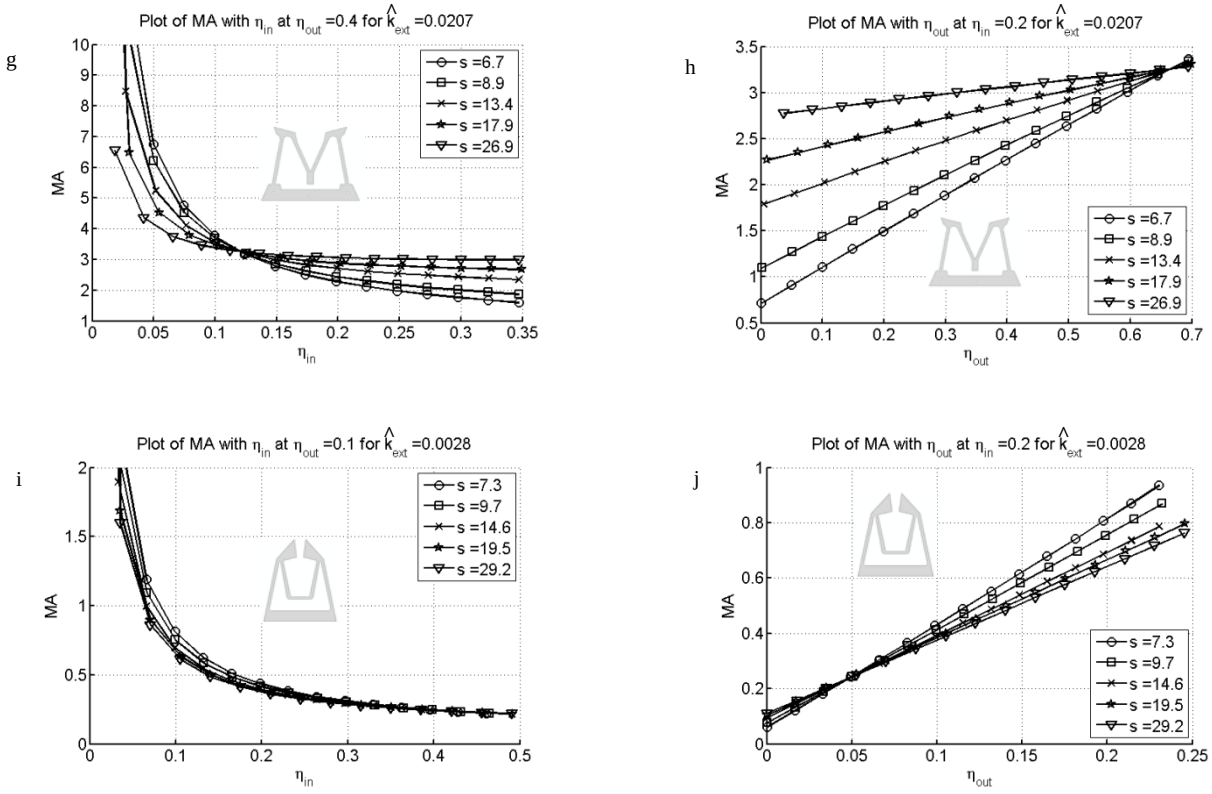


Figure 22 (a-j) Plots of MA for different slenderness ratio for constant non-dimensional $\widehat{k_{ext}}$

6. DISCUSSIONS AND CONCLUSIONS

A practical example of compliant mechanism is presented where there is positive contribution from the compliant component of MA, thereby improving the MA beyond the MA of its rigid-body counterpart.

MA of the compliant mechanism is analyzed using spring-lever (SL) model and analytical expressions for MA_s and s_k as a function of SL-parameters (n, k_{ci} and k_{co}) are derived. Based on these expressions, an optimization problem is posed for improving the MA of a compliant mechanism and two optimized compliant mechanisms are presented. This shows the feasibility of optimization using SL-parameters.

A scope of improvement in optimization always exists. The optimization problems posed are subjected to the connectivity of the beam ground structure. In order to obtain a generalized topology, 2D continuum elements may be used. Furthermore, the design space consisted of beams that may not be slender as approximated for FE runs. In this case, 2D continuum elements may provide more accurate results. The SL-parameters for optimization are obtained using small-displacement finite element analysis. However, if it is required to obtain SL-parameters for instantaneous deformed configuration of the compliant mechanism, large-displacement finite element analysis can be employed. In addition, the optimization algorithm may utilize more efficient objective functions that may produce better results in terms of element connectivity, higher MA_s and lower s_k for the given boundary conditions.

The non-dimensional maps for MA against non-dimensional parameters have been generated. These maps enable the user to obtain a compliant mechanism according to desired MA, input and output forces, external stiffness, geometric and material properties requirements.

7. REFERENCES

[1] Mallik, A. K., Ghosh, A., and Dittrich, G. *Kinematic analysis and synthesis of mechanisms*. CRC Press, 1994.

[2] Salamon, B. A., and A. Midha. "An introduction to mechanical advantage in compliant mechanisms." Journal of Mechanical Design 120.2 (1998): 311-315.

[3] Ananthasuresh, G., and Kumar, G. R. (2013). "A Study of Mechanical Advantage in Compliant Mechanisms." Proceedings of the 1st International and 16th National Conference on Machines and Mechanisms (iNaCoMM2013). IIT Roorkee.

[4] Hegde, Sudarshan, and G. K. Ananthasuresh. "A spring-mass-lever model, stiffness and inertia maps for single-input, single-output compliant mechanisms." Mechanism and Machine Theory 58 (2012): 101-119.

[5] Howell, Larry L. *Compliant mechanisms*. Wiley-Interscience, 2001.

[6] Howell, L. L., A. Midha, and T. W. Norton. "Evaluation of equivalent spring stiffness for use in a pseudo-rigid-body model of large-deflection compliant mechanisms." Journal of Mechanical Design 118 (1996): 126.

[7] Hegde, Sudarshan, and G. K. Ananthasuresh. "Design of single-input-single-output compliant mechanisms for practical applications using selection maps." Journal of Mechanical Design 132.8 (2010): 081007.

[8] Krishnan, Girish, and G. K. Ananthasuresh. "Evaluation and design of displacement-amplifying compliant mechanisms for sensor applications." Journal of Mechanical Design 130.10 (2008): 102304.

[9] Bhargav, Santosh D. B, Harish I. Varma, and G. K. Ananthasuresh. "Non-dimensional Kinetoelastostatic Maps for Compliant Mechanisms." ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, paper No. 12178 DETC. Vol. 2013.

[10] Wang, Michael Yu. "A kinetoelastic formulation of compliant mechanism optimization." Journal of Mechanisms and Robotics 1.2 (2009): 021011.

[11] Hedge, Sudarshan PhD Thesis, Mechanical Engg. IISc, 2013.

[12] Krishnan, Girish PhD Thesis, Mechnanical Engg. IISc, 2006.

[13] Qiu, Jin, Jeffrey H. Lang, and Alexander H. Slocum. "A curved-beam bistable mechanism." Microelectromechanical Systems, Journal of 13.2 (2004): 137-146.

APPENDIX

A. NONLINEAR SL-PARAMETERS

The spring-lever model (SL-model) parameters, i.e., n, k_{ci} and $k_{co,}$ as presented in [4] and [7] are valid only for small displacements of compliant mechanisms. Linear-static finite element analysis (FEA) is performed for obtaining SL-model parameters that are found to be constants. However, when the displacements of the compliant mechanism become large, the values of SL-model parameters may not remain constants. In such cases, nonlinear-static FEA (for large displacements) has to be performed for SL-model parameters, while still assuming the material in linear and elastic regime.

Finite-difference method is employed for obtaining SL-parameter values (n, k_{ci} and k_{co}) for a given input and output force (at instantaneous deformed configuration) as follows.

i. The first FEA is conducted for a given value of input and output forces applied on the compliant mechanism.

ii. The second FEA is conducted with a small increment in the input force, $F_{in,}$ applied while keeping the output force as constant. The small increment is kept within 1% of the actual value.

iii. The spring-lever ratio (n) and input-side stiffness (k_{ci}) are calculated as follows

$$n = \frac{\Delta u_{out2}}{\Delta u_{in2}} \tag{A.1}$$

$$k_{ci} = \frac{\Delta F_{in2}}{\Delta u_{in2}} \tag{A.2}$$

where, the subscript 2 refers to small increment in the input force as described in (ii). The quantities denoted by Δ in Eq. (A.1) and Eq. (A.2) refers to finite difference values from the first and second FEA.

iv. The third FEA is conducted with a small increment in output force, F_{out}, while keeping the input force as constant. Again, the small increment is kept within 1% of the actual value.

The output-side stiffness (k_{co}) is calculated as follows

$$k_{co} = \frac{\Delta F_{out3}}{n\Delta u_{in3} - \Delta u_{out3}} \tag{A.3}$$

where, the subscript 3 refers to small increment in the input force as described in (iv). The quantities denoted by Δ in Eq. (A.3) refer to finite difference values from the first and third FEA.

It should be noted that the external spring (k_{ext}) is not present while deriving SL-model parameter values as they represent the intrinsic properties of the corresponding compliant mechanism.

The plots for n, k_{ci} and k_{co} values for given slenderness ratio (s) of different compliant mechanism are given next.

1. Compliant Mechanism 1

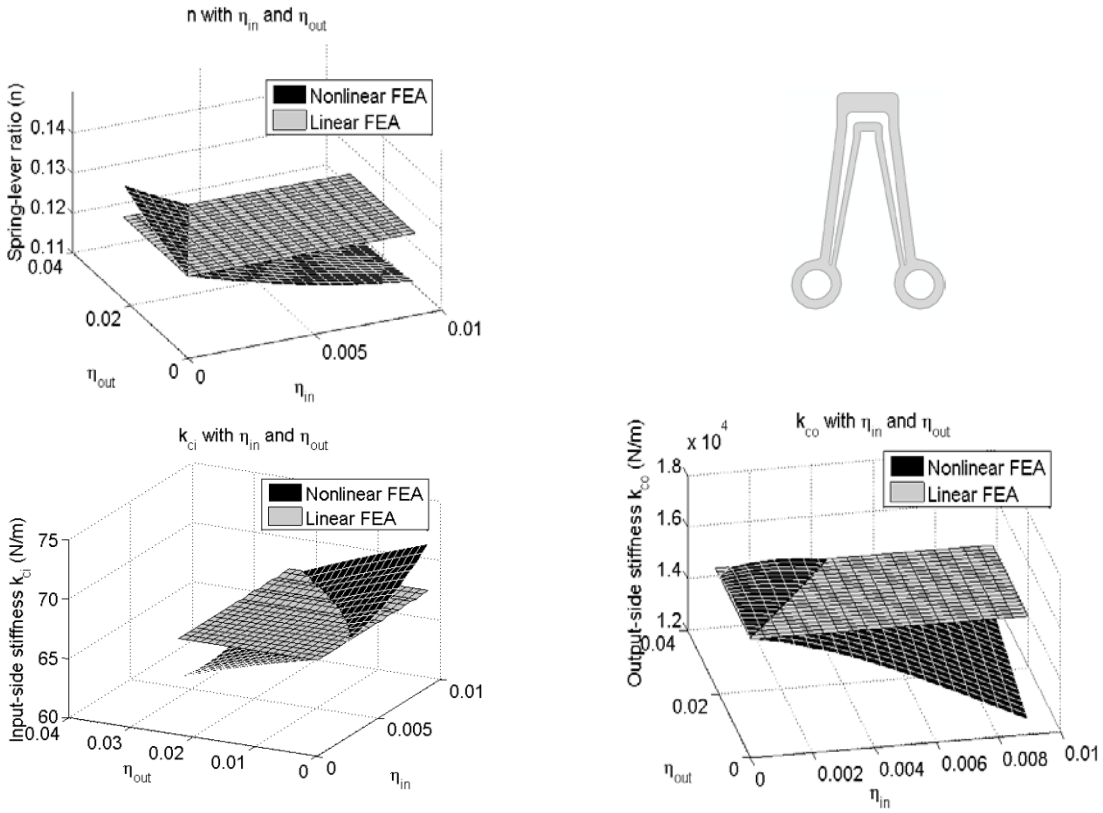


Figure 23 3-D plots of n, k_{ci} and k_{co} for given mechanism for s = 8.6

As in most of the practical applications, the force applied at the output port (F_{out}) is zero. Below are the 2-D plots when F_{out} or η_{out} is zero.

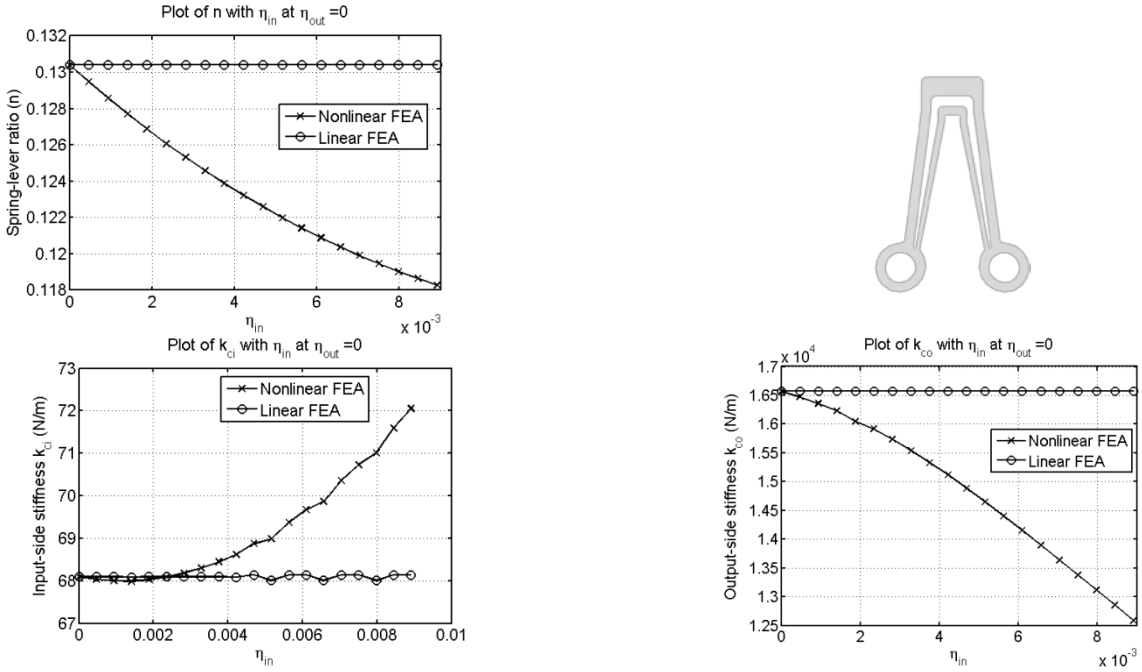


Figure 24 2-D plots of n, k_{ci} and k_{co} when η_{out} is zero for given mechanism for s = 8.6

2. Compliant Mechanism 2

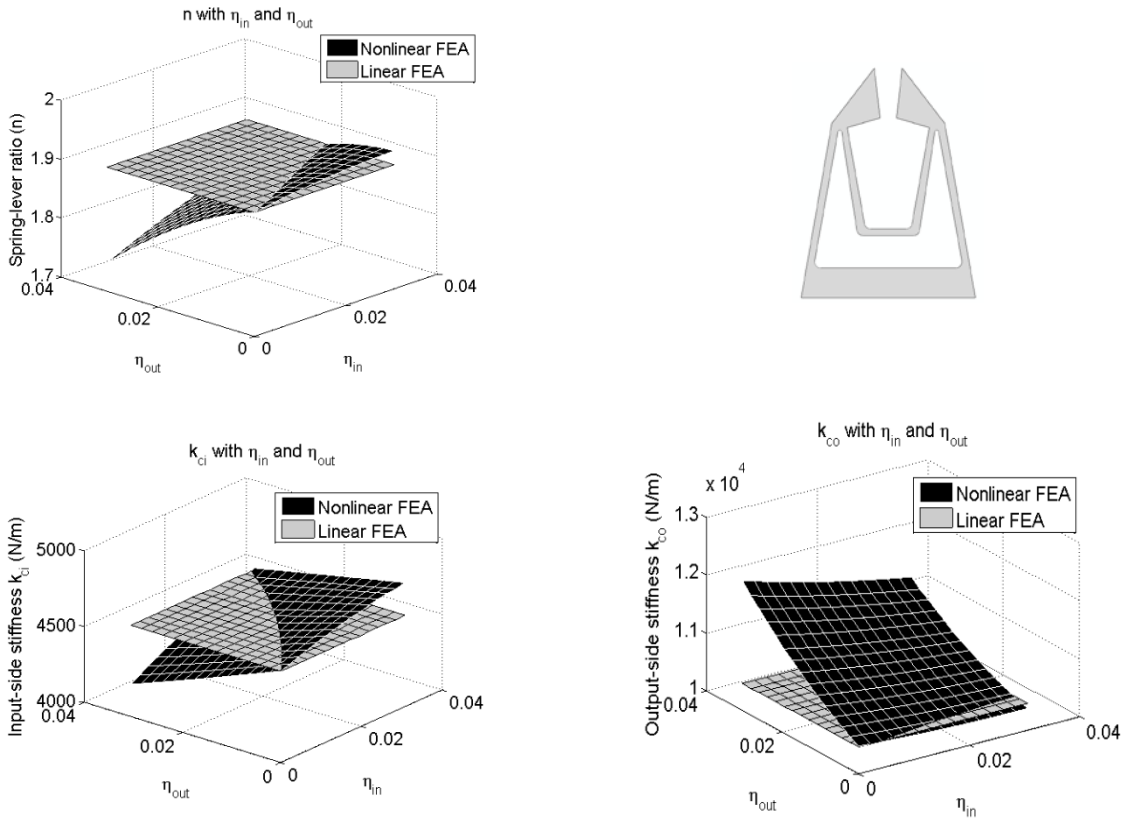


Figure 25 3-D plots of n, k_{ci} and k_{co} for given mechanism for s = 14.6

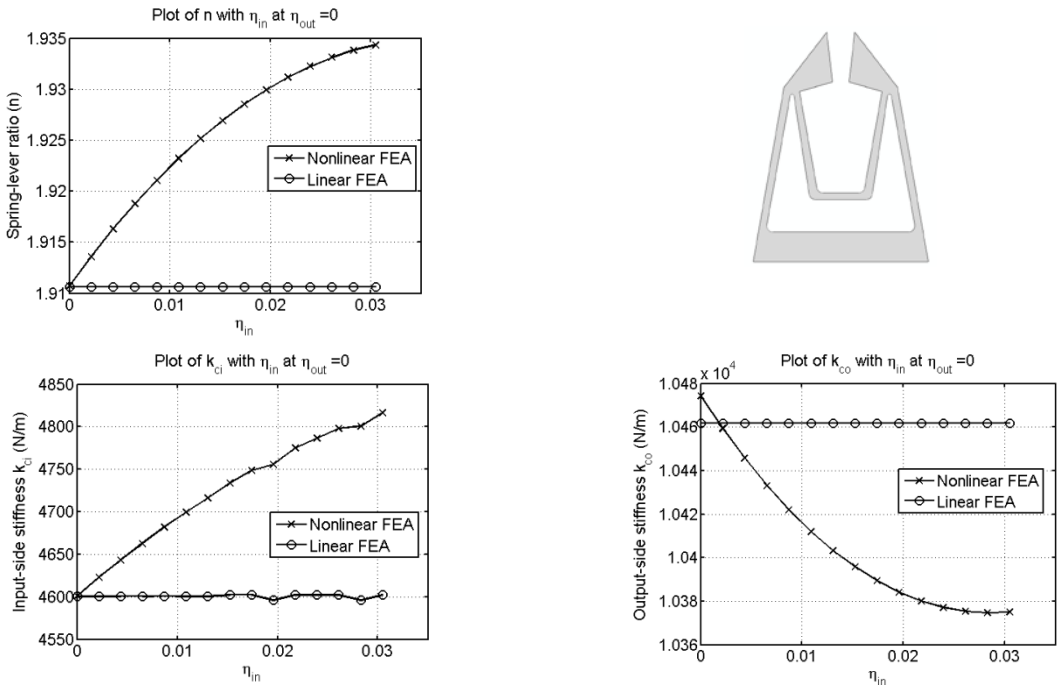


Figure 26 2-D plots of n, k_{ci} and k_{co} when η_{out} is zero for given mechanism for s = 14.6

3. Compliant Mechanism 3

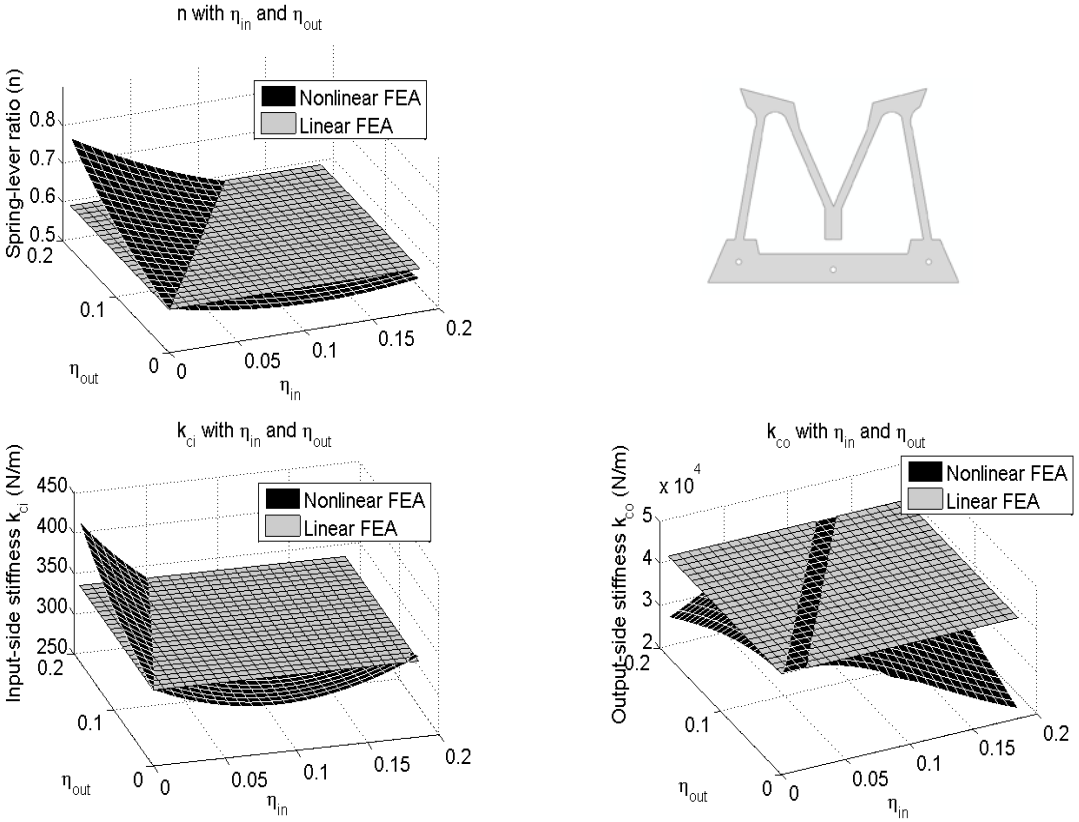


Figure 27 3-D plots of n, k_{ci} and k_{co} for given mechanism for s = 13.4

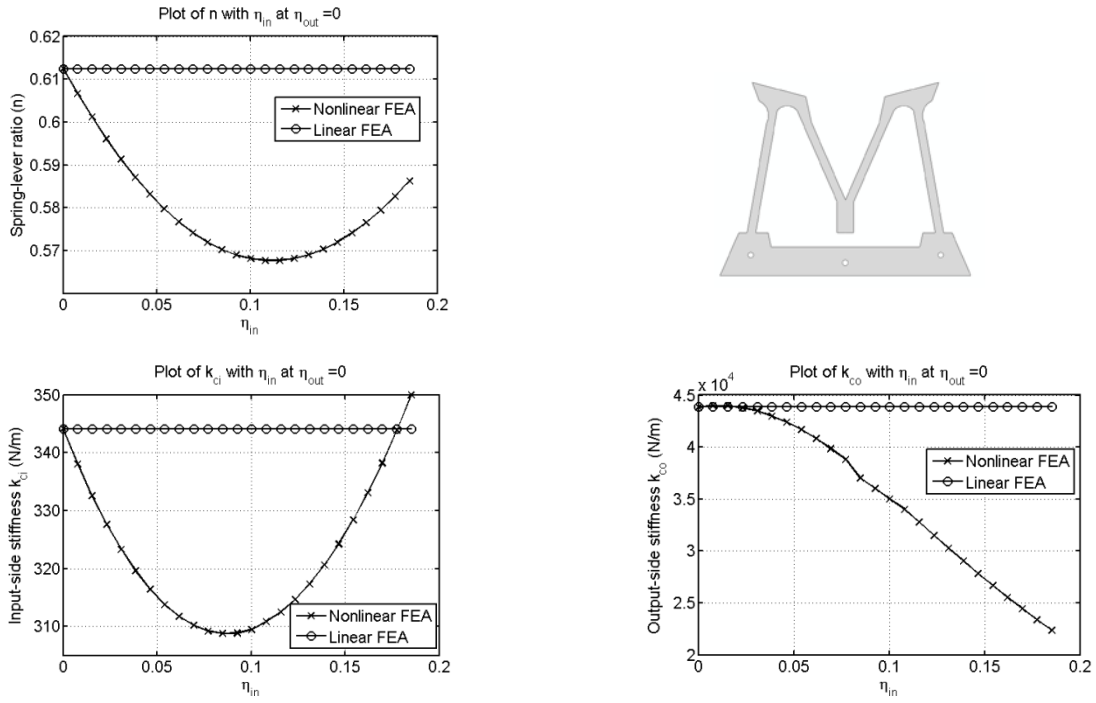


Figure 28 2-D plots of n, k_{ci} and k_{co} when η_{out} is zero for given mechanism for s = 13.4

4. Compliant mechanism 4

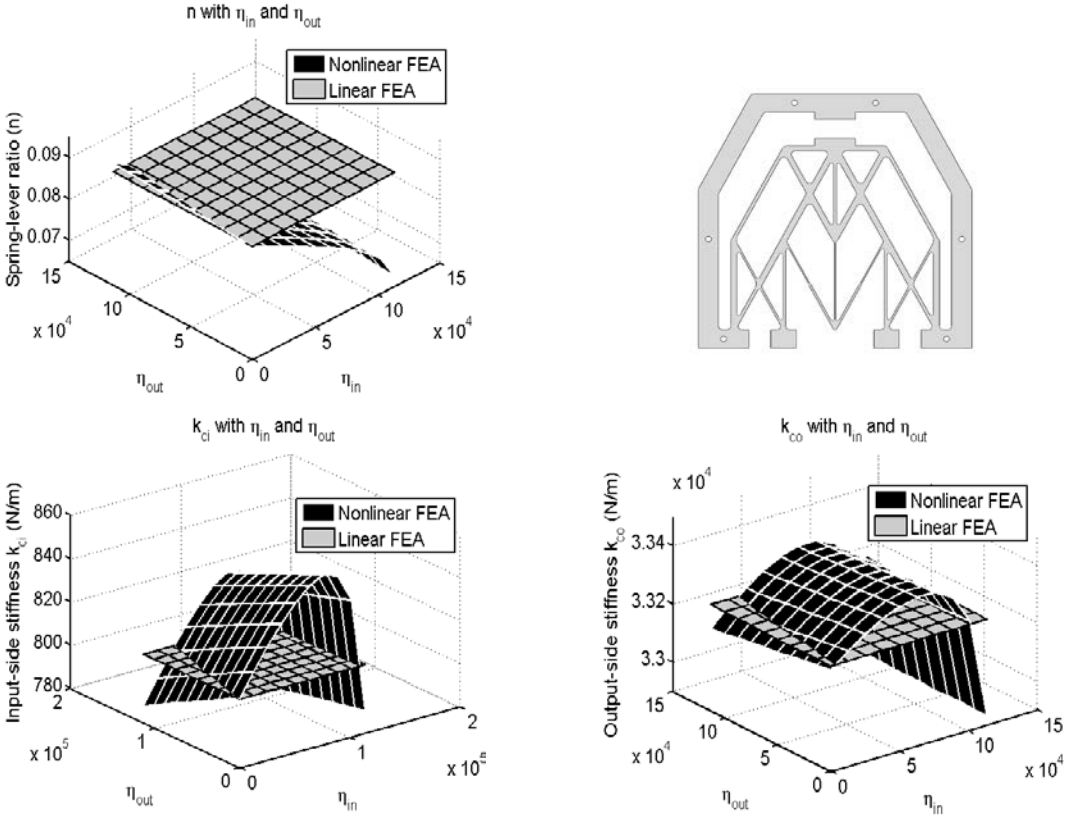


Figure 29 3-D plots of n, k_{ci} and k_{co} for given mechanism for s = 17.5

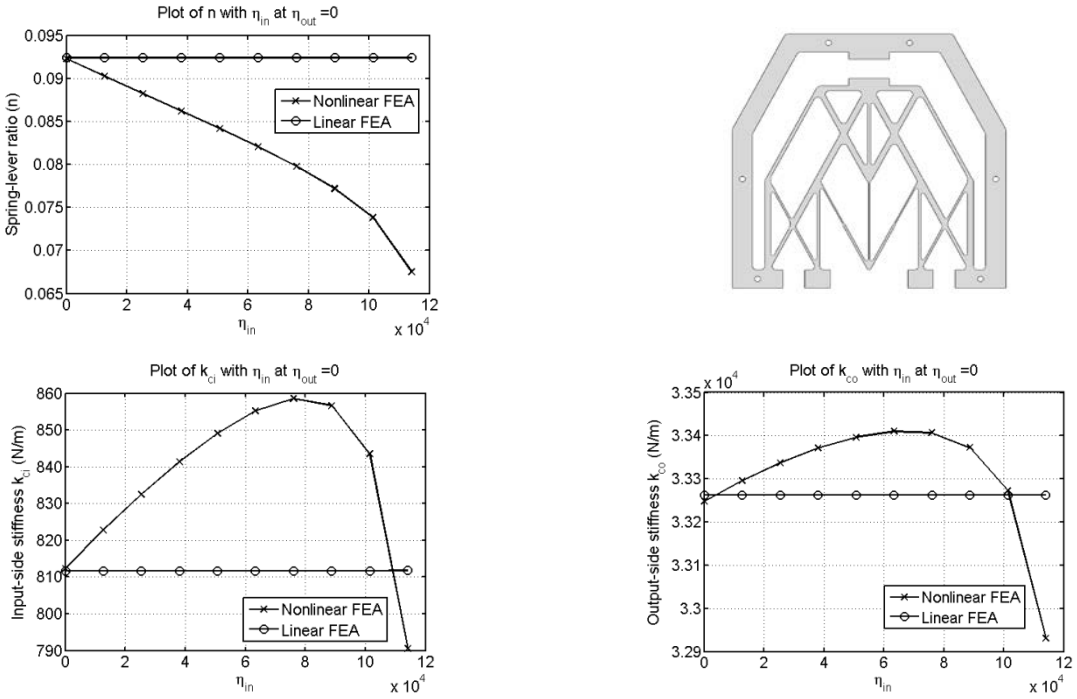


Figure 30 2-D plots of n, k_{ci} and k_{co} when η_{out} is zero for given mechanism for s = 17.5

5. Compliant mechanism 5

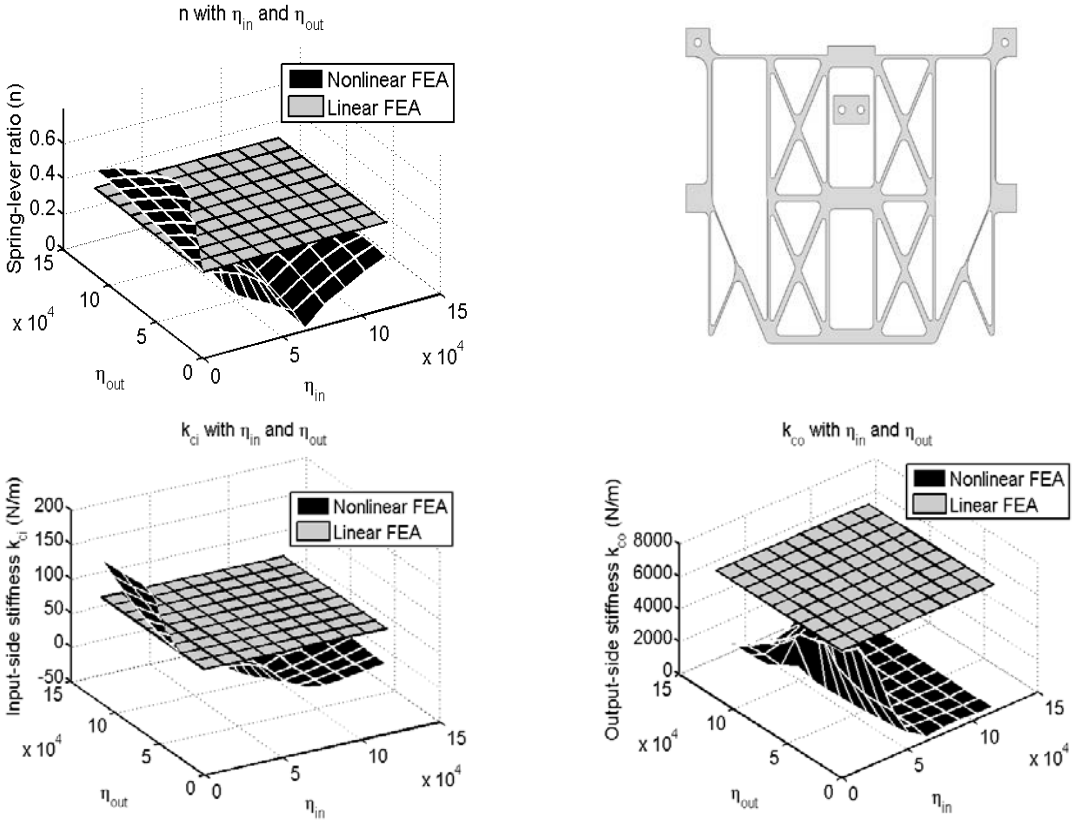


Figure 31 3-D plots of n, k_{ci} and k_{co} for given mechanism for s = 21.9

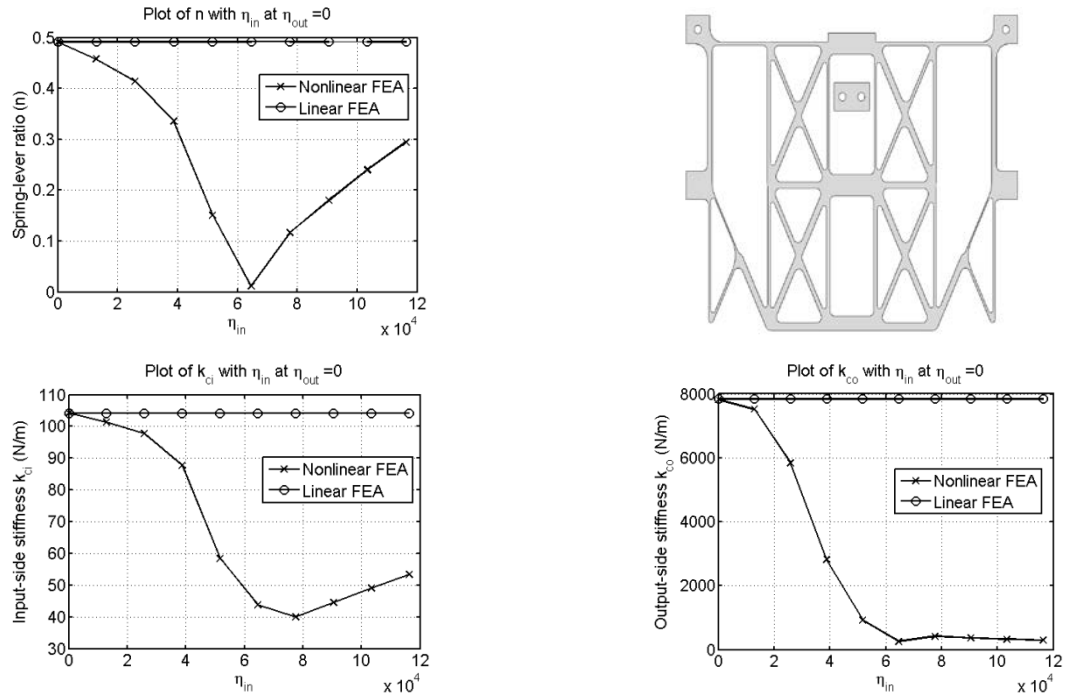


Figure 32 2-D plots of n, k_{ci} and k_{co} when η_{out} is zero for given mechanism for s = 21.9

B. GRADIENTS OF THE OBJECTIVE FUNCTION AND CONSTRAINTS OF OPTIMIZATION PROBLEM

It can be observed from Eq. (20), MA_s and s_k are functions of SL-parameters (n, k_{ci} and k_{co}) that are inherent functions of the input and output port displacement for the two linear-static FE simulations as described in [4]. Thus, MA_s and s_k are also functions of the input and output port displacement. Therefore,

$$\left\{\begin{array}{l} MA_s = MA_s\left(u_{in1}, u_{out1}, u_{in2}, u_{out2}\right) \\ s_k = s_k\left(u_{in1}, u_{out1}, u_{in2}, u_{out2}\right) \end{array}\right\} \tag{B.1}$$

Here, u_{in} and u_{out} are the input and output port displacement respectively and subscript 1 and 2 refer to linear-static FE simulation discussed in [4]. Gradient of Eq. (B.1) can be written using the adjoint method as follows.

MA_s can be written as

$$MA_s = MA_s + \lambda_1^T\left(Ku_1 - F_1\right) + \lambda_2^T\left(Ku_2 - F_2\right) \tag{B.2}$$

λ_1 and λ_2 are constant arrays of length N_{ELEM} (number of design variables for optimization). The quantities within the brackets are the equilibrium equations for first and second linear-static FE simulations.

Differentiate Eq. (B.2) with respect to i^{th} design variable, we get

$$\frac{\partial MA_s}{\partial \mathbf{d}_i} = \left[\frac{\partial MAs}{\partial u_1} + \lambda_1^T K\right]\left(\frac{\partial u_1}{\partial \mathbf{d}_i}\right) + \left[\frac{\partial MAs}{\partial u_2} + \lambda_2^T K\right]\left(\frac{\partial u_2}{\partial \mathbf{d}_i}\right) + \lambda_1^T\left[\frac{\partial K}{\partial \mathbf{d}_i}u_1 - \cancelto{0}{\frac{\partial F_1}{\partial \mathbf{d}_i}}\right] + \lambda_2^T\left[\frac{\partial K}{\partial \mathbf{d}_i}u_2 - \cancelto{0}{\frac{\partial F_2}{\partial \mathbf{d}_i}}\right] \tag{B.3}$$

Therefore,

$$\begin{array}{l} \dfrac{\partial MAs}{\partial u_1} + K\lambda_1 = 0 \Rightarrow \lambda_1 = -K^{-1}\left(\dfrac{\partial MA_s}{\partial u_1}\right) \\ \dfrac{\partial MAs}{\partial u_2} + K\lambda_2 = 0 \Rightarrow \lambda_2 = -K^{-1}\left(\dfrac{\partial MA_s}{\partial u_2}\right) \end{array} \tag{B.4}$$

Thus, Eq. (B.3)becomes

$$\frac{\partial MA_s}{\partial \mathbf{d}_i} = \lambda_1^T\left(\frac{\partial K}{\partial \mathbf{d}_i}u_1\right) + \lambda_2^T\left(\frac{\partial K}{\partial \mathbf{d}_i}u_2\right) \tag{B.5}$$

The term $\left(\partial MA_s/\partial u_1\right)$ and $\left(\partial MA_s/\partial u_2\right)$ in Eq. (B.4) or similar expression for $\left(\partial s_k/\partial u_1\right)$ and $\left(\partial s_k/\partial u_2\right)$ may be calculated as described below.

The SL-parameters are functions of input and output port displacement as

$$\left\{\begin{array}{l} n = \dfrac{u_{out1}}{u_{in1}} \\ k_{ci} = \dfrac{F_{in}}{u_{in1}} \\ k_{co} = \dfrac{F_{out}u_{in1}}{u_{in1}u_{out2} - u_{in2}u_{out1}} \end{array}\right\} \tag{B.6}$$

Substitute Eq. (B.6) in Eq. (20) and simplifying, we get

$$\left\{\begin{array}{l} MA_s = \dfrac{F_{out}u_{in1}u_{out1}}{F_{in}u_{in1}u_{out2} - F_{in}u_{in2}u_{out1} + F_{out}u_{out1}^2} \\ s_k = \dfrac{F_{in}F_{out}u_{in1}}{F_{in}u_{in1}u_{out2} - F_{in}u_{in2}u_{out1} + F_{out}u_{out1}^2} \end{array}\right\} \tag{B.7}$$

The derivatives of MA_s and s_k with respect to u_{in1}, u_{out1}, u_{in2}, u_{out2} can be obtained from Eq. (B.7) as

$$\left\{\begin{array}{l} \dfrac{\partial MA_s}{\partial u_{in1}} = \dfrac{F_{out}u_{out1}^2\left(-F_{in}u_{in2} + F_{out}u_{out1}\right)}{\left(F_{in}u_{in1}u_{out2} - F_{in}u_{in2}u_{out1} + F_{out}u_{out1}^2\right)^2} \\ \dfrac{\partial MA_s}{\partial u_{out1}} = \dfrac{-F_{out}u_{in1}\left(F_{out}u_{out1}^2 - F_{in}u_{in1}u_{out2}\right)}{\left(F_{in}u_{in1}u_{out2} - F_{in}u_{in2}u_{out1} + F_{out}u_{out1}^2\right)^2} \\ \dfrac{\partial MA_s}{\partial u_{in2}} = \dfrac{F_{in}F_{out}u_{in1}u_{out1}^2}{\left(F_{in}u_{in1}u_{out2} - F_{in}u_{in2}u_{out1} + F_{out}u_{out1}^2\right)^2} \\ \dfrac{\partial MA_s}{\partial u_{out2}} = \dfrac{F_{in}F_{out}u_{in1}^2u_{out1}}{\left(F_{in}u_{in1}u_{out2} - F_{in}u_{in2}u_{out1} + F_{out}u_{out1}^2\right)^2} \end{array}\right\} \tag{B.8}$$

$$\left\{\begin{array}{l} \dfrac{\partial s_k}{\partial u_{in1}} = \dfrac{F_{in}F_{out}u_{out1}\left(-F_{in}u_{in2} + F_{out}u_{out1}\right)}{\left(F_{in}u_{in1}u_{out2} - F_{in}u_{in2}u_{out1} + F_{out}u_{out1}^2\right)^2} \\ \dfrac{\partial s_k}{\partial u_{out1}} = \dfrac{F_{in}F_{out}u_{in1}\left(F_{in}u_{in2} - 2F_{out}u_{out1}\right)}{\left(F_{in}u_{in1}u_{out2} - F_{in}u_{in2}u_{out1} + F_{out}u_{out1}^2\right)^2} \\ \dfrac{\partial s_k}{\partial u_{in2}} = \dfrac{F_{in}^2F_{out}u_{in1}u_{out1}}{\left(F_{in}u_{in1}u_{out2} - F_{in}u_{in2}u_{out1} + F_{out}u_{out1}^2\right)^2} \\ \dfrac{\partial s_k}{\partial u_{out2}} = \dfrac{-F_{in}^2F_{out}u_{in1}^2}{\left(F_{in}u_{in1}u_{out2} - F_{in}u_{in2}u_{out1} + F_{out}u_{out1}^2\right)^2} \end{array}\right\} \tag{B.9}$$

Thus the terms, $\left(\partial MA_s/\partial u_1\right)$, $\left(\partial MA_s/\partial u_2\right)$, $\left(\partial s_k/\partial u_1\right)$ and $\left(\partial s_k/\partial u_2\right)$ can be calculated for appropriate input and output degree of freedom from Eq. (B.8) and Eq. (B.9).

After finding the gradient for MA_s and s_k, an attempt was made for obtaining global Lagrange multipliers, Λ_1 and Λ_2, using Newton-Raphson (NR) method as follows.

Problem statement: $\underset{\mathbf{d}}{Min}\left[-MA_s\right]$

Subject to, $\left\{\begin{array}{l} \Lambda_1 :\ V - V^* \le 0 \\ \Lambda_2 :\ s_k - s_k^* \le 0 \end{array}\right\}$

Data: $E\ \ V^*\ MA_s^*$, $\mathbf{d_l} \le \mathbf{d} \le \mathbf{d_u}$

Lagrangian, $$\begin{array}{l} L = -MA_s + \Lambda_1\left(V - V^*\right) + \Lambda_2\left(s_k - s_k^*\right) \\ \therefore \nabla_{\mathbf{d}}L = -\nabla_{\mathbf{d}}MA_s + \Lambda_1\nabla_{\mathbf{d}}V + \Lambda_2\nabla_{\mathbf{d}}s_k = 0 \end{array} \tag{B.10}$$

$$\mathbf{d}^{(k+1)} = \mathbf{d}^{(k)} + \left[-\nabla_d MA_s + \Lambda_1\nabla_{\mathbf{d}}V + \Lambda_2\nabla_{\mathbf{d}}s_k\right] \tag{B.11}$$

Eq. (B.11) is the optimality criteria method for update of design variable $\mathbf{d}$ for $\left(k+1\right)^{th}$ iteration.

If $\mathbf{d}_i$ and l_i represents in-plane width and length of i^{th} beam element respectively and $V = \sum_{i=1}^{N_{ELEM}} \mathbf{d}_i l_i \Rightarrow \dfrac{\partial V}{\partial \mathbf{d}_i} = l_i$

From Eq.(B.11), we can get the following expressions for every i^{th} element,

$\frac{\partial \mathbf{d}_i}{\partial \Lambda_1} = \left(\nabla_{\mathbf{d}} V\right)_i = l_i$ and $\frac{\partial \mathbf{d}_i}{\partial \Lambda_2} = \left(\nabla_{\mathbf{d}} s_k\right)_i = \frac{\partial s_k}{\partial \mathbf{d}_i}$. Here $\partial s_k / \partial d_i$ can be obtained from similar expressions for s_k as in Eq. (B.4) and Eq. (B.5).

Let $\left\{ \begin{array}{l} \Lambda = \begin{bmatrix} \Lambda_1 \\ \Lambda_2 \end{bmatrix} \\ f_1(\Lambda) = V - V^* = 0 \\ f_2(\Lambda) = s_k - s_k^* = 0 \end{array} \right\}$ and $\Lambda^{(k+1)} = \Lambda^{(k)} - \left[J_f\left(\Lambda^{(k)}\right) \right]^{-1} f\left(\Lambda^{(k)}\right)$

Here J_f is the Jacobian defined as

$$J_f = \begin{bmatrix} \frac{\partial V}{\partial \Lambda_1} & \frac{\partial V}{\partial \Lambda_2} \\ \frac{\partial s_k}{\partial \Lambda_1} & \frac{\partial s_k}{\partial \Lambda_2} \end{bmatrix} \tag{B.12}$$

Therefore,

1. $\frac{\partial V}{\partial \Lambda_1} = \sum_{i=1}^{N_{ELEM}} \frac{\partial V}{\partial \mathbf{d}_i} \frac{\partial \mathbf{d}_i}{\partial \Lambda_1} = \sum_{i=1}^{N_{ELEM}} l_i l_i = \sum_{i=1}^{N_{ELEM}} (l_i)^2$

2. $\frac{\partial V}{\partial \Lambda_2} = \sum_{i=1}^{N_{ELEM}} \frac{\partial \mathbf{d}_i}{\partial \Lambda_2} l_i = \sum_{i=1}^{N_{ELEM}} \frac{\partial s_k}{\partial \mathbf{d}_i} l_i$

3. $\frac{\partial s_k}{\partial \Lambda_1} = \sum_{i=1}^{N_{ELEM}} \frac{\partial s_k}{\partial \mathbf{d}_i} \frac{\partial \mathbf{d}_i}{\partial \Lambda_1} = \sum_{i=1}^{N_{ELEM}} \frac{\partial s_k}{\partial \mathbf{d}_i} l_i$

4. $\frac{\partial s_k}{\partial \Lambda_2} = \sum_{i=1}^{N_{ELEM}} \frac{\partial s_k}{\partial \mathbf{d}_i} \frac{\partial \mathbf{d}_i}{\partial \Lambda_2} = \sum_{i=1}^{N_{ELEM}} \frac{\partial s_k}{\partial \mathbf{d}_i} \frac{\partial s_k}{\partial \mathbf{d}_i} = \sum_{i=1}^{N_{ELEM}} \left(\frac{\partial s_k}{\partial \mathbf{d}_i} \right)^2$

Therefore, $J_f = \begin{bmatrix} \sum_{i=1}^{N_{ELEM}} (l_i)^2 & \sum_{i=1}^{N_{ELEM}} \frac{\partial s_k}{\partial \mathbf{d}_i} l_i \\ \sum_{i=1}^{N_{ELEM}} \frac{\partial s_k}{\partial \mathbf{d}_i} l_i & \sum_{i=1}^{N_{ELEM}} \left(\frac{\partial s_k}{\partial \mathbf{d}_i} \right)^2 \end{bmatrix}$ (B.13)

However, it was observed that the NR method could converge only for 10-12 optimization iterations. Hence, Matlab optimization toolbox function *fmincon* was employed for the optimization problem.

C. MATLAB CODES

a. Matlab code for the pseudo-rigid-body model

```
clc;
clear;
close all;
format long;
S.alpha1 = 5;     % Torsional Stiffness for spring 1
S.alpha2 = 8;     % Torsional Stiffness for spring 1
S.K_in = 0;       % Input Stiffness
S.theta0 = 55*pi/180;              % Initial angle
S.l = 0.25;       % Length of Link
K_out = 0;        % Workpiece Stiffness
n = 1000;
F_in = linspace(2,20,n)';        % Input Force
F_out = 27*ones(n,1);
MA_r = zeros(n,1);              % Rigid Mechanical Advantage
MA_c = zeros(n,1);              % Compliant Mechanical Advantage
MA = zeros(n,1);                % Total Mechanical Advantage
MA_2 = zeros(n,1);
u_out = zeros(n,1);             % Output Displacement
u_in = zeros(n,1);              % Input Displacement
d_theta = zeros(n,1);           % Change in angle wrt initial angle
SE = zeros(n,1);                % Strain Energy
WP = zeros(n,1);                % Work Potential
WP_in = zeros(n,1);             % Input Work
WP_out = zeros(n,1);            % Output Work
PE = zeros(n,1);                % Potential Energy
PE_2 = zeros(n,1);
d2PE_dtheta2 = zeros(n,1);      % 2nd derivative of PE wrt theta
theta_ans = zeros(n,1);         % Eqilibrium Angle
d_SE = zeros(n,1);              % Change in strain energy
delta_theta = zeros(n,1);       % Small change in angle
MA_c_2 = zeros(n,1);
F_out2 = zeros(n,1);
first = zeros(n,1);
second = zeros(n,1);
third = zeros(n,1);
theta_initial = S.theta0;       % Initial Guess
for i=1:n
   if(F_in(i) == 0.0)
```

```
        MA_c(i) = 0.0;
        MA_r(i) = tan(S.theta0);
        theta_ans(i) = S.theta0;
    else
         theta_ans(i) = NR(S,F_in(i),K_out,F_out(i),theta_initial);
         func_x(i) = func_NR(S,F_in(i),K_out,F_out(i),theta_ans(i));    % Check
        d_theta(i) = theta_ans(i) - S.theta0;
        u_out(i) = S.l*(sin(theta_ans(i)) - sin(S.theta0));
        u_in(i) = S.l*(cos(S.theta0) - cos(theta_ans(i)));
        MA_r(i) = tan(theta_ans(i));
        %% Large Angle
        MA_c(i) = -(S.K_in*S.l*S.l*(cos(S.theta0)-cos(theta_ans(i)))*sin(theta_ans(i))+...
            K_out*S.l*S.l*(sin(theta_ans(i))-sin(S.theta0))*cos(theta_ans(i))+...
            (S.alpha1+S.alpha2)*(theta_ans(i)-S.theta0))/(F_in(i)*S.l*cos(theta_ans(i)));
        MA_2(i) = F_out(i)/F_in(i);
    end
    MA(i) = MA_r(i) + MA_c(i);
    F_out2(i) = MA(i)*F_in(i);
    SE(i) = 0.5*S.K_in*u_in(i)*u_in(i) + ...
        0.5*K_out*u_out(i)*u_out(i) +...
        0.5*S.alpha1*d_theta(i)*d_theta(i)+...
        0.5*S.alpha2*d_theta(i)*d_theta(i);
    WP_in(i) = F_in(i)*u_in(i);
    WP_out(i) = F_out(i)*u_out(i);
    WP(i) = F_in(i)*u_in(i) - F_out(i)*u_out(i);
    PE(i) = SE(i) - WP(i);
    PE_2(i) = 0.5*S.K_in*u_in(i)*u_in(i) + ...
        0.5*K_out*u_out(i)*u_out(i) +...
        0.5*S.alpha1*d_theta(i)*d_theta(i)+...
        0.5*S.alpha2*d_theta(i)*d_theta(i) - ...
        (F_in(i)*u_in(i) - F_out(i)*u_out(i));
d2PE_dtheta2(i) = S.K_in*S.l*S.l*(sin(theta_ans(i))*sin(theta_ans(i)) +…
 (cos(S.theta0) - cos(theta_ans(i)))*cos(theta_ans(i))) + ...
        K_out*S.l*S.l*(cos(theta_ans(i))*cos(theta_ans(i)) - (sin(theta_ans(i)) - sin(S.theta0))*…
sin(theta_ans(i))) + (S.alpha1+S.alpha2) - F_in(i)*S.l*cos(theta_ans(i));
end
theta_deg = theta_ans*180/pi;       % Equilibrium angle in degree
%% Output
figure(1);
plot(F_in,MA,F_in,MA_r,F_in,MA_c);
```

```
grid on;
ylabel('MA','FontSize',16);
xlabel('F_i_n','FontSize',16);
legend('MA','MA_r','MA_c');
title('Mechanical Advantage v/s Input Force','FontSize',16);
figure(2);
plot(theta_deg,PE,theta_deg,SE,theta_deg,WP);
grid on;
xlabel('Equilibrium Angle (degree)','FontSize',16);
legend('Potential Energy','Strain Energy','Work Potential');
figure(3);
plot(theta_deg,d_SE);
grid on;
xlabel('Equilibrium Angle (degree)','FontSize',16);
ylabel('Change in Strain Energy','FontSize',16);
title('Change in SE','FontSize',16);
figure(4);
plot(theta_deg,MA,theta_deg,MA_r,theta_deg,MA_c);
grid on;
ylabel('MA','FontSize',16);
xlabel('Equilibrium Angle (degree)','FontSize',16);
legend('MA','MA_r','MA_c');
title('Mechanical Advantage (type-3) v/s Equilibrium Angle','FontSize',16);

%% Function derivative
function d_fx = d_func_NR(S,F_in,K_out,F_out,x)
d_fx = S.K_in*S.l*S.l*((cos(S.theta0) - cos(x))*cos(x) + sin(x)*sin(x)) +...
   K_out*S.l*S.l*((sin(x) - sin(S.theta0))*(-sin(x)) + cos(x)*cos(x)) +...
   (S.alpha1+S.alpha2) +...
   F_in*S.l*cos(x) +...
   F_out*S.l*-(sin(x));
end

%% Newton-Raphson
function x = NR(S,F_in,K_out,F_out,x0)
x = x0;
TOL = 1e-10;
n_max = 500;
n=0;
flag = 0;
```

```
while(flag == 0)
   n = n+1;
   fx = func_NR(S,F_in,K_out,F_out,x);
   d_fx = d_func_NR(S,F_in,K_out,F_out,x);
   x_new = x - fx/d_fx;
   error = (x-x_new);
   if(abs(error) <= TOL)
      x = x_new;
      flag = 1;
      return
   else
      x = x_new;
      if(n>n_max)
         flag = -1;
      end
   end
end

%% Function for finding equilibrium angle
function F = func(x,S,F_in,K_out,F_out)
F = S.K_in*S.l*S.l*(cos(S.theta0) - cos(x))*sin(x) + ...
   K_out*S.l*S.l*(sin(x) - sin(S.theta0))*cos(x) + ...
   (S.alpha1+S.alpha2)*(x-S.theta0) - ...
   F_in*S.l*sin(x) +...
   F_out*S.l*cos(x);
```

b. Matlab code for optimization

```
%% main program for optimization
clc;
clear;
close all;
format long;
construct_grid();                    % Construct grid of elements
MAT.E = 1300;              % Youngs Modulus
MAT.nu = 0.3;              % Poisson ratio
VOL = 1500;                % 1500 Upper limit for vol constraint
SK = 10.0;                 % 0.60 Upper limit for sensitivity index
NELEM = load('NELEM.txt');      % No. of elements
NNODE = load('NNODE.txt');      % No. of nodes
node  = load('node.txt');       % Nodal data
```

```
ncon = load('ncon.txt');        % Nodal connectivity matrx
Le = get_length(NELEM,ncon,node);   % Calculate length of each element
Ee = MAT.E*ones(NELEM,1);       % Youngs Modulus for each element
% Design variable is 'd' or depth of each element
d_upper = 3;                    % Upper Limit for d
d_lower = 0.001;                % Lower Limit for d
d0 = load('d0.txt');            % Initial Condition
lb = d_lower*ones(NELEM,1);     % Lower bound for design variable
ub = d_upper*ones(NELEM,1);     % Upper bound for design variable
ub(5) = 0.5*d_upper;
ub(6) = 0.5*d_upper;
be = 1*ones(NELEM,1); % Out-of-plane thickness for each element, which is kept constant
area0 = d0.*be;           % Initial area of each element
vol0 = dot(area0,Le);     % Initial Volume
input_force_node = load('input_force_node.txt');    % Input force node no.
output_force_node = load('output_force_node.txt');    % Output force node no.
dispID = load('dispID.txt');
dispVal = load('dispVal.txt');
[n0, kci0, kco0] = SL_parameters(NELEM,ncon,NNODE,…
node,Le,d0,be,input_force_node,output_force_node,dispID,dispVal,Ee);
MAs0 = find_MAs(NELEM,ncon,NNODE,…
node,Le,d0,be,input_force_node,output_force_node,dispID,dispVal,Ee);
sk0 = find_sk(NELEM,ncon,NNODE,node,Le,d0,be,input_force_node,output_force_node,dispID,dispVal,Ee);
obj_fn = @(de)find_MAs(NELEM,ncon,NNODE,…
node,Le,de,be,input_force_node,output_force_node,dispID,dispVal,Ee);
nlcon_fn = @(de)nlcon(NELEM,ncon,NNODE,…
node,Le,de,be,input_force_node,output_force_node,dispID,dispVal,Ee,SK,VOL);
optns = optimset('MaxFunEvals',50000,'MaxIter',5000,'TolFun',1e-4,'TolCon',1e-4,…
'Display','iter-detailed','GradObj','on','GradConstr','on','Algorithm','interior-point');
tic;
[de_optim,MAs_optim,exitflag,output] = fmincon(obj_fn,d0,[],[],[],[],lb,ub,nlcon_fn,optns);
time=toc
ae = de_optim.*be;        % Optimized area of each elementt
vol = dot(ae,Le);         % Volume of optimized mechanism
[n, kci, kco] = SL_parameters(NELEM,ncon,NNODE,…
node,Le,de_optim,be,input_force_node,output_force_node,dispID,dispVal,Ee);   % SL parameters for optimum
[MAs, grad] = find_MAs(NELEM,ncon,NNODE,…
node,Le,de_optim,be,input_force_node,output_force_node,dispID,dispVal,Ee);
sk = find_sk(NELEM,ncon,NNODE,…
node,Le,de_optim,be,input_force_node,output_force_node,dispID,dispVal,Ee);
```

```
plot_elements(NELEM,ncon,node,de_optim,d_upper,d_lower,[1 0 0]);     % Plot final result
plot_deflection(NELEM,ncon(:,2:3),NNODE,node,Le,de_optim,be,dispID,dispVal,Ee,[0 0.5 0]);

%% Function for calculating MAs for the given topology
function MAs = MAs(NELEM,ncon,NNODE,…
node,Le,de,be,input_force_node,output_force_node,dispID,dispVal,MAT)
[n, kci, kco] = SL_parameters(NELEM,ncon,NNODE,…
node,Le,de,be,input_force_node,output_force_node,dispID,dispVal,MAT);
% NELEM = No. of elements
% ncon = nodal connectivity matrix
% NNODE = No. of nodes
% node = nodal data
% le = vector containing length of each element
% de = vector containing depth of each element
% be = CONSTANT vector containing breadth of each element
% input_force_node = Node no. where input force ia applied
% output_force_node = Node no. where output force is applied
% MAT = Material constants (E, nu)
% dispID = dof no. for BC's
% dispVal = Prescribed dof value at BC
MAs = -n*kco/(kci+n*n*kco);

%% Function to compute sk and volume
function [c,ceq,grad_c,grad_ceq] = nlcon(NELEM,ncon,NNODE,…
node,Le,de,be,input_force_node,output_force_node,dispID,dispVal,Ee,SK,VOL)
sk = find_sk(NELEM,ncon,NNODE,node,Le,de,be,input_force_node,output_force_node,dispID,dispVal,Ee);
c(1) = sk-SK;
Ae = de.*be;
vol = dot(Ae,Le);
c(2) = vol-VOL;
gradient_sk = grad_sk_2(NELEM,ncon,NNODE,node,Le,de,be,dispID,dispVal,Ee);
gradient_V = grad_V(NELEM,Le,be);
grad_c = zeros(NELEM,2);
grad_c(:,1) = gradient_sk;
grad_c(:,2) = gradient_V;
ceq = [];
grad_ceq = [];
%% Function for nodal connectivity for all elements
function [Nelem,ncon] = nodal_connectivity(Nnode,node)
% Nnode = No. of nodes
```

```
% node = Nodal data
Nelem = (Nnode-1)*Nnode/2;      % No. of elements
ncon = zeros(Nelem,3);          % Nodal connectivity matrix
counter = 1;
for i=1:Nelem
   for j=i+1:1:Nnode
      ncon(counter,1) = counter;
      ncon(counter,2) = i;
      ncon(counter,3) = j;
      counter = counter + 1;
   end
end
%% Function for plotting defections after optimisation
function plot_deflection(NELEM,ncon,NNODE,node,Le,de,be,dispID,dispVal,Ee,color_line)
% NELEM = No. of elements
% ncon = nodal connectivity matrix
% NNODE = No. of nodes
% node = nodal data
% le = vector containing length of each element
% de = vector containing depth of each element
% be = CONSTANT vector containing breadth of each element
% input_force_node = Node no. where input force ia applied
% output_force_node = Node no. where output force is applied
% MAT = Material constants (E, nu)
% dispID = dof no. for BC's
% dispVal = Prescribed dof value at BC
% Ee = Youngs Modulus for each element
% color_line = color of line
Ine = zeros(NELEM,1);       % Moment of inertia of each element
Ae = zeros(NELEM,1);        % Area of each element
for i=1:NELEM
   Ae(i) = de(i)*be(i);
   Ine(i) = (1/12)*be(i)*(de(i)^3);
end
nx = node(:,2);
ny = node(:,3);
input_dof = load('input_dof.txt');          % Input dof
output_dof = load('output_dof.txt');        % Output dof
Fin = load('Fin.txt');                      % Input Force
Fout = load('Fout.txt');                    % Output Force
```

```
F = zeros(3*NNODE,1);
F(input_dof) = Fin;        % Fx = 4 N applied at input load
F(output_dof) = Fout;
u = fembeam(Ae, Le, Ee, nx, ny, ncon, NELEM, NNODE, F, dispID,dispVal,Ine);
scale = 1.0;        %  Scaling factor
% Plot deformation
d_upper = max(de);
d_lower = min(de);
for i=1:NELEM
   bin = (de(i)-d_lower)/(d_upper-d_lower);
   if(bin <= 0.1)
      p = 1;
      continue;
   end
   if(bin>0.1 && bin<=0.2)
      p=2;
   end
   if(bin>0.2 && bin<=0.3)
      p=3;
   end
   if(bin>0.3 && bin<=0.4)
      p=4;
   end
   if(bin>0.4 && bin<=0.5)
      p=5;
   end
   if(bin>0.5 && bin<=0.6)
      p=6;
   end
   if(bin>0.6 && bin<=0.7)
      p=7;
   end
   if(bin>0.7 && bin<=0.8)
      p=8;
   end
   if(bin>0.8 && bin<=0.9)
      p=9;
   end
   if(bin>0.9 && bin<=1)
      p=10;
```

```
    end
    x1 = nx(ncon(i,1)) + u(3*(ncon(i,1)-1)+1)*scale;
    y1 = ny(ncon(i,1)) + u(3*(ncon(i,1)-1)+2)*scale;
    x2 = nx(ncon(i,2)) + u(3*(ncon(i,2)-1)+1)*scale;
    y2 = ny(ncon(i,2)) + u(3*(ncon(i,2)-1)+2)*scale;
    figure(1);
    hold on;
    plot([x1,x2],[y1,y2],'LineStyle','-','LineWidth',p,'Color',color_line);
    hold off;
end

%% Plot all elements
function plot_elements(NELEM,ncon,node,de,d_upper,d_lower,color_line)
d_upper = max(de);
d_lower = min(de);
for i=1:NELEM
    bin = (de(i)-d_lower)/(d_upper-d_lower);
    if(bin <= 0.1)
        p = 1;
        continue;
    end
    if(bin>0.1 && bin<=0.2)
        p=2;
    end
    if(bin>0.2 && bin<=0.3)
        p=3;
    end
    if(bin>0.3 && bin<=0.4)
        p=4;
    end
    if(bin>0.4 && bin<=0.5)
        p=5;
    end
    if(bin>0.5 && bin<=0.6)
        p=6;
    end
    if(bin>0.6 && bin<=0.7)
        p=7;
    end
    if(bin>0.7 && bin<=0.8)
```

```
        p=8;
    end
    if(bin>0.8 && bin<=0.9)
        p=9;
    end
    if(bin>0.9 && bin<=1)
        p=10;
    end
    x = [node(ncon(i,2),2), node(ncon(i,3),2)];
    y = [node(ncon(i,2),3), node(ncon(i,3),3)];
    figure(1);
    hold on;
    plot(x,y,'LineWidth',p,'Color',color_line);
    hold off;
end

%% Function for calculating sk for the given topology
function sk = find_sk(NELEM,ncon,NNODE,…
node,Le,de,be,input_force_node,output_force_node,dispID,dispVal,MAT)
[n, kci, kco] = SL_parameters(NELEM,ncon,NNODE,…
node,Le,de,be,input_force_node,output_force_node,dispID,dispVal,MAT);
sk = kci*kco/(kci+n*n*kco);


%% Function for calculating Spring-lever (SL) parameters of the given mechanism
function [n, kci, kco] = SL_parameters(NELEM,ncon,NNODE,…
node,Le,de,be,input_force_node,output_force_node,dispID,dispVal,Ee)
Ine = zeros(NELEM,1);       % Moment of inertia of each element
Ae = zeros(NELEM,1);        % Area of each element
for i=1:NELEM
    Ae(i) = de(i)*be(i);
    Ine(i) = (1/12)*be(i)*(de(i)^3);
end
nx = node(:,2);
ny = node(:,3);
input_dof = load('input_dof.txt');          % Input dof
output_dof = load('output_dof.txt');        % Output dof
Fin = load('Fin.txt');                      % Input Force
Fout = load('Fout.txt');                    % Output Force
%% First FE run
```

```
F = zeros(3*NNODE,1);
F(input_dof) = Fin;        % Fx = 4 N applied at input load
u_first = fembeam(Ae, Le, Ee, nx, ny, ncon(:,2:3), NELEM, NNODE, F, dispID,dispVal,Ine);
%% Second FE run
F = 0*F;                   % Initialize to zero
F(output_dof) = Fout;      % Fy = 4 N applied at output load
u_second = fembeam(Ae, Le, Ee, nx, ny, ncon(:,2:3), NELEM, NNODE, F, dispID,dispVal,Ine);
%% Calculation for n, kci and kco
uin_first = abs(u_first(input_dof));        % Disp at input port for FEA 1
uout_first = abs(u_first(output_dof));      % Disp at output port for FEA 1
uin_second = abs(u_second(input_dof));      % Disp at input port for FEA 2
uout_second = abs(u_second(output_dof));    % Disp at output port for FEA 2
Fin = abs(Fin);
Fout = abs(Fout);
n = abs(uout_first/uin_first);              % Spring-lever ratio
kci = Fin/uin_first;                    % Input-side stiffness
kco = -Fout/(n*uin_second - uout_second);% Output-side stiffness


%% Function to construct a grid of elements and nodes
function construct_grid()
L = 50;         % Length of rectangle
B = 100;        % Breadth of rectangle
nx_div = 2;    % 4 No. of divisions in x direction
ny_div = 2;    % 4 No. of divisions in y direction
h_x = L/nx_div; % Element Size in x direction
h_y = B/ny_div; % Element Size in y direction
NNODE = (nx_div+1)*(ny_div+1) + nx_div*ny_div;  % No. of nodes
NELEM = nx_div*(ny_div+1) + ny_div*(nx_div+1) + 4*nx_div*ny_div;    % No. of elements
[nx, ny] = get_nx_ny(NNODE, h_x, h_y, L, B, nx_div, ny_div);        % x & y coordinate of nodes
node = zeros(NNODE,3);      % Nodal data
node(:,1) = 1:NNODE;
node(:,2) = nx;
node(:,3) = ny;
ncon2 = get_ncon(NELEM, nx_div, ny_div);     % Get Nodal connectivity matrix
ncon = zeros(NELEM,3);      % Nodal connectivity
ncon(:,1) = 1:NELEM;
ncon(:,2:3) = ncon2;
save('NNODE.txt','NNODE','-ascii');
save('NELEM.txt','NELEM','-ascii');
```

```
save('node.txt','node','-ascii', '-double');
save('ncon.txt','ncon','-ascii', '-double');
d0 = 2*ones(NELEM,1);                    % Initial Condition
save('d0.txt','d0','-ascii', '-double');
plot_GUI(NNODE,NELEM,h_x, h_y,nx,ny,ncon2);      % Plot GUI
dispID = [1 2 3 19 21 22 24 25 27];
dispVal = zeros(1,length(dispID));
save('dispID.txt','dispID','-ascii');
save('dispVal.txt','dispVal','-ascii');
input_force_node = 9;    % 11 Input force node no.
output_force_node = 4;    % Output force node no.
input_dof = 3*(input_force_node-1)+2;       % Input dof
output_dof = 3*(output_force_node-1)+1;     % Output dof
save('input_force_node.txt','input_force_node','-ascii');
save('output_force_node.txt','output_force_node','-ascii');
save('input_dof.txt','input_dof','-ascii');
save('output_dof.txt','output_dof','-ascii');
Fin = -10.0;
Fout = 10.0;
save('Fin.txt','Fin','-ascii');
save('Fout.txt','Fout','-ascii');


%% Function to check upper and lower constraint on design variable
function [de,bin_lower,bin_upper,bin_design] = des_var_check(de,NELEM,de_lower,de_upper)
counter_bin_lower = 1;
counter_bin_upper = 1;
counter_bin_design = 1;
bin_lower = [];
bin_upper = [];
bin_design = [];
for i=1:NELEM
   if (de(i) <= de_lower)
      de(i) = de_lower;
      bin_lower(counter_bin_lower,1) = i;
      counter_bin_lower = counter_bin_lower + 1;
      continue;
   end
   if(de(i) >= de_upper)
      de(i) = de_upper;
```

```
        bin_upper(counter_bin_upper,1) = i;
        counter_bin_upper = counter_bin_upper + 1;
        continue;
    end
    bin_design(counter_bin_design,1) = i;
    counter_bin_design = counter_bin_design + 1;
end


%% Function to obtain length of each element
function le = get_length(Nelem,ncon,node)
le = zeros(Nelem,1);
for i=1:Nelem
    sum = (node(ncon(i,2),2)-node(ncon(i,3),2))^2 +...
        (node(ncon(i,2),3)-node(ncon(i,3),3))^2;
    le(i) = sqrt(sum);
end

%% Function for gradient of onjective function f = -MAs
function grad_obj = grad_MAs(NELEM,ncon,NNODE,node,Le,de,be,dispID,dispVal,Ee)
Ine = zeros(NELEM,1);       % Moment of inertia of each element
Ae = zeros(NELEM,1);        % Area of each element
for i=1:NELEM
    Ae(i) = de(i)*be(i);
    Ine(i) = (1/12)*be(i)*(de(i)^3);
end
nx = node(:,2);
ny = node(:,3);
input_dof = load('input_dof.txt');          % Input dof
output_dof = load('output_dof.txt');        % Output dof
Fin = load('Fin.txt');                          % Input Force
Fout = load('Fout.txt');                        % Output Force
% First FE run
F = zeros(3*NNODE,1);
F(input_dof) = Fin;         % Fx applied at input load
[u_first,K1] = fembeam_with_K(Ae, Le, Ee, nx, ny, ncon(:,2:3), NELEM, NNODE, F, dispID,dispVal,Ine);
% Second FE run
F = 0*F;                    % Initialize to zero
F(output_dof) = Fout;       % Fy applied at output load
[u_second,K2] = fembeam_with_K(Ae, Le, Ee, nx, ny, ncon(:,2:3), NELEM, NNODE, F, dispID,dispVal,Ine);
```

```
uin1 = abs(u_first(input_dof));         % Disp at input port for FEA 1
uout1 = abs(u_first(output_dof));       % Disp at output port for FEA 1
uin2 = abs(u_second(input_dof));        % Disp at input port for FEA 2
uout2 = abs(u_second(output_dof));      % Disp at output port for FEA 2
Fin = abs(Fin);
Fout = abs(Fout);
% Derivatives of MAs wrt uin and uout
dMAs_duin1 = (Fout*(uout1^2)*…
            (-Fin*uin2+Fout*uout1))/((-Fin*uin2*uout1+Fout*(uout1^2)+Fin*uin1*uout2)^2);
dMAs_duout1 = -((Fout*uin1*(Fout*(uout1^2)-Fin*uin1*uout2))/…
(-Fin*uin2*uout1+Fout*(uout1^2)+Fin*uin1*uout2)^2);
dMAs_duin2 = (Fin*Fout*uin1*(uout1^2))/((-Fin*uin2*uout1+Fout*(uout1^2)+Fin*uin1*uout2)^2);
dMAs_duout2 = (Fin*Fout*(uin1^2)*uout1)/((-Fin*uin2*uout1+Fout*(uout1^2)+Fin*uin1*uout2)^2);
u_first_cut = u_first;
u_first_cut(dispID)=[];
u_second_cut = u_second;
u_second_cut(dispID) = [];
grad_obj = zeros(NELEM,1);
du1_dde_added =  zeros(3*NNODE,1);
du2_dde_added =  zeros(3*NNODE,1);
for i=1:NELEM
   grad_K = get_grad_K(be, de, Le, Ee, nx, ny, ncon(:,2:3), NELEM, NNODE, dispID,i);
% Gradient of K wrt design variable
   du1_dde = -1*K1\(grad_K*u_first_cut);
   du2_dde = -1*K2\(grad_K*u_second_cut);
   % Add the vector again
   [sm,sn] = size(dispID);
   Ndbc = sn;
   du1_dde_added = 0*du1_dde_added;
   du2_dde_added = 0*du2_dde_added;
   for iu=1:Ndbc,
      du1_dde_added(dispID(iu)) = 12345.12345;
      du2_dde_added(dispID(iu)) = 12345.12345;
   end
   iuc = 0;
   for iu=1:3*NNODE,
      if du1_dde_added(iu) == 12345.12345
         iuc = iuc+1;
      else
         du1_dde_added(iu) = du1_dde(iu-iuc);
```

```
            du2_dde_added(iu) = du2_dde(iu-iuc);
        end
    end
    duin1_dde = du1_dde_added(input_dof);     % Value of d(u_first)/d(design var) at input dof
    duout1_dde = du1_dde_added(output_dof);   % Value of d(u_first)/d(design var) at output dof
    duin2_dde = du2_dde_added(input_dof);     % Value of d(u_second)/d(design var) at input dof
    duout2_dde = du2_dde_added(output_dof);   % Value of d(u_second)/d(design var) at output dof
    grad_obj(i) = -1 * (dMAs_duin1*duin1_dde + dMAs_duout1*duout1_dde +...
        dMAs_duin2*duin2_dde + dMAs_duout2*duout2_dde);
end

%% Function for gradient of MAs using adjoint method
function gradient_MAs = grad_MAs_2(NELEM,ncon,NNODE,node,Le,de,be,dispID,dispVal,Ee)
Ine = zeros(NELEM,1);       % Moment of inertia of each element
Ae = zeros(NELEM,1);        % Area of each element
for i=1:NELEM
    Ae(i) = de(i)*be(i);
    Ine(i) = (1/12)*be(i)*(de(i)^3);
end
nx = node(:,2);
ny = node(:,3);
input_dof = load('input_dof.txt');          % Input dof
output_dof = load('output_dof.txt');        % Output dof
Fin = load('Fin.txt');                          % Input Force
Fout = load('Fout.txt');                        % Output Force
% First FE run
F = zeros(3*NNODE,1);
F(input_dof) = Fin;        % Fx applied at input load
[u_first,K1] = fembeam_with_K(Ae, Le, Ee, nx, ny, ncon(:,2:3), NELEM, NNODE, F, dispID,dispVal,Ine);
% Second FE run
F = 0*F;                   % Initialize to zero
F(output_dof) = Fout;        % Fy applied at output load
[u_second,K2] = fembeam_with_K(Ae, Le, Ee, nx, ny, ncon(:,2:3), NELEM, NNODE, F, dispID,dispVal,Ine);
uin1 = abs(u_first(input_dof));         % Disp at input port for FEA 1
uout1 = abs(u_first(output_dof));       % Disp at output port for FEA 1
uin2 = abs(u_second(input_dof));        % Disp at input port for FEA 2
uout2 = abs(u_second(output_dof));      % Disp at output port for FEA 2
Fin = abs(Fin);
Fout = abs(Fout);
% Derivatives of MAs wrt uin and uout
```

```
dMAs_duin1 = (Fout*(uout1^2)*(-Fin*uin2+Fout*uout1))/…
((-Fin*uin2*uout1+Fout*(uout1^2)+Fin*uin1*uout2)^2);
dMAs_duout1 = -((Fout*uin1*(Fout*(uout1^2)-Fin*uin1*uout2))/…
(-Fin*uin2*uout1+Fout*(uout1^2)+Fin*uin1*uout2)^2);
dMAs_duin2 = (Fin*Fout*uin1*(uout1^2))/((-Fin*uin2*uout1+Fout*(uout1^2)+Fin*uin1*uout2)^2);
dMAs_duout2 = (Fin*Fout*(uin1^2)*uout1)/((-Fin*uin2*uout1+Fout*(uout1^2)+Fin*uin1*uout2)^2);
u_first_cut = u_first;
u_first_cut(dispID)=[];
u_second_cut = u_second;
u_second_cut(dispID) = [];
% Finding lamda1 and lamda2
dMAs_du1 = zeros(3*NNODE,1);
dMAs_du1(input_dof) = dMAs_duin1;
dMAs_du1(output_dof) = dMAs_duout1;
dMAs_du1(dispID) = [];
lamda1 = -inv(K1) * dMAs_du1;
dMAs_du2 = zeros(3*NNODE,1);
dMAs_du2(input_dof) = dMAs_duin2;
dMAs_du2(output_dof) = dMAs_duout2;
dMAs_du2(dispID) = [];
lamda2 = -inv(K2) * dMAs_du2;
gradient_MAs = zeros(NELEM,1);
for i=1:NELEM
   grad_K = get_grad_K(be, de, Le, Ee, nx, ny, ncon(:,2:3), NELEM, NNODE, dispID,i);        % Gradient of K
wrt design variable
   gradient_MAs(i) = dot(lamda1,grad_K*u_first_cut) + ...
      dot(lamda2,grad_K*u_second_cut);
end


%% Function to find gradient of sk wrt design variable using adjoint method
function gradient_sk = grad_sk_2(NELEM,ncon,NNODE,node,Le,de,be,dispID,dispVal,Ee)
Ine = zeros(NELEM,1);       % Moment of inertia of each element
Ae = zeros(NELEM,1);        % Area of each element
for i=1:NELEM
   Ae(i) = de(i)*be(i);
   Ine(i) = (1/12)*be(i)*(de(i)^3);
end
nx = node(:,2);
ny = node(:,3);
```

```
input_dof = load('input_dof.txt');          % Input dof
output_dof = load('output_dof.txt');       % Output dof
Fin = load('Fin.txt');                          % Input Force
Fout = load('Fout.txt');                       % Output Force
% First FE run
F = zeros(3*NNODE,1);
F(input_dof) = Fin;        % Fx applied at input load
[u_first,K1] = fembeam_with_K(Ae, Le, Ee, nx, ny, ncon(:,2:3), NELEM, NNODE, F, dispID,dispVal,Ine);
% Second FE run
F = 0*F;                    % Initialize to zero
F(output_dof) = Fout;        % Fy applied at output load
[u_second,K2] = fembeam_with_K(Ae, Le, Ee, nx, ny, ncon(:,2:3), NELEM, NNODE, F, dispID,dispVal,Ine);
uin1 = abs(u_first(input_dof));         % Disp at input port for FEA 1
uout1 = abs(u_first(output_dof));       % Disp at output port for FEA 1
uin2 = abs(u_second(input_dof));        % Disp at input port for FEA 2
uout2 = abs(u_second(output_dof));      % Disp at output port for FEA 2
Fin = abs(Fin);
Fout = abs(Fout);
% Derivatives of sk wrt uin and uout
dsk_duin1 = (Fin*Fout*uout1*(-Fin*uin2+Fout*uout1))/…
(-Fin*uin2*uout1+Fout*(uout1^2)+Fin*uin1*uout2)^2;
dsk_duout1 = (Fin*Fout*uin1*(Fin*uin2-2*Fout*uout1))/…
(-Fin*uin2*uout1+Fout*(uout1^2)+Fin*uin1*uout2)^2;
dsk_duin2 = ((Fin^2)*Fout*uin1*uout1)/(-Fin*uin2*uout1+Fout*(uout1^2)+Fin*uin1*uout2)^2;
dsk_duout2 = -((Fin^2)*Fout*(uin1^2))/((-Fin*uin2*uout1+Fout*(uout1^2)+Fin*uin1*uout2)^2);
u_first_cut = u_first;
u_first_cut(dispID)=[];
u_second_cut = u_second;
u_second_cut(dispID) = [];
% Finding lamda1 and lamda2
dsk_du1 = zeros(3*NNODE,1);
dsk_du1(input_dof) = dsk_duin1;
dsk_du1(output_dof) = dsk_duout1;
dsk_du1(dispID) = [];
lamda1 = -inv(K1) * dsk_du1;
dsk_du2 = zeros(3*NNODE,1);
dsk_du2(input_dof) = dsk_duin2;
dsk_du2(output_dof) = dsk_duout2;
dsk_du2(dispID) = [];
lamda2 = -inv(K2) * dsk_du2;
```

```
gradient_sk = zeros(NELEM,1);
for i=1:NELEM
   grad_K = get_grad_K(be, de, Le, Ee, nx, ny, ncon(:,2:3), NELEM, NNODE, dispID,i);
   gradient_sk(i) = dot(lamda1,grad_K*u_first_cut) + ...
   dot(lamda2,grad_K*u_second_cut);
end


%% Function to find gradient of V wrt design variable
function gradient_V = grad_V(NELEM,Le,be)
gradient_V = zeros(NELEM,1);
for i=1:NELEM
   gradient_V = be(i)*Le(i);
end


%% Function for obtaining LAMDA1 and LAMDA2 using Newton-Raphson Method
function LAMDA = get_LAMDA_NR(NELEM,ncon,NNODE,node,Le,de,be,…
input_force_node,output_force_node,dispID,dispVal,Ee,SK,VOL,d_lower,d_upper)
LAMDA1_0 = -10;
LAMDA2_0 = 2700;    % 0.1
LAMDA = [LAMDA1_0; LAMDA2_0];
TOL_vol = 100;                  % Tolerance
TOL_sk =0.02 ;
n_max = 100;                % Max. iterations
n = 0;
J = zeros(2,2);             % Jacobian Matrix
flag = 1;
grad_obj = grad_MAs(NELEM,ncon,NNODE,node,Le,de,be,dispID,dispVal,Ee);
gradient_sk = grad_sk(NELEM,ncon,NNODE,node,Le,de,be,dispID,dispVal,Ee);
while (flag == 1)
   n = n+1

   grad_lag = grad_obj + LAMDA(1)*(be.*Le) + LAMDA(2)*gradient_sk;
   de_new = de + 0.0005*(grad_lag);                                % Update of design variable
   [de,bin_lower,bin_upper,bin_design] = des_var_check(de_new,NELEM,d_lower,d_upper);
   grad_obj = grad_MAs(NELEM,ncon,NNODE,node,Le,de,be,dispID,dispVal,Ee);
   gradient_sk = grad_sk(NELEM,ncon,NNODE,node,Le,de,be,dispID,dispVal,Ee);
   % Creating Jacobian
   J(1,1) = dot(Le,Le);
```

```
   J(1,2) = dot(gradient_sk,Le);
   J(2,1) = dot(gradient_sk,Le);
   J(2,2) = dot(gradient_sk,gradient_sk);
   sk = find_sk(NELEM,ncon,NNODE,node,Le,de,be,input_force_node,output_force_node,dispID,dispVal,Ee);
   f1 = dot(be.*de,Le) - VOL;    % f1 = V - V*
   f2 = sk - SK;                 % f2 = sk - sk*
   f = [f1;f2]
   LAMDA_new = LAMDA - J\f;          % Newton-Raphson
   %     if(norm(error,2) <= TOL)
   if(abs(f1) <= TOL_vol && abs(f2) <= TOL_sk)
      LAMDA = LAMDA_new
      flag = 0;
   else
      LAMDA = LAMDA_new
      if(n>n_max)
         flag = -1;
      end
   end
end
if (flag == -1)
   LAMDA = [0;0]
end
```

c. Matlab code for calculation of mechanical advantage from finite-element analysis data

```
clc;
clear;
close all;
format long;
%% Input Data
Kext = 0.5*load('stiffness.txt');   % External stiffness
kext_nd = load('kext_nd.txt');      % Non-dimensional stiffness
Fin = 2*load('Fin.txt');   % Input Force
Fout = load('Fout.txt');  % Output Force
% FE run
uin = load('INPUT_results.txt');     % Input disp for FE run
uout = load('OUTPUT_results.txt');   % Output disp for FE run
[p,q] = size(uin);
%% Calculations
MA = zeros(p,q);    % Mechanical Advantage
scale = load('scale.txt');      % Geometric scale of model
```

```
E = load('E.txt');  % Youngs Modulus
b = load('b.txt');  % depth
d = load('d.txt');  % width
l_bar = load('l_bar.txt');
s = l_bar/d;        % slenderness ratio
eta_in = Fin*s*s/(E*b*d);    % Non-Dimensional Parameter
eta_out = Fout*s*s/(E*b*d);
for i=1:p
   for j=1:q
      MA(i,j) = (Fout(i,j) + Kext*uout(i,j))/(Fin(i,j));
   end
end
%% Plots
figure(1);
mesh(eta_in(:,2:q),eta_out(:,2:q),MA(:,2:q));
xlabel('\eta_{in}');
ylabel('\eta_{out}');
zlabel('Mechanical Advantage (MA)');
titlename = strcat('Variation of MA with \eta_{in} and \eta_{out} with k_{ext} = ',num2str(Kext),' N/m');
title(titlename);
save('1.mat');
```

d. Matlab code for creating non-dimensional maps

```
clc;
clear;
close all;
format long;
m = 5;      % No. of runs
legend_names = cell(1,m);   % legend entry
stiffness = 0.5*load('kext_nd.txt');    % Non-dimensional stiffness
% Populate legend entries
for x=1:m
   filename = strcat(num2str(x),'.mat');
   load(filename);
   legend_names{x} = strcat('s =',num2str(s));
end
%% Mechanical Advantage (MA) Plots
for x=1:m
   filename = strcat(num2str(x),'.mat');
   load(filename);
```

```
    figure(1);
    hold on;
    mesh(eta_in,eta_out,MA);
    hold off;
end
figure(1);
grid on;
xlabel('\eta_{in}');
ylabel('\eta_{out}');
zlabel('MA');
title('Plot of MA with \eta_{in} and \eta_{out}');
legend(legend_names);
% Contour plots for some constant eta_out
eta_out_value = 0.1;
for x=1:m
    filename = strcat(num2str(x),'.mat');
    load(filename);
    figure(20);
    [C,h] = contour(eta_in,MA,eta_out,[eta_out_value eta_out_value]);
    [p q] = size(C);
        % Plot of MA with eta_in with constant value of eta_out mentioned above
    figure(2);
    switch(x)
        case 1
            hold on;
            plot(C(1,2:q),C(2,2:q),'MarkerSize',15,'Marker','x','Color',[0 0 0]);
            hold off;
        case 2
            hold on;
            plot(C(1,2:q),C(2,2:q),'MarkerSize',15,'Marker','square','Color',[0 0 0]);
            hold off;
        case 3
            hold on;
            plot(C(1,2:q),C(2,2:q),'MarkerSize',15,'Marker','v','Color',[0 0 0]);
            hold off;
        case 4
            hold on;
            plot(C(1,2:q),C(2,2:q),'MarkerSize',15,'Marker','o','Color',[0 0 0]);
            hold off;
        case 5
```

```
            hold on;
            plot(C(1,2:q),C(2,2:q),'MarkerSize',15,'Marker','pentagram','Color',[0 0 0]);
            hold off;
        case 6
            hold on;
            plot(C(1,2:q),C(2,2:q),'MarkerSize',15,'Marker','+','Color',[0 0 0]);
            hold off;
        case 7
            hold on;
            plot(C(1,2:q),C(2,2:q),'MarkerSize',15,'Marker','^','Color',[0 0 0]);
            hold off;
        otherwise
            hold on;
            plot(C(1,2:q),C(2,2:q),'Color',[rand rand rand]);
            hold off;
    end
end
figure(2);
axis([0 0.5 0 2]);
grid on;
xlabel('\eta_{in}');
ylabel('MA');
% ylabel('abcd $$\hat{k_{ci}}$$ xyz','Interpreter','Latex');
titlename = strcat('Plot of MA with \eta_{in} at \eta_{out} = ',num2str(eta_out_value),' for k_{ext} =
',num2str(stiffness));
title(titlename);
legend(legend_names);
close 20;
% Contour plots for some constant eta_in
eta_in_value = 0.2;
for x=1:m
    filename = strcat(num2str(x),'.mat');
    load(filename);
    figure(20);
    [C,h] = contour(eta_out,MA,eta_in,[eta_in_value eta_in_value]);
    [p q] = size(C);
    % Plot of MA with eta_in with constant value of eta_in mentioned above
    figure(3);
    switch(x)
        case 1
```

```
            hold on;
            plot(C(1,2:q),C(2,2:q),'MarkerSize',15,'Marker','x','Color',[0 0 0]);
            hold off;
        case 2
            hold on;
            plot(C(1,2:q),C(2,2:q),'MarkerSize',15,'Marker','square','Color',[0 0 0]);
            hold off;
        case 3
            hold on;
            plot(C(1,2:q),C(2,2:q),'MarkerSize',15,'Marker','v','Color',[0 0 0]);
            hold off;
        case 4
            hold on;
            plot(C(1,2:q),C(2,2:q),'MarkerSize',15,'Marker','o','Color',[0 0 0]);
            hold off;
        case 5
            hold on;
            plot(C(1,2:q),C(2,2:q),'MarkerSize',15,'Marker','pentagram','Color',[0 0 0]);
            hold off;
        case 6
            hold on;
            plot(C(1,2:q),C(2,2:q),'MarkerSize',15,'Marker','+','Color',[0 0 0]);
            hold off;
        case 7
            hold on;
            plot(C(1,2:q),C(2,2:q),'MarkerSize',15,'Marker','^','Color',[0 0 0]);
            hold off;
        otherwise
            hold on;
            plot(C(1,2:q),C(2,2:q),'Color',[rand rand rand]);
            hold off;
    end
end
figure(3);
grid on;
xlabel('\eta_{out}');
ylabel('MA');
titlename = strcat('Plot of MA with \eta_{out} at \eta_{in} = ',num2str(eta_in_value),' for k_{ext} =
',num2str(stiffness));
title(titlename);
```

```
legend(legend_names);
close 20;
% Contour plot for uout = 0
uoutvalue = 0;
for x=1:m
   filename = strcat(num2str(x),'.mat');
   load(filename);
   figure(20);
   [C,h] = contour(eta_in,eta_out,uout,[uoutvalue uoutvalue]);
   [p q] = size(C);
   figure(4);
   switch(x)
      case 1
         hold on;
         plot(C(1,2:q),C(2,2:q),'MarkerSize',15,'Marker','x','Color',[0 0 0]);
         hold off;
      case 2
         hold on;
         plot(C(1,2:q),C(2,2:q),'MarkerSize',15,'Marker','square','Color',[0 0 0]);
         hold off;
      case 3
         hold on;
         plot(C(1,2:q),C(2,2:q),'MarkerSize',15,'Marker','v','Color',[0 0 0]);
         hold off;
      case 4
         hold on;
         plot(C(1,2:q),C(2,2:q),'MarkerSize',15,'Marker','o','Color',[0 0 0]);
         hold off;
      case 5
         hold on;
         plot(C(1,2:q),C(2,2:q),'MarkerSize',15,'Marker','pentagram','Color',[0 0 0]);
         hold off;
      case 6
         hold on;
         plot(C(1,2:q),C(2,2:q),'MarkerSize',15,'Marker','+','Color',[0 0 0]);
         hold off;
      case 7
         hold on;
         plot(C(1,2:q),C(2,2:q),'MarkerSize',15,'Marker','^','Color',[0 0 0]);
         hold off;
```

```
        otherwise
            hold on;
            plot(C(1,2:q),C(2,2:q),'Color',[rand rand rand]);
            hold off;
    end
end
figure(4);
grid on;
xlabel('\eta_{in}');
ylabel('\eta_{out}');
titlename = strcat('Plot of \eta_{out} with \eta_{in} when u_{out}=',num2str(uoutvalue),' for k_{ext} = ',num2str(stiffness));
title(titlename);
legend(legend_names);
close 20;
```

D. PYTHON CODES

a. Python code for creating finite-element (FE) model

```
## Python code for Model-10
# Creating file for tracking time
timefile='timefile.txt'
time_track = open(timefile,'w')
from datetime import datetime
time_track.write(str(datetime.now()))
time_track.write("\n")
time_track.close()
# Defining parameters
scalemodel = 1.0
E_Youngs = 1.3e9
poissonratiomaterial = 0.45
depth = scalemodel * 0.010
width1 = scalemodel * 0.0074
width2 = scalemodel * 0.006
total_length = scalemodel * (130e-3 + 100e-3 + 30e-3)
l_bar = scalemodel * (2*130e-3 + 2*100e-3 + 2*30e-3)/5 # l_bar for full model instead of half model
d_bar = (2*width1 + 2*width1 + width2)/5   # b_bar for full model instead of half model
Kext = 5000.0
kext_nd = Kext*l_bar/(depth*d_bar*E_Youngs)
# Writing parameters to a file
parameterfile = open('scale.txt','w')
```

```
parameterfile.write(str(scalemodel))
parameterfile.close()
parameterfile = open('E.txt','w')
parameterfile.write(str(E_Youngs))
parameterfile.close()
parameterfile = open('b.txt','w')
parameterfile.write(str(depth))
parameterfile.close()
parameterfile = open('d.txt','w')
parameterfile.write(str(d_bar))
parameterfile.close()
parameterfile = open('l_bar.txt','w')
parameterfile.write(str(l_bar))
parameterfile.close()
parameterfile.close()
parameterfile = open('kext_nd.txt','w')
parameterfile.write(str(kext_nd))
parameterfile.close()
stiffness_file = open('stiffness.txt','w')
stiffness_file.write(str(Kext))
stiffness_file.close()
# Creating Sketch
from part import *
from material import *
from section import *
from optimization import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
mdb.models['Model-1'].ConstrainedSketch(name='__profile__', sheetSize=5.0)
mdb.models['Model-1'].sketches['__profile__'].ConstructionLine(angle=90.0,
    point1=(0.0, -0.075))
mdb.models['Model-1'].sketches['__profile__'].VerticalConstraint(addUndoState=
    False, entity=mdb.models['Model-1'].sketches['__profile__'].geometry[2])
```

```
mdb.models['Model-1'].sketches['__profile__'].ConstructionLine(angle=0.0,
    point1=(0.225, 0.0))
mdb.models['Model-1'].sketches['__profile__'].HorizontalConstraint(
    addUndoState=False, entity=
    mdb.models['Model-1'].sketches['__profile__'].geometry[3])
mdb.models['Model-1'].sketches['__profile__'].FixedConstraint(entity=
    mdb.models['Model-1'].sketches['__profile__'].geometry[3])
mdb.models['Model-1'].sketches['__profile__'].FixedConstraint(entity=
    mdb.models['Model-1'].sketches['__profile__'].geometry[2])
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0.0, 0.0), point2=(
    0.2, 0.4))
mdb.models['Model-1'].sketches['__profile__'].CoincidentConstraint(
    addUndoState=False, entity1=
    mdb.models['Model-1'].sketches['__profile__'].vertices[0], entity2=
    mdb.models['Model-1'].sketches['__profile__'].geometry[2])
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0.2, 0.4), point2=(
    0.275, 0.225))
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0.275, 0.225),
    point2=(0.4, 0.225))
mdb.models['Model-1'].sketches['__profile__'].HorizontalConstraint(
    addUndoState=False, entity=
    mdb.models['Model-1'].sketches['__profile__'].geometry[6])
mdb.models['Model-1'].sketches['__profile__'].AngularDimension(line1=
    mdb.models['Model-1'].sketches['__profile__'].geometry[4], line2=
    mdb.models['Model-1'].sketches['__profile__'].geometry[5], textPoint=(
    0.210232853889465, 0.278846085071564), value=21.0)
mdb.models['Model-1'].sketches['__profile__'].AngularDimension(line1=
    mdb.models['Model-1'].sketches['__profile__'].geometry[5], line2=
    mdb.models['Model-1'].sketches['__profile__'].geometry[6], textPoint=(
    0.237543553113937, 0.24023175239563), value=101.0)
mdb.models['Model-1'].sketches['__profile__'].ObliqueDimension(textPoint=(
    0.026375025510788, 0.284710198640823), value=0.13, vertex1=
    mdb.models['Model-1'].sketches['__profile__'].vertices[0], vertex2=
    mdb.models['Model-1'].sketches['__profile__'].vertices[1])
mdb.models['Model-1'].sketches['__profile__'].ObliqueDimension(textPoint=(
    0.0572492629289627, 0.0451269000768662), value=0.1, vertex1=
    mdb.models['Model-1'].sketches['__profile__'].vertices[1], vertex2=
    mdb.models['Model-1'].sketches['__profile__'].vertices[2])
mdb.models['Model-1'].sketches['__profile__'].ObliqueDimension(textPoint=(
    0.0926303714513779, 0.0175947099924088), value=0.03, vertex1=
```

```
    mdb.models['Model-1'].sketches['__profile__'].vertices[2], vertex2=
    mdb.models['Model-1'].sketches['__profile__'].vertices[3])
mdb.models['Model-1'].Part(dimensionality=TWO_D_PLANAR, name=
    'ORIGINAL_Model_10', type=DEFORMABLE_BODY)
mdb.models['Model-1'].parts['ORIGINAL_Model_10'].BaseWire(sketch=
    mdb.models['Model-1'].sketches['__profile__'])
del mdb.models['Model-1'].sketches['__profile__']
# Creating scaled part
mdb.models['Model-1'].Part(compressFeatureList=ON, name='Model_10',
    objectToCopy=mdb.models['Model-1'].parts['ORIGINAL_Model_10'], scale=scalemodel)
# Create reference point
mdb.models['Model-1'].parts['Model_10'].DatumPointByOffset(point=
    mdb.models['Model-1'].parts['Model_10'].vertices[1], vector=(0.050*scalemodel, 0.0,
    0.0))
mdb.models['Model-1'].parts['Model_10'].ReferencePoint(point=
    mdb.models['Model-1'].parts['Model_10'].datums[2])
# Properties
mdb.models['Model-1'].parts['Model_10'].Set(edges=
    mdb.models['Model-1'].parts['Model_10'].edges.getSequenceFromMask(('[#3 ]',
    ), ), name='SET_BEAM1')
mdb.models['Model-1'].parts['Model_10'].Set(edges=
    mdb.models['Model-1'].parts['Model_10'].edges.getSequenceFromMask(('[#4 ]',
    ), ), name='SET_BEAM2')
mdb.models['Model-1'].Material(name='Polypropylene')
mdb.models['Model-1'].materials['Polypropylene'].Elastic(table=((E_Youngs,
    poissonratiomaterial), ))
mdb.models['Model-1'].RectangularProfile(a=depth, b=width1, name='Rect1')
mdb.models['Model-1'].RectangularProfile(a=depth, b=width2, name='Rect2')
mdb.models['Model-1'].BeamSection(consistentMassMatrix=False, integration=
    DURING_ANALYSIS, material='Polypropylene', name='Beam1', poissonRatio=0.0,
    profile='Rect1', temperatureVar=LINEAR)
mdb.models['Model-1'].BeamSection(consistentMassMatrix=False, integration=
    DURING_ANALYSIS, material='Polypropylene', name='Beam2', poissonRatio=0.0,
    profile='Rect2', temperatureVar=LINEAR)
mdb.models['Model-1'].parts['Model_10'].SectionAssignment(offset=0.0,
    offsetField='', offsetType=MIDDLE_SURFACE, region=
    mdb.models['Model-1'].parts['Model_10'].sets['SET_BEAM1'], sectionName=
    'Beam1', thicknessAssignment=FROM_SECTION)
mdb.models['Model-1'].parts['Model_10'].SectionAssignment(offset=0.0,
    offsetField='', offsetType=MIDDLE_SURFACE, region=
```

```
    mdb.models['Model-1'].parts['Model_10'].sets['SET_BEAM2'], sectionName=
    'Beam2', thicknessAssignment=FROM_SECTION)
mdb.models['Model-1'].parts['Model_10'].assignBeamSectionOrientation(method=
    N1_COSINES, n1=(0.0, 0.0, -1.0), region=Region(
    edges=mdb.models['Model-1'].parts['Model_10'].edges.getSequenceFromMask(
    mask=('[#7 ]', ), )))
# Assembly
mdb.models['Model-1'].rootAssembly.DatumCsysByDefault(CARTESIAN)
mdb.models['Model-1'].rootAssembly.Instance(dependent=ON, name='Model_10-1',
    part=mdb.models['Model-1'].parts['Model_10'])
# Creating load-step
mdb.models['Model-1'].StaticStep(initialInc=0.1, maxInc=0.1, name='Step-1',
    nlgeom=ON, previous='Initial')
# Creating external spring
mdb.models['Model-1'].ConnectorSection(name='Ext_spring', translationalType=
    AXIAL)
mdb.models['Model-1'].sections['Ext_spring'].setValues(behaviorOptions=(
    ConnectorElasticity(table=((Kext, ), ), independentComponents=(),
    components=(1, )), ))
mdb.models['Model-1'].rootAssembly.WirePolyLine(mergeWire=False, meshable=False
    , points=((
    mdb.models['Model-1'].rootAssembly.instances['Model_10-1'].referencePoints[3],
    mdb.models['Model-1'].rootAssembly.instances['Model_10-1'].vertices[1]), ))
mdb.models['Model-1'].rootAssembly.Set(edges=
    mdb.models['Model-1'].rootAssembly.edges.getSequenceFromMask(('[#1 ]', ), )
    , name='Wire-1-Set-1')
mdb.models['Model-1'].rootAssembly.SectionAssignment(region=
    mdb.models['Model-1'].rootAssembly.sets['Wire-1-Set-1'], sectionName=
    'Ext_spring')
mdb.models['Model-1'].rootAssembly.DatumCsysByThreePoints(coordSysType=
    CARTESIAN, origin=
    mdb.models['Model-1'].rootAssembly.instances['Model_10-1'].referencePoints[3]
    , point1=
    mdb.models['Model-1'].rootAssembly.instances['Model_10-1'].vertices[1])
mdb.models['Model-1'].rootAssembly.sectionAssignments[0].getSet()
mdb.models['Model-1'].rootAssembly.ConnectorOrientation(localCsys1=
    mdb.models['Model-1'].rootAssembly.datums[6], region=
    mdb.models['Model-1'].rootAssembly.allSets['Wire-1-Set-1'])
# Mesh
mdb.models['Model-1'].parts['Model_10'].seedPart(deviationFactor=0.1,
```

```
    minSizeFactor=0.1, size=0.004*scalemodel)
mdb.models['Model-1'].parts['Model_10'].setElementType(elemTypes=(ElemType(
    elemCode=B22, elemLibrary=STANDARD), ), regions=(
    mdb.models['Model-1'].parts['Model_10'].edges.getSequenceFromMask(('[#7 ]',
    ), ), ))
mdb.models['Model-1'].parts['Model_10'].generateMesh()
# Creating input and putput node-set
mdb.models['Model-1'].parts['Model_10'].Set(name='INPUT', nodes=
    mdb.models['Model-1'].parts['Model_10'].nodes.getSequenceFromMask(mask=(
    '[#8 ]', ), ))
mdb.models['Model-1'].parts['Model_10'].Set(name='OUTPUT', nodes=
    mdb.models['Model-1'].parts['Model_10'].nodes.getSequenceFromMask(mask=(
    '[#2 ]', ), ))
# Boundary conditions
mdb.models['Model-1'].DisplacementBC(amplitude=UNSET, createStepName='Initial',
    distributionType=UNIFORM, fieldName='', localCsys=None, name='Fix', region=
    Region(
    vertices=mdb.models['Model-1'].rootAssembly.instances['Model_10-1'].vertices.getSequenceFromMask(
    mask=('[#1 ]', ), ), referencePoints=(
    mdb.models['Model-1'].rootAssembly.instances['Model_10-1'].referencePoints[3],
    )), u1=SET, u2=SET, ur3=SET)
mdb.models['Model-1'].DisplacementBC(amplitude=UNSET, createStepName='Initial',
    distributionType=UNIFORM, fieldName='', localCsys=None, name='Vert',
    region=
    mdb.models['Model-1'].rootAssembly.instances['Model_10-1'].sets['INPUT'],
    u1=SET, u2=UNSET, ur3=SET)
# Save
savefilename = 'Model_10_' + str(int(Kext)) + '.cae'
mdb.saveAs(savefilename)
# Simulate runs
execfile('FE_run.py')
# Writing closing time
time_track = open(timefile,'a')
from datetime import datetime
time_track.write(str(datetime.now()))
time_track.close()

## Python code for scissors model
# Creating file for tracking time
timefile='timefile.txt'
time_track = open(timefile,'w')
```

```
from datetime import datetime
time_track.write(str(datetime.now()))
time_track.write("\n")
time_track.close()
# Parameters
scalemodel = 1.0
radiusofcircle = 20.5165965e-3
y1 = 0.164605492802795
x1 = 0.0268490843290746
vert1 = 25.319219410299e-3
hori1 = 27.0e-3
y2 = 152.15643031128e-3
x2 = 42.1243095149255e-3
vert2 = 12.299832857306e-3
E_Youngs = 1.3e9
poissonratiomaterial = 0.45
G_shearmodulus = E_Youngs/(2*(1+poissonratiomaterial))
depth = 0.001 * scalemodel
width1 = 0.01 * scalemodel
width2_start = 0.004802855 * scalemodel
width2_end = 0.007009778 * scalemodel
total_length = scalemodel*((146.993E-03 + vert1 + hori1) + (143.229E-03 + vert2 + 11.725E-03))
l_bar = total_length/6
#limit = 143.229e-3 * scalemodel
#tau = 0.0154083*limit*limit/2 + 0.00480286*limit
d_bar = (width1*((146.993E-03 + vert1 + hori1)*scalemodel) + width2_end*((143.229E-03 + vert2 + 11.725E-
03)*scalemodel))/total_length
Kext = 0.0
# Writing parameters to a file
parameterfile = open('scale.txt','w')
parameterfile.write(str(scalemodel))
parameterfile.close()
parameterfile = open('E.txt','w')
parameterfile.write(str(E_Youngs))
parameterfile.close()
parameterfile = open('b.txt','w')
parameterfile.write(str(depth))
parameterfile.close()
parameterfile = open('d.txt','w')
parameterfile.write(str(d_bar))
```

```
parameterfile.close()
parameterfile = open('l_bar.txt','w')
parameterfile.write(str(l_bar))
parameterfile.close()
stiffness_file = open('stiffness.txt','w')
stiffness_file.write(str(Kext))
stiffness_file.close()
# sketch
from part import *
from material import *
from section import *
from optimization import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
mdb.models['Model-1'].ConstrainedSketch(name='__profile__', sheetSize=0.5)
mdb.models['Model-1'].sketches['__profile__'].ConstructionLine(angle=0.0,
    point1=(0.005, 0.0))
mdb.models['Model-1'].sketches['__profile__'].HorizontalConstraint(
    addUndoState=False, entity=
    mdb.models['Model-1'].sketches['__profile__'].geometry[2])
mdb.models['Model-1'].sketches['__profile__'].ConstructionLine(angle=90.0,
    point1=(0.0, 0.0))
mdb.models['Model-1'].sketches['__profile__'].VerticalConstraint(addUndoState=
    False, entity=mdb.models['Model-1'].sketches['__profile__'].geometry[3])
mdb.models['Model-1'].sketches['__profile__'].FixedConstraint(entity=
    mdb.models['Model-1'].sketches['__profile__'].geometry[2])
mdb.models['Model-1'].sketches['__profile__'].FixedConstraint(entity=
    mdb.models['Model-1'].sketches['__profile__'].geometry[3])
mdb.models['Model-1'].sketches['__profile__'].CircleByCenterPerimeter(center=(
    0.0, 0.0), point1=(0.01, -0.02))
mdb.models['Model-1'].sketches['__profile__'].CoincidentConstraint(
    addUndoState=False, entity1=
    mdb.models['Model-1'].sketches['__profile__'].vertices[0], entity2=
```

```
    mdb.models['Model-1'].sketches['__profile__'].geometry[2])
mdb.models['Model-1'].sketches['__profile__'].RadialDimension(curve=
    mdb.models['Model-1'].sketches['__profile__'].geometry[4], radius=
    radiusofcircle, textPoint=(-0.0321817472577095, -0.0232926364988089))
mdb.models['Model-1'].sketches['__profile__'].Spot(point=(0.0275, 0.07))
mdb.models['Model-1'].sketches['__profile__'].VerticalDimension(textPoint=(
    -0.044392853975296, 0.0539091899991035), value=y1, vertex1=
    mdb.models['Model-1'].sketches['__profile__'].vertices[0], vertex2=
    mdb.models['Model-1'].sketches['__profile__'].vertices[2])
mdb.models['Model-1'].sketches['__profile__'].HorizontalDimension(textPoint=(
    0.00948844850063324, 0.171912297606468), value=x1, vertex1=
    mdb.models['Model-1'].sketches['__profile__'].vertices[0], vertex2=
    mdb.models['Model-1'].sketches['__profile__'].vertices[2])
mdb.models['Model-1'].sketches['__profile__'].ConstructionLine(point1=(
    0.0268490843290746, 0.164605492802795), point2=(0.0225, 0.1525))
mdb.models['Model-1'].sketches['__profile__'].CoincidentConstraint(
    addUndoState=False, entity1=
    mdb.models['Model-1'].sketches['__profile__'].vertices[2], entity2=
    mdb.models['Model-1'].sketches['__profile__'].geometry[5])
mdb.models['Model-1'].sketches['__profile__'].AngularDimension(line1=
    mdb.models['Model-1'].sketches['__profile__'].geometry[3], line2=
    mdb.models['Model-1'].sketches['__profile__'].geometry[5], textPoint=(
    0.0222168117761612, 0.189627200365067), value=7.28018211729673)
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0.00822185972944567,
    0.0187971209099942), point2=(0.0268490843290746, 0.164605492802795))
mdb.models['Model-1'].sketches['__profile__'].CoincidentConstraint(
    addUndoState=False, entity1=
    mdb.models['Model-1'].sketches['__profile__'].vertices[3], entity2=
    mdb.models['Model-1'].sketches['__profile__'].geometry[4])
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0.0268490843290746,
    0.164605492802795), point2=(0.0268490843290747, 0.21))
mdb.models['Model-1'].sketches['__profile__'].VerticalConstraint(addUndoState=
    False, entity=mdb.models['Model-1'].sketches['__profile__'].geometry[7])
mdb.models['Model-1'].sketches['__profile__'].ObliqueDimension(textPoint=(
    -0.0430961847305298, 0.188463658094406), value=vert1, vertex1=
    mdb.models['Model-1'].sketches['__profile__'].vertices[5], vertex2=
    mdb.models['Model-1'].sketches['__profile__'].vertices[6])
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0.0268490843290746,
    0.189924712213094), point2=(0.06, 0.189924712213094))
mdb.models['Model-1'].sketches['__profile__'].HorizontalConstraint(
```

```
    addUndoState=False, entity=
    mdb.models['Model-1'].sketches['__profile__'].geometry[8])
mdb.models['Model-1'].sketches['__profile__'].PerpendicularConstraint(
    addUndoState=False, entity1=
    mdb.models['Model-1'].sketches['__profile__'].geometry[7], entity2=
    mdb.models['Model-1'].sketches['__profile__'].geometry[8])
mdb.models['Model-1'].sketches['__profile__'].ObliqueDimension(textPoint=(
    0.0448111444711685, 0.19775116443634), value=hori1, vertex1=
    mdb.models['Model-1'].sketches['__profile__'].vertices[6], vertex2=
    mdb.models['Model-1'].sketches['__profile__'].vertices[7])
mdb.models['Model-1'].sketches['__profile__'].ConstructionLine(angle=90.0,
    point1=(0.0538490843290746, 0.189924712213094))
mdb.models['Model-1'].sketches['__profile__'].CoincidentConstraint(
    addUndoState=False, entity1=
    mdb.models['Model-1'].sketches['__profile__'].vertices[7], entity2=
    mdb.models['Model-1'].sketches['__profile__'].geometry[9])
mdb.models['Model-1'].sketches['__profile__'].VerticalConstraint(addUndoState=
    False, entity=mdb.models['Model-1'].sketches['__profile__'].geometry[9])
mdb.models['Model-1'].sketches['__profile__'].Spot(point=(0.0425, 0.115))
mdb.models['Model-1'].sketches['__profile__'].VerticalDimension(textPoint=(
    0.0771897286176682, 0.0753825083374977), value=y2, vertex1=
    mdb.models['Model-1'].sketches['__profile__'].vertices[0], vertex2=
    mdb.models['Model-1'].sketches['__profile__'].vertices[8])
mdb.models['Model-1'].sketches['__profile__'].HorizontalDimension(textPoint=(
    0.0195010453462601, -0.0316804870963097), value=x2,
    vertex1=mdb.models['Model-1'].sketches['__profile__'].vertices[0], vertex2=
    mdb.models['Model-1'].sketches['__profile__'].vertices[8])
mdb.models['Model-1'].sketches['__profile__'].ConstructionLine(point1=(
    0.0421243095149255, 0.15215643031128), point2=(0.0375, 0.1325))
mdb.models['Model-1'].sketches['__profile__'].CoincidentConstraint(
    addUndoState=False, entity1=
    mdb.models['Model-1'].sketches['__profile__'].vertices[8], entity2=
    mdb.models['Model-1'].sketches['__profile__'].geometry[10])
mdb.models['Model-1'].sketches['__profile__'].AngularDimension(line1=
    mdb.models['Model-1'].sketches['__profile__'].geometry[3], line2=
    mdb.models['Model-1'].sketches['__profile__'].geometry[10], textPoint=(
    0.0455578565597534, 0.233590453863144), value=10.0002334864253)
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0.017252291178238,
    0.0111035661411734), point2=(0.0421243095149255, 0.15215643031128))
mdb.models['Model-1'].sketches['__profile__'].CoincidentConstraint(
```

```
    addUndoState=False, entity1=
    mdb.models['Model-1'].sketches['__profile__'].vertices[9], entity2=
    mdb.models['Model-1'].sketches['__profile__'].geometry[4])
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0.0421243095149255,
    0.15215643031128), point2=(0.0421243095149253, 0.167499999990687))
mdb.models['Model-1'].sketches['__profile__'].VerticalConstraint(addUndoState=
    False, entity=mdb.models['Model-1'].sketches['__profile__'].geometry[12])
mdb.models['Model-1'].sketches['__profile__'].ObliqueDimension(textPoint=(
    0.0394190289080143, 0.162210464477539), value=vert2, vertex1=
    mdb.models['Model-1'].sketches['__profile__'].vertices[11], vertex2=
    mdb.models['Model-1'].sketches['__profile__'].vertices[12])
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0.0421243095149255,
    0.164456263168586), point2=(0.0538490843159288, 0.164456263169291))
mdb.models['Model-1'].sketches['__profile__'].HorizontalConstraint(
    addUndoState=False, entity=
    mdb.models['Model-1'].sketches['__profile__'].geometry[13])
mdb.models['Model-1'].sketches['__profile__'].PerpendicularConstraint(
    addUndoState=False, entity1=
    mdb.models['Model-1'].sketches['__profile__'].geometry[12], entity2=
    mdb.models['Model-1'].sketches['__profile__'].geometry[13])
mdb.models['Model-1'].sketches['__profile__'].CoincidentConstraint(
    addUndoState=False, entity1=
    mdb.models['Model-1'].sketches['__profile__'].vertices[13], entity2=
    mdb.models['Model-1'].sketches['__profile__'].geometry[9])
mdb.models['Model-1'].Part(dimensionality=TWO_D_PLANAR, name='ORIGINAL_Scissors_Model_1'
    , type=DEFORMABLE_BODY)
mdb.models['Model-1'].parts['ORIGINAL_Scissors_Model_1'].BaseWire(sketch=
    mdb.models['Model-1'].sketches['__profile__'])
del mdb.models['Model-1'].sketches['__profile__']
# Creating Scaled Part
mdb.models['Model-1'].Part(compressFeatureList=ON, name='Scissors_Model_1',
    objectToCopy=mdb.models['Model-1'].parts['ORIGINAL_Scissors_Model_1'], scale=scalemodel)
# Creating Reference Point
mdb.models['Model-1'].parts['Scissors_Model_1'].DatumPlaneByPrincipalPlane(
    offset=0.0, principalPlane=XZPLANE)
mdb.models['Model-1'].parts['Scissors_Model_1'].PartitionEdgeByDatumPlane(
    datumPlane=mdb.models['Model-1'].parts['Scissors_Model_1'].datums[2],
    edges=
    mdb.models['Model-1'].parts['Scissors_Model_1'].edges.getSequenceFromMask((
    '[#8 ]', ), ))
```

```
mdb.models['Model-1'].parts['Scissors_Model_1'].ReferencePoint(point=
    mdb.models['Model-1'].parts['Scissors_Model_1'].vertices[4])
# Properties
mdb.models['Model-1'].parts['Scissors_Model_1'].Set(edges=
    mdb.models['Model-1'].parts['Scissors_Model_1'].edges.getSequenceFromMask((
    '[#238 ]', ), ), name='CIRCLE')
mdb.models['Model-1'].parts['Scissors_Model_1'].Set(edges=
    mdb.models['Model-1'].parts['Scissors_Model_1'].edges.getSequenceFromMask((
    '[#1c0 ]', ), ), name='SET_BEAM1')
mdb.models['Model-1'].parts['Scissors_Model_1'].Set(edges=
    mdb.models['Model-1'].parts['Scissors_Model_1'].edges.getSequenceFromMask((
    '[#7 ]', ), ), name='SET_BEAM2')
mdb.models['Model-1'].Material(name='Polypropylene')
mdb.models['Model-1'].materials['Polypropylene'].Elastic(table=((E_Youngs,
    poissonratiomaterial), ))
mdb.models['Model-1'].RectangularProfile(a=depth, b=width1, name='Rect1')
mdb.models['Model-1'].RectangularProfile(a=depth, b=width2_start, name=
    'Rect2_start')
mdb.models['Model-1'].RectangularProfile(a=depth, b=width2_end, name=
    'Rect2_end')
mdb.models['Model-1'].BeamSection(consistentMassMatrix=False, integration=
    DURING_ANALYSIS, material='Polypropylene', name='Beam1', poissonRatio=0.0,
    profile='Rect1', temperatureVar=LINEAR)
mdb.models['Model-1'].parts['Scissors_Model_1'].SectionAssignment(offset=0.0,
    offsetField='', offsetType=MIDDLE_SURFACE, region=
    mdb.models['Model-1'].parts['Scissors_Model_1'].sets['SET_BEAM1'],
    sectionName='Beam1', thicknessAssignment=FROM_SECTION)
mdb.models['Model-1'].parts['Scissors_Model_1'].assignBeamSectionOrientation(
    method=N1_COSINES, n1=(0.0, 0.0, -1.0), region=
    mdb.models['Model-1'].parts['Scissors_Model_1'].sets['SET_BEAM1'])
#mdb.models['Model-1'].BeamSection(alphaDamping=0.0, beamShape=TAPERED,
#    betaDamping=0.0, centroid=(0.0, 0.0), compositeDamping=0.0,
#    consistentMassMatrix=False, dependencies=0, integration=BEFORE_ANALYSIS,
#    name='Beam2', outputPts=((0.0, 0.0), ), poissonRatio=0.0, profile=
#    'Rect2_start', profileEnd='Rect2_end', shearCenter=(0.0, 0.0), table=((
#    E_Youngs, G_shearmodulus), ), temperatureDependency=OFF, thermalExpansion=
#    OFF)
#mdb.models['Model-1'].parts['Scissors_Model_1'].SectionAssignment(offset=0.0,
#    offsetField='', offsetType=MIDDLE_SURFACE, region=Region(
#    edges=mdb.models['Model-1'].parts['Scissors_Model_1'].edges.getSequenceFromMask(
```

```
#    mask=('[#1 ]', ), )), sectionName='Beam2', thicknessAssignment=
#    FROM_SECTION)
#mdb.models['Model-1'].BeamSection(consistentMassMatrix=False, integration=
#    DURING_ANALYSIS, material='Polypropylene', name='Beam3', poissonRatio=0.0,
#    profile='Rect2_end', temperatureVar=LINEAR)
#mdb.models['Model-1'].parts['Scissors_Model_1'].SectionAssignment(offset=0.0,
#    offsetField='', offsetType=MIDDLE_SURFACE, region=Region(
#    edges=mdb.models['Model-1'].parts['Scissors_Model_1'].edges.getSequenceFromMask(
#    mask=('[#6 ]', ), )), sectionName='Beam3', thicknessAssignment=
#    FROM_SECTION)
mdb.models['Model-1'].BeamSection(consistentMassMatrix=False, integration=
    DURING_ANALYSIS, material='Polypropylene', name='Beam2', poissonRatio=0.0,
    profile='Rect2_end', temperatureVar=LINEAR)
mdb.models['Model-1'].parts['Scissors_Model_1'].SectionAssignment(offset=0.0,
    offsetField='', offsetType=MIDDLE_SURFACE, region=
    mdb.models['Model-1'].parts['Scissors_Model_1'].sets['SET_BEAM2'],
    sectionName='Beam2', thicknessAssignment=FROM_SECTION)
mdb.models['Model-1'].parts['Scissors_Model_1'].assignBeamSectionOrientation(
    method=N1_COSINES, n1=(0.0, 0.0, -1.0), region=
    mdb.models['Model-1'].parts['Scissors_Model_1'].sets['SET_BEAM2'])
mdb.models['Model-1'].BeamSection(consistentMassMatrix=False, integration=
    DURING_ANALYSIS, material='Polypropylene', name='Beam_circle',
    poissonRatio=0.0, profile='Rect1', temperatureVar=LINEAR)
mdb.models['Model-1'].parts['Scissors_Model_1'].SectionAssignment(offset=0.0,
    offsetField='', offsetType=MIDDLE_SURFACE, region=
    mdb.models['Model-1'].parts['Scissors_Model_1'].sets['CIRCLE'],
    sectionName='Beam_circle', thicknessAssignment=FROM_SECTION)
mdb.models['Model-1'].parts['Scissors_Model_1'].assignBeamSectionOrientation(
    method=N1_COSINES, n1=(0.0, 0.0, -1.0), region=
    mdb.models['Model-1'].parts['Scissors_Model_1'].sets['CIRCLE'])
# Creating Assembly
mdb.models['Model-1'].rootAssembly.Instance(dependent=ON, name=
    'Scissors_Model_1-1', part=mdb.models['Model-1'].parts['Scissors_Model_1'])
# Creating Load-Step
mdb.models['Model-1'].StaticStep(initialInc=0.1, maxInc=0.1, name='Step-1',
    nlgeom=ON, previous='Initial')
# Creating rigid-body of circle
mdb.models['Model-1'].RigidBody(bodyRegion=
    mdb.models['Model-1'].rootAssembly.instances['Scissors_Model_1-1'].sets['CIRCLE']
    , name='RIGID_CIRCLE', refPointRegion=Region(referencePoints=(
```

```
    mdb.models['Model-1'].rootAssembly.instances['Scissors_Model_1-1'].referencePoints[4],
    )))
# Creating External Spring
#mdb.models['Model-1'].ConnectorSection(name='Ext_spring', translationalType=
#    AXIAL)
#mdb.models['Model-1'].sections['Ext_spring'].setValues(behaviorOptions=(
#    ConnectorElasticity(table=((Kext, ), ), independentComponents=(),
#    components=(1, )), ))
#mdb.models['Model-1'].rootAssembly.WirePolyLine(mergeWire=False, meshable=False
#    , points=((
#    mdb.models['Model-1'].rootAssembly.instances['Scissors_Model_1-1'].vertices[9],
#    mdb.models['Model-1'].rootAssembly.instances['Scissors_Model_1-1'].vertices[3]),
#    ))
#mdb.models['Model-1'].rootAssembly.Set(edges=
#    mdb.models['Model-1'].rootAssembly.edges.getSequenceFromMask(('[#1 ]', ), )
#    , name='Wire-1-Set-1')
#mdb.models['Model-1'].rootAssembly.SectionAssignment(region=
#    mdb.models['Model-1'].rootAssembly.sets['Wire-1-Set-1'], sectionName=
#    'Ext_spring')
#mdb.models['Model-1'].rootAssembly.DatumCsysByThreePoints(coordSysType=
#    CARTESIAN, origin=
#    mdb.models['Model-1'].rootAssembly.instances['Scissors_Model_1-1'].vertices[9]
#    , point1=
#    mdb.models['Model-1'].rootAssembly.instances['Scissors_Model_1-1'].vertices[3])
#mdb.models['Model-1'].rootAssembly.sectionAssignments[0].getSet()
#mdb.models['Model-1'].rootAssembly.ConnectorOrientation(localCsys1=
#    mdb.models['Model-1'].rootAssembly.datums[6], region=
#    mdb.models['Model-1'].rootAssembly.allSets['Wire-1-Set-1'])
# Mesh the model
mdb.models['Model-1'].parts['Scissors_Model_1'].seedPart(deviationFactor=0.1,
    minSizeFactor=0.1, size=0.004*scalemodel)
mdb.models['Model-1'].parts['Scissors_Model_1'].setElementType(elemTypes=(
    ElemType(elemCode=B22, elemLibrary=STANDARD), ), regions=(
    mdb.models['Model-1'].parts['Scissors_Model_1'].edges.getSequenceFromMask((
    '[#3ff ]', ), ), ))
mdb.models['Model-1'].parts['Scissors_Model_1'].generateMesh()
# Creating input and putput node-set
mdb.models['Model-1'].parts['Scissors_Model_1'].Set(name='INPUT', nodes=
    mdb.models['Model-1'].parts['Scissors_Model_1'].nodes.getSequenceFromMask(
    mask=('[#10 ]', ), ))
```

```
mdb.models['Model-1'].parts['Scissors_Model_1'].Set(name='OUTPUT', nodes=
    mdb.models['Model-1'].parts['Scissors_Model_1'].nodes.getSequenceFromMask(
    mask=('[#8 ]', ), ))
# Boundary Conditions
mdb.models['Model-1'].DisplacementBC(amplitude=UNSET, createStepName='Initial',
    distributionType=UNIFORM, fieldName='', localCsys=None, name='Fix', region=
    Region(
    edges=mdb.models['Model-1'].rootAssembly.instances['Scissors_Model_1-1'].edges.getSequenceFromMask(
    mask=('[#100 ]', ), )), u1=SET, u2=SET, ur3=SET)
mdb.models['Model-1'].DisplacementBC(amplitude=UNSET, createStepName='Initial',
    distributionType=UNIFORM, fieldName='', localCsys=None, name='Vert',
    region=
    mdb.models['Model-1'].rootAssembly.instances['Scissors_Model_1-1'].sets['OUTPUT']
    , u1=SET, u2=UNSET, ur3=SET)
# Save the model
savefilename = 'scissors_model_' + str(int(Kext)) + '.cae'
mdb.saveAs(savefilename)
# Simulate runs
execfile('FE_run.py')
execfile('FE_run_Second.py')
execfile('FE_run_Third.py')
# Writing closing time
time_track = open(timefile,'a')
from datetime import datetime
time_track.write(str(datetime.now()))
time_track.close()

## Python code for M-model
# Creating file for tracking time
timefile='timefile.txt'
time_track = open(timefile,'w')
from datetime import datetime
time_track.write(str(datetime.now()))
time_track.write("\n")
time_track.close()
# Defining parameters
scalemodel = 1.0
length1 = scalemodel * 0.089468776
length2 = scalemodel * 0.07177473
datumdist = scalemodel * (-0.015)
```

```
E_young = 1.3e9
poissonratio = 0.45
depth = 0.001
width = 0.006
d_bar = width
l_bar = (2*length1 + 2*length2)/4
s = l_bar/d_bar
Kext = 1000.0
kext_nd = Kext*l_bar/(depth*d_bar*E_young)
# Writing parameters to a file
parameterfile = open('scale.txt','w')
parameterfile.write(str(scalemodel))
parameterfile.close()
parameterfile = open('E.txt','w')
parameterfile.write(str(E_young))
parameterfile.close()
parameterfile = open('b.txt','w')
parameterfile.write(str(depth))
parameterfile.close()
parameterfile = open('d.txt','w')
parameterfile.write(str(d_bar))
parameterfile.close()
parameterfile = open('l_bar.txt','w')
parameterfile.write(str(l_bar))
parameterfile.close()
parameterfile = open('s.txt','w')
parameterfile.write(str(s))
parameterfile.close()
parameterfile = open('kext_nd.txt','w')
parameterfile.write(str(kext_nd))
parameterfile.close()
stiffness_file = open('stiffness.txt','w')
stiffness_file.write(str(Kext))
stiffness_file.close()
# sketch module
from part import *
from material import *
from section import *
from optimization import *
from assembly import *
```

```
from step import *
from interaction import *
from load import *
from mesh import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
mdb.models['Model-1'].ConstrainedSketch(name='__profile__', sheetSize=5.0)
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0.0, 0.0), point2=(
    0.225, 0.325))
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0.225, 0.325),
    point2=(0.3, 0.2))
mdb.models['Model-1'].sketches['__profile__'].undo()
mdb.models['Model-1'].sketches['__profile__'].undo()
mdb.models['Model-1'].sketches['__profile__'].ConstructionLine(angle=90.0,
    point1=(0.0, 0.075))
mdb.models['Model-1'].sketches['__profile__'].VerticalConstraint(addUndoState=
    False, entity=mdb.models['Model-1'].sketches['__profile__'].geometry[2])
mdb.models['Model-1'].sketches['__profile__'].ConstructionLine(angle=0.0,
    point1=(-0.35, 0.0))
mdb.models['Model-1'].sketches['__profile__'].HorizontalConstraint(
    addUndoState=False, entity=
    mdb.models['Model-1'].sketches['__profile__'].geometry[3])
mdb.models['Model-1'].sketches['__profile__'].FixedConstraint(entity=
    mdb.models['Model-1'].sketches['__profile__'].geometry[3])
mdb.models['Model-1'].sketches['__profile__'].FixedConstraint(entity=
    mdb.models['Model-1'].sketches['__profile__'].geometry[2])
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0.0, 0.0), point2=(
    0.15, 0.35))
mdb.models['Model-1'].sketches['__profile__'].CoincidentConstraint(
    addUndoState=False, entity1=
    mdb.models['Model-1'].sketches['__profile__'].vertices[0], entity2=
    mdb.models['Model-1'].sketches['__profile__'].geometry[2])
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0.15, 0.35), point2=
    (0.225, 0.225))
mdb.models['Model-1'].sketches['__profile__'].AngularDimension(line1=
    mdb.models['Model-1'].sketches['__profile__'].geometry[4], line2=
    mdb.models['Model-1'].sketches['__profile__'].geometry[3], textPoint=(
    0.420960068702698, 0.217533156275749), value=80.0)
```

```
mdb.models['Model-1'].sketches['__profile__'].AngularDimension(line1=
    mdb.models['Model-1'].sketches['__profile__'].geometry[4], line2=
    mdb.models['Model-1'].sketches['__profile__'].geometry[5], textPoint=(
    0.188068151473999, 0.115010306239128), value=33.627367)
mdb.models['Model-1'].sketches['__profile__'].ObliqueDimension(textPoint=(
    -0.0578657388687134, 0.27345472574234), value=length1, vertex1=
    mdb.models['Model-1'].sketches['__profile__'].vertices[0], vertex2=
    mdb.models['Model-1'].sketches['__profile__'].vertices[1])
mdb.models['Model-1'].sketches['__profile__'].ObliqueDimension(textPoint=(
    0.0900472700595856, 0.0596405491232872), value=length2, vertex1=
    mdb.models['Model-1'].sketches['__profile__'].vertices[1], vertex2=
    mdb.models['Model-1'].sketches['__profile__'].vertices[2])
mdb.models['Model-1'].Part(dimensionality=TWO_D_PLANAR, name='M_Model', type=
    DEFORMABLE_BODY)
mdb.models['Model-1'].parts['M_Model'].BaseWire(sketch=
    mdb.models['Model-1'].sketches['__profile__'])
del mdb.models['Model-1'].sketches['__profile__']
# Create datum point for external spring
mdb.models['Model-1'].parts['M_Model'].DatumPointByOffset(point=
    mdb.models['Model-1'].parts['M_Model'].vertices[2], vector=(0.0, datumdist,
    0.0))
mdb.models['Model-1'].parts['M_Model'].ReferencePoint(point=
    mdb.models['Model-1'].parts['M_Model'].datums[2])
# properties
mdb.models['Model-1'].Material(name='Polypropylene')
mdb.models['Model-1'].materials['Polypropylene'].Elastic(table=((E_young,
    poissonratio), ))
mdb.models['Model-1'].RectangularProfile(a=depth, b=width, name='Rect')
mdb.models['Model-1'].BeamSection(consistentMassMatrix=False, integration=
    DURING_ANALYSIS, material='Polypropylene', name='Beam', poissonRatio=0.0,
    profile='Rect', temperatureVar=LINEAR)
mdb.models['Model-1'].parts['M_Model'].SectionAssignment(offset=0.0,
    offsetField='', offsetType=MIDDLE_SURFACE, region=Region(
    edges=mdb.models['Model-1'].parts['M_Model'].edges.getSequenceFromMask(
    mask=('[#3 ]', ), )), sectionName='Beam', thicknessAssignment=FROM_SECTION)
mdb.models['Model-1'].parts['M_Model'].assignBeamSectionOrientation(method=
    N1_COSINES, n1=(0.0, 0.0, -1.0), region=Region(
    edges=mdb.models['Model-1'].parts['M_Model'].edges.getSequenceFromMask(
    mask=('[#3 ]', ), )))
# assembly
```

```
mdb.models['Model-1'].rootAssembly.DatumCsysByDefault(CARTESIAN)
mdb.models['Model-1'].rootAssembly.Instance(dependent=ON, name='M_Model-1',
    part=mdb.models['Model-1'].parts['M_Model'])
mdb.models['Model-1'].rootAssembly.regenerate()
# step
mdb.models['Model-1'].StaticStep(initialInc=0.1, maxInc=0.1, minInc=1e-05,
    name='Step-1', nlgeom=ON, previous='Initial')
# External Spring
mdb.models['Model-1'].ConnectorSection(name='Ext_spring', translationalType=
    AXIAL)
mdb.models['Model-1'].sections['Ext_spring'].setValues(behaviorOptions=(
    ConnectorElasticity(table=((Kext, ), ), independentComponents=(),
    components=(1, )), ))
mdb.models['Model-1'].rootAssembly.WirePolyLine(mergeWire=False, meshable=False
    , points=((
    mdb.models['Model-1'].rootAssembly.instances['M_Model-1'].referencePoints[3],
    mdb.models['Model-1'].rootAssembly.instances['M_Model-1'].vertices[2]), ))
mdb.models['Model-1'].rootAssembly.Set(edges=
    mdb.models['Model-1'].rootAssembly.edges.getSequenceFromMask(('[#1 ]', ), )
    , name='Wire-1-Set-1')
mdb.models['Model-1'].rootAssembly.SectionAssignment(region=
    mdb.models['Model-1'].rootAssembly.sets['Wire-1-Set-1'], sectionName=
    'Ext_spring')
mdb.models['Model-1'].rootAssembly.DatumCsysByThreePoints(coordSysType=
    CARTESIAN, origin=
    mdb.models['Model-1'].rootAssembly.instances['M_Model-1'].referencePoints[3]
    , point1=
    mdb.models['Model-1'].rootAssembly.instances['M_Model-1'].vertices[2])
mdb.models['Model-1'].rootAssembly.sectionAssignments[0].getSet()
mdb.models['Model-1'].rootAssembly.ConnectorOrientation(localCsys1=
    mdb.models['Model-1'].rootAssembly.datums[6], region=
    mdb.models['Model-1'].rootAssembly.allSets['Wire-1-Set-1'])
# mesh
mdb.models['Model-1'].parts['M_Model'].seedPart(deviationFactor=0.1,
    minSizeFactor=0.1, size=0.003*scalemodel)
mdb.models['Model-1'].parts['M_Model'].setElementType(elemTypes=(ElemType(
    elemCode=B22, elemLibrary=STANDARD), ), regions=(
    mdb.models['Model-1'].parts['M_Model'].edges.getSequenceFromMask(('[#3 ]',
    ), ), ))
mdb.models['Model-1'].parts['M_Model'].generateMesh()
```

```
# creation of input and output nodal set
mdb.models['Model-1'].parts['M_Model'].Set(name='INPUT', nodes=
    mdb.models['Model-1'].parts['M_Model'].nodes.getSequenceFromMask(mask=(
    '[#2 ]', ), ))
mdb.models['Model-1'].parts['M_Model'].Set(name='OUTPUT', nodes=
    mdb.models['Model-1'].parts['M_Model'].nodes.getSequenceFromMask(mask=(
    '[#4 ]', ), ))
# Boundary Conditions
mdb.models['Model-1'].DisplacementBC(amplitude=UNSET, createStepName='Initial',
    distributionType=UNIFORM, fieldName='', localCsys=None, name='Fix', region=
    Region(
    vertices=mdb.models['Model-1'].rootAssembly.instances['M_Model-1'].vertices.getSequenceFromMask(
    mask=('[#1 ]', ), ), referencePoints=(
    mdb.models['Model-1'].rootAssembly.instances['M_Model-1'].referencePoints[3],
    )), u1=SET, u2=SET, ur3=SET)
mdb.models['Model-1'].DisplacementBC(amplitude=UNSET, createStepName='Initial',
    distributionType=UNIFORM, fieldName='', localCsys=None, name='Vert',
    region=
    mdb.models['Model-1'].rootAssembly.instances['M_Model-1'].sets['OUTPUT'],
    u1=SET, u2=UNSET, ur3=SET)
# save
savefilename = 'M_model_beam_' + str(int(Kext)) + '.cae'
mdb.saveAs(savefilename)
# Simulate runs
execfile('FE_run.py')
# Writing closing time
time_track = open(timefile,'a')
from datetime import datetime
time_track.write(str(datetime.now()))
time_track.close()
```

b. Python code for first FE run

```
## Python Script for recursive runs
from part import *
from material import *
from section import *
from optimization import *
from assembly import *
from step import *
from interaction import *
```

```
from load import *
from mesh import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
# Creating Stepfile
stepfilename = 'Stepfile_FIRST.txt'
stepfile = open(stepfilename,'w')
stepfile.write('Step                    Time\n')
stepfile.close()
# Create new input and output results file
INPUT_resultfile = 'INPUT_results.txt'
OUTPUT_resultfile = 'OUTPUT_results.txt'
INPUT_myoutfile = open(INPUT_resultfile,'w')
OUTPUT_myoutfile = open(OUTPUT_resultfile,'w')
# Creating input and output grid files
Fin_file = 'Fin.txt'
Fout_file = 'Fout.txt'
Fin_gridfile = open(Fin_file,'w')
Fout_gridfile = open(Fout_file,'w')
runs = 15
Fin_start=0.01
Fout_start= 0.01
Fin_end = 20.0
Fout_end = 20.0
Fin_incre = (Fin_end-Fin_start)/(runs-1)
Fout_incre = (Fout_end-Fout_start)/(runs-1)
Fout = Fout_start
detailfile = open('runs.txt','w')
detailfile.write(str(runs))
detailfile.close()
detailfile = open('Fin_start_end.txt','w')
detailfile.write(str(Fin_start))
detailfile.write("\n")
detailfile.write(str(Fin_end))
detailfile.close()
detailfile = open('Fout_start_end.txt','w')
detailfile.write(str(Fout_start))
detailfile.write("\n")
```

```
detailfile.write(str(Fout_end))
detailfile.close()
counter = 1
for y in range(1,runs+1,1):
	Fin = Fin_start
	for x in range(1,runs+1,1):
		# Input Force
		mdb.models['Model-1'].ConcentratedForce(cf2=-1*Fin, createStepName=
			'Step-1', field='', localCsys=None, name='Fin'
			, region=
			mdb.models['Model-1'].rootAssembly.instances['Model_10-1'].sets['INPUT'])
		# Output Force
		mdb.models['Model-1'].ConcentratedForce(cf1=-1*Fout, createStepName=
			'Step-1', field='', localCsys=None, name='Fout'
			, region=
			mdb.models['Model-1'].rootAssembly.instances['Model_10-1'].sets['OUTPUT'])
		jobname = 'Individual_runs'
		mdb.Job(atTime=None, contactPrint=OFF, description='', echoPrint=OFF,
			explicitPrecision=SINGLE, getMemoryFromAnalysis=False, historyPrint=OFF,
			memory=20, memoryUnits=PERCENTAGE, model='Model-1', modelPrint=OFF,
			multiprocessingMode=DEFAULT, name=jobname, nodalOutputPrecision=
			FULL, numCpus=1, queue=None, scratch='', type=ANALYSIS, userSubroutine='',
			waitHours=0, waitMinutes=0)
		job = mdb.jobs[jobname]
		job.submit()
		job.waitForCompletion()
		# Writing output to resultfile
		import odbAccess
		outputodbfile = jobname + '.odb'
		odb = session.openOdb(outputodbfile)
		timeFrame = odb.steps['Step-1'].frames[-1]
		displacement = timeFrame.fieldOutputs['U']
		N_in = odb.rootAssembly.instances['MODEL_10-1'].nodeSets['INPUT']
		N_out = odb.rootAssembly.instances['MODEL_10-1'].nodeSets['OUTPUT']
		NDisp_in = displacement.getSubset(region=N_in)
		NDisp_out = displacement.getSubset(region=N_out)
		# Write output
		for v in NDisp_in.values:
			INPUT_myoutfile.write(str(v.dataDouble[1]))
			INPUT_myoutfile.write(" ")
```

```
                for v in NDisp_out.values:
                        OUTPUT_myoutfile.write(str(v.dataDouble[0]))
                        OUTPUT_myoutfile.write(" ")
                odb.close()
                # Writing Input and output grid
                Fin_gridfile.write(str(Fin))
                Fin_gridfile.write(" ")
                Fout_gridfile.write(str(Fout))
                Fout_gridfile.write(" ")
                Fin = Fin + Fin_incre
                # Printing steptime
                from datetime import datetime
                stepfile = open(stepfilename,'a')
                stepfile.write(str(counter))
                stepfile.write("     ")
                stepfile.write(str(datetime.now()))
                stepfile.write("\n")
                stepfile.close()
                counter = counter + 1
                #print datetime.now()
        # Adding blank line in the output file
        Fin_gridfile.write("\n")
        Fout_gridfile.write("\n")
        INPUT_myoutfile.write("\n")
        OUTPUT_myoutfile.write("\n")
        Fout = Fout + Fout_incre
# Closing all files
INPUT_myoutfile.close()
OUTPUT_myoutfile.close()
Fin_gridfile.close()
Fout_gridfile.close()
```

c. Python code for second FE run

```
## Python Script for recursive runs
detailfile = open('runs.txt','r')
runs = int(detailfile.readline())
detailfile.close()
detailfile = open('Fin_start_end.txt','r')
Fin_start = float(detailfile.readline())
Fin_end = float(detailfile.readline())
```

```
detailfile.close()
detailfile = open('Fout_start_end.txt','r')
Fout_start = float(detailfile.readline())
Fout_end = float(detailfile.readline())
detailfile.close()
from part import *
from material import *
from section import *
from optimization import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
# Creating Stepfile
stepfilename = 'Stepfile_SECOND.txt'
stepfile = open(stepfilename,'w')
stepfile.write('Step                    Time\n')
stepfile.close()
# Create new input and output results file
INPUT_resultfile = 'INPUT_results_SECOND.txt'
OUTPUT_resultfile = 'OUTPUT_results_SECOND.txt'
INPUT_myoutfile = open(INPUT_resultfile,'w')
OUTPUT_myoutfile = open(OUTPUT_resultfile,'w')
# Creating input and output grid files
Fin_file = 'Fin_SECOND.txt'
Fout_file = 'Fout_SECOND.txt'
Fin_gridfile = open(Fin_file,'w')
Fout_gridfile = open(Fout_file,'w')
Fin_incre = (Fin_end-Fin_start)/(runs-1)
Fout_incre = (Fout_end-Fout_start)/(runs-1)
d_Fin = 0.005*Fin_incre
d_Fout = 0.0
Fout = Fout_start + d_Fout
counter = 1
for y in range(1,runs+1,1):
```

```
Fin = Fin_start + d_Fin
for x in range(1,runs+1,1):
        # Input Force
        mdb.models['Model-1'].ConcentratedForce(cf2=-1*Fin, createStepName=
                'Step-1', field='', localCsys=None, name='Fin'
                , region=
                mdb.models['Model-1'].rootAssembly.instances['Model_10-1'].sets['INPUT'])
        # Output Force
        mdb.models['Model-1'].ConcentratedForce(cf1=-1*Fout, createStepName=
                'Step-1', field='', localCsys=None, name='Fout'
                , region=
                mdb.models['Model-1'].rootAssembly.instances['Model_10-1'].sets['OUTPUT'])
        jobname = 'Individual_runs'
        mdb.Job(atTime=None, contactPrint=OFF, description='', echoPrint=OFF,
                explicitPrecision=SINGLE, getMemoryFromAnalysis=False, historyPrint=OFF,
                memory=20, memoryUnits=PERCENTAGE, model='Model-1', modelPrint=OFF,
                multiprocessingMode=DEFAULT, name=jobname, nodalOutputPrecision=
                FULL, numCpus=1, queue=None, scratch='', type=ANALYSIS, userSubroutine='',
                waitHours=0, waitMinutes=0)
        job = mdb.jobs[jobname]
        job.submit()
        job.waitForCompletion()
        # Writing output to resultfile
        import odbAccess
        outputodbfile = jobname + '.odb'
        odb = session.openOdb(outputodbfile)
        timeFrame = odb.steps['Step-1'].frames[-1]
        displacement = timeFrame.fieldOutputs['U']
        N_in = odb.rootAssembly.instances['MODEL_10-1'].nodeSets['INPUT']
        N_out = odb.rootAssembly.instances['MODEL_10-1'].nodeSets['OUTPUT']
        NDisp_in = displacement.getSubset(region=N_in)
        NDisp_out = displacement.getSubset(region=N_out)
        # Write output
        for v in NDisp_in.values:
                INPUT_myoutfile.write(str(v.dataDouble[1]))
                INPUT_myoutfile.write(' ')
        for v in NDisp_out.values:
                OUTPUT_myoutfile.write(str(v.dataDouble[0]))
                OUTPUT_myoutfile.write(' ')
        odb.close()
```

```
                    # Writing Input and output grid
                    Fin_gridfile.write(str(Fin))
                    Fin_gridfile.write(" ")
                    Fout_gridfile.write(str(Fout))
                    Fout_gridfile.write(" ")
                    Fin = Fin + Fin_incre
                    # Printing steptime
                    from datetime import datetime
                    stepfile = open(stepfilename,'a')
                    stepfile.write(str(counter))
                    stepfile.write("     ")
                    stepfile.write(str(datetime.now()))
                    stepfile.write("\n")
                    stepfile.close()
                    counter = counter + 1
                    #print datetime.now()
            # Adding blank line in the output file
            Fin_gridfile.write("\n")
            Fout_gridfile.write("\n")
            INPUT_myoutfile.write("\n")
            OUTPUT_myoutfile.write("\n")
            Fout = Fout + Fout_incre
# Writing increments
incrementfile = open('d_Fin.txt','w')
incrementfile.write(str(d_Fin))
incrementfile.close()
# Closing all files
INPUT_myoutfile.close()
OUTPUT_myoutfile.close()
Fin_gridfile.close()
Fout_gridfile.close()
```

d. Python code for third FE run

```
## Python Script for recursive runs
detailfile = open('runs.txt','r')
runs = int(detailfile.readline())
detailfile.close()
detailfile = open('Fin_start_end.txt','r')
Fin_start = float(detailfile.readline())
Fin_end = float(detailfile.readline())
```

```
detailfile.close()
detailfile = open('Fout_start_end.txt','r')
Fout_start = float(detailfile.readline())
Fout_end = float(detailfile.readline())
detailfile.close()
from part import *
from material import *
from section import *
from optimization import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
# Creating Stepfile
stepfilename = 'Stepfile_THIRD.txt'
stepfile = open(stepfilename,'w')
stepfile.write('Step                    Time\n')
stepfile.close()
# Create new input and output results file
INPUT_resultfile = 'INPUT_results_THIRD.txt'
OUTPUT_resultfile = 'OUTPUT_results_THIRD.txt'
INPUT_myoutfile = open(INPUT_resultfile,'w')
OUTPUT_myoutfile = open(OUTPUT_resultfile,'w')
# Creating input and output grid files
Fin_file = 'Fin_THIRD.txt'
Fout_file = 'Fout_THIRD.txt'
Fin_gridfile = open(Fin_file,'w')
Fout_gridfile = open(Fout_file,'w')
Fin_incre = (Fin_end-Fin_start)/(runs-1)
Fout_incre = (Fout_end-Fout_start)/(runs-1)
d_Fin = 0.0
d_Fout = 0.005*Fout_incre
Fout = Fout_start + d_Fout
counter = 1
for y in range(1,runs+1,1):
```

```
Fin = Fin_start + d_Fin
for x in range(1,runs+1,1):
        # Input Force
        mdb.models['Model-1'].ConcentratedForce(cf2=-1*Fin, createStepName=
                'Step-1', field='', localCsys=None, name='Fin'
                , region=
                mdb.models['Model-1'].rootAssembly.instances['Model_10-1'].sets['INPUT'])
        # Output Force
        mdb.models['Model-1'].ConcentratedForce(cf1=-1*Fout, createStepName=
                'Step-1', field='', localCsys=None, name='Fout'
                , region=
                mdb.models['Model-1'].rootAssembly.instances['Model_10-1'].sets['OUTPUT'])
        jobname = 'Individual_runs'
        mdb.Job(atTime=None, contactPrint=OFF, description='', echoPrint=OFF,
                explicitPrecision=SINGLE, getMemoryFromAnalysis=False, historyPrint=OFF,
                memory=20, memoryUnits=PERCENTAGE, model='Model-1', modelPrint=OFF,
                multiprocessingMode=DEFAULT, name=jobname, nodalOutputPrecision=
                FULL, numCpus=1, queue=None, scratch='', type=ANALYSIS, userSubroutine='',
                waitHours=0, waitMinutes=0)
        job = mdb.jobs[jobname]
        job.submit()
        job.waitForCompletion()
        # Writing output to resultfile
        import odbAccess
        outputodbfile = jobname + '.odb'
        odb = session.openOdb(outputodbfile)
        timeFrame = odb.steps['Step-1'].frames[-1]
        displacement = timeFrame.fieldOutputs['U']
        N_in = odb.rootAssembly.instances['MODEL_10-1'].nodeSets['INPUT']
        N_out = odb.rootAssembly.instances['MODEL_10-1'].nodeSets['OUTPUT']
        NDisp_in = displacement.getSubset(region=N_in)
        NDisp_out = displacement.getSubset(region=N_out)
        # Write output
        for v in NDisp_in.values:
                INPUT_myoutfile.write(str(v.dataDouble[1]))
                INPUT_myoutfile.write(' ')
        for v in NDisp_out.values:
                OUTPUT_myoutfile.write(str(v.dataDouble[0]))
                OUTPUT_myoutfile.write(' ')
        odb.close()
```

```
                    # Writing Input and output grid
                    Fin_gridfile.write(str(Fin))
                    Fin_gridfile.write(" ")
                    Fout_gridfile.write(str(Fout))
                    Fout_gridfile.write(" ")
                    Fin = Fin + Fin_incre
                    # Printing steptime
                    from datetime import datetime
                    stepfile = open(stepfilename,'a')
                    stepfile.write(str(counter))
                    stepfile.write("     ")
                    stepfile.write(str(datetime.now()))
                    stepfile.write("\n")
                    stepfile.close()
                    counter = counter + 1
                    #print datetime.now()
            # Adding blank line in the output file
            Fin_gridfile.write("\n")
            Fout_gridfile.write("\n")
            INPUT_myoutfile.write("\n")
            OUTPUT_myoutfile.write("\n")
            Fout = Fout + Fout_incre
# Writing increments
incrementfile = open('d_Fout.txt','w')
incrementfile.write(str(d_Fout))
incrementfile.close()
# Closing all files
INPUT_myoutfile.close()
OUTPUT_myoutfile.close()
Fin_gridfile.close()
Fout_gridfile.close()
```