6-26-2015

Modeling and Optimizing IPMC Microgrippers

Justin Simpson

Justin Simpson
Candidate

Mechanical Engineering
Department

This thesis is approved, and it is acceptable in quality and form for publication:

Approved by the Thesis Committee:

Dr. Ronald Lumia , Chairperson

Dr. Tariq Khraishi

Dr. Juan Heinrich

Modeling and Optimizing IPMC Microgrippers

BY

Justin Simpson

Bachelor of Science Mechanical Engineering

THESIS

Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

Mechanical Engineering

The University of New Mexico

Albuquerque, New Mexico

May, 2015

ACKNOWLEDGMENTS

I heartily acknowledge Dr. Ronald Lumia, my advisor and dissertation chair, for allowing me the opportunity to learn and use the research methods developed in his research laboratory. Also, I would like to thank him for encouraging me and pushing me to produce quality work. His teachings and guidance will remain with me as I continue my career.

I also thank my committee members, Dr. Juan Heinrich and Dr. Tariq Khraishi, for their recommendations pertaining to this study.

To my friend, Manuel Martinez, who helped me throughout this entire process and taught me a great deal in this area which helped me in completing this thesis.

And finally to my family, who gave me immeasurable support over the years and pushed me to do my best.

MODELING AND OPTIMIZING IPMC MICROGRIPPERS

By

Justin Simpson

B.S., Mechanical Engineering, University of New Mexico, 2011

M.S., Mechanical Engineering, University of New Mexico, 2015

ABSTRACT

A FEA (Finite Element Analysis) model was used to determine the change in performance that results from varying the size and shape of IPMC (Ionic Polymer Metal Composite) fingers. Using Comsol Multiphysics and modeFRONTIER, these fingers were modeled and optimized for both force exerted and deflection. Using the Comsol model, we were able to determine the tip deflection and force output of many different IPMC fingers which were verified experimentally. Then, using modeFRONTIER we were able to optimize the fingers to determine the best shape and area depending on whether a high force or deflection was desired.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

Chapter 1. INTRODUCTION

1.1 IPMCs

1.1.1 Discovery of IPMCs and Fabrication

An Ionic Polymer-Metal Composite (IPMC) is a type of electro-active polymer (EAP) that deflects in the presence of an electric field [1]. Electro-activity is the movement of mobile ions. The higher the electro-activity, the higher the tendency for ionic motion inside the IPMCs. These materials also produce an electric field when physically bent. IPMCs are referred to as artificial muscles, due to their ability to mimic natural muscles. These unique abilities have led to much interest for using them in many fields, including aerospace and bio-engineering. Interest in IPMC microgrippers is also growing, as these microgrippers are capable of grasping small objects without damaging them, due to their large range of deflection and small grasping force.

IPMCs typically consist of a synthetic polymer film with ionic properties, such as Nafion, that is plated on both sides with a noble metal such as platinum or gold, as seen in Fig. 1.

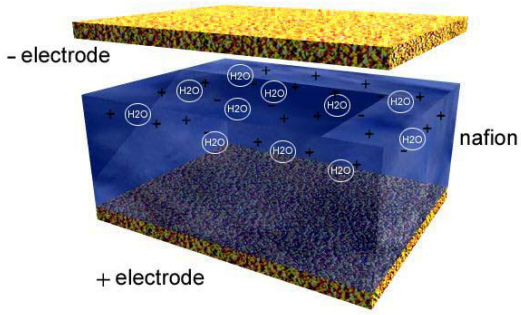


Figure 1. IPMC Configuration

IPMCs are manufactured using a deposition process to create a thin electrode, usually one to five microns thick. The electrodes are anchored to the polymer by thin dendrites created using this

process. This electrode is not a continuous metal layer; it is instead tightly packed metal grains. In the case of platinum, the metal grains are platinum salt deposits, rather than a layer of platinum. The noble metal layers store opposing charges, much like a parallel plate capacitor [2]. This depletes the anode of its positive charge and the cathode receives a high positive charge. In response, the cathode expands due to electrostatic charges. This storage of charge also causes ions in the polymer to migrate and collect on one side, also seen in Fig. 1. This migration of ions causes large water molecules held in the IPMC to migrate as well. The migration of the large water molecules to one side of the IPMC causes a swelling effect, while the other side is losing a large number of ions, causing a shrinking effect, as seen in Fig. 2 [3].

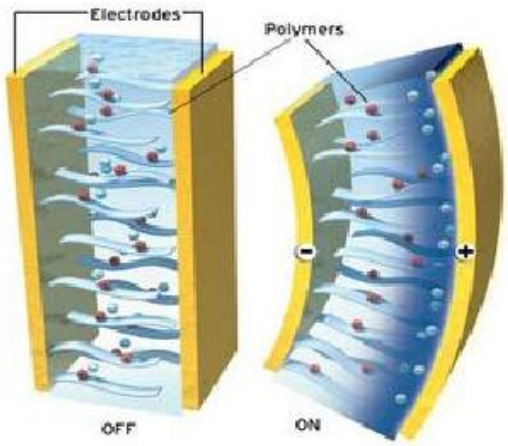


Figure 2. IPMC Water and ion migration

Though IPMCs cannot provide substantial force as an actuator, they are flexible, require low actuation voltage, conform to the object being grasped, and can operate in a wet environment. Depending on the way they are cut, IPMCs typically have a large area with a very little thickness. The area of these microgrippers is usually on the order of square millimeters, while the thickness is around hundreds of microns. The amount of metal on these microgrippers is small compared to the amount of polymer, causing the whole microgripper to act more as a

polymer than a metal. This characteristic causes them to be flexible and lightweight. Due to their flexibility and high electro-activity, they are also able to deflect large amounts with a very low actuation voltage, i.e., around 1 volt [3]. Also due to their thickness and high degree of flexibility, these microgrippers are capable of conforming to the surface of the object they grasp, as seen in Fig. 3.

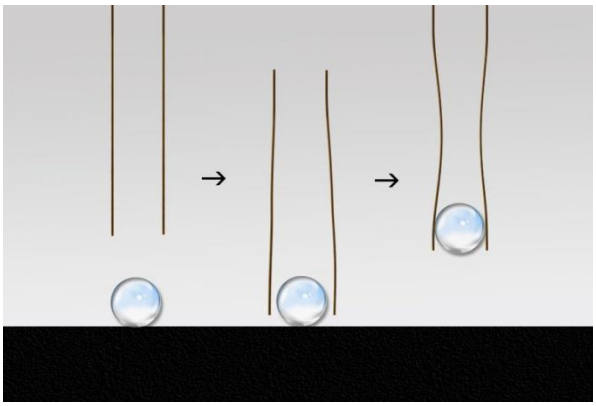


Figure 3. IPMC Conforming to objects

This is extremely important in micromanipulation of objects, especially cell manipulation. The gripper is able to securely grasp a cell without damaging it. IPMCs also do not generate high forces when grasping objects. This is a valuable characteristic, as biological cells are easily damaged by high forces. This makes IPMCs better suited for micromanipulation than other gripping technologies that are rigid and exert higher forces. As stated above, IPMCs are able to operate in wet environments and also require low actuation voltages. This allows IPMCs to be used in micromanipulation with moist cells that may be in aqueous environments. The low actuation voltage also decreases the chances of damaging the cell, due to voltage differences across the cell. All of these characteristics make them well suited for bio-micromanipulation applications in the form of microgrippers.

These actuators can be processed into certain geometries and complex shapes needed for certain tasks, using an automated laser cutter and CAD software. This laser operates at a cutting

frequency of 532 nm, which ablates the platinum layers. This technique is far superior to early techniques, in which scissors or a scalpel were used. Using a laser, we can make complex shapes and can cut the IPMC to any size. This is extremely important when making small actuators for micromanipulation.

1.1.2 IPMC Actuation Abilities

Although the migration of ions and solvent, due to an induced voltage, is considered to be the driving force behind actuation of IPMCs, many are still researching the chemical structure and physical mechanisms associated with actuation. The model most widely used to describe the chemical structure of Nafion was introduced by Hsu in 1981. This model was based on wide-angle and small-angle x-ray diffraction studies. When in the hydrated state, Nafion displays phase separation and forms two distinct regions, hydrophilic and hydrophobic. Gierke et al. [4] described the hydrophilic regions as 4 nm spherical inverted micellar structures that are separated by a distance of 5 nm and connected by 1 nm diameter micro-channels, as seen in Fig. 4.

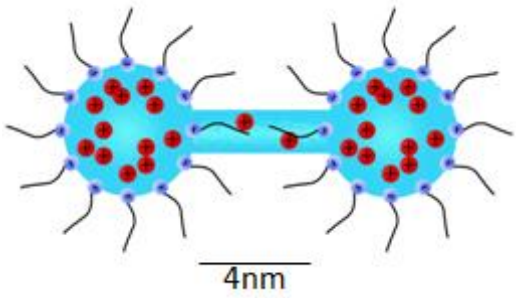


Figure 4. Micelle connected with channel

These clusters form a cubic grid called the cluster-network morphology. As can be seen in Fig. 4, the solvent and cations are contained in the micelle and channel. In an IPMC, the hydrophobic region is made up of the fluorocarbon chains of Nafion.

Hydration level and type of solvents and cations being used have a great effect on IPMC actuation [5]. When a voltage is applied to an IPMC, an electric field is set up through the thickness of the Nafion, which produces an electrostatic force on the cations. These cations are then driven through the channel, as seen above. Depending on the level of hydration and type of solvent being used, the cations may migrate easily. A fully hydrated sample will allow more migration than a dehydrated sample. This becomes quite evident on the macroscopic level when testing different samples, as only hydrated samples are capable of movement. Nemat-Nasser and Li proposed a model that demonstrated the increase in concentration of cations at the cathode, resulting in a fast expansion [6]. Solvent molecules attached to cations also migrate towards the cathode. The combined migration of solvent and cations results in a fast actuation response towards the cathode that may last several minutes. This migration of cations towards the anode leaves a depleted region of cations in the anode. These clusters slowly redistribute, causing a decrease in actuation, known as back relaxation. The anions that migrate towards the anode during this process repel one another, also causing relaxation. This relaxation causes a bending back towards the anode.

The model used in this thesis assumes the actuation of the IPMC is explained by the electrostatic interaction between the micellar clusters only, meaning the cations in anode clusters migrate to the clusters in the cathode. The anode will be void of cations, resulting in negatively charged clusters, which repel each other. Likewise, the cathode will be filled with cations, which also repel each other. Although it may seem both sides, anode and cathode, will have positive pressure, it will be seen that the force production on the cathode is much higher than that of the anode. This causes actuation towards the cathode.

1.1.3 IPMC Sensing Abilities

While most IPMCs are used as actuators, they can also be used as sensors. As stated above, they produce an electric field when physically bent. Nemat-Nasser and Li suggest the imposed deflection causes production of stress in the backbone polymer which leads to displacement of charges in the micelle clusters [6]. When relaxed, the cations and anions in either electrode are balanced. When deformed, the cations are shifted according to the magnitude of deformation. Hydrostatic pressure, caused by the stress in the backbone polymer, may also cause the flow of water and cations from high pressure regions to regions of low pressure. Sadeghipour et al. used IPMCs as hydrogen pressure transducers in 1992 to make a smart accelerometer in machinery [7]. This transducer was an IPMC held between two electrodes that transmitted a voltage when squeezed. In contrast to actuation, the IPMC can be used in sensing abilities whether it is wet or dry. It was later shown that IPMCs work better as sensors when they are dry [8].

1.1.4 Simultaneous Actuator/Sensor

Attempts have been made to develop a device that is capable of combining both actuation and sensing capabilities in an IPMC. An IPMC "sandwich" was created in which two IPMC fingers were cut to the same dimensions and glued to each other. The actuator was a 200 micron thick finger, while the sensor was 60 microns thick. This was done as thinner fingers are better for sensing capabilities and thicker fingers are better suited for actuation. When actuated, the thicker finger would deflect the thinner finger, which generated a small voltage. This made it possible to track the movement of the "sandwich." When actuated, the thicker finger created an electromagnetic field which was detected by the sensor. The voltage created by the actuator was much higher than the voltage created by the movement in the sensor, so the readings were

incorrect. A layer of gold leaf connected to ground was then placed in between the sensor and actuator, which rid the system of interference, but led to rigidity. Recently at the University of New Mexico, researchers have made a simultaneous sensor/actuator by cutting one surface of the IPMC into two electrically separate components, as seen in Fig. 5.

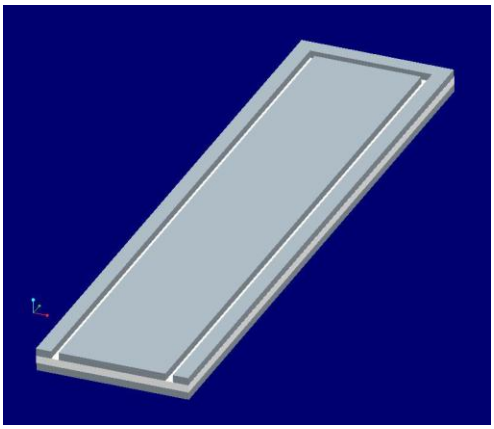


Figure 5. Channel cut by laser

This finger is capable of actuation, while detecting its displacement. The channel is created by removing small amounts of electrode on either side, leaving the polymer layer intact. As stated above, the electrodes are not solid metal; rather they are tightly packed metallic clusters. These clusters may become more tightly packed or may separate when the polymer bends, which leads to a change in the resistance of the electrode surface. This change in resistance can be measured which can then be used to determine the amount of deflection in the IPMC. This design is far superior to the "sandwich," as only one IPMC finger must be cut, reducing waste of material. The channel also does not add rigidity to the entire structure, leading to greater actuation. The size of the channel may also be made very small, reducing waste of actuation abilities. These channels are also easy to cut using the mentioned laser cutting technique.

1.2 IPMC Microgrippers

1.2.1 Microgripper and Simultaneous Sensor/Actuator

A microgripper consists of two IPMC fingers used for actuation that are held together with a holder connected to a power supply, as seen in Fig. 6. The two fingers of matching size will be situated with a slight gap in between them. They are fixed in the holder as cantilever beams. The holder has electrodes that connect at the base of the fingers so a voltage or current signal can be applied.

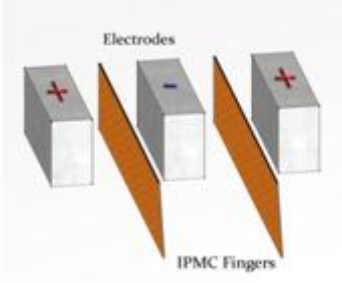


Figure 6. IPMC Microgripper

As can be seen, the two fingers will actuate towards the middle, as the cations will flow towards the cathode. In most cases, a small negative or positive voltage is applied to one face of the electrode, usually 2 volts, as a high voltage may damage the finger, while the other face is connected to ground. When the voltage is applied, the fingers will actuate, closing the gap in between them, grasping any object located in this gap.

Attempts have also been made to assemble the simultaneous sensor/actuator. This device is much more complex than single finger grippers, as the change in resistance on the surface of the electrodes must also be measured. The current size and shape of the simultaneous sensor/actuator can be seen in Fig. 7.

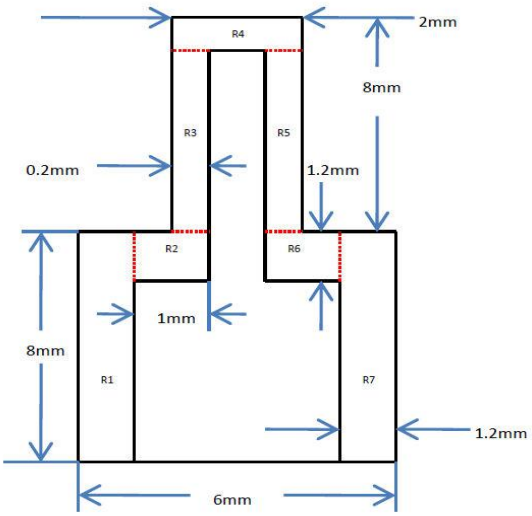


Figure 7. Dimensions of Sensor/Actuator

The bottom half of this finger is considered to be the key. This key is the area held by our custom made holder. This holder requires careful detail as three wires must be attached to each side, as seen in Fig. 8. The device was made to hold the fingers while ensuring there is good contact between the fingers and copper leads that must be attached to the holder.

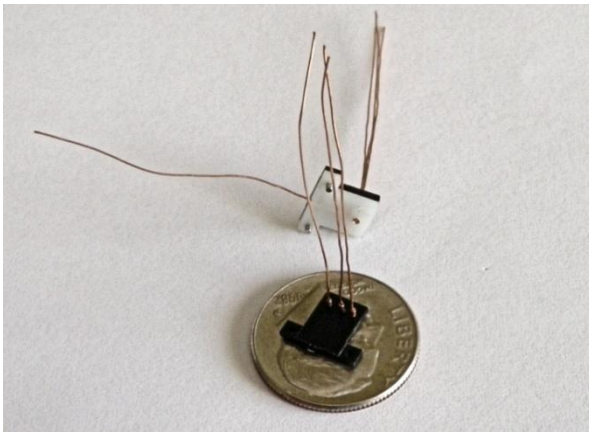


Figure 8. Custom Holder

1.2.2 Manufacturing IPMC Fingers

Early IPMC research consisted of simply cutting out any IPMC finger and testing it. This cutting was done using scissors or a scalpel. IPMC material is expensive and this cutting

procedure is wasteful. In order to cut complex and precise geometries and also to cut the channel mentioned above, another method had to be installed. This precision was achieved by installing an IPMC cutting workstation that is run by accurate motion software and connected to a Nd: YAG laser (Signatone 1160) with a green light frequency of wavelength of 532 nm, as seen in Fig. 9. This ablates the noble metal (platinum) leaving the polyelectrolytic membrane intact. Nafion does not absorb the green light, allowing the laser to penetrate through both sides of the platinum on the IPMC.

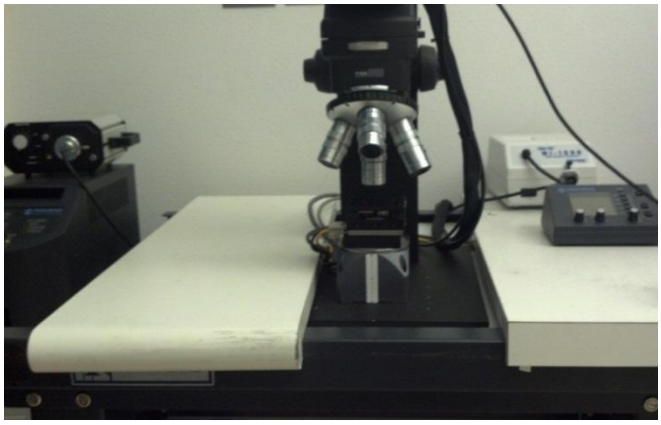


Figure 9. Signatone 1160 Laser Cutting Station

The laser cutting station is equipped with a set of Parker MX80L linear programmable stages, seen in Fig. 10. These stages are connected to a LabVIEW computer that tells the stages where to move, as seen in Fig. 11. The geometry of any IPMC finger can be made in a CAD program and can be exported to NI motion software that converts the CAD model into motion profiles that will be executed by the stages. These profiles will be stored in LabVIEW codes, which will run the stages and the laser.

Figure 10. Parker Microstage

Figure 11. Cutting Station Connections

This technique is far superior to previous techniques of cutting an IPMC free hand with a scalpel. Using this technology, we can cut any size and shape of IPMC desired. Although the Nafion is still intact, we are able to remove the microgripper easily with a scalpel, as the laser leaves identifiable channels that are easily traced. This is also important in increasing quality and number of cuts needed, decreasing waste of expensive IPMC material.

Because the Nafion stays intact, simultaneous sensor/actuators are possible, as the channel can be cut in between the two. Other features can also be cut into IPMC fingers, such as lines, to mimic real hands or fingers. This may be done by cutting several lines into one IPMC piece to make several free fingers that can be actuated individually. A twisting motion in IPMC

fingers may also be accomplished by cutting individual fingers in a larger piece and applying different voltages and signals to the individual fingers.

1.2.3 Microgripper Holders

As stated above, there must be a way to hold these fingers while making a connection with the surface of the electrodes. This is accomplished using specialized holders, usually made in house. These devices are made using manual techniques, such as rapid prototype processes or by modifying electrical components, as seen in Fig. 12 and Fig. 13.

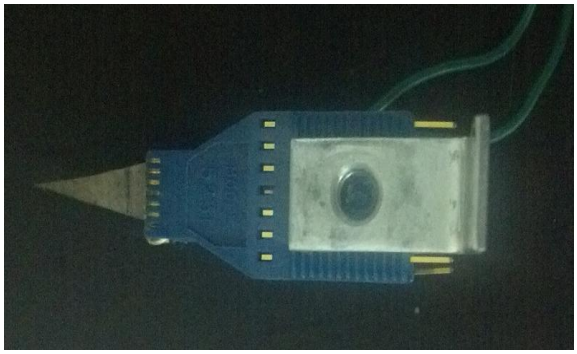


Figure 12. Finger in electroded holder (Top View)

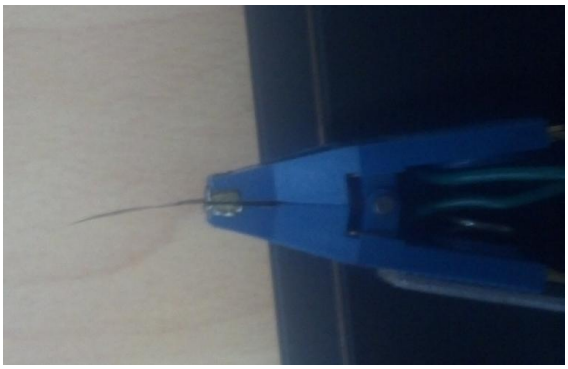


Figure 13. Finger in electroded holder (Side View)

These holders are constructed using modified IC test clips with copper plates attached to the clamping area on either side of the IPMC finger. In the beginning, the geometry of these fingers was quite simple, so the simple IC clips were sufficient. These clips are also only capable of holding and actuating one finger. The purpose of these fingers is to be able to grasp an object,

meaning two fingers need to be actuated simultaneously. In the case of the simultaneous sensor/actuator, the geometry and connections are very complex, making the holder hard to manufacture, also making it expensive. This holder was created using rapid prototyping technology.

1.2.4 IPMC Control

In order to control these IPMCs and ensure repeatability, a control system was established. Using the electroded holder seen above and supplying signals output by a computer running LabVIEW interfaced with a National Instruments DAQ board, we are able to ensure repeatability and control. As can be seen in Fig. 14, LabVIEW and the DAQ board are used in most of the processes.

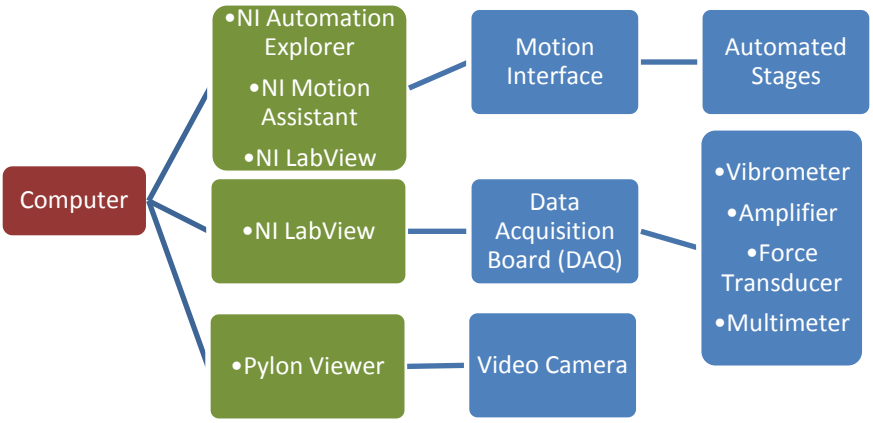


Figure 14. Computer Connections

This system is capable of supplying output voltage to the IPMC fingers, as well as taking readings given by the vibrometer and force transducer. These output voltage signals are sent as analog signals to the electrodes connected to the IPMC finger, causing actuation. Multiple signals can also be sent to segmented fingers in order to cause twisting or different movements.

Current research has been focusing on using a control loop to maintain a certain position, accounting for back relaxation. This control loop actuates the IPMC finger while simultaneously taking resistance readings in the finger to determine if the finger is relaxing and will supply additional voltage in order to correct for the back relaxation that occurs naturally. This is beneficial as the finger can maintain its position for an extended period of time.

Using this equipment, an IPMC microgripper robot was also created. This IPMC microgripper was attached to the moving stages and programmed to move to certain positions, actuate, and grasp objects.

1.3 IPMC Force Model and Force Scanner

1.3.1 Force Scanner Overview

Recently at UNM, a force scan of multiple IPMC fingers was created. This scanner was used to create topographic maps of various fingers and their gripping strengths at numerous positions, as seen in Fig. 15. These maps were important as they were compared to modeled force scans. As can be seen, the force is very high at 0 mm, where the IPMC is connected to the electroded holder, and small towards the end of the IPMC.

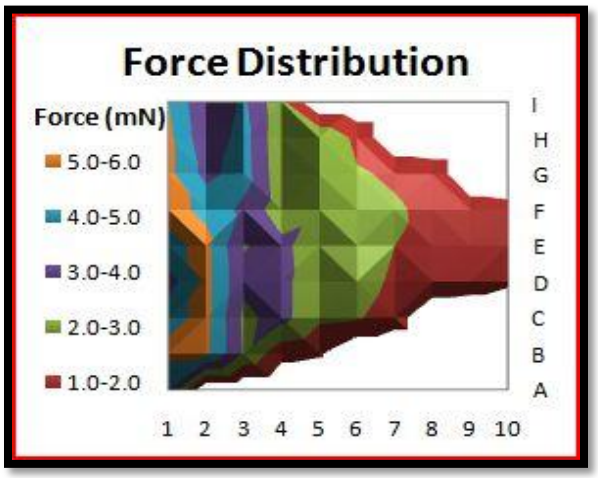


Figure 15. IPMC Force Scan

This scanner was created using the custom holder explained above. The IPMC finger was inserted into the custom holder which is attached to the Parker microstages. Using an Aurora Force Transducer, the force applied by the finger was measured, as seen in Fig. 16.

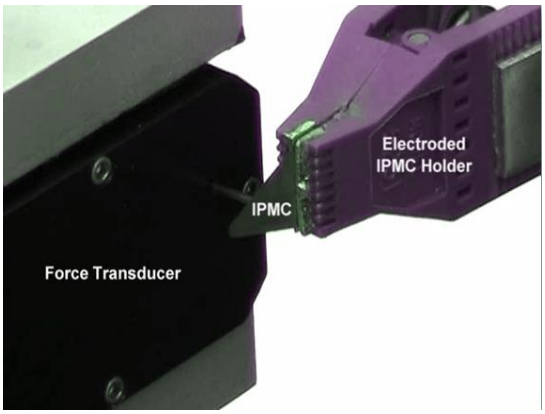


Figure 16. IPMC and Force Transducer

LabVIEW is used to control the whole system. Once in place, LabVIEW outputs a voltage signal to the finger so it actuates into the force transducer and the transducer outputs a signal to LabVIEW giving the force applied. LabVIEW then outputs a signal to the microstages forcing them to move a certain distance, and the process is repeated multiple times, as seen in Fig. 17. The forces applied are then plotted in a 3D map, which is shown above.

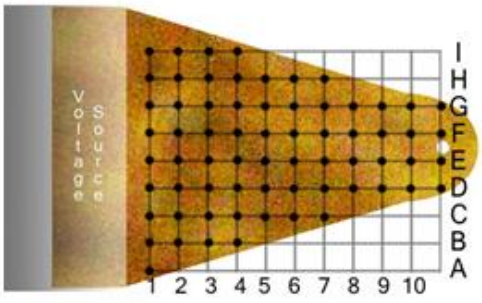


Figure 17. Testing Points

1.3.2 IPMC Force Model

An IPMC Force Model was then created using Comsol Multiphysics and Matlab. Comsol is a type of FEA software used for simulations of coupled physical phenomena. Comsol uses CAD modeling to make objects and attributes can be added to the CAD model such as structural mechanics and electrical conduction. This model will be explained in much greater detail later, but the impact of the model will be discussed here. This novel model was used to make topographic force maps of several IPMC fingers and was compared to the experimental results. This model was quite accurate when compared to the experimental results. Although in its early stages and somewhat burdensome to run, the model worked. The results of one test can be seen in Fig. 18. This model is important as it is the first of its kind. Using this model, any IPMC finger can be created and tested to see if is capable of achieving a desired force. Simulation is much more beneficial compared to experimentally testing potential designs to see if they are suited for the task. Experimentally testing wastes both time and money, while the simulation is relatively fast and does not waste material.

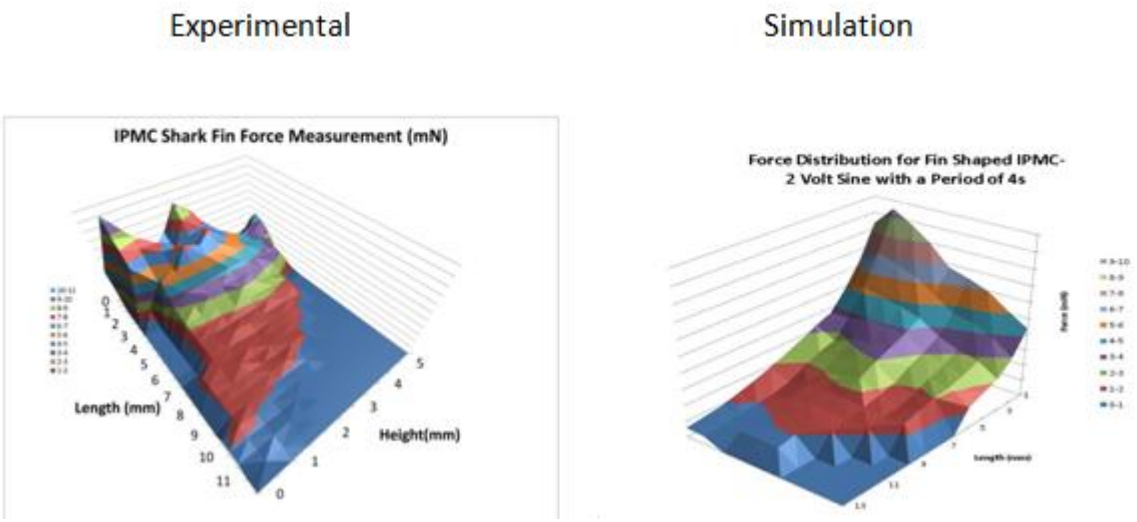


Figure 18. Experimental vs. Simulation

While using this model to experiment with other finger shapes, it was found that different sizes and shapes have drastic changes in results. Most research focuses on simple rectangle fingers. Shapes such as triangles were not of much interest. Using this model, right triangles and isosceles triangles were modeled and then tested experimentally to determine the validity of the model. The results were quite surprising. Two different tests were used to compare the fingers. First, the finger output vs. size was modeled and tested. To do the experiments, the force measurements were the same as above, except only a single point was measured 2 mm in from the tip of the finger. The deflection measurements have a different process. Using LabVIEW, a square wave signal of 0-2 volts is applied. Using a Polytec laser Doppler vibrometer (Polytec model # OFV-551 & OFV-5000), the deflection of the IPMC is measured. The IPMC is held perpendicular to the vibrometer's sensor head with the electroded holder. The IPMC starts out straight and deflects when the voltage is applied. The vibrometer measures the amount of displacement throughout the process and the total displacement of the IPMC is measured. Two fingers were modeled and tested experimentally. One was a 7x17 mm rectangular finger and the other was an isosceles triangle with the same height and base but half the area, as can be seen in Fig. 19.

Figure 19. Rectangle and Triangle Fingers

The results can be seen in Table 1. The error in force and deflection for the rectangle is 23.37% and 2.77%, respectively. The error for the 7x17 mm triangle for force and deflection is 13.3% and 4.92%, respectively. As can be seen, the force output of the triangle is roughly half the force produced by the rectangle. This can be explained by the difference in area. Decreasing the area by half should have a proportional decrease in force produced, as force is a function of the area of IPMC fingers. The deflection for the triangle is smaller than the rectangle, but not quite half. Although it has half the area, the triangle has less material than the rectangle and is able to deflect easier.

Table 1. Output vs. Size

Gripper Dimension	7x17 mm rectangle	7x17 mm triangle
Experimental Force	2.0 mN	1.16 mN
Experimental Deflection	1.4 mm	1.13 mm
Modeled Force	2.6 mN	1.22 mN
Modeled Deflection	1.44 mm	0.98 mm

Next, the finger output vs. shape was tested. The same 7x17 mm rectangle was used and new 14x17 mm isosceles and right triangles were used, as seen in Fig. 20.

Figure 20. Rectangle and Triangle Fingers

The results can be seen in Table 2. The error in force and deflection for the isosceles triangle is 14.38% and 23.5%, respectively. The error in force and deflection for the right triangle is 13.96% and 6.26%, respectively. As can be seen, the force applied by the isosceles triangle is higher than the force exerted by the rectangle, but the force exerted by the right triangle finger was similar to the rectangle, while the deflection was higher than the other two. These results are very surprising.

Table 2. Output vs. Shape

Gripper Dimension	7x17 mm rectangle	14x17 mm Isosceles triangle	14x17 mm Right triangle
Experimental Force	2.0 mN	2.92 mN	2.28 mN
Experimental Deflection	1.4 mm	1.5 mm	1.85 mm
Modeled Force	2.6 mN	2.5 mN	2.65 mN
Modeled Deflection	1.44 mm	1.96 mm	1.735 mm

These tests prove there are many ways to design fingers depending on the design goals. If a large force is desired, an isosceles triangle finger may be better suited, but if deflection is needed, a smaller triangle is capable of deflecting a high degree compared to the rectangle finger. This lead to the conclusion that an optimization package is needed to better determine the best way to design these fingers.

1.4 Purpose Statement

Although there have been many advances in IPMC technology, there is still a disconnect in modeling and understanding the exact behavior of these IPMC microgrippers [6]. The models that do describe an IPMC are insufficient in predicting force output and actuation. Although

they are capable of predicting some behavior, they are not capable of predicting the behavior of any arbitrary size and shape of microgripper [8]. A model that predicts the behavior of any IPMC is beneficial to create IPMC fingers that meet design goals. This model also needs to be able to determine the best design for a given finger in order to reduce waste. An optimization package incorporated with the current model would be beneficial to research, as a finger can be modeled using different parameters. The current model can determine force applied by the finger, but it does not determine the best way to design the finger. An optimization package could test many different ways to design a finger, depending on the desired force, deflection, or both. These can be given as design goals and the model will change according to these goals. This will optimize the finger while reducing waste. The model can also be scaled up or down to determine the effects of larger or smaller fingers to determine the relationship between size and force or deflection. The model will also be used to focus on small IPMC fingers, where there is no evidence whether they can produce a suitable force with very small dimensions [9]. Using an optimization package, this can be done very easily, as the process will be done autonomously as long as the model has design goals.

1.5 Contribution

This model will be a more cost effective way to predict the force output and deflection of any arbitrary microgripper than simply cutting and testing. Many users currently cut out a microgripper before they have any knowledge of its force output or deflection and test it, which increases the cost and wastes material if the microgripper does not produce the desired characteristics. The model will eliminate the aspect of guessing if the microgripper is suitable and eliminate unnecessary cutting involved in microgripper research, as the microgripper will be modeled before it is cut out. It will also lead to optimization as users will be able to design

microgrippers for certain purposes and determine whether the gripper they designed was the easiest and least wasteful way to achieve the actuation force or deflection desired. This model will use Comsol Multiphysics and Matlab to determine the force and deflection of any IPMC finger. The existing model will be simplified into one process. The existing model takes many sub-models to get to the final design. The new model will be made into one process that is very easy to use. The new model will also incorporate optimization software. ModeFRONTIER is a powerful design package that is capable of optimizing IPMC fingers given many design parameters. Using this model that incorporates all of the steps into one model, we can change the width and length of any IPMC finger and determine the effects. Also, we can limit the area of the finger to reduce waste and still meet design goals. These design goals will be deflection, force, and both. The user will be able to input a desired force or deflection and the model will optimize for the smallest area. The model will also determine the best size and shape to meet these design goals. This process will be easy to use, as the model will have distinct inputs, such as length, width, desired force, etc. This model will be beneficial as many iterations will be run to determine the best design, instead of designing an arbitrary finger and testing to see if it meets the goals.

Chapter 2. LITERATURE REVIEW

2.1 Actuation

Discovered in 1992, the actuation properties of IPMCs have become more understood and advanced through the years. These properties were learned by Oguro et al. in 1992 when bending them under applied voltages [10]. Because of this discovery, many others experimented to explain the dynamics behind the actuation properties. Shortly after, dehydration was observed by Kanno et al. when the IPMC stopped actuating after a certain amount of time [11]. This led to a belief that water molecules with mobile ions were responsible for the electric response in IPMCs. More evidence of this dependence on hydration was observed by Bar-Cohen when an IPMC was actuated underwater hundreds of thousands of times with little change in response [12].

Greater understanding of the mechanisms behind actuation was continually being discovered. One discovery was that the polymer contained in IPMCs consists of mobile cations and water that move when the outer metal layers are charged [3]. These water molecules bond to the positively charged cations and migrate towards the cathode. This migration is responsible for some of the actuation in the material, but many believed it did not account for the fast response. It was soon discovered that Coulombic forces between the charges in the electrodes were responsible for this fast reaction. These forces caused the migration of hydrated cations towards the cathode, causing the IPMC to swell towards the cathode side and shrink on the anode side [13].

Researchers also experimented with other solvents to determine their effects. Nemat-Nasser and Wu demonstrated that sodium ions worked better than lithium and hydrogen atoms when using Nafion based IPMC [14]. They used different cations to change the force and

displacement of different IPMCs. The IPMC containing sodium ions produced a greater force than the others.

Once researchers began to understand the driving force of IPMCs, they began experimenting to improve the material. Shahinpoor also experimented with many different polymers. One problem with the IPMC material was attaching the polymer in between the two electrodes to create the sandwich. Nafion is commonly used as the polymer, but it was difficult to attach as it was non-reactive. A solution was achieved when chemical etchants were used on the Nafion before it was bonded to the metal electrodes. This etching produced microscopic ridges in the Nafion that the metal electrode was able to attach to. The metal formed dendrites that were able to anchor in the Nafion [15]. This also led to an interest into the structure of the metal electrodes. Shahinpoor and Kim studied these metals and their effects on the IPMC [16]. They observed that the surface morphology of the IPMC is characterized by a granular nano-roughness of the order of approximately 50 nm. This characteristic is responsible for producing a high level of electrical resistance, yet provides a porous layer that allows water movement in and out of the Nafion.

Due to their dependence on hydration, IPMCs have some limitations. Hydrolysis is a very common occurrence in IPMCs when the voltage is raised above 1.23 volts [17]. Once it is raised above this limit, hydrolysis occurs very quickly causing the IPMC to desiccate, producing hydrogen gas at its electrodes, thus losing its actuation capabilities. Hydrolysis is common in air as there is no shielding to contain the hydration once the voltage is high enough. When the IPMC is actuated in water, hydrolysis is not a concern as the IPMC will continue to be hydrated. Although hydrolysis occurs at a low actuation voltage, the IPMC is still able to actuate with

voltages smaller than 1.23 volts. This is especially important in micromanipulation, as a high voltage may damage cells.

2.2 Sensing

Sadeghipour et al. learned of the sensing capabilities of IPMCs while using them as hydrogen pressure transducers [7]. Due to this, researchers became increasingly interested in understanding and explaining the science behind the sensing capability of IPMCs. This capability lies in the presence of the mobile charges contained in the negatively charged polymer. When an IPMC is deformed, one side of the polymer compresses causing the negative charges to become more compact. Due to the compression on one side, the other side simultaneously expands, spreading the molecules. As a result, the mobile cations migrate to the expanded side due to a decrease in concentration. Sampling the electrodes, a voltage difference is noticed as cations and water molecules are moving. Although these characteristics seem similar to actuation, it has been proven experimentally that the voltage produced when being physically deformed would have to be amplified by two orders of magnitude to actuate the same piece to the desired location [9]. Although this signal is smaller in magnitude, it is still very useful for sensing. This may prove useful in biomedical and engineering purposes, as IPMCs can be used as a simultaneous sensor/actuator.

Several design applications have been proposed that utilized the IPMC as simultaneous actuator and sensor. Brunetto et al. [18] have proposed the use of IPMCs in a cantilever configuration as a vibration sensor. Their prototype consisted of a system that imposed vibration to the base of a cantilever, and used a circuit to measure the tip deflection with respect to the base.

Bonomo et al. [19] presented a prototype of a tactile sensor for biomedical applications

that utilized an IPMC as both an actuator and sensor. Their sensor was able to detect both contact force and the relative hardness of the tissue compared to a control sample. The actuator IPMC in their prototype was used to bend the sensor membrane around an object. When an object came into contact with the system, it limited the actuator vibration amplitude which resulted in the sensor output signal decreasing. The actuator vibration amplitude also depended on the stiffness of the object. Stiffer objects allowed for no deflection of the actuator, whereas less stiff materials resulted in an amplitude of vibrations that was proportional to the stiffness. Although their sensor output signal was affected by noise, the actuator was able to deform objects that had a Young's modulus under 1kPa.

As stated above, these IPMCs can be used as simultaneous sensors and actuators. Preliminary research consisted of two IPMC grippers glued to each other. One IPMC was used to sense its deflection, while the other actuated. Although it worked, the extra stiffness of the added sensor hindered the actuation of the whole system. More advanced combinations were soon discovered by Kruusamae et al. [20] when they discovered a channel can be cut in the IPMC surface creating two separate components. This led to a microgripper that is capable of actuating when given an input voltage, which causes a change in resistance in the outer channel. This change in resistance can be used for sensing. These advances led to a need for better understanding and modeling these complex IPMCs.

2.3 Modeling

Most early models to explain macroscopic qualities were based solely on experimentation. Researchers take the experiments and fit equations around their results to obtain their models. Kanno et al. developed a model that described the relationship between the input voltage and the change in current [11]. They were also able to relate the change in current

to deformation. They described the model of an IPMC as having three phases: mechanical, electrical, and stress generation. This model applied a simple circuit that resembled a RLC circuit, but this model only described what happened at the initial actuation. These models are accurate in describing the immediate response of the sudden increase in the current, but new models aim to predict the effects of transient behavior. Current theories are trying to describe this transient behavior, but it is very complex as it is due to a chemical-electrical-mechanical reaction. These models also do not incorporate the sensing capabilities of this material. Although researchers understood actuation was caused by movement of ions, there was no clear explanation behind the dynamics of sensing.

Many researchers are actively trying to model and understand IPMCs as artificial muscles. Using these models, researchers have discovered many important attributes, including material strains, Coulombic forces, current induced, and transportation of ions. Although these models have led to better understanding, there is little knowledge of the interactions between the forces inside the IPMC. This is apparent as there are many different theories as to what causes the actuation. Nemat-Nasser has developed models to understand this behavior using the material properties of these IPMCs, including mechanical properties, electrodynamics, and chemistry [21]. His models led him to believe that actuation is due to electrostatic forces that exist due to the redistribution of charges. The areas high in cations will swell, while the other side with fewer will shrink. He believes the swelling occurs due to electrostatic interactions rather than the migration of cations and water. Branco and Dente also developed a model using electric field distribution and other electromechanical properties [22]. Their work was based on Shahinpoor and Nemat-Nasser, but they also neglected hydration and considered actuation to be

caused by electrical effects of mobile ions. Many assumptions are made in these models and they only apply to the gripper being tested.

For these reasons, most researchers simply use closed loop control of IPMCs to stabilize the force output. Bhat and Kim [23] acknowledge open-loop position and force responses are not repeatable. Therefore this closed-loop control is critical in ensuring repeatability and reliability.

Currently, many researchers are using finite element methods to predict IPMC behavior. Lopes and Branco successfully modeled large, simple IPMCs and compared them to experimental results [24]. They used Comsol to determine their displacements when subjected to an induced voltage. Their model bases actuation on the repulsive electrostatic forces that exist between anions in the IPMC. Pugal was also able to successfully model the electrokinetic migration of ions in Comsol [25]. This simulation was able to model the tip displacement of an oscillating actuator. Pugal et al. also used Comsol to determine the instantaneous electric current induced by a charge in a FEA model using the Ramo-Shockley theorem [26]. They studied the effects of ionic motion on electrode voltage and current. Although the model worked, deformation studies were difficult to accomplish, as meshing techniques led to extreme computational load.

Current methods at UNM are also using finite element models to understand and predict the behavior of IPMCs [27]. This method starts with a model that predicts ionic concentration in an IPMC given an input voltage. The concentration is predicted starting in Comsol, where a box is modeled with ion transport on the surface due to a given input voltage. It is then exported into Matlab to predict the concentration throughout the whole box which represents an IPMC. Matlab is then used to create a grid to encompass any IPMC. This grid is needed as the model

aims to predict the behavior of any IPMC. These models are then used to determine the stress concentration in the modeled IPMC to use in the distributed force simulation. This stress distribution is then imported into the model, representing the stresses between ions, which cause deflection in the IPMC. Then, a model is created where a force transducer is in contact with an IPMC actuator. This simulation will then predict the force of the IPMC on the transducer. The model is then exported into Matlab where it is programmed to run iteratively to model different locations where the transducer may be placed.

This model is capable of predicting the force generated by an IPMC on an object. This model has not been used to test very small microgrippers. There is a belief that with very small microgrippers there will be a change in behavior. The current model being used at UNM will be expanded upon and simplified in order to predict the behavior of small IPMC microgrippers and will be made into a single model that will be much easier to learn and use than the previous model currently being used at UNM. Optimization software, modeFRONTIER, will also be added to the model in order to optimize fingers for force or deflection.

Chapter 3. IPMC FORCE MODEL

3.1 Diagram of Models

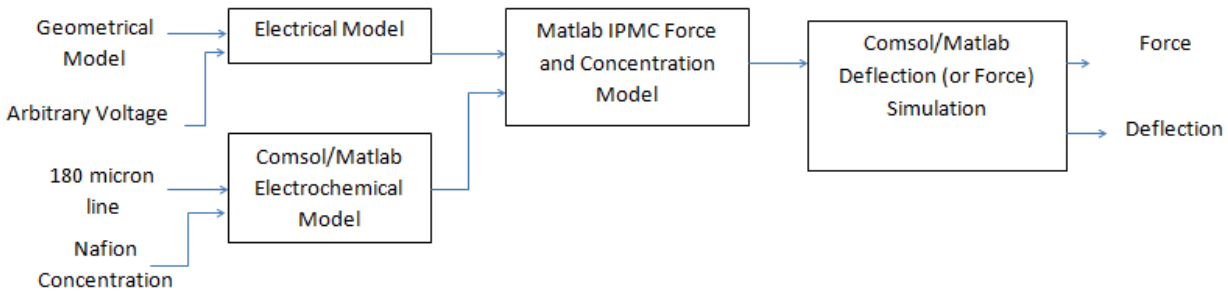


Figure 21. Smaller models incorporated in complete model

3.2 Approach Overview

As can be seen in Fig. 21, the current model consists of several different sub-models that must be combined to make the complete model. These models correspond to the coupled electrochemomechanical processes that are responsible for the actuation of a cantilevered IPMC finger. The end result will be a computer simulation of the force or deflection of any finger. The complete model that consists of the smaller models will be made into one complete code. This must be done as modeFRONTIER must interface with the code, and this code must have the complete model in order to change the model parameters.

All the computer models will be made in Comsol Multiphysics and Matlab. Comsol is a type of FEA software used for simulations of coupled physical phenomena. Comsol uses CAD modeling to make objects and attributes can be added to the CAD model such as structural mechanics and electrical conduction. After any model is completed, it can be represented graphically in Comsol. Comsol is also interfaced with Matlab. Comsol is built to run with

Matlab, meaning any model built in Comsol can be saved as an M-file in Matlab and edited. These models can be used to run simulations iteratively to allow for greater design capability.

The complete model will predict the force output and deflection. It will use the solvers mentioned to predict the electrochemomechanical transduction processes that cause the actuator to deflect. The smaller models will consist of a geometrical, voltage, concentration and force distribution. The complete model will be interfaced with modeFRONTIER and many different optimization tests will be run. These optimization tests will include optimizing a finger to achieve the highest force, the highest deflection, or the combined highest force and deflection, all in different test runs.

Section 3.3 will cover the process of creating domains, i.e., the IPMC finger. Section 3.4 covers the process of meshing domains in Comsol. Section 3.5 will cover the process of creating the smaller models, as seen below. This will cover the process of using the electrical model and using the migration of ions in Comsol to determine the concentration of cations and anions in the Nafion when applied with a voltage. Using the concentration of these ions, the force at any point is determined using a novel force equation. This force inside the finger is then incorporated in Comsol and used to determine the behavior of the finger. Below is a brief outline of the complete process.

- 3.5.2 Comsol/Matlab Electrochemical Model

  This step creates a base model that yields the ionic concentration through the thickness of IPMC as a function of input voltage and time. This model contains a single line that is easily meshed. The model provides an accurate representation of the migration of ions due to a given voltage signal.

- 3.5.3 Electrical Model

  This step contains the creation of an electric model that predicts the voltage distribution throughout an arbitrarily shaped IPMC finger based on the input voltage. This involves modeling and solving the IPMC finger in Comsol and then exporting the solution to Matlab in order to extract the voltage at any given point in the finger.

- 3.5.4 Matlab Force Model

  The results of 3.5.2 and 3.5.3 will be saved as m-files and opened in Matlab. Based on the voltage distribution, the ionic concentration distribution is predicted for any IPMC finger. Given this concentration, Matlab is then used to predict the stress field throughout the material. Lastly, the stress field is saved as a text file to be imported back into the Comsol model.

- 3.5.5 Comsol/Matlab Force and Deflection Simulation

  Two simulations will then be created in Comsol using the stress field predicted by the force model in Matlab. The first simulation is simply the deflection of the finger when modeled as a cantilevered beam. The second simulation is a stress strain simulation. This simulation involves the force experienced by a force transducer in contact with the IPMC finger.

3.3 Geometrical Modeling

The process begins with the geometrical modeling of any IPMC in Comsol, as can be seen in Fig. 22. This requires the creation of three domains to represent the Nafion polymer and two metal electrodes. Using the modeling tools, any shape desired can be created. The user can choose to model simple blocks using the block function, or can make complex shapes with the Bezier polygon tool, depending on the desired shape of their finger. First, the finger must be modeled. This can be any complex shape, but for this study only rectangles and triangles are used. As mentioned earlier these shapes had quite different results. An application named "Comsol 4.3b with Matlab" can be seen on the Desktop and must first be opened so the two programs will communicate with each other and any model can be saved as an m-file and opened in Matlab. Comsol 4.3b is then opened and a 3D space dimension is chosen. Then the user must specify the physics to be used. To start, the electrical currents (ec) under AC/DC will be used. The electrical modeling will be explained later, but it is easier to add physics before the IPMC finger is created. The geometry is then created using a variety of CAD tools. In the case of the rectangle finger, a simple block is used. As seen in Fig. 22, a 7x17 mm block is created using the block geometry and given a thickness. Two identical blocks will then be created and placed on top of the block already created. This is done in order to model the three layers of an IPMC finger. The three layers must also be offset by the thickness of each layer to make a sandwich. As stated above, an IPMC is made of three distinct layers. In our case, we have two layers of platinum and one thicker layer of Nafion. The IPMC material will then be specified by the user. Comsol has a variety of built-in materials in the material library. The user can also change the properties of the material, if desired, as can be seen in Fig. 23.

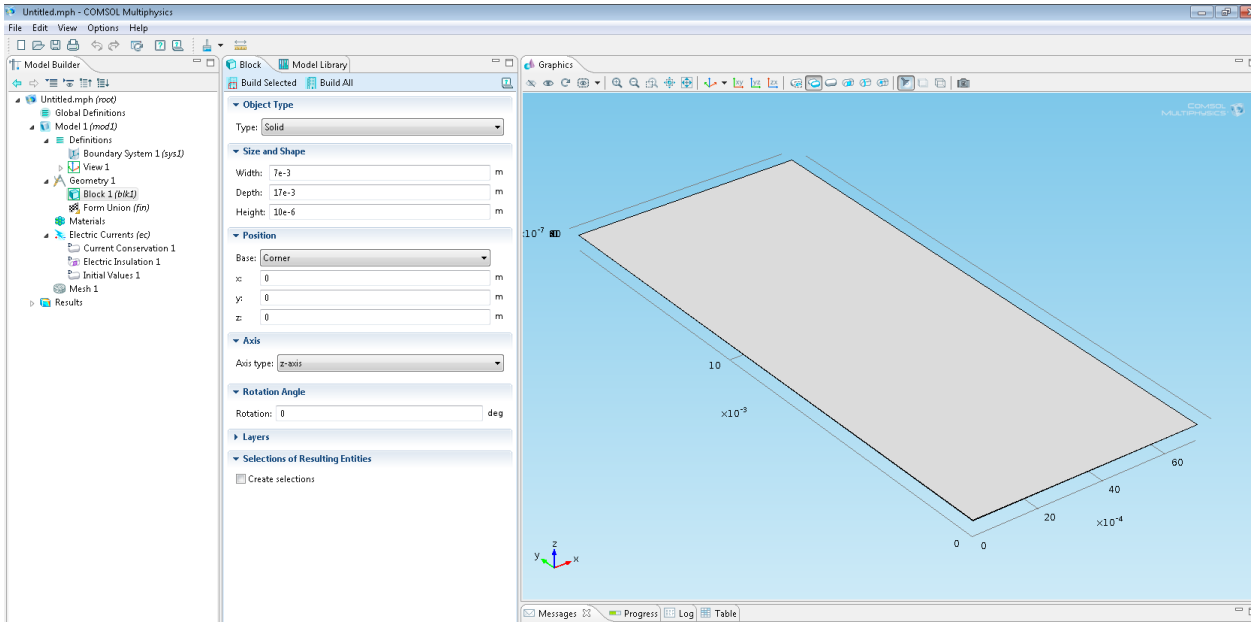


Figure 22. 7x17 mm IPMC finger

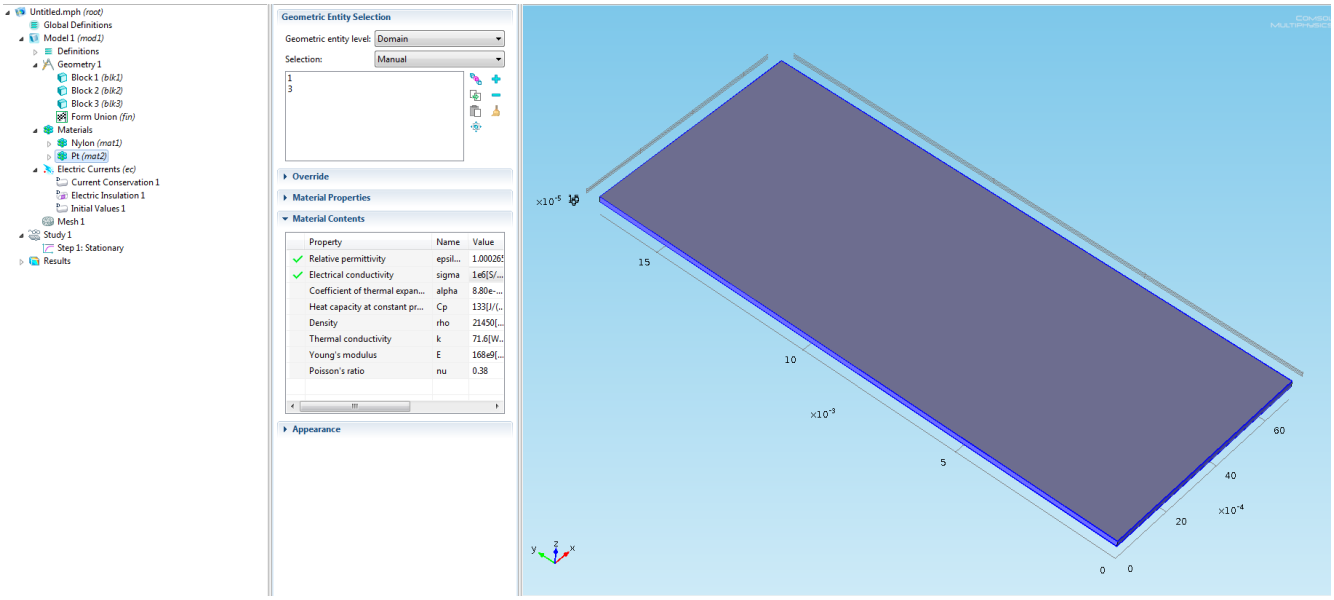


Figure 23. 7x17 IPMC Material Properties

Complex shapes are also easily drawn using the CAD tools. In the case of the triangles, a work plane must be created to use the Bezier Polygon tool. This is easily done using the geometry tool, as seen in Fig. 24. Once the work plane is added, the plane geometry feature is used to

create the Bezier Polygon and make a triangle, or any other shape. The Bezier Polygon must then be extruded to the thickness desired and a single triangle is created. This process can be duplicated to create three layers, to make the sandwich, but the work planes must be offset by the desired thickness of IPMC layers. The material is once again specified by the user and the model is ready to be used to determine the electric field created throughout the finger when applied a given voltage. When the electric currents physics is specified as the physics to use, Comsol determines if all material properties needed to solve the problem are supplied.

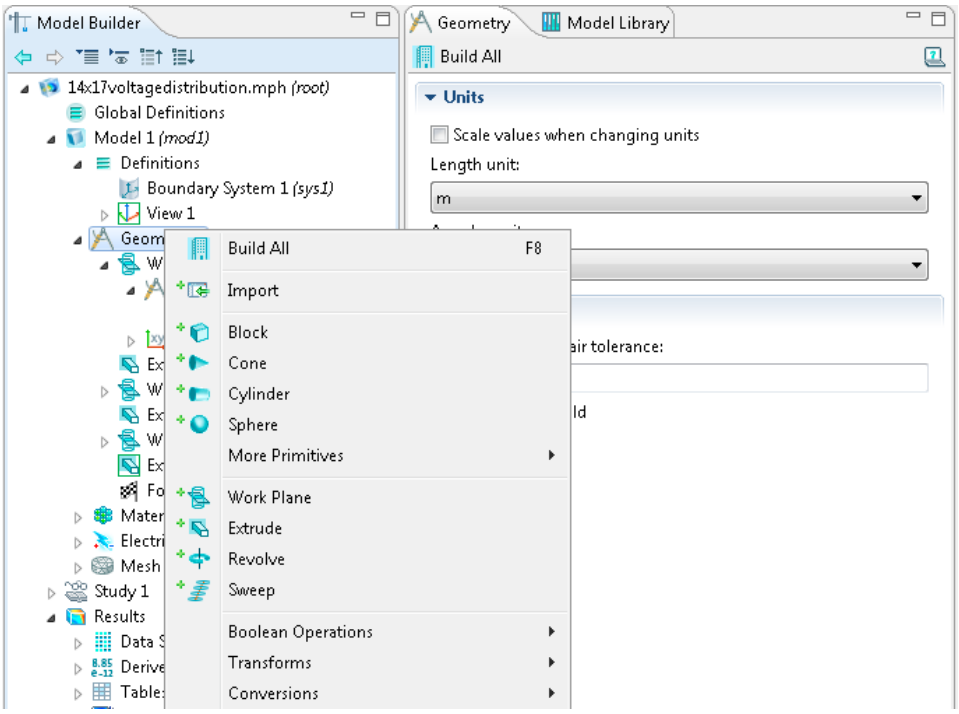


Figure 24. Work Plane in Comsol

3.4 Meshing IPMC Fingers

Meshing is one of the most important parts, and one of the most challenging, when designing and testing these fingers. The geometry being modeled must be meshed correctly or the model will not give correct results. The challenge is to mesh the electrical and chemical

gradients in such a way that they are represented correctly. Due to the thickness of IPMC fingers, this is hard to do. These gradients are extremely high and occur through this thickness. The thickness of IPMC fingers is usually around 200 microns, while the width and length may be 10-20 mm. In order to mesh such thin elements, automatic meshes in Comsol create meshes with hundreds of thousands of elements, which most computers are unable to solve. This is less of a problem when using electrical currents, but when predicting force and deflection it proves to be a problem. In order to mesh thin geometries, a couple of methods can be used. The first method involves using a rectangular swept mesh. This mesh creates a rectangular grid on the face of the geometry of interest. The grid is swept across the geometry to create a mesh with solid rectangular elements. This meshing technique is favorable when the IPMC finger is a simple rectangular solid, but does not work well when complex geometries are being used. In the complex cases, the easiest mesh to use is a user-controlled mesh that enables the user to scale the mesh in the direction of the thickness of the IPMC finger, as seen in Fig. 25. This mesh creates tetrahedral or triangular elements. Using a scaling factor, we are able to get as many elements throughout the thickness as possible. This scaling factor depends on the model being solved and must be changed according to different models. The user can choose a predefined mesh ranging from extremely coarse to extremely fine and then change the scaling factor to choose the best mesh. Another complication arises when trying to mesh the IPMC finger and the force transducer straw that is represented as a small glass cylinder in Comsol. The area where the straw meets the finger must be meshed by using the user-controlled mesh parameters and keeping the predefined mesh degree for the finger close to the degree for the transducer, i.e., extra fine for the finger and fine for the transducer straw.

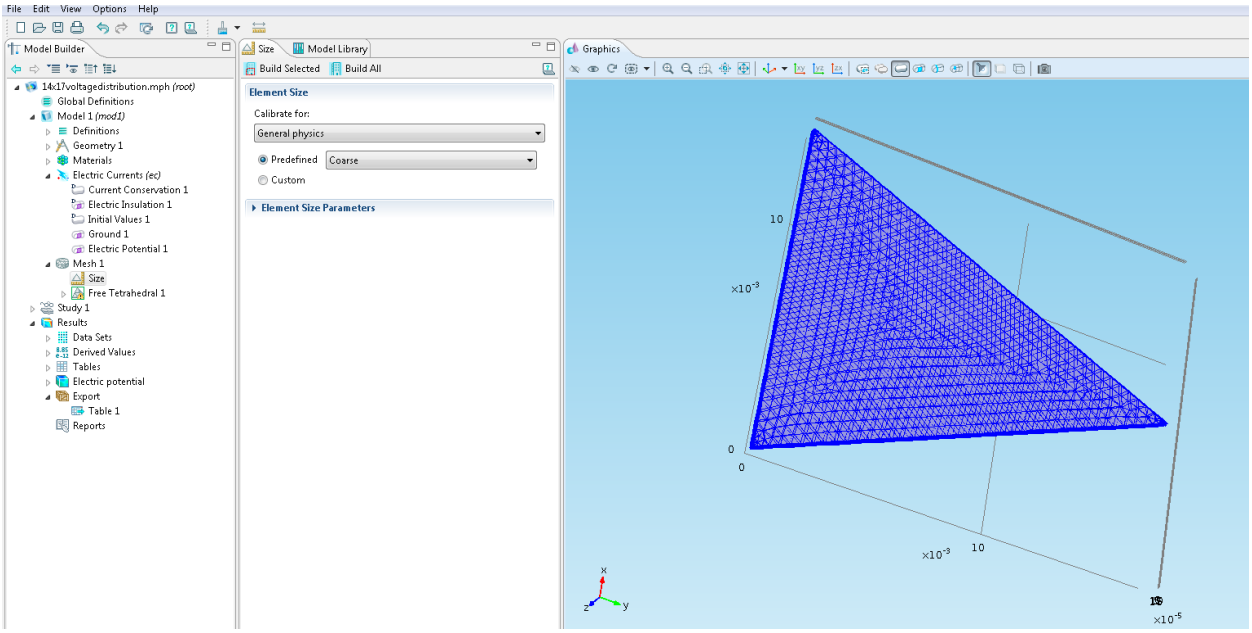


Figure 25. Meshing IPMC Fingers

3.5 IPMC Force and Deflection Model

3.5.1 Approach Overview

This model is created to predict the deflection and force output of an IPMC finger as a function of its electrical, chemical and mechanical properties. As mentioned above, the process is divided into smaller sub-models, as can be seen in Fig. 26. The process of putting these models together can be seen in Fig. 27.

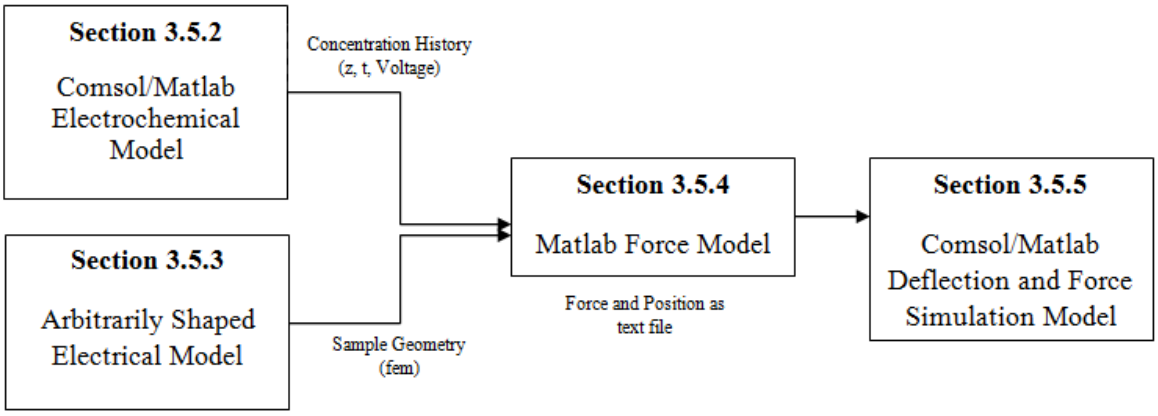


Figure 26. Model Diagram

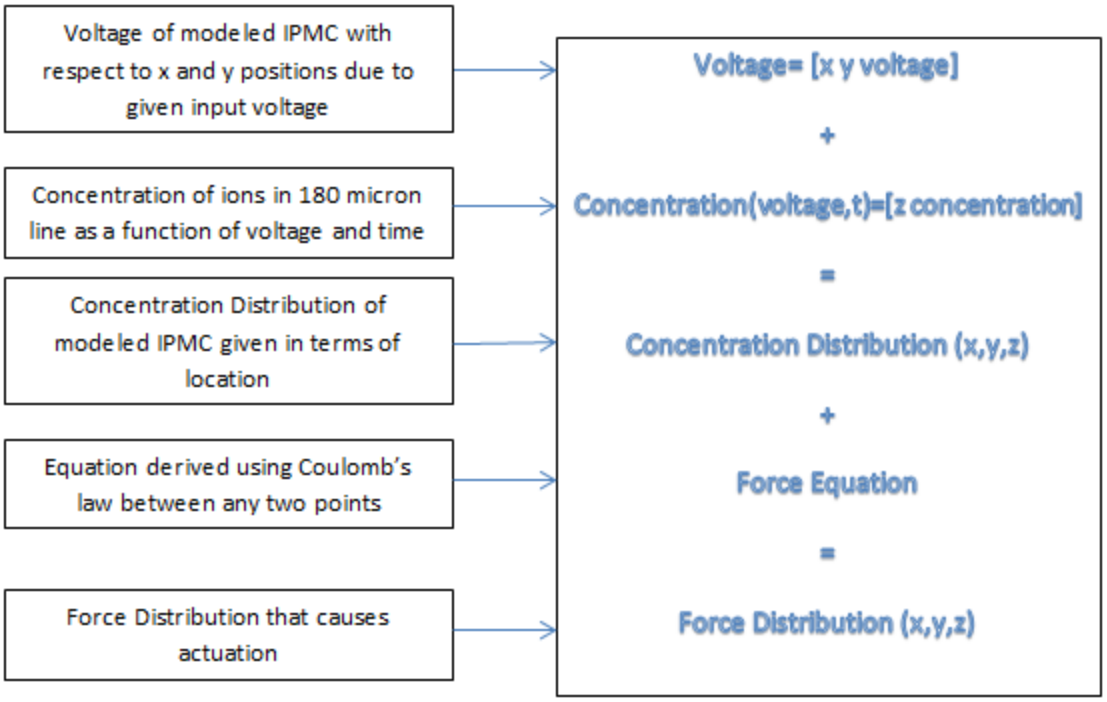


Figure 27. Force Modeling Process

The model starts with an electrochemical model that predicts the distribution of cations due to the imposed electrical field (Section 3.5.2). This model is used in all IPMC models, as this is just a representation of the ion movement. This will be the same in all IPMC fingers.

This is done using a concentration distribution modeled in Comsol. A 180 micron line is modeled in Comsol, representing the electroactive Nafion layer. In this model, the platinum layers have been omitted, as we are interested in the distribution of the cations and their movement in Nafion. An input voltage is applied to the model as a function of time and the concentration throughout the material is predicted. The ionic response will be accounted for at every possible input value at any given point. These concentration values will then be used to determine the stress values inside the finger. A separate model is created in Comsol that predicts the electric field distribution for any arbitrarily shaped IPMC finger (Section 3.5.3). The results of both of these models will be saved in Matlab where the stresses throughout the material will be predicted (Section 3.5.4). Finally, Comsol and Matlab will be used to simulate the deflection and force exerted by the IPMC finger on a transducer (Section 3.5.5).

3.5.2 Comsol/Matlab Electrochemical Model

3.5.2.1 Approach Overview

This model will describe the behavior of an IPMC finger as a function of its electrical, chemical, and mechanical properties. This is started using the electrochemical model that describes the redistribution of cations due to an applied voltage. This model will consist of a 180 micron line that represents the electroactive Nafion layer. This model does not contain the electrode regions, as we are concerned with capturing the distribution of cations in the Nafion. This shape is very easy to mesh as it is simply a line. This model is chosen, as it is important to capture the cation concentration gradient at the two distinct boundary layers that form near the electrodes during actuation. One of the layers will have a very high concentration of cations near the cathode and the other is depleted of cations near the anode. The goal of this model is to predict the distribution of cations due to an arbitrary DC signal as a function of time. Once the

model is solved in Comsol, it will be converted into an m-file, where it will be run iteratively. This must be done, as the response will be predicted at any voltage less than the input voltage. This will be used to simulate the response of the finger at any point, especially points far from where the voltage was applied, as the voltage drops. In our case, the model will involve an input signal of 2 volts, so the model will be run iteratively in Matlab to predict the responses for the values between 0-2V inputs. This process captures the ionic response for any possible input value experienced by points in the finger. These responses will be saved as a matrix in Matlab and referenced in the force calculations.

3.5.2.2 Concentration Distribution Theory

In this section, we are interested in the electro migration of cations under an imposed electric field through a porous medium. This model uses Comsol's Transport of Diluted Species. This physics uses the Nernst-Planck equation for ion transport, to model the flux of cations. This equation contains terms describing fluid velocity, diffusion, and electrophoretic migration for the surrounding medium. Comsol uses the equation:

$$R = \delta_{ts}\frac{\partial c}{\partial t} + \nabla \cdot (-D\nabla c - zu_m Fc\nabla V + c\boldsymbol{u}) \qquad 3.1$$

Where R is a reaction term, D is the diffusivity (m^2/s), c is the concentration (mol/m^3), z is the charge number (unit less), u_m is the mobility (s*mol/kg), F is Faraday's constant (C/mol), and u is the initial velocity of species (m/s). cu is zero as the medium containing the cations is not a flowing liquid. The system is also conservative with respect to the number of cations as the domain is isolated, meaning R will be zero. A brief outline is now given to explain the process, in order, of how we will create this model.

- Geometry is created using Comsol's CAD tools

    - 180 micron line is created to represent electroactive layer (Nafion)
- Physics is added to the model
    - Physics is added to base geometry using application nodes
    - Transport of Diluted Species is added to model to model migration of cations through porous medium
    - Electric field is added to model using variables
- Model is meshed
    - Geometry is meshed using a scaling factor
- Model is solved and Results are plotted
    - Solver is selected and problem is solved
    - Concentration of ions is plotted
- Model is exported to Matlab
    - Model is saved as a Model M-File and can be opened in Matlab
    - "Comsol 4.3b with Matlab" icon must be selected to open connection with Matlab
- Variables added in Matlab M-File
    - M-file is converted into a function to allow inputs and return outputs
    - Model performs parametric sweeps over many voltages to create history of ionic concentrations
- Ionic Concentration history of IPMC is exported as a text file

3.5.2.3 Modeling

First, the model is created using a single domain representing the electroactive polymer, as seen in Fig. 28. This line is created as explained in Section 3.3. This method is used to create a 180 micron line to represent the Nafion layer. Physics will then be added to the base model.

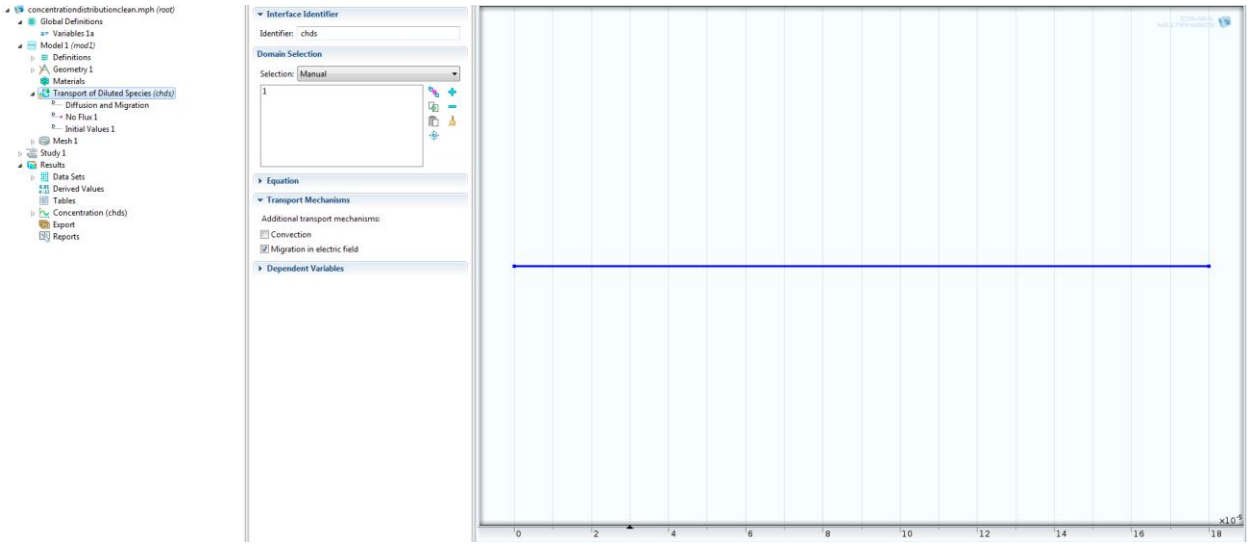


Figure 28. Concentration Model

After the geometry has been created, Transport of Diluted Species application mode is added to the model by choosing the physics, as seen in Fig. 29. This application mode predicts the distribution of cations due to an induced electric field. The electric field will be added as a function in the model. This is necessary for solving the electro migration problem. This is done under Global Definitions>Variables, this can be seen in Fig. 30. We can add "Van" and "Vcat" to correspond to the voltage at the anode and the voltage at the cathode, respectively. Now, we will define the physics.

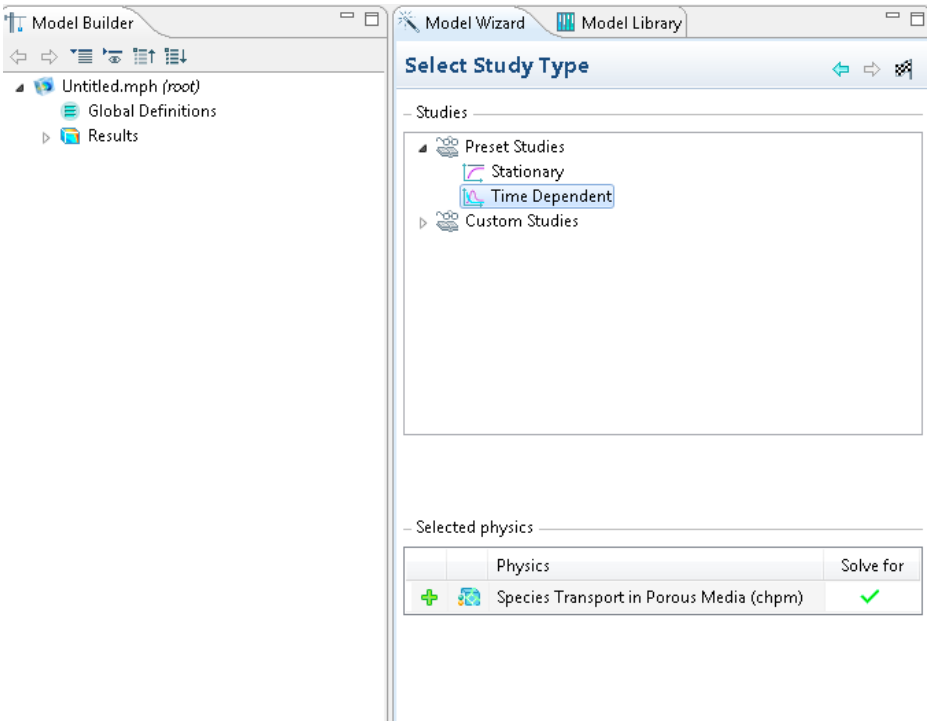


Figure 29. Physics Added

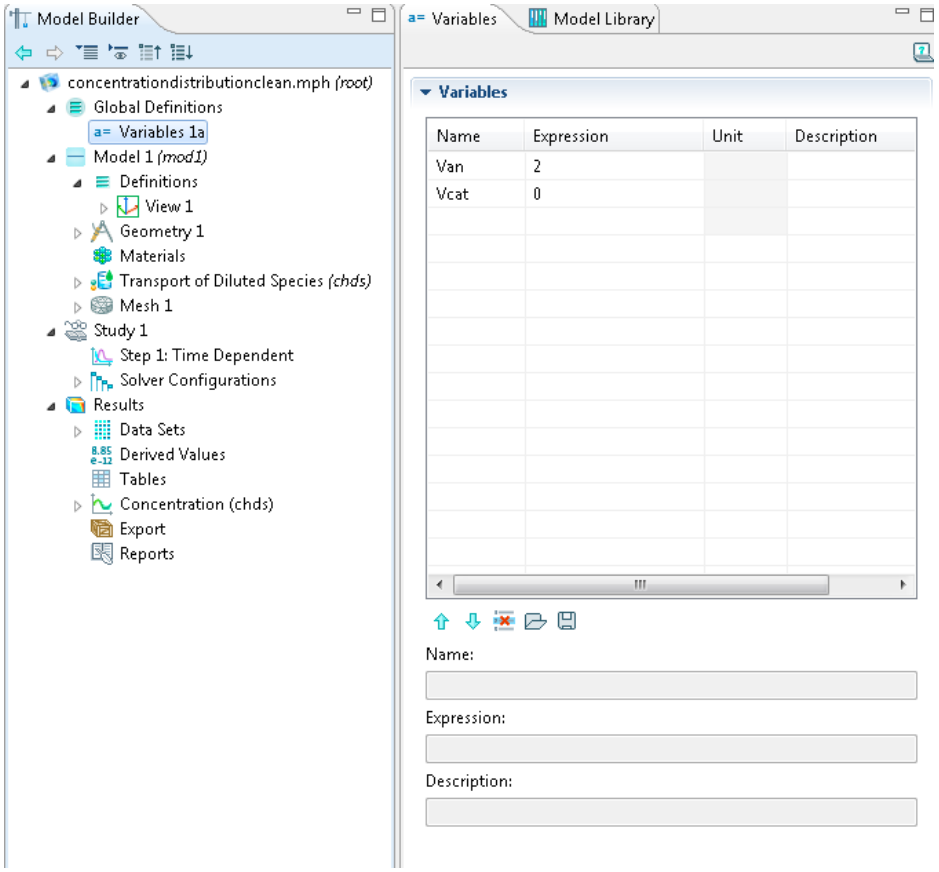


Figure 30. Variables Added

We begin by adding "Diffusion and Migration" to the model. This is accomplished by right-clicking "Transport of Diluted Species" and choosing "Diffusion and Migration." Under Model Inputs, we will add an electric potential, as can be seen in Fig. 31. This voltage potential will be a function of "Vcat" and "Van." As stated above, there will be no flux, so we will choose the two ends to have a no flux condition. The model must then be meshed as described in Section 3.4. A user-controlled mesh will be used. Clicking on the size option, we can select a custom mesh with a maximum element size of 1e-6 m, as seen in Fig. 32. This creates an element for every micron, or 180 elements.

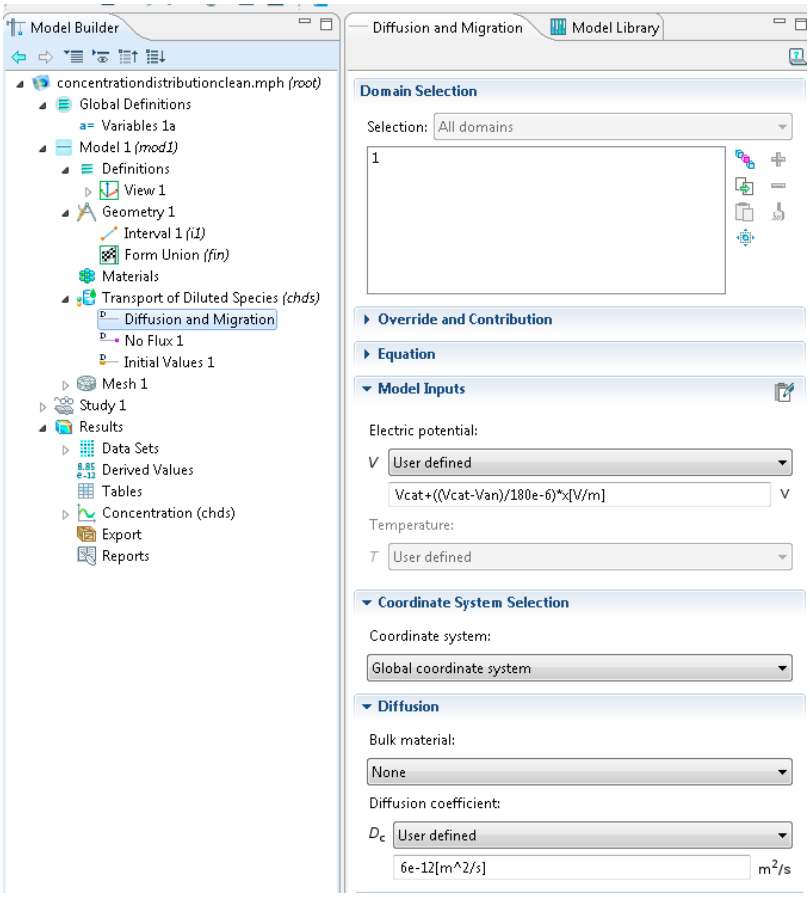


Figure 31. Electric Potential

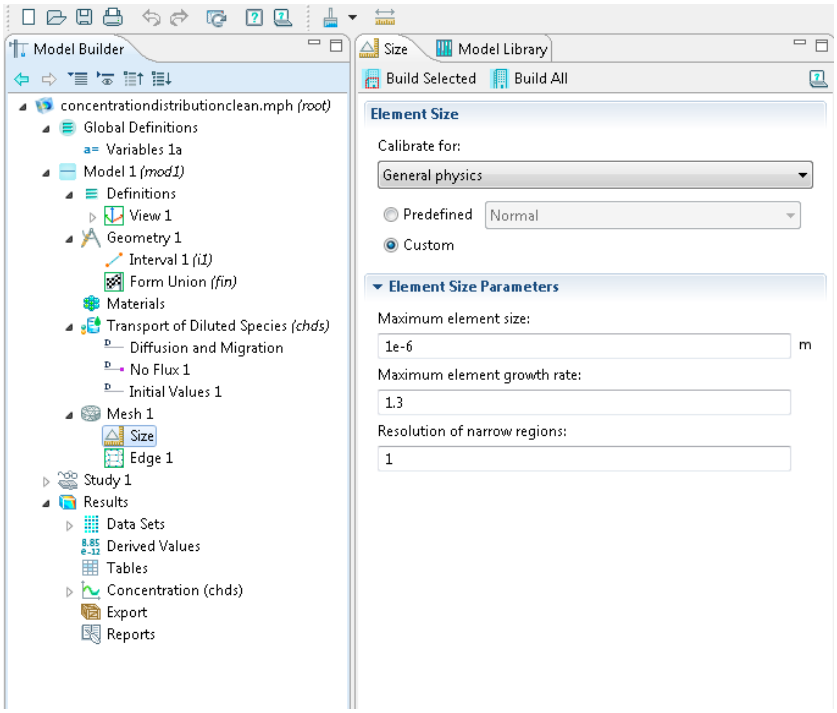


Figure 32. Meshing

Opening the solver branch, we can see the solver is already configured to be a time dependent problem. The solver configurations branch contains information about the type of analysis, the solver settings and the solver being used. The "Results While Solving" also contains different plots that may be represented graphically. In most cases, the automatic solver is used and the results chosen by Comsol are the correct results. Comsol automatically chooses solvers based on the problem type. After the settings in the solver parameters are chosen, the equal sign in the toolbar at the top of the screen is selected and the problem is solved and the results will be displayed graphically. Plotting options are also available, as seen in Fig. 33. Right-clicking "Results," many different plots can be added to the model. In this case, Comsol automatically plots the concentration of the line, as can be seen in Fig. 34. Other plot types are available, such as slice, sub domain, deformed and boundary plots. The model is then complete.

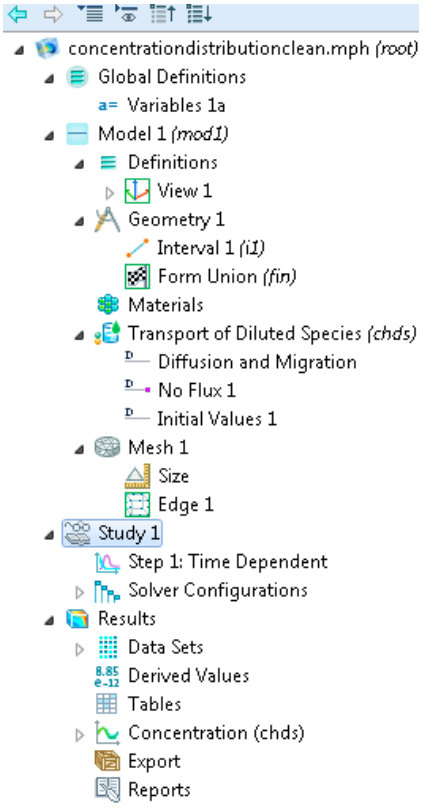


Figure 33. Solver and Plots

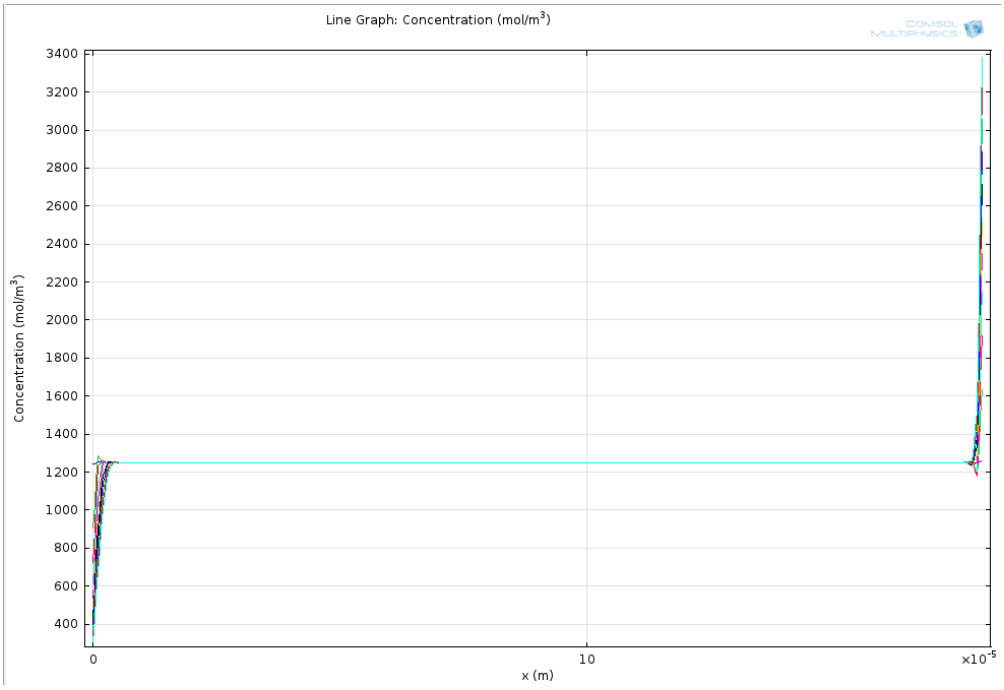


Figure 34. Concentration Plot

Next, we must export the model to Matlab as an m-file. The completed model will include the CAD model, solvers, and plots that were discussed. Comsol will store everything that was discussed in an m-file. This includes things that may have been deleted or numerous plots that may not be needed. This will be stored in the m-file and can become quite large. In order to save a model that is clean, there is a way to rid the code of anything that has been deleted. Clicking "File" and choosing "Reset History" will clean the model of any unwanted code. The "Save As Model M-File" can then be selected under "File." This will produce an m-file that can be edited and run in Matlab. In order for Matlab to understand the language used in the code, Comsol must be connected to Matlab, as described earlier.

Next, we will examine the m-file of the Comsol model. This file contains many lines of code that indicate whether the model contains parameters, data or variables. The line that contains variables can be seen in Appendix A1. Other processes such as geometry and meshing can also be seen. This code is not in a form where it can accept input variables; instead it is a history of what has been done in Comsol. Using this m-file, we will then create a program to yield concentration values for specific voltage inputs.

First, a variable containing a vector of solution times is created. This is used to indicate to the solver when solutions are returned. A voltage variable will also be created in the m-file. This will also be a vector that contains an evenly spaced series of numbers from zero to the maximum applied voltage. The last variable will be an empty matrix that is used as space for the storage of concentration values and is in the form "Concentration (z, time, voltage)." The rest of the model code is then placed in a for loop whose number of iterations is equal to the number of elements in the voltage vector. Next, the voltage signal will be changed. The variable controlling the voltage vector will be "V." This value will be replaced with a variable string.

In our case, we want to increment through the voltage vector, so we will change the model.variable to model.variable('var1').set('Va', strcat (num2str (V_a)), '[V]'). This will cause the voltage to change with each iteration and evolve over time.

Next, the code must be able to extract and store the solutions. This is done using mphinterp. The solution to the model at the end of each iteration will be stored as a FEM. Mphinterp will reference the fem for a solution at a given point in time and space. The format will be extracted as "con= mphinterp (fem,'c','coord','z','T',tf)'." The concentration at any point is the variable of interest which is shown as 'c'. We will place mphinterp into two "for loops" to make a concentration profile for the cations through the thickness of the Nafion. The first loop increments by time steps to extract the concentration profile for every solution time. The second loop increments by the height in the z-direction, returning the concentration at values through the thickness of the finger. These results are then stored in an array according to the voltage, time and z-position.

3.5.3 Electrical Model

3.5.3.1 Approach Overview

We created the model that is able to predict the distribution of cations through an IPMC finger due to an induced voltage in the last section. Next, the electrical distribution in the IPMC will be modeled in Comsol. This is done as the input voltage will decrease in strength as the distance from the application point is increased. As a result, the actuation of the finger will also decrease in strength, as actuation is proportional to input voltage. The model is created using the modeling techniques described in Section 3.3. The three domains representing the IPMC layers will be modeled. Then, the finger will be meshed and solved. Lastly, it will be exported to Matlab as a fem structure to be incorporated into the force model.

3.5.3.2 Electrical Currents Theory

Electrical currents, combines Ohm's law and Poisson's equation into one equation, given as:

$$-\nabla \cdot (\sigma \Delta V - J^e) = Q_j \qquad 3.2$$

where sigma is the conductivity (S/m), V is the voltage (V), J^e is the externally generated current density (A/m^2), and Q_j is a current source (A). Q_j is zero as the Nafion does not generate any current during actuation. As stated above, the top and bottom domains have electric potential boundaries where they contact the electrodes. So, $V = V_0$. In the case of the boundary with the ground condition, $V_0 = 0$ and the other boundary is equal to the value specified.

The simulation will be run in Comsol using this theory and the voltage distribution inside the IPMC will be predicted, once the mesh has been assigned. A brief outline is now given to explain the process of modeling the voltage distribution.

- Geometry is created using Comsol's CAD tools
    - Three domains are created that represent the thick layer of Nafion and the two thinner electrodes (Platinum)
    - Layers can be created using the block tool or drawn on work planes
- Physics is added to model
    - Electric Currents is added to model
- Add boundary conditions and sub domain settings to model
    - Materials are selected for geometry (Nafion and Platinum)
    - Electric Potential and Ground applied to edges
- Mesh Geometry
    - Geometry is meshed using user-controlled mesh

- Model is solved and results are plotted
    - Solver is selected automatically by Comsol
    - Plots can be changed to sub domain or volume to represent voltage distribution
- Solution is exported to Matlab
    - Solution is exported as m-file (Appendix A2) to be called as a fem structure into extract function (Appendix A3)

3.5.3.3 Modeling

This model will predict the electric potential in an arbitrarily shaped IPMC finger. In this case, we will model a 7x17 mm rectangle finger. First, we must model the voltage distribution. Starting by clicking on Comsol 4.3b with Matlab, a command window will open that connects Matlab to Comsol. Then, selecting Comsol 4.3b, Comsol opens. Selecting 3D, AC/DC>Electric Currents (ec), Stationary, Finish, the finger is ready to be built. Any shape can be modeled using the Geometry tool. The three blocks will be built, representing two layers of platinum and one of Nafion, as seen in Fig. 35. This is done by right-clicking geometry and selecting "block." The block is extruded to the thickness of the first layer of platinum. This can then be duplicated to make two more layers that have the same area, but the Nafion will be thicker.

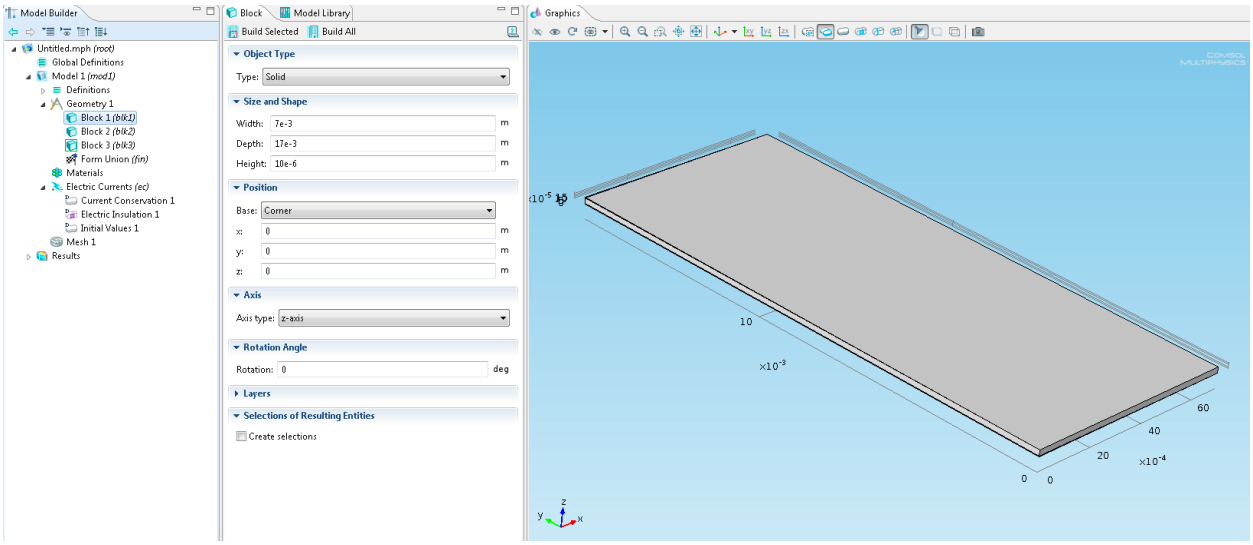


Figure 35. Arbitrary Finger

The materials must then be assigned to the layers. Using Materials>MEMS>Metals>Pt and assigning to Block 1 and Block 3 and Materials>Built In>Nylon to Block 2. Although these materials already have assigned values, the values for platinum must be changed a little to account for the material being platinum salt. In this case, the electrical conductivity will be changed to 1e6. Now, we must assign the physical properties to the model. This is accomplished by assigning the ground and electric potential to the left top and bottom of the finger. This can be seen in Fig. 36 and Fig. 37.

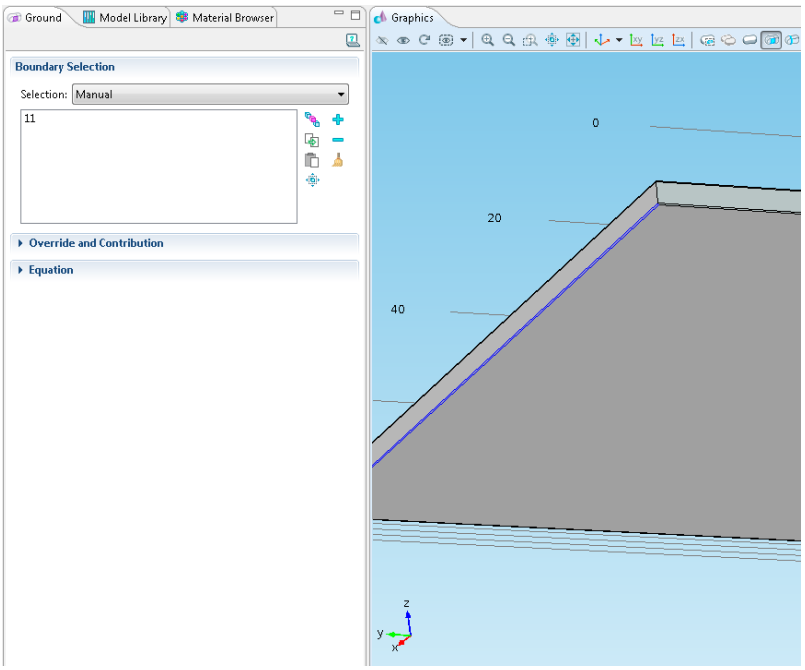


Figure 36. Ground

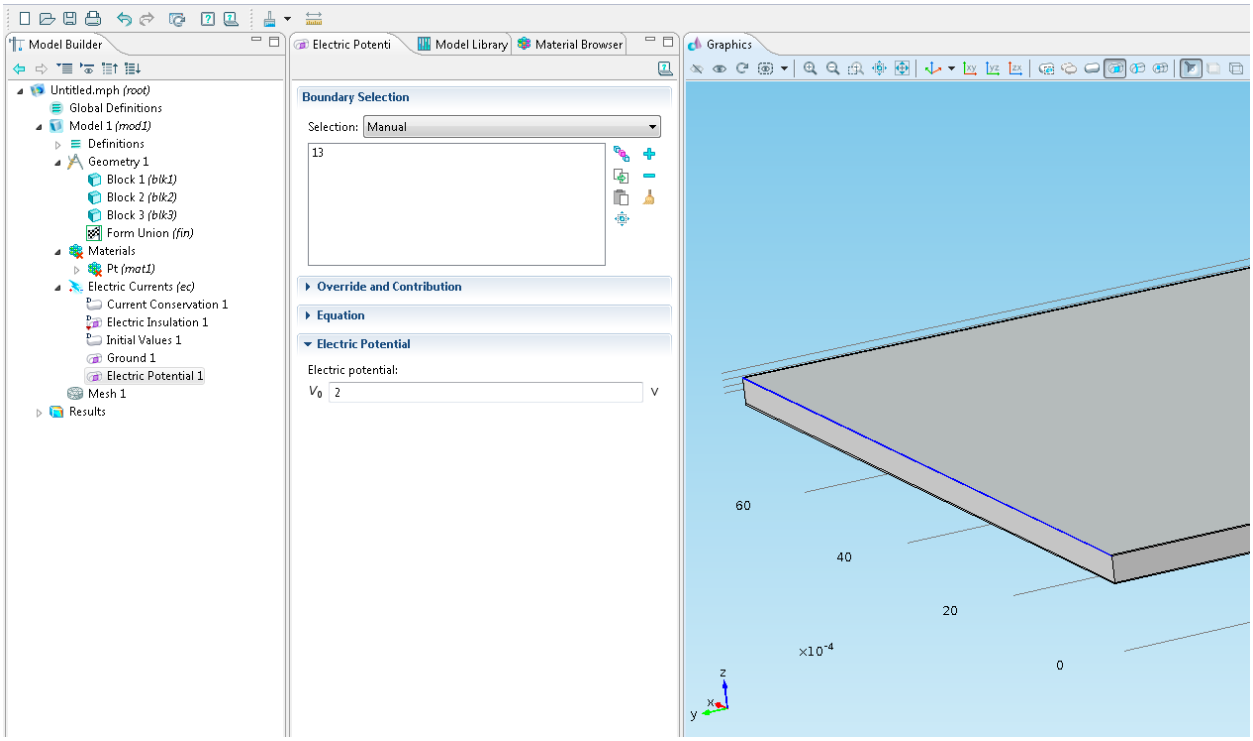


Figure 37. Electric Potential

Next is the mesh. Using Mesh>User Controlled>Size>Normal the finger is meshed, as seen in Fig. 38. The model can then be solved. Comsol automatically chooses the best solvers and plots

the results, as seen in Fig. 39. The plot can be changed by right-clicking Results>3D Plot Group and choosing the type of plot desired. Once the model is completed it will also be saved as a model m-file. This m-file will not be edited; it will simply be called as a fem in later codes.

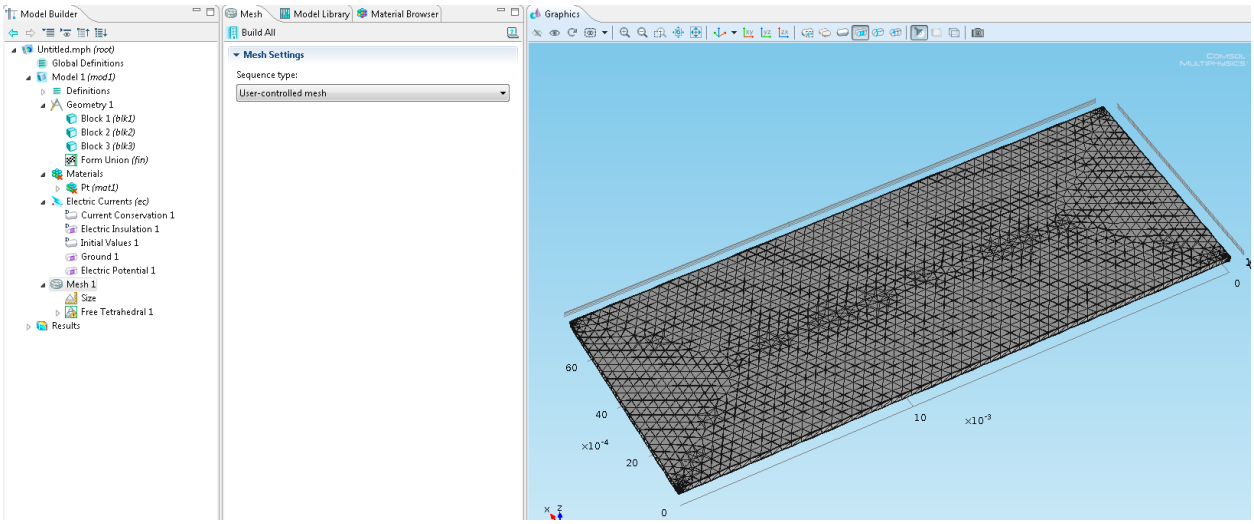


Figure 38. Meshing

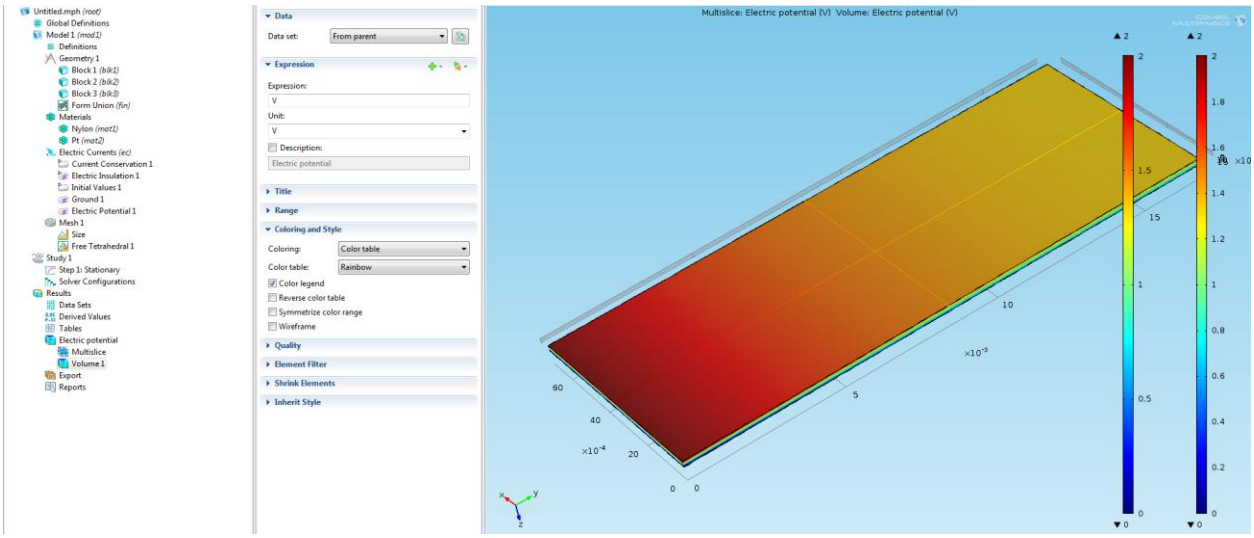


Figure 39. Results

3.5.4 Matlab Force Model

3.5.4.1 Approach Overview

As seen in the previous sections, we now have a model capable of predicting the concentration distribution through the thickness of the IPMC finger given a voltage input and we also have a model capable of predicting the voltage distribution across the surfaces of an arbitrarily shaped IPMC finger. Now, we can create a simulation to determine the force distribution throughout an IPMC finger. The voltage distribution can be used along with the concentration tables to determine the local concentration values throughout the IPMC finger. This will be accomplished programmatically using Matlab and the results will be converted into a distributed force which will be used to drive the IPMC finger during actuation simulations.

3.5.4.2 Force Model Theory

As discussed in Section 1.1.2, the basic structure of Nafion is considered to have two regions, hydrophobic and hydrophilic. These regions contain micellar structures forming a grid. When hydrated, the solvent collects in the micelle and the channels that connect them. When actuated, the cations in the micelle are driven towards the cathode. Cations are added to the micelle near the cathode, positively charging the micelle. Micelle near the cathode will be positively charged, forcing a repulsive force between them. This is also true near the anode, where the micelle will be negatively charged. The force between these two spheres, or two points, is calculated using Coulomb's law given as:

$$F_{cluster-cluster} = \frac{k_e q_1 q_2}{d^2} = \frac{k_e\left(e(c_+ - c_-)\right)^2}{d^2} = \frac{k_e(e\Delta c_+)^2}{d^2} \qquad 3.3$$

$$\text{And} \qquad k_e = \frac{1}{4\pi\epsilon}$$

This expression states that the force (F) between any two points is proportional to their charge (q) times a constant divided by the square of the distance (d) between them. The next expression applies to the case of two nested spheres that have the same charge. We will use this in the case of our IPMC, because the concentration does not vary much. The third expression states the difference between the number of anions and cations is equal to the change in concentration since their numbers are initially equal.

But, our electrochemical model predicts the concentration of cations in moles per cubic meter and equation 3.3 uses the change in concentration of an individual cluster. An equation will be needed to convert the molar cation concentration into cluster concentration. This equation is given by:

$$\Delta c_+ = \frac{\Delta\overline{c_+} * N_A}{N_{cluster}} \qquad 3.4$$

$$N_{cluster} = (\frac{1m}{d})^3$$

where $\Delta\overline{c_+}$ is the molar concentration of cations, N_A is Avogadro's number, and $N_{cluster}$ is the number of clusters per cubic meter. The number of clusters is calculated by dividing the total number of cations by the number of clusters per cubic meter, assuming the cations are initially distributed equally. Using these equations, equation 3.3 can be simplified to:

$$F_{cluster-cluster} = \frac{k_e(e\Delta\overline{c_+}N_A)^2}{(N_{cluster}\, d)^2} \qquad 3.5$$

But, we want an equation in terms of layers, not clusters. This is important because the repulsion forces between layers causes the actuation in IPMCs. It has been proven that the force in any direction is equal to the force exerted on a plane normal to that direction. This force only depends on the force that the two layers of cluster exert on each other. This implies each plane

experiences the same force as all the other planes, so the force is not increased by adding layers. This simplifies equation 3.5 to:

$$F_{layer-layer} = F_{cluster-cluster} * N_{cluster}^{2/3} \quad 3.6$$

Once again, this equation must be adjusted to account for multiple layers, as equation 3.6 only applies to double layer systems. This equation will be adjusted to account for the presence of multiple layers. Assuming even separation and a large number of layers, the combined force of each additional layer can be approximated as $\frac{1}{x^2}$ with a solution of $\frac{\pi^2}{6}$. Combining these equations, equation 3.6 can be simplified as:

$$F_{IPMC} = \frac{\pi^2}{k6} F_{cluster-cluster} N_{cluster}^{2/3} \quad 3.7$$

Another brief outline will be given to explain the process of the force and deflection model.

- Voltage distributions extracted from FEM structure
    - FEM Structure contains the voltage distributions of the two electrodes produced by the electrical model
    - Voltage is extracted using mphinterp and returns the voltage at any position
- Voltage is used to determine ionic concentration
    - Voltage potentials are used to create concentration matrix that defines the distribution of cations.
- Force Matrix is produced
    - Concentration matrix is converted into matrix of forces using force equation

- Force matrix is stored in Comsol model to simulate actuation.

3.5.4.3 Deflection and Force Modeling

In Section 3.5.3, a FEM structure was exported as an m-file into Matlab. This FEM contained the voltage distribution of an arbitrarily shaped IPMC finger. This distribution will be extracted from the m-file using an extract function (Appendix A3) and stored as a matrix. Again, we will use mphinterp to extract this information, but this time the function will be passed a matrix. This matrix will contain spatial coordinates of the model and at each point in the matrix, a voltage will be extracted. Now, we have a matrix that contains information about the voltage and position for the entire actuator. This will now be applied to the concentration history that was solved for in Section 3.5.2. In this section, we created a model to simulate ionic concentration in Nafion. The results of this model were exported to Matlab as an m-file and processed into an array. These results will be combined to create an array of concentration values for the entire finger as a function of time. The array is in the form Concentration (voltage, time) = [z concentration], where z is the thickness being sampled. This array returns the distribution of cations at any point and time. Now, we will develop a model that uses this concentration distribution to describe the stress in the material. This is quite easy, as all the variables are known, as we just developed the concentration matrix. The force is then calculated at every point using Equation 3.7. This will return another matrix in the form Force = [x y z Force]. The force will be calculated and can be saved as a text file which will be returned to Comsol.

3.5.5 Comsol/Matlab Force and Deflection Modeling

The force concentration is then uploaded into the Comsol model, using a simple property tool. This is done using an interpolation function that loads the text file acquired earlier into the IPMC finger that was modeled in the electrical modeling section. The text file contains four arguments, three spatial coordinates and the force at every coordinate. By assigning "Force" as the interpolation function and its location in the file, Comsol is able to load this into the modeled finger. In this case though, different physics will be used, as we are interested in the deflection and force applied by the finger on the glass transducer straw. The model will once again be 3D and the physics used will be Structural Mechanics>Solid Mechanics. The model must then either be rebuilt or simply copied from the electrical model, but in the new model, the IPMC finger will simply be one body, instead of three layers. This is done to avoid three very thin layers when meshing. The three layers will move as one when actuated, so this is a reasonable assumption.

Starting with a new model and choosing 3D>Structural Mechanics>Solid Mechanics, Stationary, Finish, we will finish the process. This time, only one block is needed. The block will be built and the material assigned. The material will be Nylon, but the material properties will be changed to mimic a composite material of Nafion and Platinum. The text file will be loaded using Global Definitions>Functions>Interpolation. The data source is a file and can be found by browsing the computer files and selecting the desired text file. The "Function Name" will be "Force" and "Position" will be "4." The Solid Mechanics section will then be selected and the fixed constraint will be assigned on the far left end of the finger, as seen in Fig. 40.

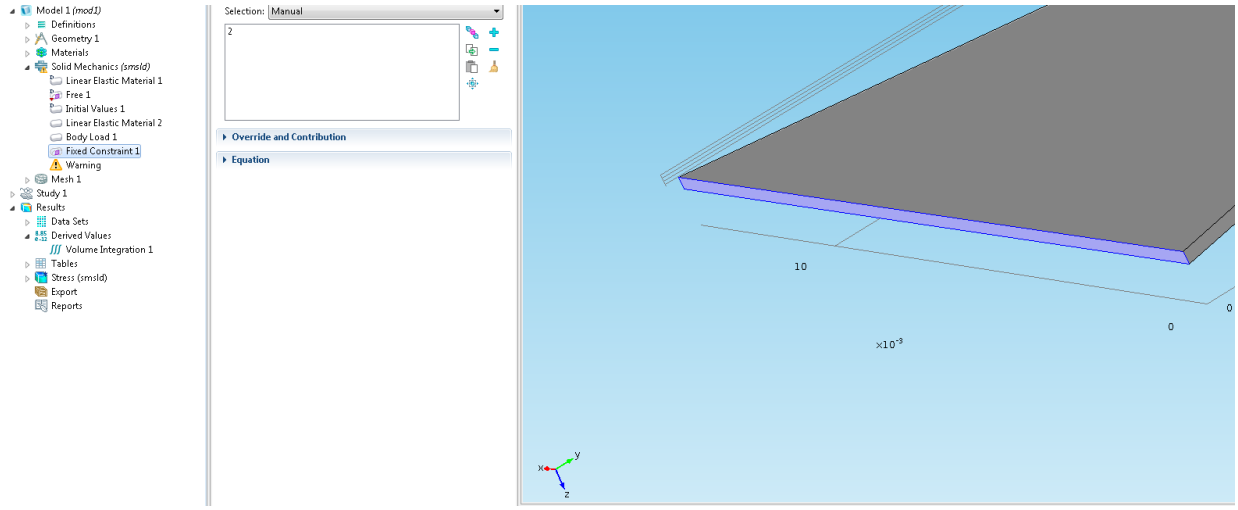


Figure 40. Fixed Constraint

The finger will be modeled as a cantilever beam that is fixed at one end where it is held by the electroded holder. A body load must then be added to the model. This is done using the Solid Mechanics option. Choosing the IPMC finger as the domain, the body load will be recognized as "Force," which was loaded in the text file above, as seen in Fig. 41.

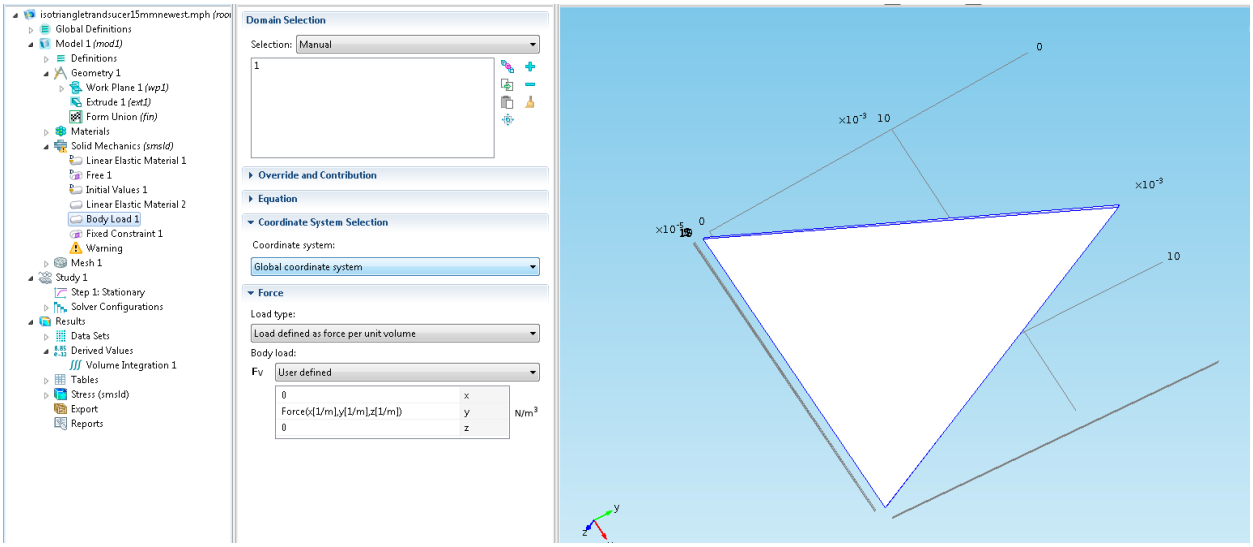


Figure 41. Body Load

In order to load the body load into the finger, the force must be defined as a body load. This means the inputs must be unitless, so the force is entered as "Force(x [1/m], y [1/m], z [1/m]).

The finger must then be meshed. This is very important in this stage. Once again, the user-controlled mesh will be used, but this time, an extra fine mesh will be used and the scaling factor under Free Tetrahedral>Scale Geometry>z-direction can be used. The model can then be run. The results are then plotted automatically. The plot desired in the deflection model will be a Volume Plot with Total Displacement as the "Expression." The finger will actuate due to the migration of cations and the force produced inside the IPMC, as seen in Fig. 42. The total displacement can be computed using the Derived Values tool, under point evaluation and choosing the far right tip of the finger that deflects, as seen in Fig. 43. Total displacement is already chosen when using this tool. Under the Solid Mechanics menu in the evaluation window is Solid Mechanics>Displacement>Total Displacement, which will give the total displacement of the tip of the finger.

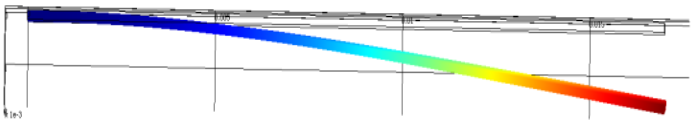


Figure 42. Finger Deflection

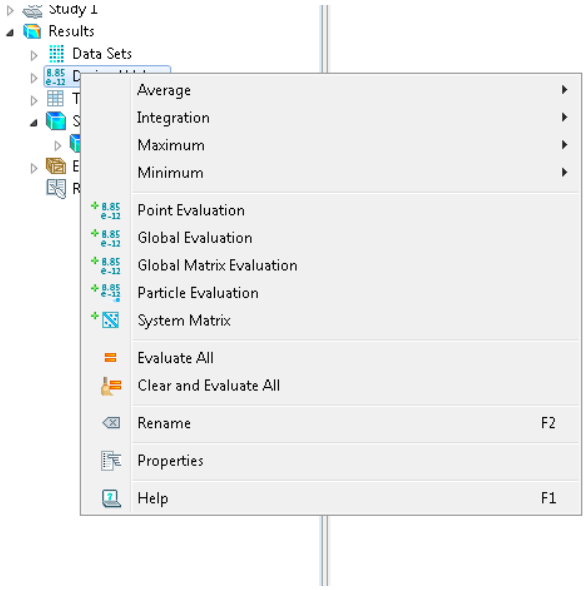


Figure 43. Deflection Measurement

The straw cylinder representing the force transducer can then be modeled in the previous deflection model. A cylinder will be modeled in Comsol to represent the straw on the force transducer. The same simulation will be run that predicts the deflection of the IPMC, which will deflect into the force transducer. This will be done in the same way the deflection was measured. The only difference is a cylinder will be modeled to represent the straw on the force transducer. The straw will have a diameter and height of one millimeter. The cylinder will be offset by the thickness of the finger and will be located two mm in from the end of the tip of the finger. This is done in all models to keep the location of the transducer consistent and also to avoid having the transducer at the very tip of the finger which causes errors in meshing. The cylinder will be assigned "Silica Glass" material, which is preloaded in Comsol. The bottom of the cylinder will also have a fixed constraint, so it will not move and will be in compression when the finger contacts it. The mesh for the finger will be the same, while the mesh on the straw can be "Fine" as it is not as thin. A fixed constraint must be placed on the bottom of the cylinder, as seen in Fig. 45, and the model can be solved. The force exerted on the load cell will

then be predicted by Comsol, once again using the “Derived Values” tool. Using Derived Values>Volume Integration>Solid Mechanics>Reactions>Reaction Force, the force exerted on the cylinder will be given in mN.

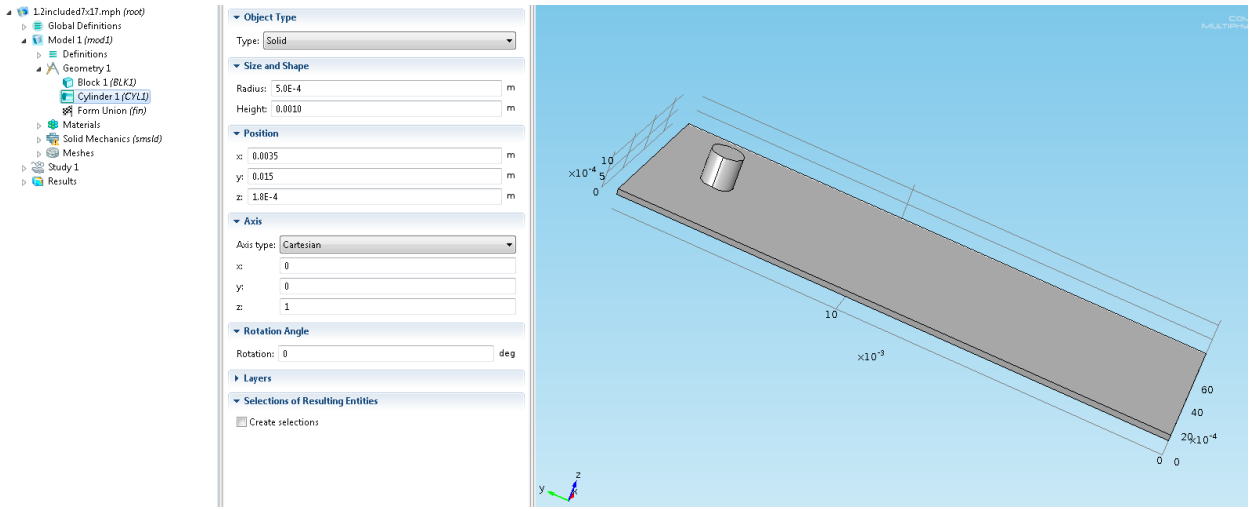


Figure 44. IPMC and Cylinder

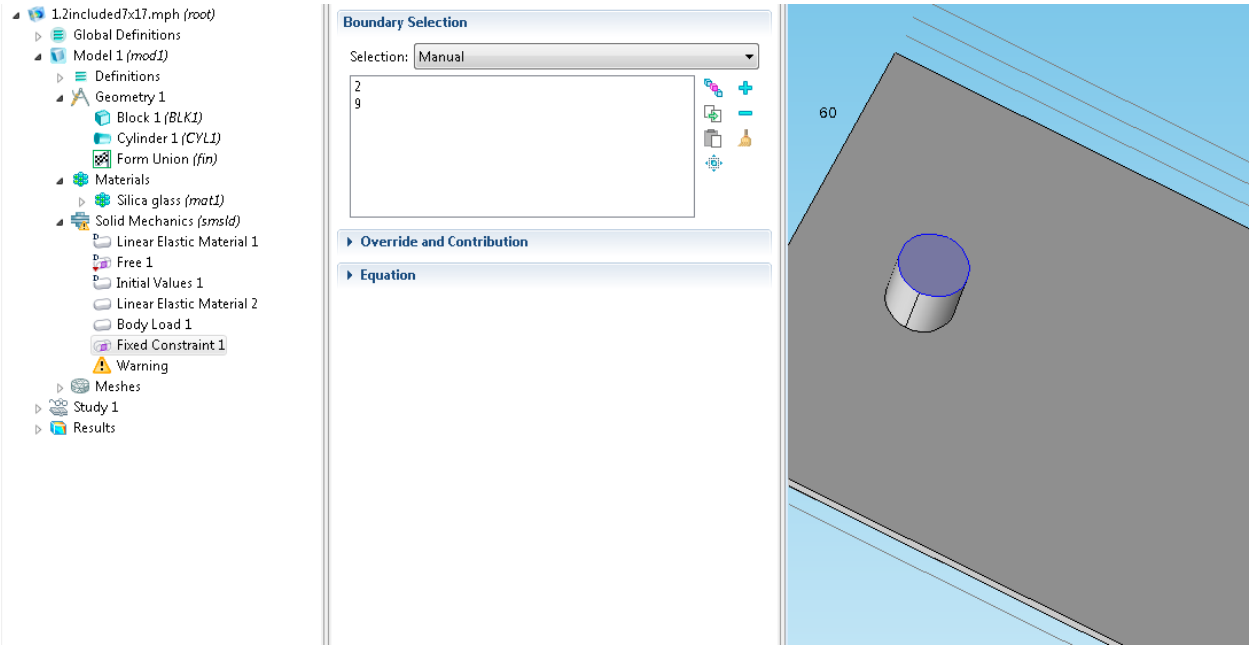


Figure 45. Cylinder Constraint

The force exerted on the cylinder can be computed using the Derived Values>Volume Integration, under Expression, as seen in Fig. 46 and Fig. 47.

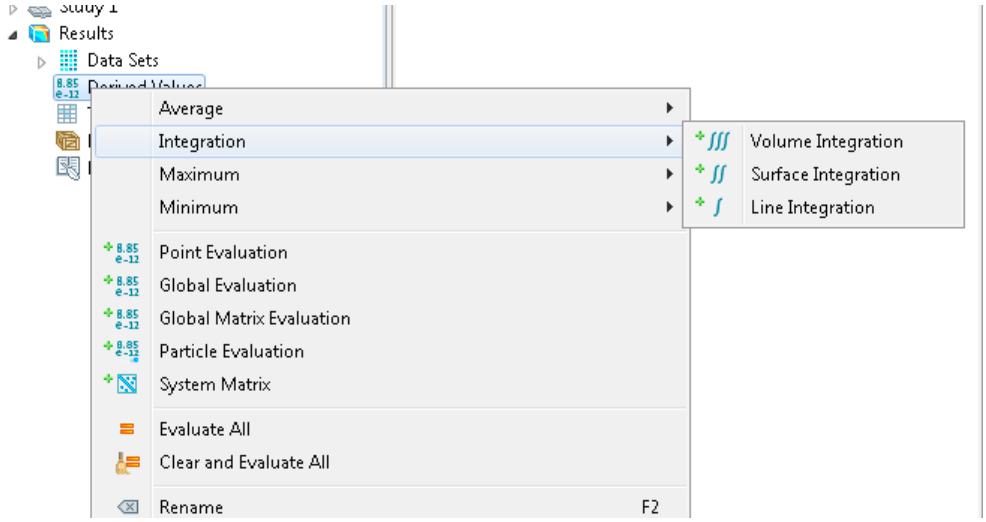


Figure 46. Reaction Force on Cylinder

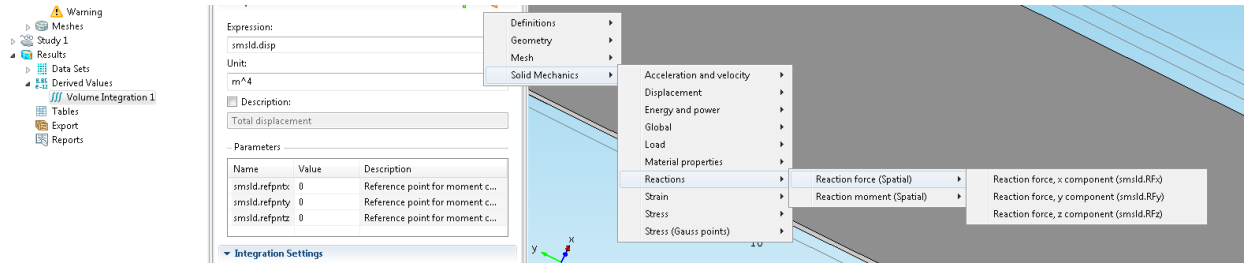


Figure 47. Reaction Force

3.6 Results

In this chapter, we created several sub-models in order to predict the deflection and force of an arbitrarily shaped IPMC finger. The first model was an electrochemical model that predicted the distribution of cations due to an induced electric field. Next, we developed a model that predicted the voltage distribution in the IPMC finger. These models were then combined to predict the ionic concentration due to the voltage. This concentration distribution was then used

to determine the electrostatic forces inside the IPMC finger. Using these models, we then were able to model many different fingers and test them experimentally to determine the validity of the models.

Two fingers with different areas were compared to determine their force and deflection. They were modeled and then tested experimentally. One was a 7x17 mm rectangular finger and the other was an isosceles triangle with the same height and base but half the area, as can be seen in Fig. 48.

Figure 48. Rectangle and Triangle Fingers

The results can be seen in Table 3. The error in force and deflection for the rectangle is 23.37% and 2.77%, respectively. The error for the 7x17 mm triangle for force and deflection is 13.3% and 4.92%, respectively.

Table 3. Output vs. Size

Gripper Dimension	7x17 mm rectangle	7x17 mm triangle
Experimental Force	2.0 mN	1.16 mN
Experimental Deflection	1.4 mm	1.13 mm
Modeled Force	2.6 mN	1.22 mN
Modeled Deflection	1.44 mm	0.98 mm

The same 7x17 mm rectangle was used and new 14x17 mm isosceles and right triangles were used, as seen in Fig. 49.

Figure 49. Rectangle and Triangle Fingers

The results can be seen in Table 4. The error in force and deflection for the isosceles triangle is 14.38% and 23.5%, respectively. The error in force and deflection for the right triangle is 13.96% and 6.26%, respectively.

Table 4. Output vs. Shape

Gripper Dimension	7x17 mm rectangle	14x17 mm Isosceles triangle	14x17 mm Right triangle
Experimental Force	2.0 mN	2.92 mN	2.28 mN
Experimental Deflection	1.4 mm	1.5 mm	1.85 mm
Modeled Force	2.6 mN	2.5 mN	2.65 mN
Modeled Deflection	1.44 mm	1.96 mm	1.735 mm

Examining the results, we can see the model is quite accurate when predicting the force and deflection.

Chapter 4. OPTIMIZATION in modeFRONTIER

4.1 Introduction

The main goal in this thesis is to optimize IPMC fingers for force or deflection. Comsol has an optimization package, but it was not powerful enough to accomplish this goal. ModeFRONTIER was suggested as an alternative to optimize these fingers for force or deflection. ModeFRONTIER is a multi-objective optimization and design package designed by Esteco SpA. It is written to couple with computer aided engineering (CAE) software, computer aided drafting (CAD) software, finite element structural analysis, and computational fluid dynamics (CFD) software. It is a GUI driven software in which optimization is accomplished by modifying the input variables assigned by the user, and analyzing the outputs as they are defined as objectives or constraints. The logic of the optimization is built around a "workflow" structure, which is built with interconnected nodes. ModeFRONTIER is capable of direct integration using CAE and CAD nodes or can be used to connect to other external programs using scripting. It uses design of experiments (DOE), robust design tools, and optimization algorithms to achieve optimization. These DOEs consist of random generator sequences, orthogonal and iterative techniques, and factorial DOEs. To achieve robustness analysis, it also includes Monte Carlo and Latin hypercube. Monte Carlo is a class of computational algorithms that relies on repeated random sampling. These are run numerous times in order to obtain the probabilistic distribution of an unknown entity. Latin hypercube sampling (LHS) is a statistical method used to generate a sample of parameter values from a multidimensional distribution. These methods are effective for single design optimization, but multi-objective problems were also considered. This was accomplished using a non-dominated sorting genetic algorithm (NSGA-II). NSGA-II generates evenly distributed Pareto designs in a fast and efficient manner.

4.2 Building a Complete Model

Using the codes described above and altering them, they can be combined into one large code, as seen in Appendix A5. This is just one sample of the code, as it was done numerous times for different shapes. This code was able to predict the force and deflection of a given finger in one complete m-file, instead of going through the process described in Chapter 3. The optimization process focused on three shapes, but any arbitrary shape can be modeled this way. This is the first time optimization has been achieved using these codes and the shapes were kept basic. As can be seen in Appendix A5, the code starts the same as the above mentioned codes. The electrical modeling is the beginning of the m-file. The blocks are built according to the size and shape desired. In the new model, a table must be created that calculated the total area of the finger being built. This is important as one of the optimization studies will involve changing length and width of the finger while keeping the area constant. This table is exported as a text file that contains the area of the finger. Once the area is calculated, the voltage is extracted using the same extraction function used above. The concentration and force is again calculated in the same way as was calculated in the previous section. The major difference in this program is the deflection and force modeling is then computed in the m-file, rather than using Comsol. This is important as modeFRONTIER is unable to connect to Comsol directly. Many software packages are able to link with modeFRONTIER, but Comsol is not one of them. But, we are able to input an m-file into modeFRONTIER, that modeFRONTIER is able to read and modify. This file contains the entire process explained in the previous chapter, but switching back and forth from Comsol to Matlab and running the entire modeling process is not required. The entire process is contained in the m-file. One code is used for the modeling of the deflection of the finger. This code builds the finger and applies all the physics and meshing. It will then solve for the

deflection of the finger. A table is created at the end of this code that outputs the deflection of the finger. This is important as modeFRONTIER will be able to read this output and optimize for the deflection. The same goes for the force applied by the finger. The code contains all of the information to build the finger and the force transducer and again solves the model and predicts the force generated by the finger on the transducer. Another text file is output that can be read by modeFRONTIER, and once again, modeFRONTIER can optimize for the force exerted by the finger. Finally, a third version of the code was written where a deflection test is run and then another force measurement is run in the same code. The important part of the code is the prediction of the stress developed inside the finger. This is calculated before any deflection or force measurements are taken, so the code is able to reuse the stress prediction in both the force and deflection modeling.

First, the deflection model is run without the force transducer, so the finger is able to deflect. The code, seen in Appendix A5, is used to model the finger and calculate the stresses developed inside the IPMC. The deflection measurement is then run in the same way that was run in Chapter 3, except this time it is run using an m-file that is stored in modeFRONTIER. The deflection is calculated and output as a text file. This text file will then be used in modeFRONTIER. modeFRONTIER will take this text file and use it as an output to maximize for, therefore maximizing the deflection of the finger. Then, the force transducer is added to the model and the model is solved again. Appendix A5 will be modified, and the force transducer will be added to the model. The force measurement will once again be measured in the same way described in Chapter 3, except this time it is run as an m-file that is stored in modeFRONTIER. The force is again output as a text file that can be read by modeFRONTIER. modeFRONTIER will then take this text file and use it as an output to maximize for, again

maximizing for the force applied by the finger. Finally, a third code will be used to model the force and deflection of the finger in the same code (Appendix A6). This is done by again using Appendix A5 and adding the force transducer, while keeping the deflection measurement already achieved. The code will have two outputs, the deflection and the force, that are output as text files. modeFRONTIER takes these text files and maximizes for them. But, although modeFRONTIER tries to maximize for both, there will always be a trade off. It will never be an exact optimization of the finger for both force and deflection, it will usually solve for a middle ground, where there will be a similar force and deflection. This is accomplished by solving the deflection code first, then restarting the process, solving for the force exerted. These codes were used to optimize rectangles, right triangles, and isosceles triangles for force or deflection. These shapes were chosen due to the interesting experimental data that was discussed above.

4.3 Optimization in modeFRONTIER

ModeFRONTIER must be able to access the m-files that represent the entire finger model. This is done using an EasyDriver node in modeFRONTIER. The EasyDriver node is able to create a link between modeFRONTIER and third party software, such as Comsol. This EasyDriver node uses the script or m-file. This is done by setting up a script and rules to drive the process. In our case, the entire process of optimization is built around the EasyDriver, as it will access our m-files, some examples can be seen in Appendix A5 and Appendix A6. EasyDriver can be found under Script Nodes and the format of the EasyDriver node can be seen in Fig. 50. Once in this node, the EasyDriver must be edited. Once in Edit EasyDriver, the m-file needed can be added by browsing under the "Add" selection and selecting the m-file. The m-file will then be loaded into the EasyDriver, as seen in Fig. 51. The entire m-file can be seen in the EasyDriver window.

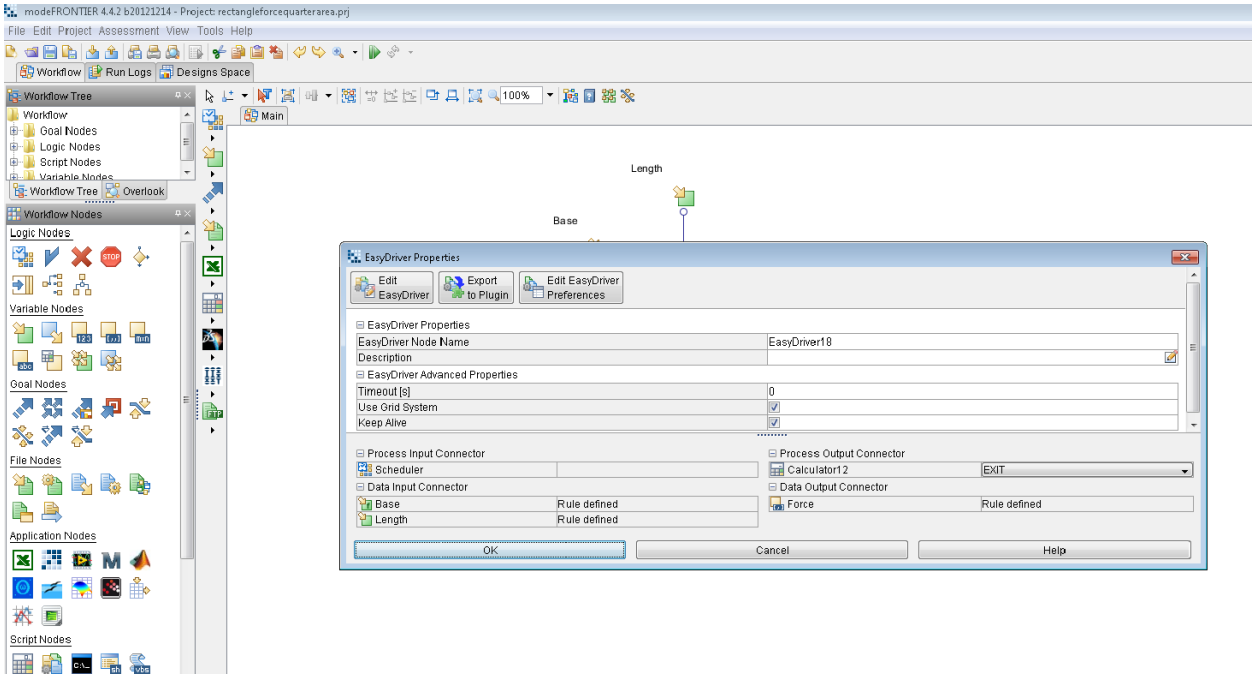


Figure 50. EasyDriver

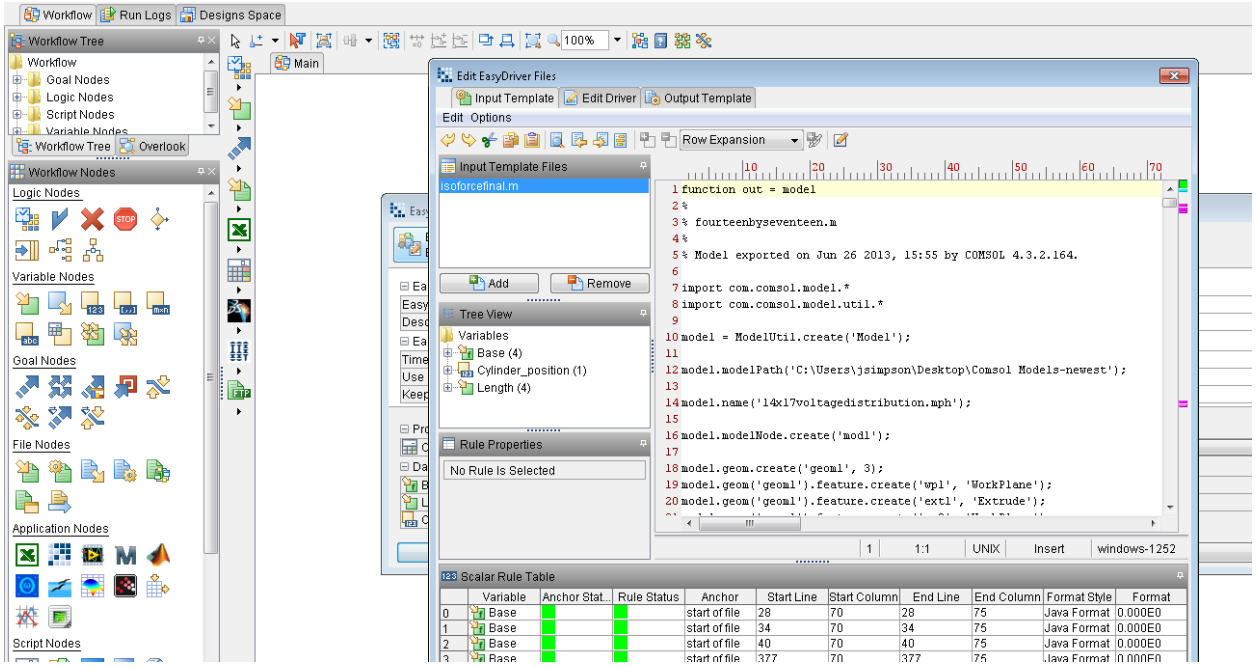


Figure 51. EasyDriver with M-File

Now, modeFRONTIER knows what m-file to use and the inputs, such as base and length, will be selected in the m-file in order to optimize the model for deflection (or force). Next, the inputs that will be modified to optimize the model for force or deflection will be parameterized. The optimization process described in this thesis focuses on changing the width and length of an IPMC finger, while keeping the area the same. So, the inputs will simply be the width and length of the finger. The width of the finger is considered the base in modeFRONTIER. The Input File can be found under File Nodes. Adding two input variables, we will be able to change the base and length of the finger, as seen in Fig. 52. modeFRONTIER needs to know two input variables, as it will change these variables in the m-file. If it does not have two input variables, it will simply change one or the other and leave the other the same. modeFRONTIER works by modifying the m-file and changing the variables that are selected by the user. In our case, we wish to change the width (or length) of the finger, but the width and length must be changed in the m-file, accordingly. Width and length are considered to be input variables, but they are really modifiers telling modeFRONTIER what to change. Also, when running the model, we will add an expression to either the width or length that expresses it in terms of the other.

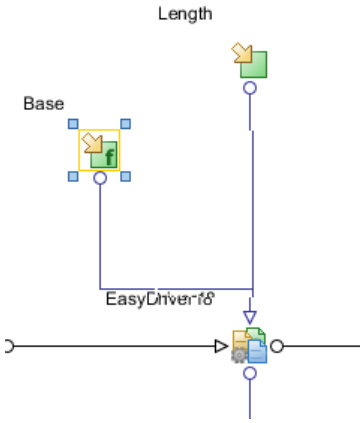


Figure 52. Input Nodes

These nodes will be connected to the EasyDriver where we can select the base and length in the m-file, so modeFRONTIER will change these when optimizing the model for force or deflection. Once again, entering Edit EasyDriver and highlighting the base, as shown in Fig. 53, and right-clicking, a rule can be added. This is done by first clicking on "Base," highlighting the base parameter in the m-file, and right-clicking, then Add Rule. Since there are three blocks in the model, three bases are selected. The same process is used to select the length of the finger.

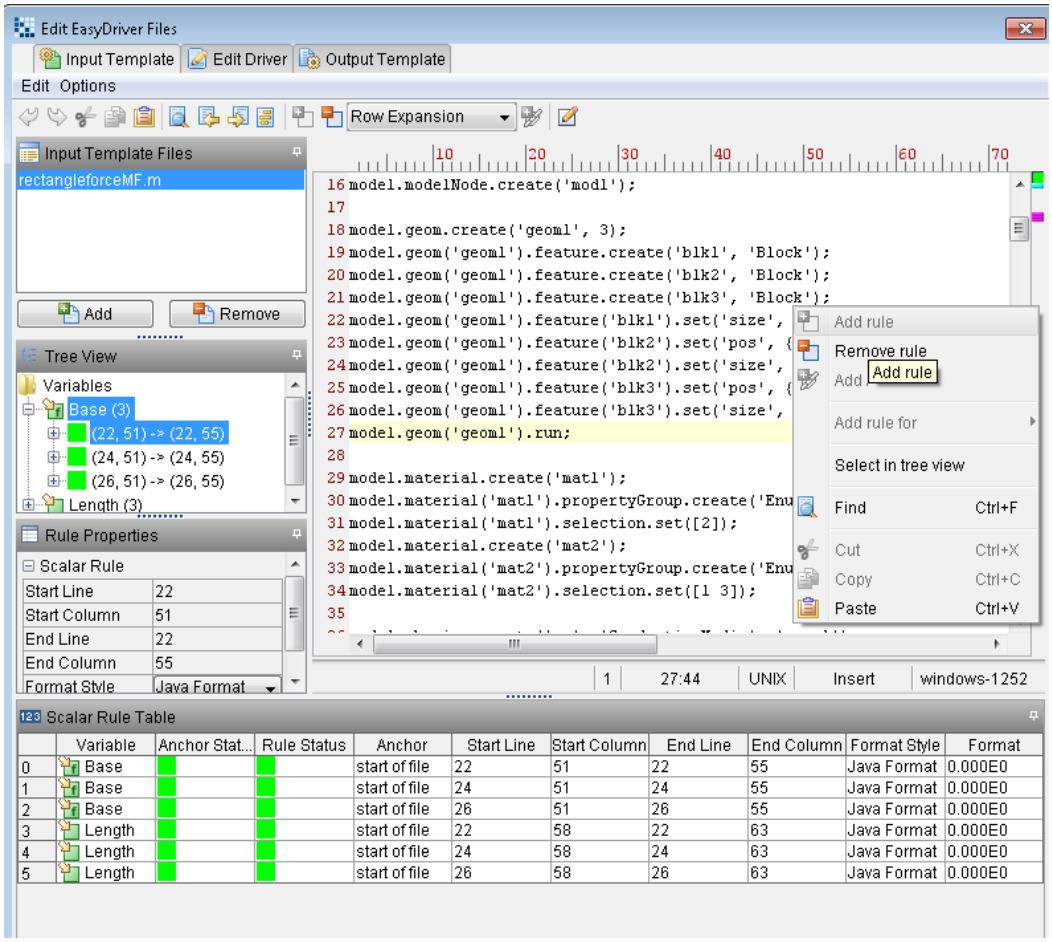


Figure 53. Adding Rules

Next, modeFRONTIER needs to know what to optimize for, in this case either Force or Deflection. This is accomplished using an output node. This is needed as the m-file outputs the deflection or force of the model as a text file and modeFRONTIER needs to be able to read this text file. The output node will be named accordingly, either Deflection or Force, and will be

passed to a design objective node. This node tells modeFRONTIER that we are maximizing the force (or deflection) of this model. An exit node is placed after the EasyDriver node to tell modeFRONTIER the model is complete. Now, we must add a scheduler node. This node tells modeFRONTIER the specific DOE to use. In the case of the single objective optimization, where we are optimizing for either deflection or force, a SIMPLEX scheduler is used. This node can be opened and the maximum number of designs is specified by the user. In most cases 60 designs are enough, as the model usually converges many designs before this. The scheduler node also has a DOE node attached to it. In our case, the Uniform Latin Hypercube will be the best DOE to use. The number of designs must be specified by the user and the Add DOE Sequence must be selected, as seen in Fig. 54.

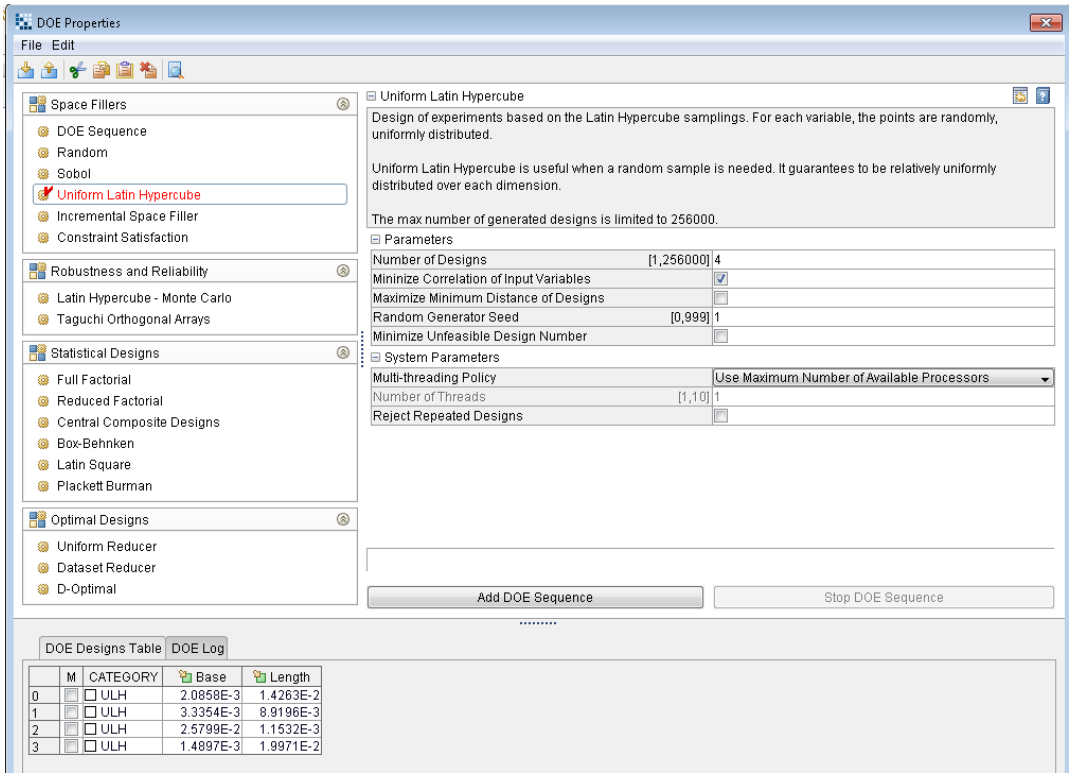


Figure 54. DOE Properties

Earlier, we parameterized the base and length, but they are not yet constrained in terms of one or the other. Seen in the bottom of modeFRONTIER are the input variables. Clicking on base, we can change it from a variable to an expression. The base can be expressed as a function of length, or vice versa, to constrain the area. In our case, the base will be "1.19E-4/Length" for the rectangle as the area of a 7x17 mm rectangle is 119 mm^2. Now, the process is complete to optimize for a single design goal, such as deflection and was done for a rectangle, isosceles triangle, and a right triangle. In order to optimize for force, we must add a node in which the cylinder, representing the force transducer, will move 2 mm in from the tip. The length of the finger will change and the force exerted by the finger must be read at the same position for each iteration. This is accomplished using a calculator node that moves the cylinder depending on the length, as seen in Fig. 55. The cylinder position is selected the same way the base and length selections were made in the EasyDriver and can be seen in Fig. 56. This process was done to maximize the force for a rectangle, isosceles triangle, and a right triangle and the results will be discussed later.

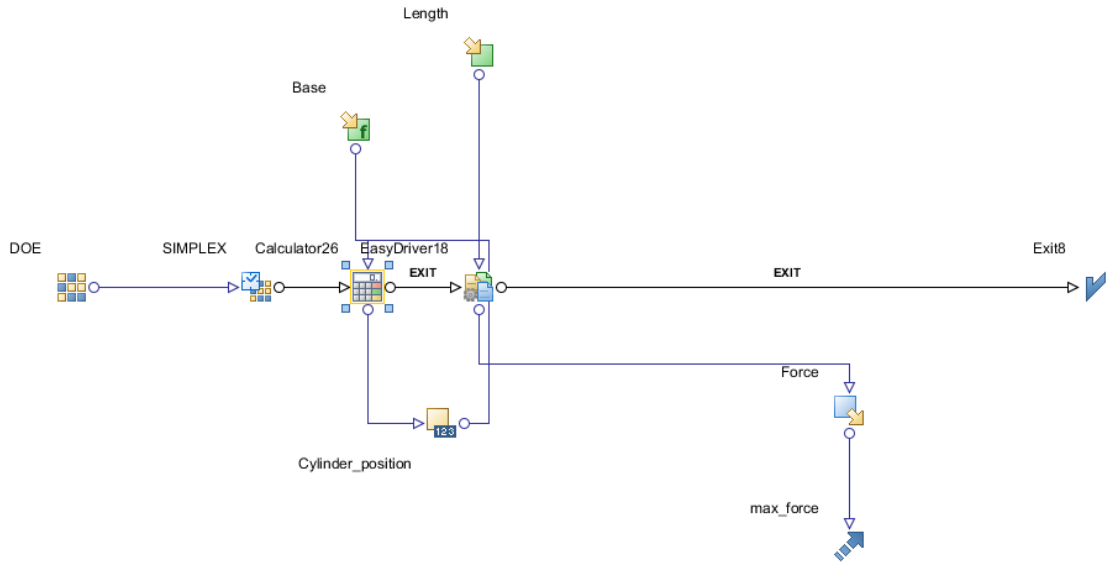


Figure 55. Cylinder Moving

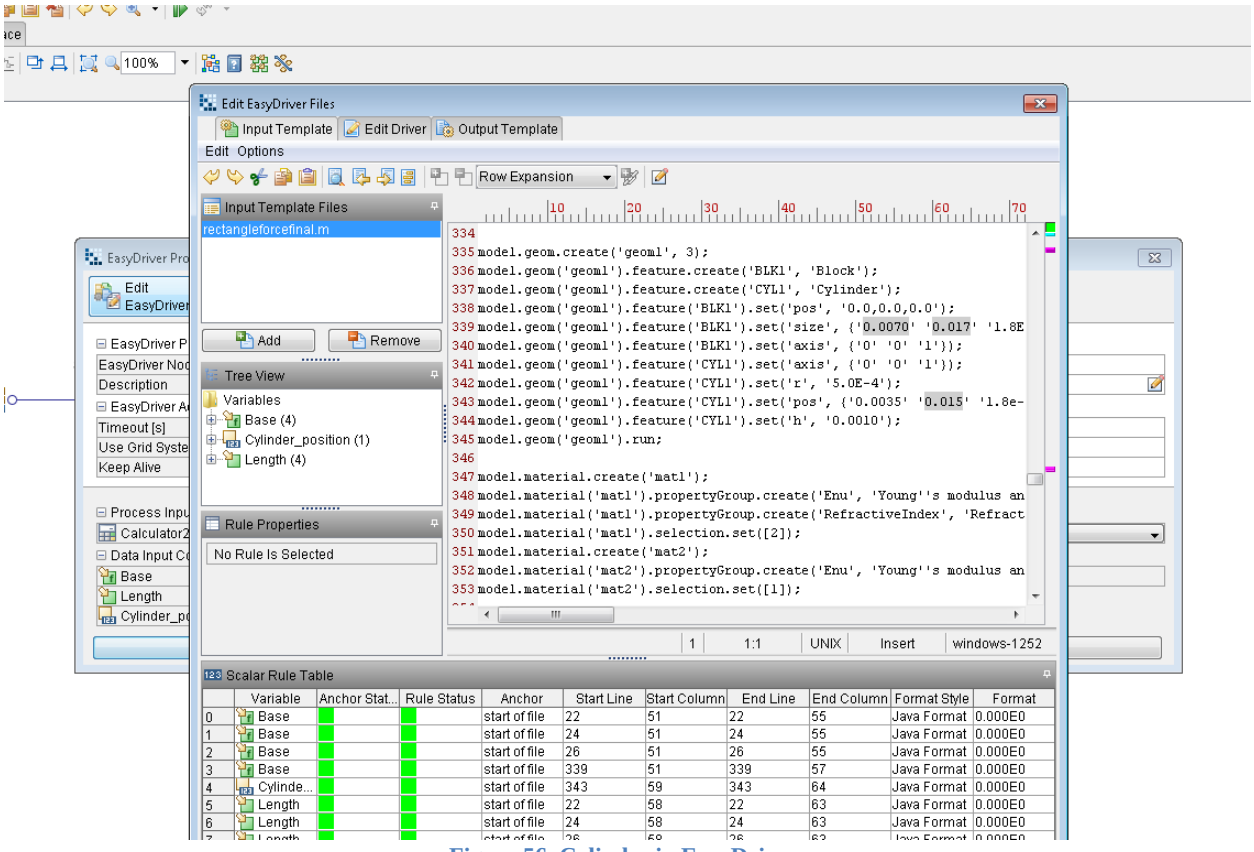


Figure 56. Cylinder in EasyDriver

Next, the process was changed in order to minimize the area, while obtaining a certain force or deflection. A desired force (or deflection) was chosen, such as 2 mN (or 2mm), and the area is minimized. This is done to choose the smallest finger, while still meeting a design goal. The process is similar to the previous process, but this time a force (or deflection) constraint will be incorporated into modeFRONTIER, while the design objective will be minimizing the area, as seen in Fig. 57. The EasyDriver process will be the same, including selecting the base and length. The only difference is "Area" and "Deflection" will be the output variables, while the objective will be to minimize area and deflection will have a constraint on it. Under "Constraint Properties" for deflection, the type can be set to "Equal To" and the limit can be set by the user. This process was repeated for all three shapes, for both force and deflection.

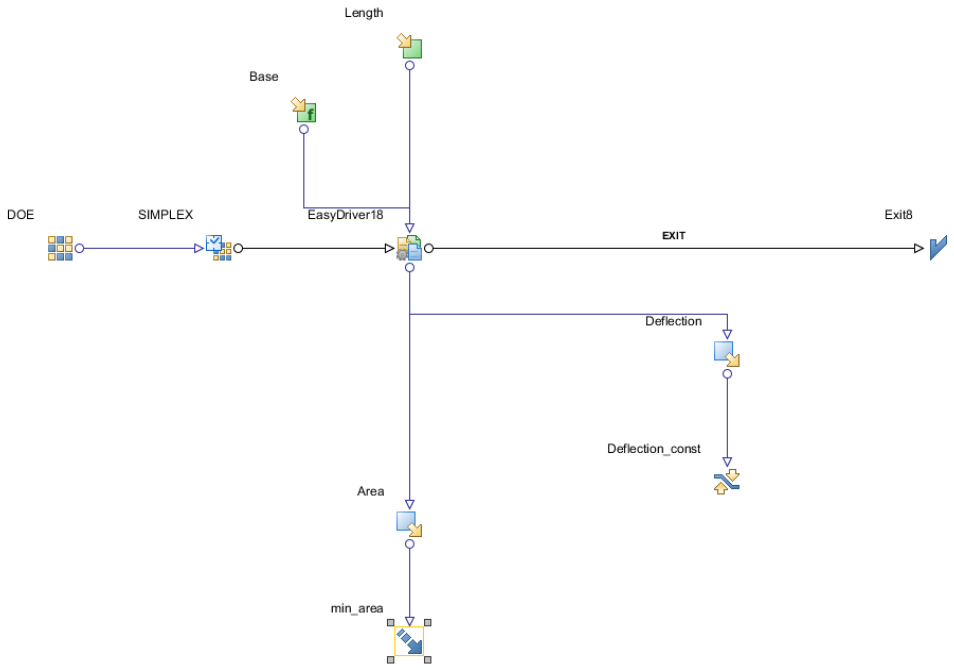


Figure 57. Minimizing Area

A third optimization process was completed, in which force and deflection were both optimized. The m-file was modified so the model was built twice in the same file and both force

and deflection were output by the file. In this case, the process is the same, except "Force" and "Deflection" are both design objectives and were both outputs to be maximized, as seen in Fig. 58. As stated above, both cannot be maximized, but an optimal finger that achieves an intermediate deflection and force can be created. The scheduler in this case will be the "NSGA-II." Once modeFRONTIER converges on a solution, the results can be seen in the Designs Space. This was again repeated for all three shapes.

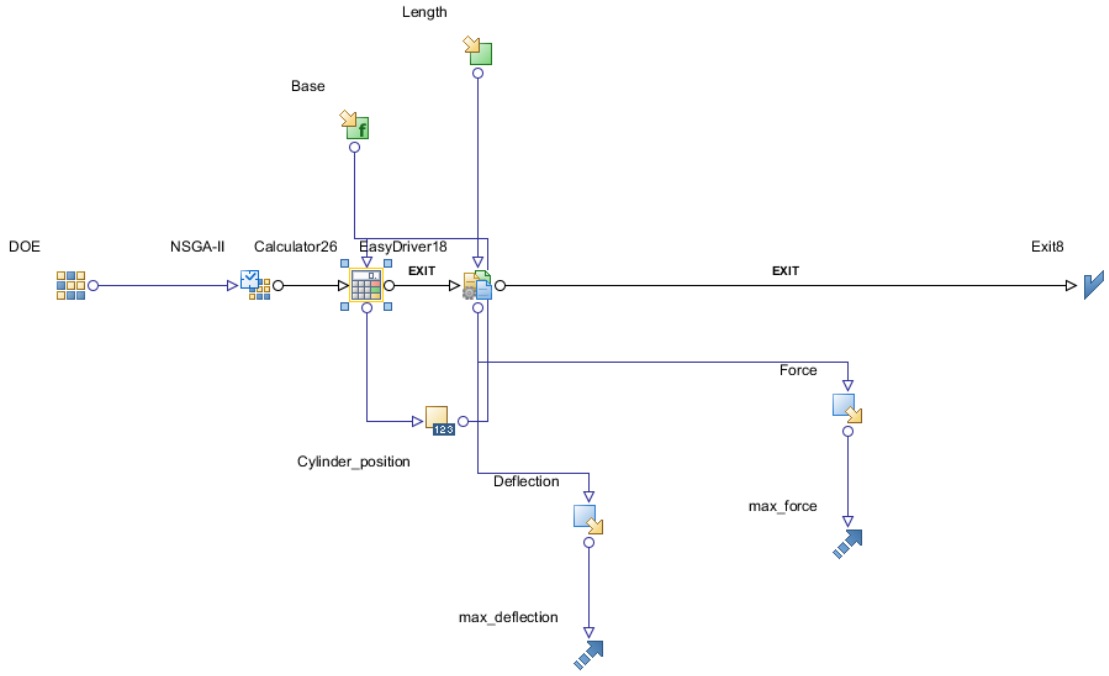


Figure 58. Maximizing Deflection and Force

4.3.1 Early Optimization

A 7x17 mm rectangle was modeled and input into modeFRONTIER. ModeFRONTIER ran different scenarios with different lengths and widths. These dimensions were used to optimize the fingers for force (or deflection). This was done manually in Comsol. This was very tedious, but led to some interesting conclusions. Examining Table 5, we can see a thin base and a longer length leads to more deflection, while a wide base and a shorter length leads to more force exerted. Also, choosing the middle ground leads to an intermediate deflection and force. This process was repeated to make the fingers smaller, as seen in Table 6 and Table 7. The results again are promising, as a wider base and shorter finger leads to a larger force exerted and vice versa for deflection.

Table 5 Full Rectangle Optimization

Full Rectangle Optimization (119mm^2)

Base (mm)	Length (mm)	Displacement (mm)	Force (mN)
6.4358	18.49	3.10	2.03
6.9655	17.084	1.718	2.52
7.9323	15.002	1.571	3.76

Table 6. Half Rectangle Optimization

Half Rectangle Optimization (59.5 mm^2)

Base (mm)	Length (mm)	Displacement (mm)	Force (mN)
3.8459	15.471	3.35	1.38
7.322	8.1262	0.528	2.49
11.899	5.0	0.055	3.87

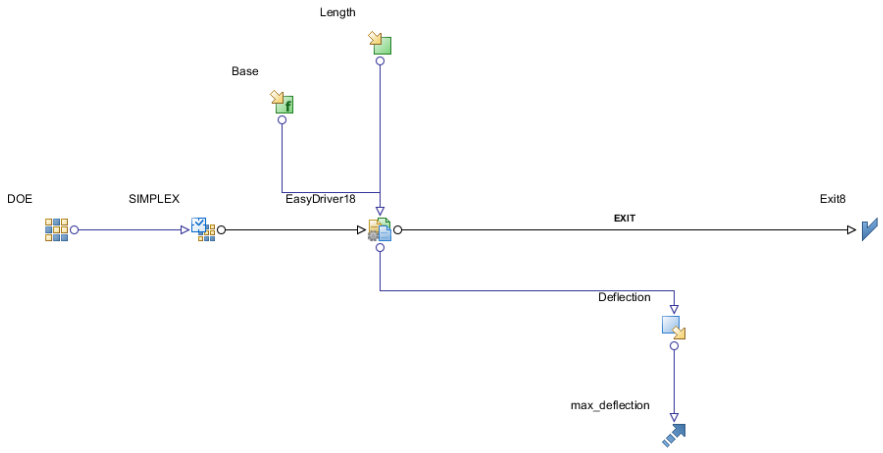
Table 7. Quarter Rectangle Optimization

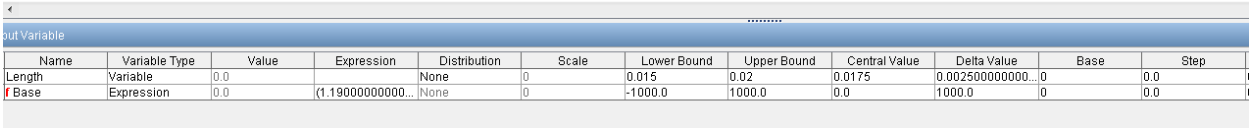
Quarter Rectangle Optimization (29.75 mm^2)

Base (mm)	Length (mm)	Displacement (mm)	Force (mN)
2.086	14.263	1.448	0.859
2.975	10.0	0.970	0.9813
14.8	2.0	0.013	1.547

4.3.2 Deflection Optimization

Next, modeFRONTIER was used to optimize the fingers for deflection. The same setup is used for each shape, but the script in the EasyDriver is changed according to the shape, as seen in Fig. 59. As can be seen, the base and length are the two input variables and base is made a function of length to keep the area the same. There are also limits set on the length, which in turn limits the base. These limits are arbitrary and can be set by the user. In our case, we limited the lengths for all shapes between 15 and 20 mm. As stated, these are arbitrary values and can be chosen by the user. If these limits are not set, modeFRONTIER would never converge on an answer and the finger would just become infinitely long by infinitely small. Gravity is also not included in any of these models, but if the finger was infinitely long, at some point gravity would overcome the stiffness of the finger and it would not be able to hold itself upright.





ut Variable

Name	Variable Type	Value	Expression	Distribution	Scale	Lower Bound	Upper Bound	Central Value	Delta Value	Base	Step
Length	Variable	0.0		None	0	0.015	0.02	0.0175	0.002500000000...	0	0.0
f Base	Expression	0.0	(1.19000000000...	None	0	-1000.0	1000.0	0.0	1000.0	0	0.0

Figure 59. Deflection Optimization

As seen in Fig. 60, the deflection of a 7x17 mm rectangle was optimized for deflection. ModeFRONTIER runs through the optimization process until the best design is acquired. Limits were set in modeFRONTIER to keep the finger under a certain width and length. As seen in Table 8, Design 20 is the best design for the given shape and modeFRONTIER is finished optimizing the shape. This is accomplished with a 5.95 mm by 20 mm finger. This makes sense as a finger with a thin base and a longer length is able to deflect more. The upper limit for length was set to 20 mm. If the limit were set to a higher value, the finger would continue to grow in length until the upper limit was met.

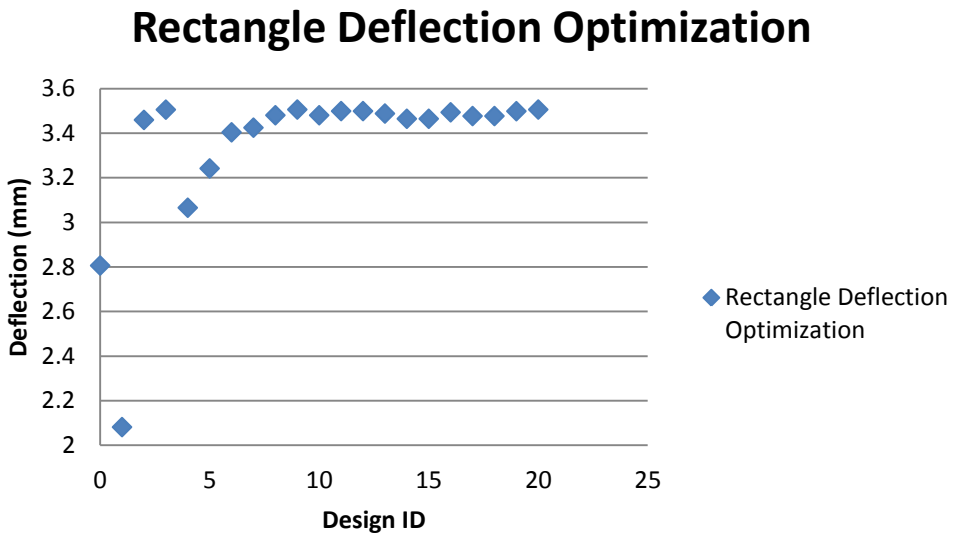


Figure 60. Rectangle Deflection

Table 8. Rectangle Design IDs

Design ID	Base (mm)	Length (mm)	Deflection (mm)
0	6.436	18.490	2.806
1	6.966	17.084	2.082
2	5.981	19.897	3.459
3	5.950	20.000	3.505
4	6.183	19.245	3.065
5	6.064	19.623	3.242
6	6.007	19.811	3.404

7	5.978	19.906	3.424
8	5.964	19.953	3.480
9	5.957	19.976	3.506
10	5.964	19.953	3.480
11	5.954	19.988	3.499
12	5.954	19.988	3.499
13	5.961	19.965	3.488
14	5.955	19.982	3.464
15	5.955	19.982	3.464
16	5.959	19.971	3.493
17	5.958	19.973	3.477
18	5.958	19.973	3.477
19	5.956	19.979	3.498
20	**5.957**	**19.978**	**3.506**

The process is then repeated for the isosceles triangle. As seen in Fig. 61 and Table 9, modeFRONTIER again converges to give the best design of an isosceles triangle. This finger is 11.9 mm width by 19.96 mm long. Again the length limit was set to 20 mm, so the optimization process chooses the finger that has the longest length. These results are promising, as we have seen a thinner base with a longer length is a better design when deflection is the design goal.

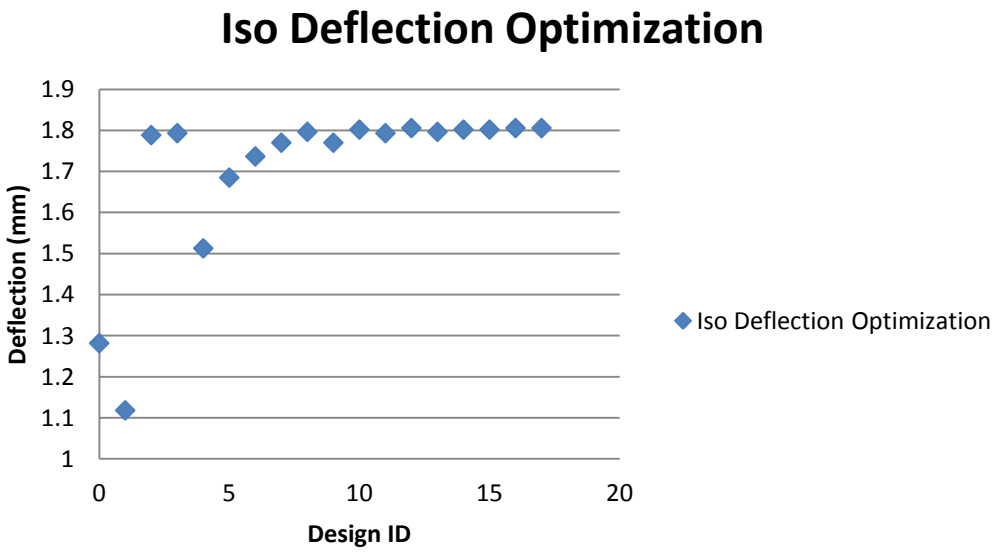


Figure 61. Iso Deflection Optimization

Table 9. Iso IDs

Design ID	Base (mm)	Length (mm)	Deflection (mm)
0	12.872	18.490	1.282
1	13.931	17.084	1.118
2	11.962	19.897	1.789
3	11.900	20.000	1.793
4	12.367	19.245	1.513
5	12.129	19.623	1.685
6	12.013	19.811	1.737
7	11.956	19.906	1.770
8	11.928	19.953	1.797
9	11.956	19.906	1.770
10	11.914	19.976	1.802
11	11.900	20.000	1.793
12	11.921	19.965	1.806
13	11.928	19.953	1.797
14	11.918	19.971	1.802
15	11.918	19.971	1.802
16	11.925	19.959	1.806
17	**11.923**	**19.962**	**1.806**

The process is repeated for a right triangle. Again, seen in Fig. 62 and Table 10, modeFRONTIER converges on a design of a thin base and longer length to achieve greater deflection. This finger is 11.9 mm wide by 19.99 mm long. As can be seen, the two triangles have similar dimensions even though they are different shapes.

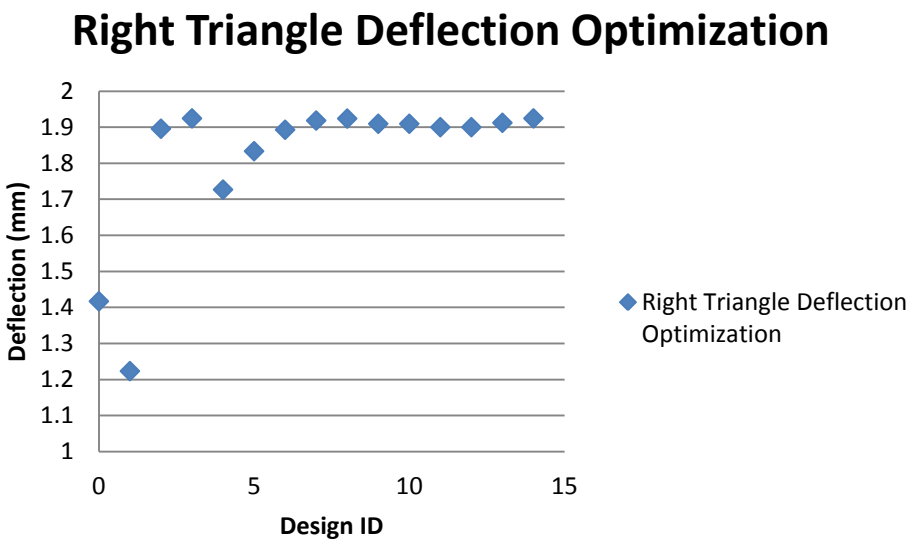


Figure 62. Right Triangle Deflection

Table 10. Right Design IDs

Design ID	Base (mm)	Length (mm)	Deflection (mm)
0	12.872	18.490	1.417
1	13.931	17.084	1.223
2	11.962	19.897	1.896
3	11.900	20.000	1.925
4	12.367	19.246	1.727
5	12.129	19.623	1.834
6	12.013	19.811	1.893
7	11.956	19.906	1.919
8	11.928	19.953	1.924
9	11.914	19.976	1.910
10	11.914	19.976	1.910
11	11.907	19.988	1.900
12	11.907	19.988	1.900
13	11.904	19.994	1.912
14	**11.902**	**19.997**	**1.925**

Now, we will compare the results. As seen in Fig. 63, the three shapes are compared. The rectangle achieves the greatest deflection with the same area as the triangles, due to the limits set on the fingers. This is achieved as the rectangle base becomes very thin and the length

is long compared to the base. The triangles cannot become as thin compared to their base, so they cannot deflect as much. When looking at the ratio of width to length, the rectangle is almost a 1:4 ratio, while the triangles are around 1:2. This means the rectangle becomes very long compared to the width, while the triangles do not If the triangles were able to become much longer and meet the same ratio, they too would be able to deflect more. As stated above, the length was limited, and in this case it was limited to 20 mm. This is an arbitrary number that can be changed. This is set as modeFRONTIER would never converge if there were no limit on the length. The triangles would be able to deflect more given a longer length and shorter base, but a limit must be set. This shows a rectangle may be the best design if a certain length is desired. If the user needs a finger to be less than a certain length, this can be changed in modeFRONTIER.

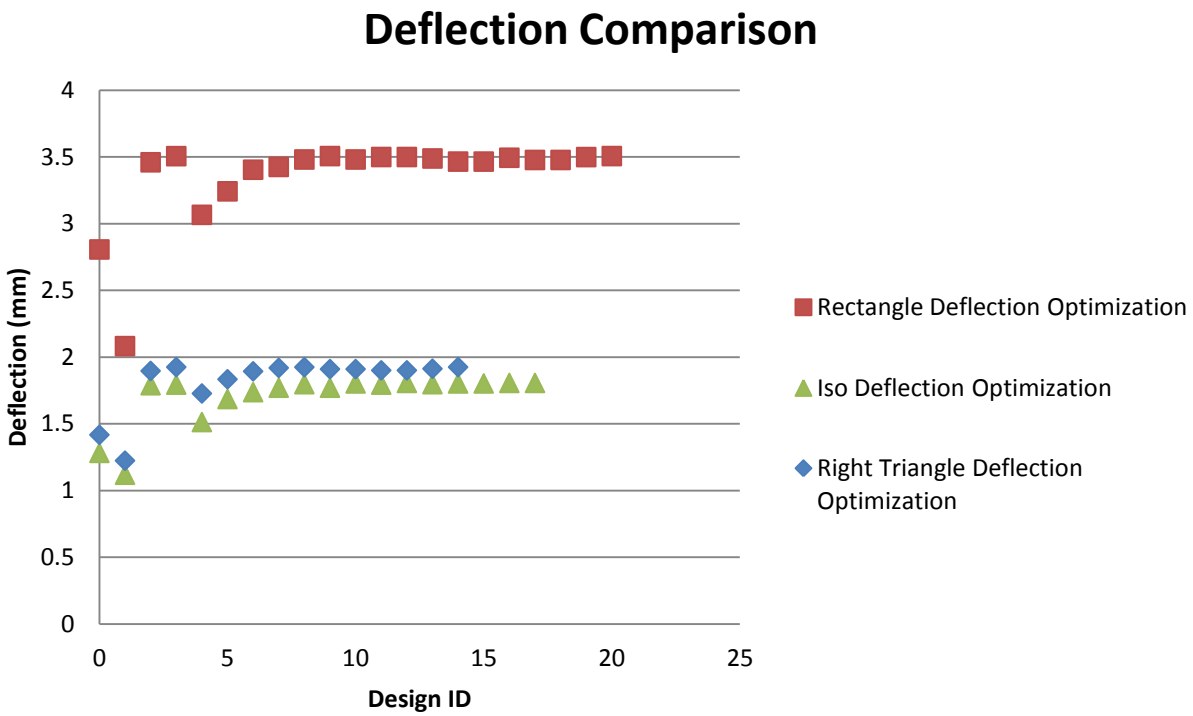
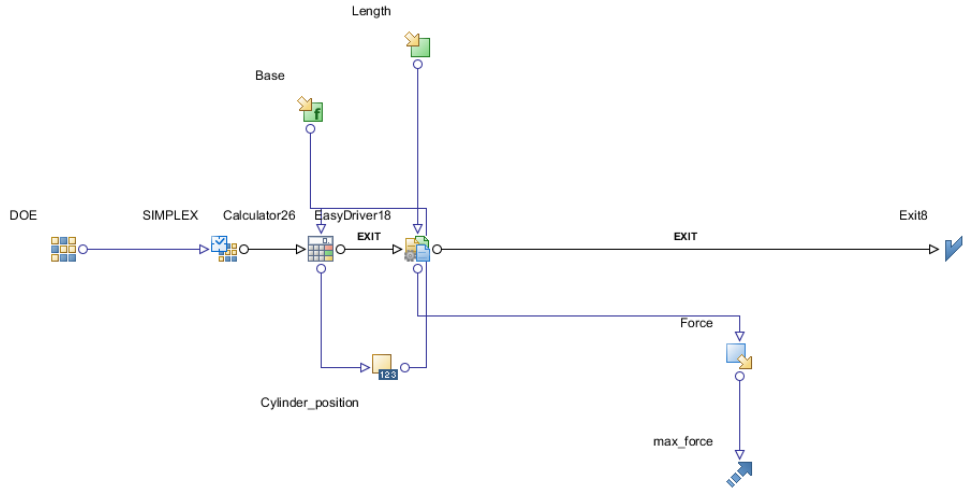


Figure 63. Shape Comparison

4.3.3 Force Optimization

The force exerted by the three shapes was then optimized in modeFRONTIER, as seen in Fig. 64. The same setup is used for each shape, but the script in the EasyDriver is changed according to the shape. As can be seen, the base and length are the two input variables and base is made a function of length to keep the area the same. There are also limits set on the length, which in turn limits the base. The length limit was again set to 15 to 20 mm. This again was arbitrary and can be changed to anything. The cylinder moving node is the cylinder that represents the force transducer that will always move 2 mm in from the tip of the finger to keep the readings consistent.



ut Variable

Name	Variable Type	Value	Expression	Distribution	Scale	Lower Bound	Upper Bound	Central Value	Delta Value	Base
Length	Variable	0.0		None	0	0.015	0.02	0.0175	0.002500000000...	0
Base	Expression	0.0	(1.19000000000...	None	0	-1000.0	1000.0	0.0	1000.0	0

Figure 64. Force Optimization

As seen in Fig. 65 and Table 11, modeFRONTIER converges on a design. Once again, a limit was set at the lower end of the length of the finger at 15 mm. This was done as modeFRONTIER would never converge on a design if there was no limit on the length or width. A wider base with a shorter length is the best design to achieve the greatest force. The rectangle finger dimensions are 7.92 mm wide by 15.02 mm long.

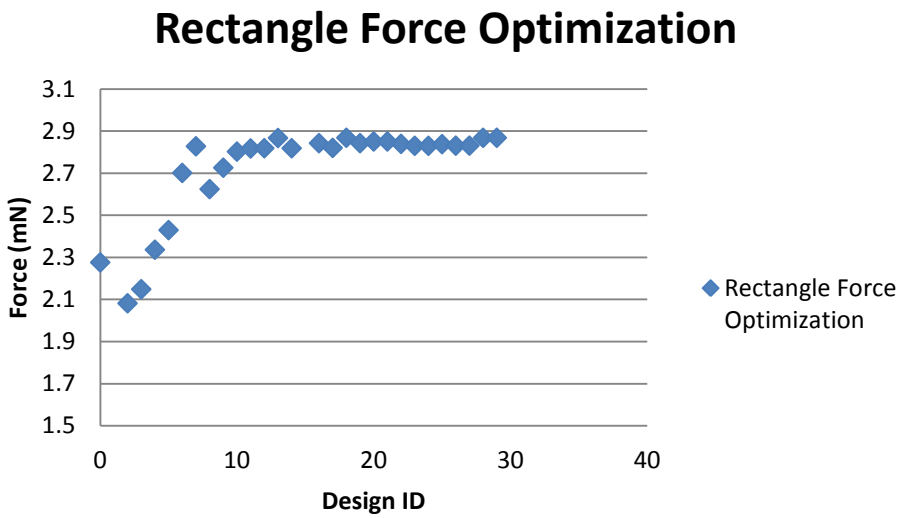


Figure 65. Rectangle Force Optimization

Table 11. Rectangle Design IDs

Design ID	Base (mm)	Length (mm)	Force (mN)
0	6.436	18.490	2.276
1	5.981	19.897	2.082
2	6.200	19.193	2.149
3	6.690	17.787	2.336
4	6.966	17.084	2.430
5	7.590	15.678	2.701
6	7.933	15.000	2.828
7	7.418	16.042	2.624
8	7.667	15.521	2.726
9	7.798	15.261	2.803
10	7.865	15.130	2.817
11	7.899	15.065	2.818
12	7.916	15.033	2.867

13	7.899	15.065	2.818
14	7.925	15.016	2.843
15	7.908	15.049	2.820
16	7.920	15.024	2.868
17	7.925	15.016	2.843
18	7.918	15.028	2.850
19	7.918	15.028	2.850
20	7.923	15.020	2.839
21	7.919	15.026	2.830
22	7.919	15.026	2.830
23	7.922	15.022	2.838
24	7.921	15.023	2.831
25	7.921	15.023	2.831
26	7.920	15.025	2.868
27	**7.920**	**15.025**	**2.868**

The force exerted by the isosceles triangle was then optimized, as seen in Fig. 66 and Table 12. The results are interesting as the triangle base and length are almost the same. Once again, a wider base with a shorter length is the best design goal, as the finger dimensions are 15.86 mm wide by 15.01 mm long. The triangle would continue to become wider and shorter if there was no limit. There must be a limit in order for modeFRONTIER to converge on a solution.

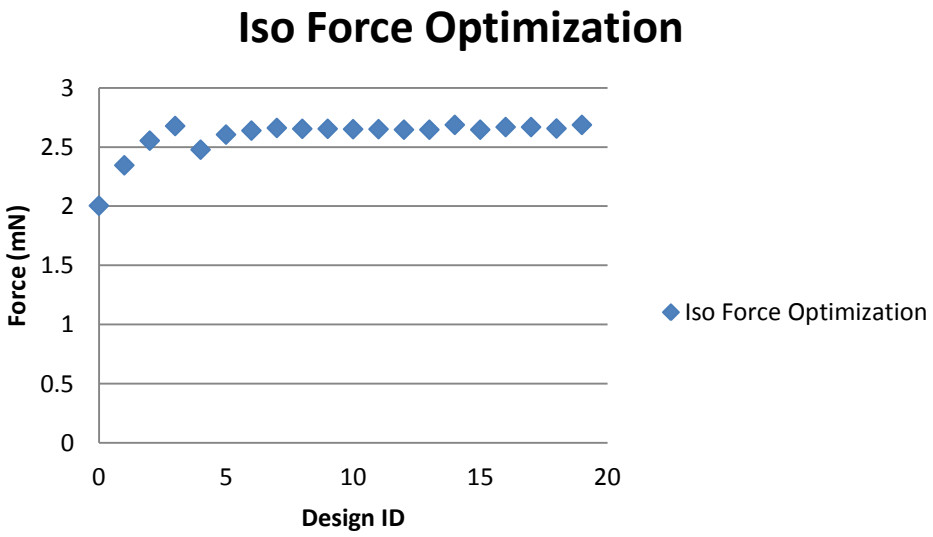


Figure 66. Iso Force Optimization

Table 12. Iso Design IDs

Design ID	Base (mm)	Length (mm)	Force (mN)
0	12.872	18.490	2.005
1	13.931	17.084	2.346
2	15.181	15.678	2.555
3	15.867	15.000	2.678
4	14.836	16.042	2.478
5	15.334	15.521	2.606
6	15.596	15.261	2.640
7	15.730	15.130	2.662
8	15.798	15.065	2.655
9	15.798	15.065	2.655
10	15.832	15.033	2.652
11	15.832	15.033	2.652
12	15.849	15.016	2.647
13	15.849	15.016	2.647
14	15.858	15.008	2.688
15	15.849	15.016	2.647
16	15.862	15.004	2.670
17	15.862	15.004	2.670
18	15.854	15.012	2.658
19	**15.860**	**15.006**	**2.688**

The design of the right triangle was then optimized for force, as seen in Fig. 67 and Table 13. Again, the base and length are almost the same, and a wider base with a shorter length is desired. The dimensions of this finger are 15.85 mm wide by 15.02 mm long.

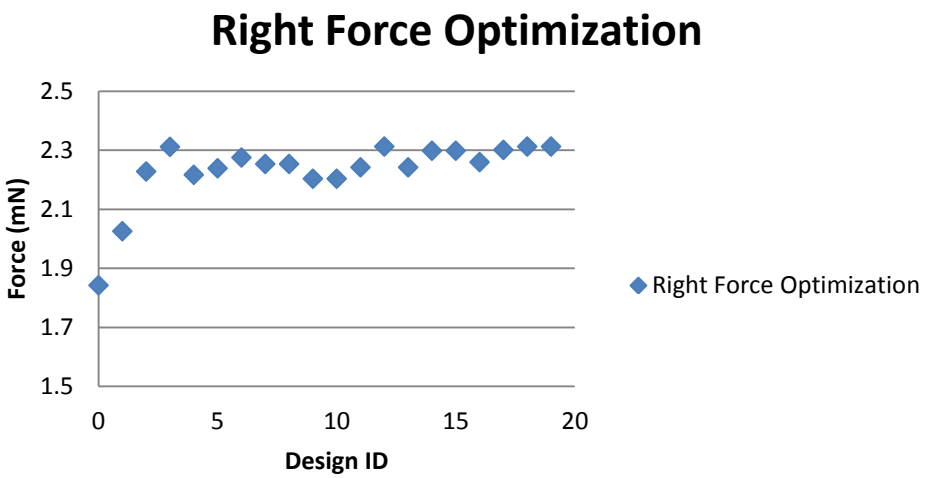


Figure 67. Right Triangle Force Optimization

Table 13. Right Triangle Design IDs

Design ID	Base (mm)	Length (mm)	Force (mN)
0	12.872	18.490	1.843
1	13.931	17.084	2.026
2	15.181	15.678	2.228
3	15.867	15.000	2.311
4	14.836	16.042	2.217
5	15.334	15.521	2.239
6	15.596	15.261	2.275
7	15.730	15.130	2.254
8	15.730	15.130	2.254
9	15.798	15.065	2.204
10	15.798	15.065	2.204
11	15.832	15.033	2.242
12	15.849	15.016	2.313
13	15.832	15.033	2.242
14	15.858	15.008	2.298
15	15.858	15.008	2.298
16	15.841	15.024	2.260
17	15.854	15.012	2.301
18	15.845	15.020	2.313
19	**15.847**	**15.018**	**2.313**

The three shapes were then compared, as seen in Fig. 68. The limit on these shapes plays a role in the optimization. Due to the shape of the triangles, the limits play a greater role than they do

on the rectangle. This can be seen as the rectangle finger can become much smaller in one dimension compared to the other, while the triangles dimensions are much closer to each other. This is due to the lower limit we set on the triangles. If we set the limit much lower for the length, the force would be much higher. We can see the triangles are going to the smallest possible limit for the length and trying to maximize their base, but this is not possible due to the limit. This can be seen in Table 14, as all fingers approach the limits to maximize either force or deflection.

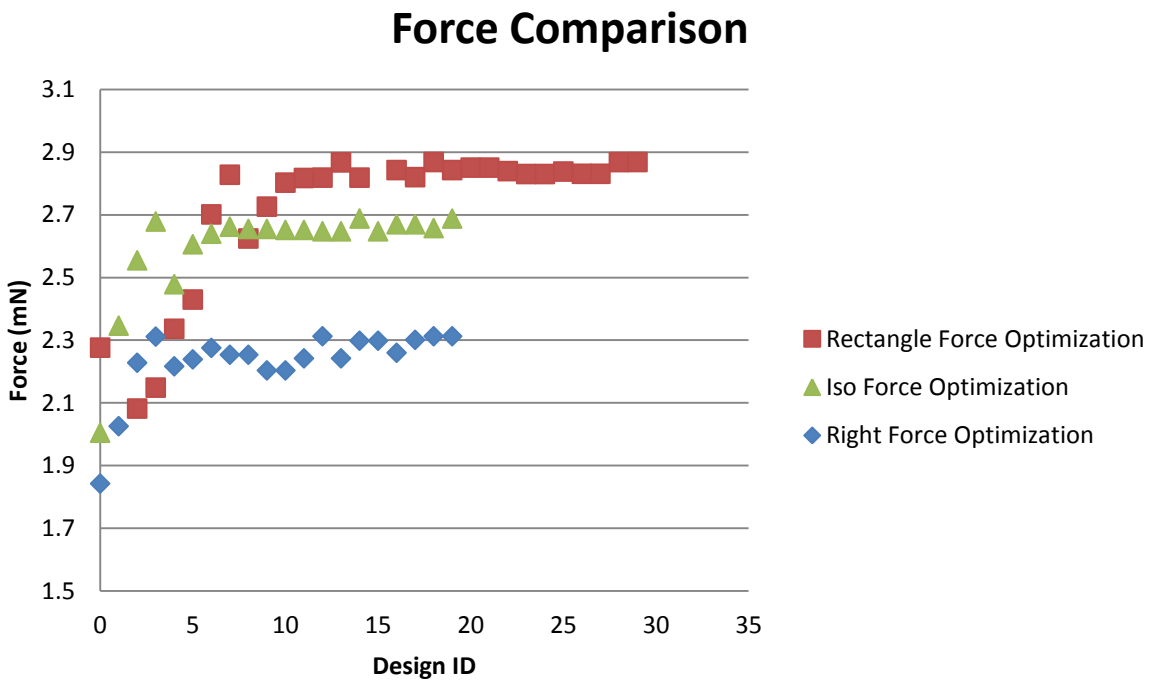


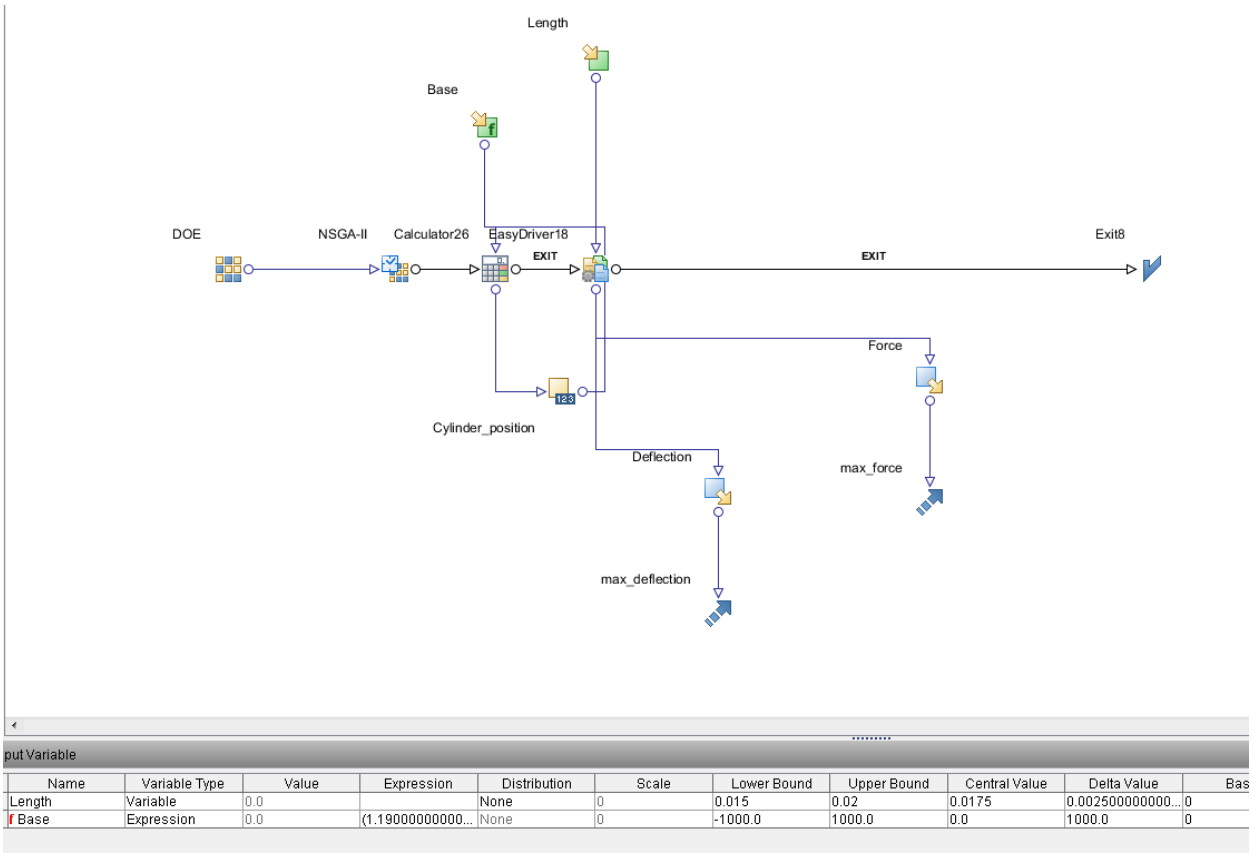
Figure 68. Shape Comparison

Table 14. Final Results

			Final Results			
Shape	Width (mm)	Length (mm)	Length Limit (mm)	Deflection (mm)	Force (mN)	Area (mm^2)
Rectangle	5.957	19.978	15-20	3.506	N/A	119
Iso Triangle	11.923	19.962	15-20	1.806	N/A	119
Right Triangle	11.902	19.997	15-20	1.925	N/A	119
Rectangle	7.92	15.025	15-20	N/A	2.868	119
Iso Triangle	15.86	15.006	15-20	N/A	2.688	119
Right Triangle	15.847	15.018	15-20	N/A	2.313	119

4.3.4 Optimization Factor

We defined the optimization factor, or the Index of Performance, as the product of the force and deflection in order to optimize each shape in terms of both. This is an arbitrary factor that we defined in order to optimize these fingers for both force and deflection, as there are no current equations or factors to represent our goal. This is important as design goals may include both the force and deflection. The setup in modeFRONTIER can be seen in Fig. 69. In this case, the force and deflection are both output variables. The limit on the length is again set to 15-20 mm. As stated earlier, a NSGA-II is used and a maximum number of designs are selected in the DOE. In our case we set the limit to 1000 and this limit was never reached by any optimization study.



put Variable

Name	Variable Type	Value	Expression	Distribution	Scale	Lower Bound	Upper Bound	Central Value	Delta Value	Bas
Length	Variable	0.0		None	0	0.015	0.02	0.0175	0.002500000000...	0
f Base	Expression	0.0	(1.19000000000...	None	0	-1000.0	1000.0	0.0	1000.0	0

Figure 69. Force and Deflection Optimization

The rectangle optimization of both force and deflection can be seen in Fig. 70 and the Design IDs can be seen in Table 15. In the case of the rectangle, the best design for achieving the highest force and deflection is Design 154. The shape of this finger is 5.95 mm width by 20 mm length.

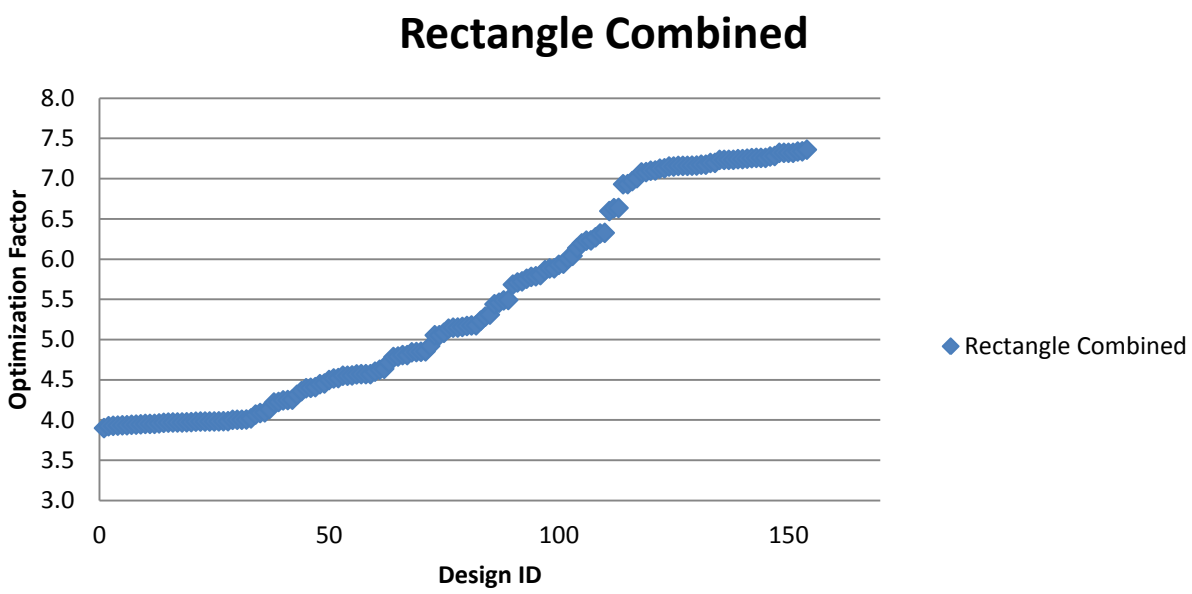


Figure 70. Rectangle Combined

Table 15. Rectangle Combined

Design ID	Base (mm)	Length (mm)	Deflection (mm)	Force (mN)	Index of Performance
1	7.908	15.048	1.383	2.820	3.900
2	7.861	15.139	1.402	2.798	3.924
3	7.928	15.011	1.386	2.835	3.928
4	7.933	15.000	1.390	2.828	3.931
5	7.890	15.083	1.398	2.815	3.935
6	7.912	15.040	1.390	2.831	3.935
7	7.899	15.065	1.398	2.818	3.941
8	7.922	15.022	1.389	2.838	3.943
9	7.886	15.089	1.402	2.815	3.947
10	7.931	15.004	1.384	2.855	3.952
11	7.931	15.004	1.384	2.855	3.952
12	7.927	15.012	1.387	2.850	3.952
13	7.929	15.009	1.387	2.853	3.958
14	7.869	15.122	1.409	2.815	3.966
15	7.892	15.078	1.406	2.823	3.968
16	7.932	15.002	1.395	2.846	3.970
17	7.932	15.002	1.395	2.846	3.970
18	7.932	15.002	1.395	2.846	3.970
19	7.894	15.075	1.409	2.820	3.972
20	7.847	15.165	1.421	2.798	3.974
21	7.900	15.063	1.410	2.823	3.980

22	7.902	15.059	1.414	2.816	3.981
23	7.902	15.060	1.414	2.816	3.981
24	7.865	15.130	1.413	2.817	3.982
25	7.901	15.062	1.414	2.816	3.982
26	7.930	15.006	1.391	2.864	3.984
27	7.904	15.056	1.404	2.838	3.986
28	7.862	15.136	1.415	2.817	3.986
29	7.849	15.162	1.428	2.804	4.003
30	7.916	15.033	1.396	2.867	4.003
31	7.916	15.034	1.396	2.867	4.003
32	7.806	15.244	1.443	2.776	4.006
33	7.895	15.073	1.418	2.836	4.020
34	7.791	15.273	1.453	2.800	4.068
35	7.753	15.348	1.485	2.752	4.087
36	7.779	15.298	1.466	2.793	4.096
37	7.788	15.280	1.473	2.807	4.134
38	7.618	15.622	1.561	2.703	4.218
39	7.619	15.620	1.561	2.706	4.225
40	7.605	15.648	1.573	2.698	4.245
41	7.609	15.640	1.571	2.705	4.248
42	7.655	15.545	1.556	2.733	4.253
43	7.563	15.734	1.603	2.691	4.314
44	7.584	15.690	1.607	2.712	4.359
45	7.466	15.940	1.664	2.642	4.396
46	7.510	15.845	1.643	2.678	4.400
47	7.471	15.929	1.660	2.656	4.409
48	7.470	15.931	1.662	2.675	4.446
49	7.463	15.945	1.675	2.660	4.454
50	7.437	16.002	1.705	2.638	4.497
51	7.451	15.972	1.692	2.670	4.517
52	7.387	16.110	1.719	2.630	4.522
53	7.352	16.187	1.754	2.595	4.551
54	7.348	16.195	1.755	2.593	4.551
55	7.430	16.016	1.718	2.652	4.554
56	7.338	16.217	1.771	2.579	4.568
57	7.333	16.228	1.773	2.578	4.570
58	7.339	16.215	1.763	2.593	4.571
59	7.339	16.215	1.763	2.593	4.571
60	7.308	16.284	1.788	2.573	4.601
61	7.315	16.268	1.797	2.573	4.625
62	7.314	16.271	1.799	2.578	4.637

63	7.255	16.402	1.832	2.574	4.716
64	7.234	16.450	1.859	2.573	4.783
65	7.245	16.424	1.857	2.579	4.789
66	7.157	16.627	1.911	2.516	4.807
67	7.157	16.627	1.911	2.516	4.807
68	7.158	16.625	1.909	2.536	4.842
69	7.133	16.684	1.924	2.518	4.845
70	7.138	16.671	1.932	2.510	4.849
71	7.202	16.523	1.899	2.556	4.854
72	7.099	16.763	1.969	2.499	4.920
73	7.035	16.914	2.029	2.491	5.054
74	6.983	17.042	2.068	2.445	5.054
75	6.969	17.076	2.088	2.435	5.084
76	7.391	16.101	1.840	2.791	5.136
77	7.002	16.994	2.077	2.479	5.148
78	6.998	17.005	2.080	2.476	5.149
79	7.017	16.960	2.062	2.501	5.157
80	6.937	17.153	2.124	2.433	5.169
81	6.910	17.221	2.138	2.421	5.176
82	6.933	17.164	2.132	2.428	5.177
83	6.896	17.255	2.161	2.423	5.235
84	6.903	17.238	2.164	2.443	5.286
85	6.863	17.340	2.203	2.410	5.310
86	6.765	17.590	2.303	2.362	5.440
87	6.775	17.565	2.287	2.387	5.458
88	6.849	17.376	2.248	2.441	5.487
89	6.842	17.391	2.266	2.424	5.492
90	6.614	17.993	2.487	2.285	5.683
91	6.587	18.066	2.493	2.291	5.710
92	6.595	18.045	2.490	2.299	5.724
93	6.631	17.947	2.469	2.331	5.756
94	6.567	18.121	2.532	2.282	5.779
95	6.558	18.145	2.530	2.287	5.787
96	6.644	17.910	2.469	2.350	5.801
97	6.531	18.222	2.568	2.283	5.863
98	6.521	18.247	2.587	2.274	5.884
99	6.525	18.237	2.587	2.275	5.886
100	6.499	18.312	2.608	2.273	5.929
101	6.495	18.323	2.622	2.267	5.943
102	6.450	18.449	2.672	2.248	6.008
103	6.460	18.421	2.679	2.255	6.040

104	6.435	18.494	2.751	2.231	6.137
105	6.355	18.725	2.805	2.209	6.195
106	6.366	18.693	2.783	2.239	6.230
107	6.391	18.621	2.790	2.234	6.233
108	6.356	18.722	2.819	2.225	6.272
109	6.350	18.742	2.825	2.237	6.319
110	6.334	18.788	2.873	2.202	6.326
111	6.294	18.906	2.984	2.211	6.597
112	6.268	18.986	2.979	2.227	6.634
113	6.162	19.311	3.104	2.138	6.636
114	6.022	19.761	3.311	2.093	6.931
115	6.042	19.695	3.297	2.103	6.932
116	6.023	19.757	3.338	2.088	6.970
117	6.021	19.765	3.351	2.091	7.006
118	6.015	19.785	3.384	2.092	7.078
119	6.008	19.806	3.380	2.095	7.082
120	6.099	19.510	3.264	2.175	7.099
121	6.089	19.543	3.293	2.157	7.101
122	6.079	19.576	3.320	2.146	7.124
123	6.004	19.821	3.396	2.099	7.129
124	5.994	19.852	3.416	2.093	7.151
125	5.990	19.868	3.427	2.086	7.151
126	5.978	19.907	3.424	2.090	7.159
127	6.003	19.824	3.399	2.106	7.159
128	5.955	19.984	3.464	2.067	7.160
129	5.954	19.985	3.457	2.072	7.161
130	5.998	19.840	3.440	2.082	7.163
131	6.015	19.783	3.405	2.107	7.174
132	6.015	19.784	3.405	2.107	7.174
133	5.996	19.847	3.438	2.093	7.195
134	5.995	19.850	3.434	2.096	7.196
135	5.962	19.960	3.465	2.088	7.234
136	5.962	19.960	3.465	2.088	7.234
137	5.961	19.961	3.488	2.074	7.234
138	5.961	19.962	3.488	2.074	7.234
139	5.966	19.946	3.487	2.077	7.243
140	5.966	19.946	3.487	2.077	7.243
141	5.969	19.935	3.478	2.084	7.249
142	5.968	19.941	3.483	2.083	7.255
143	5.960	19.966	3.487	2.082	7.258
144	5.960	19.966	3.487	2.082	7.258

145	5.960	19.968	3.487	2.082	7.258
146	5.959	19.970	3.493	2.083	7.276
147	5.989	19.869	3.486	2.089	7.281
148	5.974	19.919	3.532	2.073	7.322
149	5.974	19.919	3.532	2.073	7.322
150	5.974	19.918	3.532	2.073	7.322
151	5.974	19.921	3.532	2.073	7.322
152	5.967	19.942	3.535	2.076	7.337
153	5.957	19.976	3.506	2.094	7.340
154	**5.953**	**19.991**	**3.511**	**2.096**	**7.360**

This was repeated for the isosceles triangle, as seen in Fig. 71 and Table 16. The highest optimization factor was accomplished by Design 161. This finger was 12.1 mm wide by 19.7 mm long. As can be seen, the deflection and force are similar values. If a finger is desired of both a higher degree of force and deflection, this is the best design.

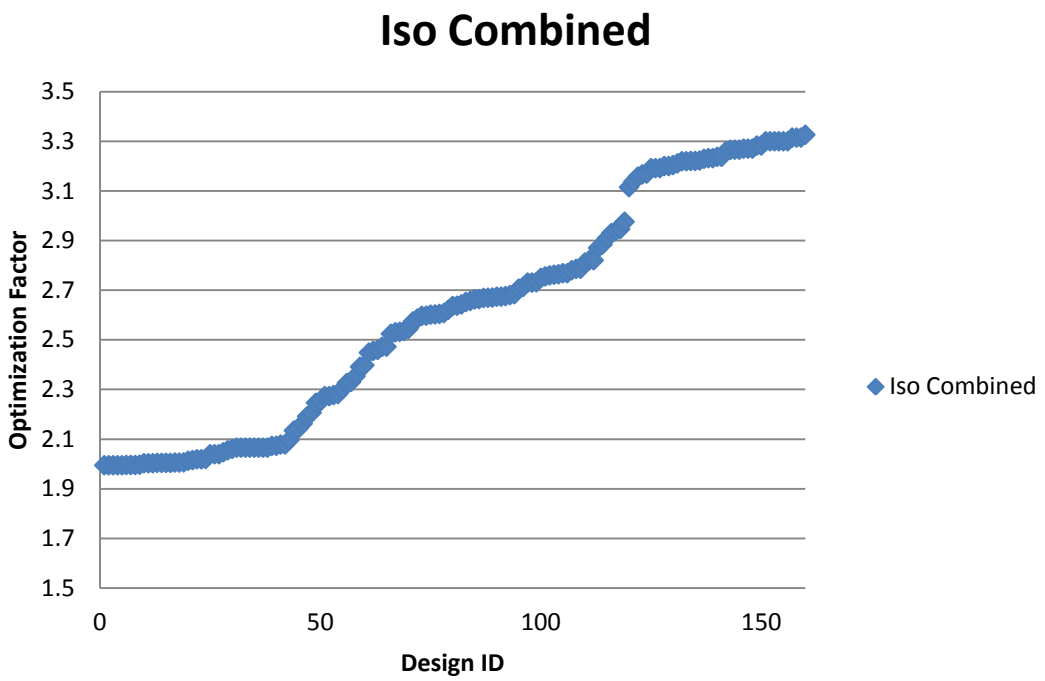


Figure 71. Iso Combined

Table 16. Iso Combined Design IDs

Design ID	Base (mm)	Length (mm)	Deflection (mm)	Force (mN)	Index of Performance
1	15.866	15.000	0.745	2.678	1.994
2	15.866	15.001	0.745	2.678	1.994
3	15.865	15.001	0.745	2.678	1.994
4	15.866	15.001	0.745	2.678	1.994
5	15.867	15.000	0.745	2.678	1.994
6	15.862	15.004	0.747	2.670	1.995
7	15.862	15.004	0.747	2.670	1.995
8	15.863	15.004	0.747	2.670	1.995
9	15.798	15.065	0.752	2.655	1.996
10	15.839	15.026	0.751	2.668	2.002
11	15.836	15.029	0.751	2.668	2.002
12	15.837	15.028	0.751	2.668	2.002
13	15.832	15.033	0.756	2.652	2.004
14	15.831	15.034	0.756	2.652	2.004
15	15.831	15.034	0.756	2.652	2.004
16	15.831	15.034	0.756	2.652	2.004
17	15.758	15.103	0.760	2.639	2.005
18	15.759	15.103	0.760	2.639	2.005
19	15.815	15.049	0.755	2.656	2.006
20	15.854	15.012	0.757	2.658	2.012
21	15.735	15.125	0.764	2.638	2.015
22	15.786	15.077	0.756	2.670	2.018
23	15.787	15.076	0.756	2.670	2.018
24	15.788	15.075	0.757	2.667	2.020
25	15.822	15.042	0.759	2.685	2.039
26	15.822	15.042	0.759	2.685	2.039
27	15.824	15.041	0.759	2.685	2.039
28	15.745	15.116	0.764	2.678	2.047
29	15.750	15.111	0.770	2.670	2.055
30	15.680	15.179	0.778	2.649	2.060
31	15.690	15.169	0.772	2.675	2.065
32	15.860	15.007	0.768	2.688	2.065
33	15.860	15.006	0.768	2.688	2.065
34	15.855	15.011	0.768	2.688	2.065
35	15.857	15.009	0.768	2.688	2.065
36	15.860	15.006	0.768	2.688	2.065

37	15.861	15.005	0.768	2.688	2.065
38	15.857	15.009	0.768	2.688	2.065
39	15.423	15.431	0.809	2.561	2.072
40	15.723	15.137	0.779	2.660	2.073
41	15.589	15.267	0.789	2.632	2.078
42	15.548	15.307	0.792	2.624	2.078
43	15.595	15.261	0.789	2.658	2.096
44	15.394	15.460	0.817	2.610	2.134
45	15.388	15.467	0.822	2.606	2.142
46	15.210	15.648	0.842	2.569	2.162
47	15.110	15.752	0.859	2.551	2.191
48	14.996	15.871	0.874	2.524	2.206
49	14.995	15.872	0.892	2.519	2.246
50	15.033	15.831	0.884	2.548	2.251
51	14.827	16.052	0.912	2.492	2.273
52	14.829	16.049	0.912	2.492	2.273
53	14.813	16.067	0.920	2.476	2.277
54	14.911	15.962	0.901	2.532	2.281
55	14.880	15.995	0.912	2.523	2.301
56	14.668	16.226	0.947	2.457	2.326
57	14.704	16.186	0.951	2.451	2.330
58	14.475	16.443	0.966	2.436	2.353
59	14.380	16.551	0.991	2.414	2.391
60	14.481	16.435	0.983	2.439	2.397
61	14.314	16.627	1.027	2.383	2.448
62	14.282	16.665	1.023	2.401	2.455
63	14.217	16.741	1.032	2.382	2.459
64	14.210	16.749	1.043	2.370	2.472
65	14.298	16.645	1.032	2.394	2.472
66	14.119	16.857	1.069	2.361	2.524
67	14.071	16.914	1.081	2.341	2.530
68	13.996	17.004	1.087	2.330	2.532
69	13.996	17.005	1.091	2.324	2.535
70	14.141	16.830	1.065	2.388	2.544
71	13.950	17.061	1.096	2.348	2.574
72	12.821	18.563	1.299	1.990	2.585
73	12.805	18.587	1.306	1.987	2.596
74	13.505	17.624	1.180	2.200	2.597
75	13.760	17.296	1.130	2.303	2.601
76	13.976	17.030	1.107	2.350	2.602
77	12.784	18.617	1.312	1.985	2.604

78	12.951	18.377	1.279	2.038	2.607
79	14.063	16.923	1.098	2.386	2.620
80	13.926	17.091	1.113	2.368	2.636
81	13.896	17.127	1.122	2.350	2.637
82	12.668	18.788	1.345	1.965	2.642
83	13.770	17.284	1.159	2.288	2.652
84	13.618	17.477	1.167	2.276	2.656
85	12.672	18.782	1.361	1.957	2.663
86	12.638	18.832	1.356	1.964	2.664
87	13.802	17.244	1.154	2.313	2.668
88	12.721	18.709	1.339	1.994	2.669
89	12.561	18.947	1.366	1.954	2.670
90	13.428	17.724	1.226	2.181	2.673
91	13.428	17.724	1.226	2.181	2.673
92	12.712	18.722	1.341	1.996	2.676
93	12.606	18.880	1.368	1.959	2.680
94	13.006	18.299	1.293	2.075	2.683
95	13.667	17.414	1.182	2.289	2.706
96	13.601	17.499	1.191	2.278	2.713
97	13.219	18.005	1.279	2.134	2.730
98	13.224	17.998	1.279	2.134	2.730
99	13.244	17.970	1.290	2.118	2.731
100	13.116	18.145	1.301	2.113	2.749
101	13.057	18.227	1.324	2.081	2.755
102	13.168	18.075	1.303	2.118	2.759
103	13.179	18.059	1.306	2.114	2.762
104	12.543	18.975	1.379	2.004	2.764
105	13.134	18.121	1.304	2.122	2.768
106	13.134	18.121	1.304	2.122	2.768
107	13.050	18.237	1.334	2.084	2.780
108	13.139	18.114	1.319	2.112	2.785
109	12.532	18.991	1.447	1.925	2.786
110	13.105	18.161	1.323	2.124	2.811
111	12.513	19.021	1.466	1.924	2.820
112	12.514	19.019	1.466	1.924	2.820
113	12.433	19.143	1.490	1.926	2.870
114	12.539	18.981	1.447	1.994	2.884
115	12.457	19.106	1.456	1.998	2.909
116	12.232	19.457	1.554	1.885	2.930
117	12.295	19.357	1.546	1.901	2.939
118	12.270	19.397	1.543	1.908	2.946

119	12.214	19.485	1.570	1.895	2.975
120	12.153	19.583	1.673	1.861	3.115
121	12.132	19.618	1.685	1.862	3.137
122	12.119	19.639	1.698	1.859	3.157
123	12.072	19.714	1.703	1.859	3.166
124	12.140	19.605	1.691	1.874	3.169
125	11.942	19.929	1.753	1.821	3.191
126	11.939	19.934	1.753	1.821	3.191
127	12.016	19.806	1.733	1.841	3.191
128	12.047	19.756	1.718	1.862	3.200
129	12.045	19.759	1.718	1.862	3.200
130	11.907	19.989	1.761	1.819	3.204
131	12.096	19.676	1.722	1.864	3.211
132	12.031	19.783	1.731	1.860	3.219
133	12.031	19.783	1.731	1.860	3.219
134	12.020	19.800	1.738	1.853	3.220
135	12.021	19.798	1.738	1.853	3.220
136	11.935	19.941	1.764	1.827	3.222
137	11.956	19.906	1.770	1.824	3.229
138	12.052	19.748	1.735	1.863	3.232
139	12.049	19.753	1.735	1.863	3.232
140	12.008	19.820	1.762	1.837	3.237
141	12.007	19.821	1.762	1.837	3.237
142	11.902	19.996	1.793	1.818	3.260
143	11.949	19.918	1.785	1.829	3.265
144	11.970	19.883	1.779	1.835	3.266
145	11.974	19.876	1.779	1.835	3.266
146	11.981	19.865	1.771	1.846	3.270
147	11.977	19.872	1.771	1.846	3.270
148	11.975	19.875	1.771	1.846	3.270
149	11.914	19.977	1.802	1.821	3.282
150	11.911	19.982	1.802	1.821	3.282
151	11.919	19.969	1.802	1.831	3.299
152	11.921	19.965	1.802	1.831	3.299
153	11.918	19.970	1.802	1.831	3.299
154	11.918	19.970	1.802	1.831	3.299
155	11.920	19.967	1.802	1.831	3.299
156	11.917	19.971	1.802	1.831	3.299
157	11.938	19.937	1.805	1.836	3.314
158	11.939	19.935	1.805	1.836	3.314
159	11.937	19.939	1.805	1.836	3.314

160	11.927	19.955	1.817	1.830	3.326
161	**12.080**	**19.701**	**1.727**	**1.934**	**3.340**

Finally, the right triangle was optimized for both force and deflection, as seen in Fig. 72 and Table 17. The best optimization factor is achieved by Design 116. As seen in this case and the previous isosceles triangle, a base of around 12 mm and a length of around 19.6 mm give the best design in both cases. The dimensions of the right triangle finger are 12.06 mm wide by 19.73 mm long. Again, the force and deflection values are similar to the isosceles triangle results. This shows the optimization process is close to choosing the best design in both cases, instead of having a one-sided design goal. The best design chosen was the design with intermediate values for both force and deflection.

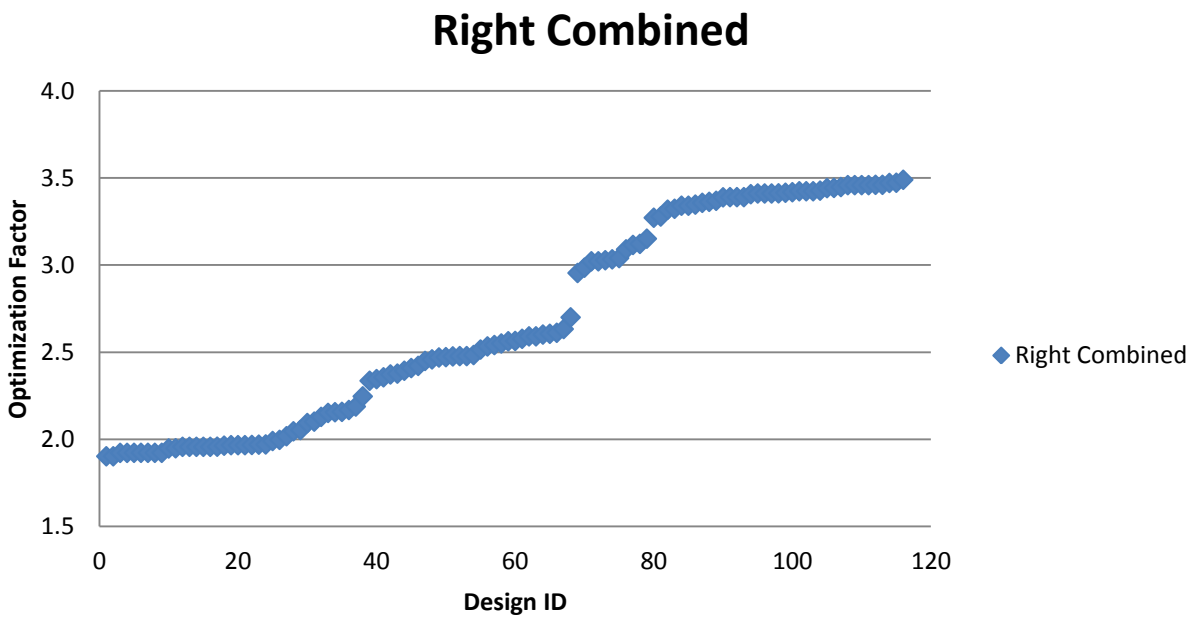


Figure 72. Right Combined

Table 17. Right Triangle Design IDs

Design ID	Base (mm)	Length (mm)	Deflection (mm)	Force (mN)	Index of Performance
1	15.863	15.004	0.840	2.265	1.903

2	15.863	15.004	0.840	2.265	1.903
3	15.857	15.009	0.837	2.298	1.923
4	15.858	15.009	0.837	2.298	1.923
5	15.833	15.032	0.858	2.242	1.923
6	15.831	15.034	0.858	2.242	1.923
7	15.832	15.033	0.858	2.242	1.923
8	15.831	15.034	0.858	2.242	1.923
9	15.831	15.034	0.858	2.242	1.923
10	15.815	15.049	0.859	2.268	1.948
11	15.822	15.042	0.865	2.253	1.949
12	15.867	15.000	0.847	2.311	1.958
13	15.867	15.000	0.847	2.311	1.958
14	15.867	15.000	0.847	2.311	1.958
15	15.865	15.002	0.847	2.311	1.958
16	15.867	15.000	0.847	2.311	1.958
17	15.724	15.136	0.870	2.252	1.958
18	15.746	15.115	0.867	2.266	1.964
19	15.839	15.027	0.865	2.275	1.967
20	15.836	15.029	0.865	2.275	1.967
21	15.838	15.027	0.865	2.275	1.967
22	15.764	15.098	0.873	2.255	1.968
23	15.854	15.012	0.856	2.301	1.970
24	15.788	15.075	0.871	2.264	1.971
25	15.588	15.268	0.880	2.262	1.991
26	15.848	15.017	0.864	2.313	1.998
27	15.592	15.264	0.892	2.262	2.018
28	15.541	15.314	0.911	2.245	2.047
29	15.534	15.322	0.902	2.277	2.054
30	15.377	15.478	0.929	2.255	2.094
31	15.210	15.648	0.961	2.187	2.102
32	15.313	15.542	0.940	2.264	2.129
33	15.143	15.717	0.965	2.230	2.152
34	15.170	15.689	0.958	2.251	2.156
35	15.146	15.714	0.968	2.230	2.158
36	15.126	15.735	0.973	2.229	2.169
37	15.031	15.834	0.989	2.213	2.189
38	15.053	15.811	0.997	2.255	2.247
39	14.549	16.358	1.064	2.196	2.337
40	14.536	16.373	1.080	2.173	2.346
41	14.368	16.565	1.114	2.116	2.356
42	14.535	16.375	1.081	2.194	2.373

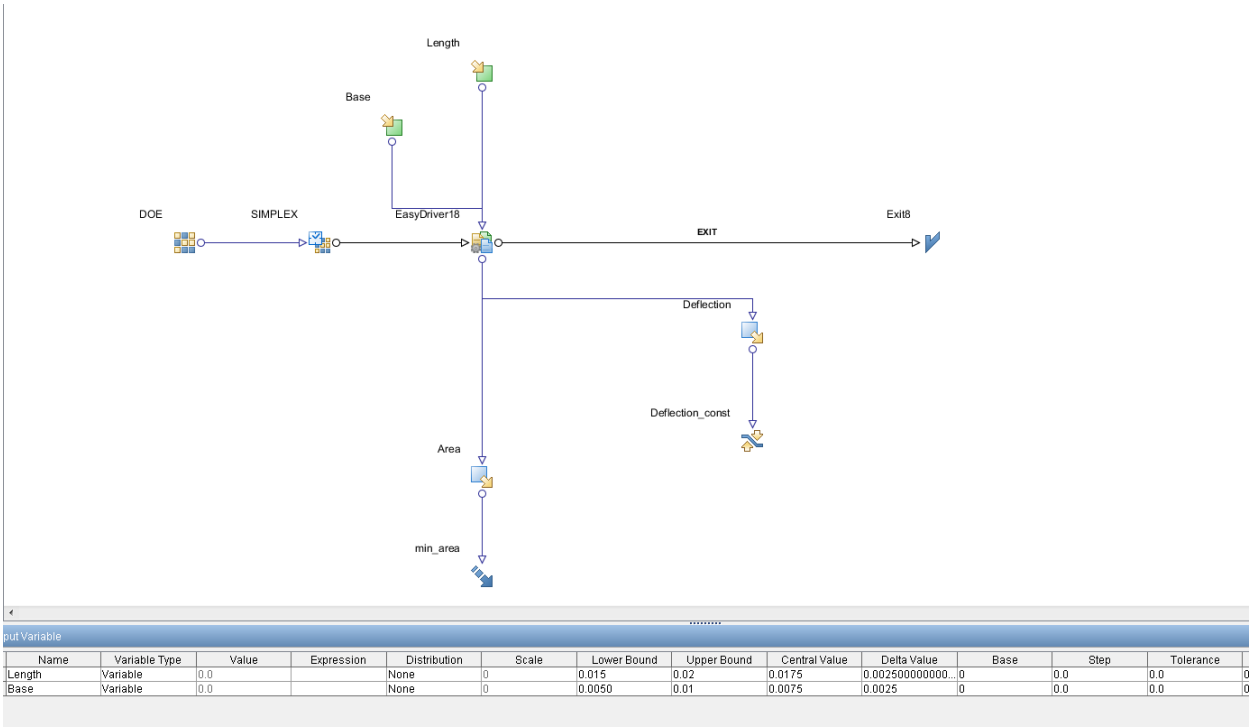
43	14.462	16.457	1.097	2.167	2.377
44	14.392	16.537	1.123	2.132	2.394
45	14.314	16.627	1.127	2.137	2.409
46	13.509	17.618	1.257	1.927	2.422
47	13.485	17.650	1.274	1.924	2.450
48	13.435	17.714	1.286	1.913	2.459
49	13.740	17.322	1.259	1.961	2.470
50	14.101	16.878	1.193	2.073	2.472
51	14.118	16.858	1.187	2.087	2.476
52	13.931	17.084	1.223	2.026	2.478
53	13.932	17.083	1.223	2.026	2.478
54	14.012	16.986	1.183	2.100	2.484
55	14.044	16.947	1.181	2.131	2.516
56	13.134	18.121	1.346	1.882	2.533
57	13.996	17.005	1.213	2.095	2.542
58	13.959	17.050	1.229	2.076	2.551
59	13.798	17.249	1.247	2.055	2.563
60	14.071	16.914	1.195	2.146	2.565
61	13.920	17.098	1.225	2.103	2.577
62	13.116	18.145	1.375	1.884	2.591
63	13.877	17.151	1.232	2.103	2.591
64	13.013	18.290	1.401	1.857	2.602
65	13.901	17.121	1.231	2.116	2.606
66	12.989	18.323	1.397	1.870	2.612
67	13.033	18.261	1.396	1.886	2.633
68	12.830	18.551	1.465	1.843	2.700
69	12.710	18.726	1.566	1.887	2.955
70	12.712	18.722	1.564	1.908	2.985
71	12.658	18.802	1.597	1.892	3.022
72	12.657	18.804	1.597	1.892	3.022
73	12.668	18.788	1.602	1.890	3.028
74	12.630	18.845	1.651	1.838	3.033
75	12.609	18.875	1.633	1.862	3.040
76	12.601	18.888	1.613	1.916	3.090
77	12.510	19.025	1.664	1.873	3.116
78	12.453	19.111	1.658	1.883	3.122
79	12.484	19.064	1.678	1.878	3.152
80	12.283	19.377	1.774	1.845	3.272
81	12.142	19.601	1.793	1.829	3.279
82	12.130	19.620	1.834	1.810	3.318
83	12.127	19.626	1.818	1.828	3.323

84	12.009	19.818	1.852	1.804	3.341
85	12.007	19.821	1.852	1.804	3.341
86	12.096	19.675	1.831	1.828	3.347
87	11.925	19.958	1.885	1.782	3.357
88	12.047	19.757	1.848	1.820	3.363
89	12.058	19.737	1.859	1.812	3.369
90	11.997	19.838	1.877	1.806	3.389
91	12.031	19.783	1.872	1.811	3.390
92	12.031	19.783	1.872	1.811	3.390
93	12.034	19.778	1.872	1.811	3.390
94	11.909	19.985	1.900	1.793	3.407
95	11.914	19.977	1.910	1.786	3.411
96	11.911	19.981	1.910	1.786	3.411
97	11.910	19.984	1.910	1.786	3.411
98	11.991	19.849	1.882	1.813	3.412
99	11.904	19.993	1.912	1.786	3.415
100	11.978	19.870	1.889	1.810	3.418
101	11.926	19.957	1.917	1.786	3.424
102	11.926	19.956	1.917	1.786	3.424
103	11.927	19.955	1.917	1.786	3.424
104	12.040	19.767	1.896	1.808	3.427
105	11.920	19.966	1.933	1.781	3.442
106	12.016	19.806	1.872	1.839	3.443
107	11.900	20.000	1.925	1.792	3.448
108	11.938	19.937	1.925	1.797	3.459
109	11.937	19.939	1.925	1.797	3.459
110	11.938	19.936	1.925	1.797	3.459
111	11.935	19.941	1.925	1.797	3.459
112	11.957	19.904	1.896	1.825	3.461
113	11.961	19.897	1.896	1.825	3.461
114	11.969	19.884	1.919	1.809	3.471
115	11.939	19.935	1.923	1.806	3.473
116	**12.063**	**19.730**	**1.859**	**1.877**	**3.490**

4.3.5 Rectangle Deflection and Force Limits

The deflection and force limits of these three shapes were then optimized. This was done by minimizing the area of the finger while achieving a deflection or force set by the user. The rectangle base was limited to 5 to 10 mm and the length was set to 15 to 20 mm. This was an

arbitrary number set by the user and can be changed. Because the area of the rectangle is simply length times width, the width was kept between 5 and 10 mm to match the area of the triangles. In the first case, the desired deflection was set to "Equal to" 2 mm. The modeFRONTIER setup can be seen in Fig. 73. The limits can be seen in the bottom and the minimize area node can be seen in the figure. This setup will also be used for the force limit optimization study. The only difference is the model will optimize for force instead of deflection. The limits will be the same and the force will be set to equal to 2 mN.



Name	Variable Type	Value	Expression	Distribution	Scale	Lower Bound	Upper Bound	Central Value	Delta Value	Base	Step	Tolerance
Length	Variable	0.0		None	0	0.015	0.02	0.0175	0.002500000000...	0	0.0	0.0
Base	Variable	0.0		None	0	0.0050	0.01	0.0075	0.0025	0	0.0	0.0

Figure 73. Rectangle Deflection Limit

As seen in Fig. 74, the deflection of a rectangle ranges significantly when changing the shape of the finger. But, as can be seen, a finger with an area of 160 mm^2 is capable of the most deflection. Also, a finger with an area of 100 mm^2 is capable of deflecting the same amount as fingers with a larger area. If a deflection of 2 mm or higher is desired, a finger with an area of 90

mm^2 is sufficient. Also, as seen in Fig. 74, many fingers deflect much less than the smaller finger and have a larger area. This shows this optimization process can be very crucial in designing fingers, as we can eliminate waste and achieve certain design goals.

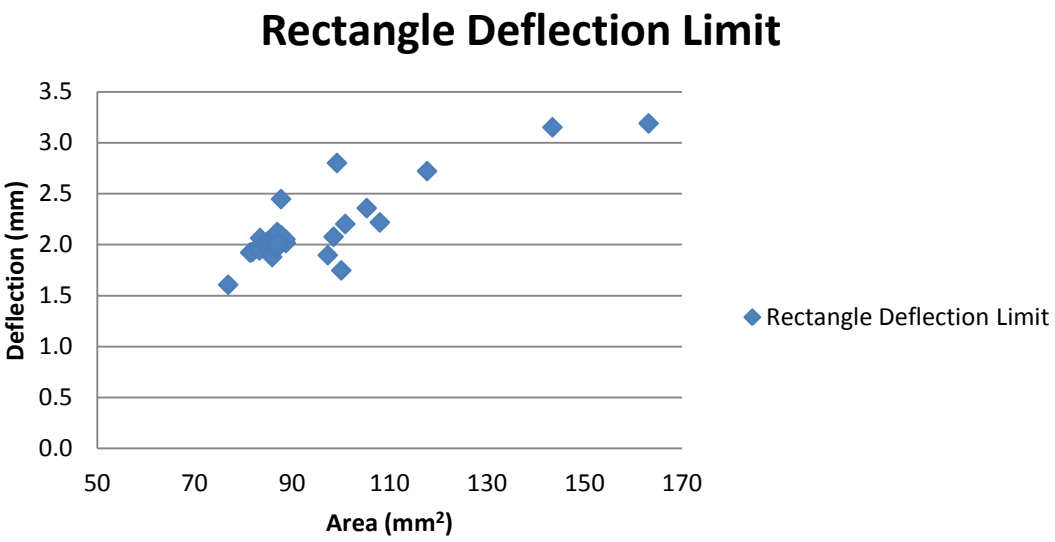


Figure 74. Rectangle Deflection Limit

The force limit was then optimized for the rectangle, as seen in Fig. 75. The limits were again the same and the force was set to equal to 2 mN. Examining the results, we can see a finger with an area of 90 mm^2 is capable of exerting 2 mN of force. As expected, larger fingers are capable of exerting higher forces, but we are interested in exerting a certain force while minimizing the area, which modeFRONTIER accomplished.

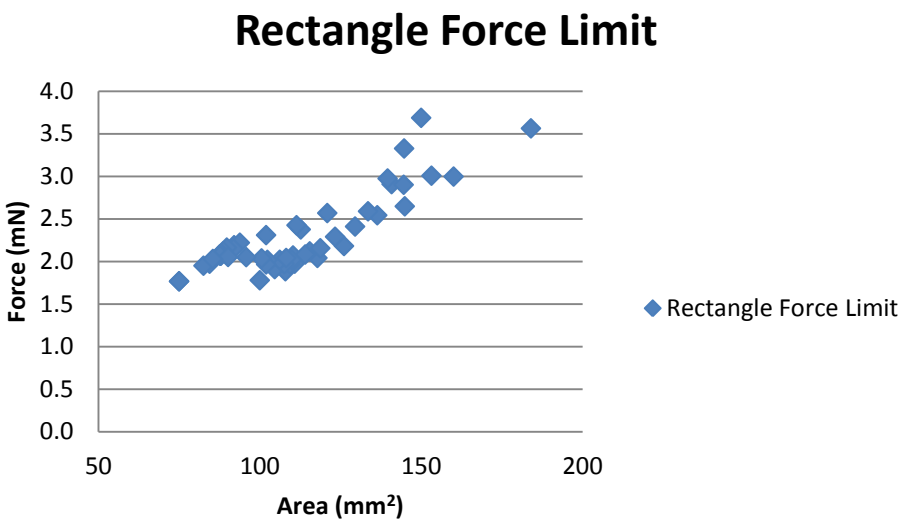
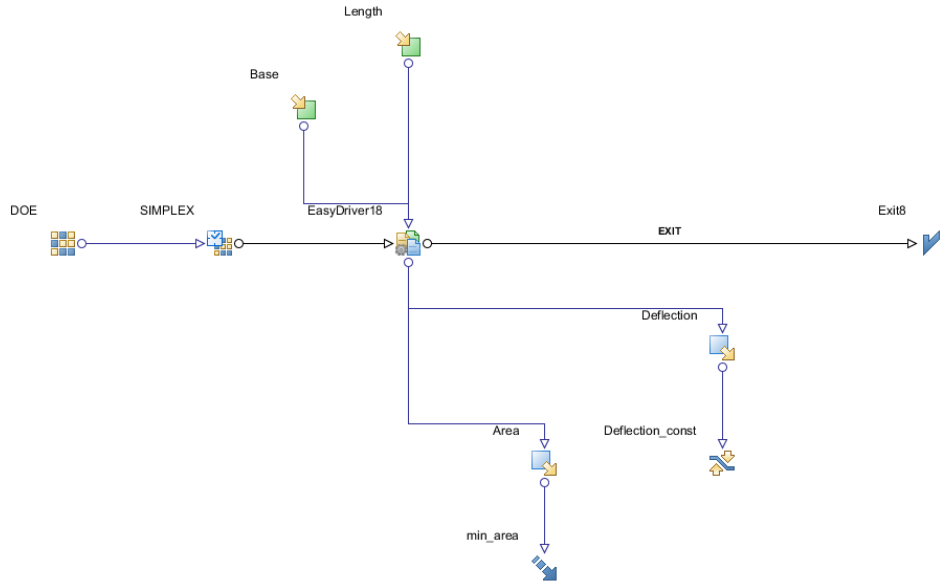


Figure 75. Rectangle Force Limit

4.3.6 Iso Triangle Deflection and Force Limits

The same process used to determine the force or deflection limits for the rectangle was used to optimize for the isosceles triangle deflection or force limits. In this case, the base was limited to 7 to 14 mm and the length was limited to 15 to 20 mm. The deflection was set to "Equal to" 2 mm again. The modeFRONTIER setup can be seen in Fig. 76. The limits can be seen in the bottom and the minimize area node can be seen in the figure. This setup will also be used for the force limit optimization study. The only difference is the model will optimize for force instead of deflection. The limits will be the same and the force will be set to equal to 2 mN.



out Variable

Name	Variable Type	Value	Expression	Distribution	Scale	Lower Bound	Upper Bound	Central Value	Delta Value	Base	
Length	Variable	0.0		None	0	0.015	0.02	0.0175	0.002500000000...	0	0.0
Base	Variable	0.0		None	0	0.0070	0.014	0.0105	0.0035	0	0.0

Figure 76. Iso Triangle Deflection Limit

As can be seen in Fig. 77, the deflection varies when changing the area. The important thing to notice is a finger with an area of 70 mm^2 is capable of deflecting up to 2 mm, while fingers with a larger area are not able to deflect as much. This is due to the shape of the finger. So, if a finger with a large deflection is desired, a smaller finger can be used to obtain this deflection.

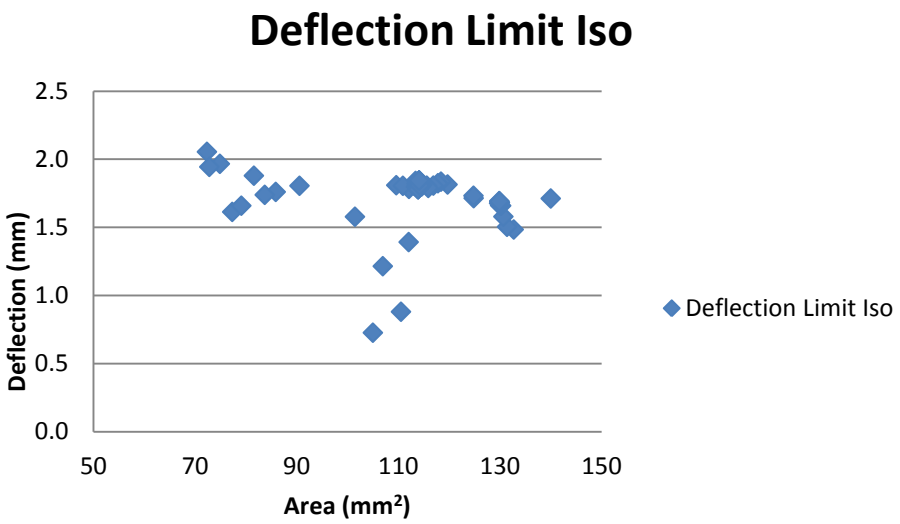


Figure 77. Iso Deflection Limit

Next, the force was optimized while minimizing the area, as seen in Fig. 78. A finger with an area of 90 mm^2 is capable of 2 mN of force. Also, fingers with a higher area are capable of the same force, but wastes material. A finger with a smaller area is achieved with the same force production.

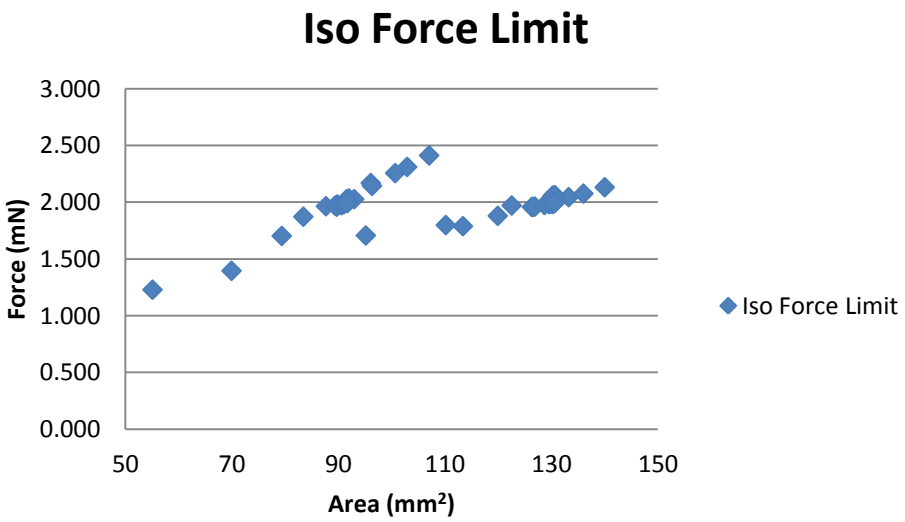
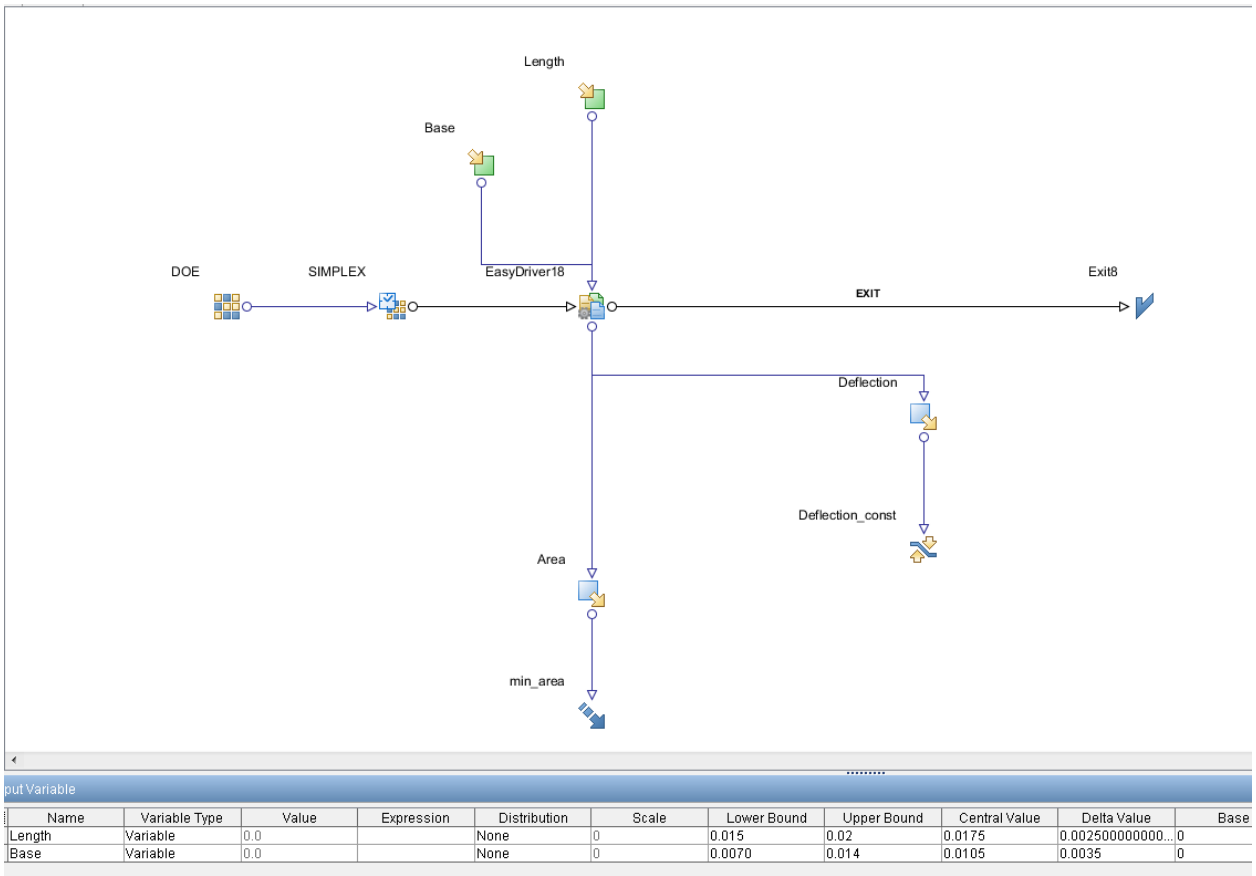


Figure 78. Iso Triangle Force Limit

4.3.7 Right Triangle Deflection and Force Limits

The same process was used to optimize for the deflection or force limits of a right triangle, using the process stated in Section 4.3.5. In this case, the base was limited to 7 to 14 mm and the length was limited to 15 to 20 mm. The deflection was set to "Equal to" 2 mm again. The modeFRONTIER setup can be seen in Fig. 79. The limits can be seen in the bottom and the minimize area node can be seen in the figure. This setup will also be used for the force limit optimization study. The only difference is the model will optimize for force instead of deflection. The limits will be the same and the force will be set to equal to 2 mN.



put Variable

Name	Variable Type	Value	Expression	Distribution	Scale	Lower Bound	Upper Bound	Central Value	Delta Value	Base
Length	Variable	0.0		None	0	0.015	0.02	0.0175	0.002500000000...	0
Base	Variable	0.0		None	0	0.0070	0.014	0.0105	0.0035	0

Figure 79. Right Triangle Deflection Limit

As can be seen in Fig. 80, the deflection once again varies when changing the area. The important thing to notice is a finger with an area of 100 mm^2 is capable of deflecting up to 2 mm, while fingers with a larger area are not able to deflect as much. This is due to the shape of the finger. So, if a finger with a large deflection is desired, a smaller finger can be used to obtain this deflection.

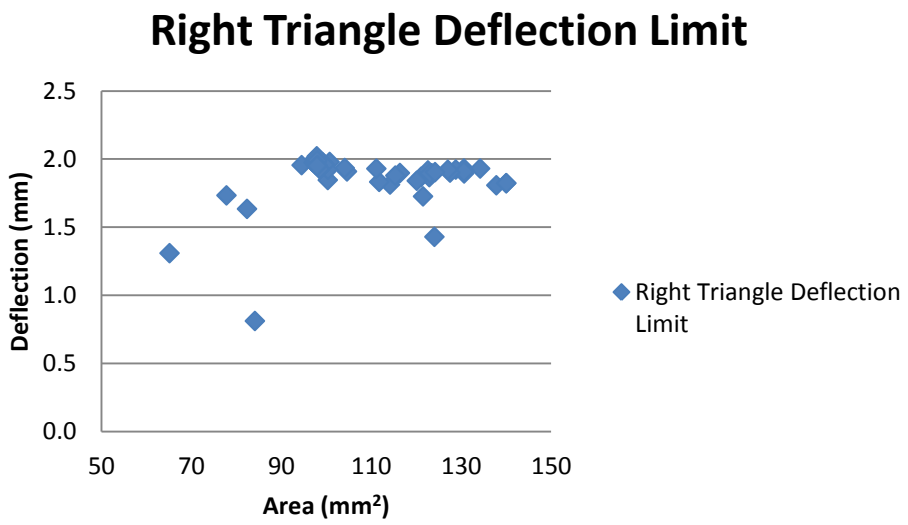


Figure 80. Right Triangle Deflection Limit

Next, the force was optimized while minimizing the area, as seen in Fig. 81. A finger with an area of 110 mm^2 is capable of 2 mN of force. Also, fingers with a higher area are capable of the same force, but wastes material. A finger with a smaller area is achieved with the same force production.

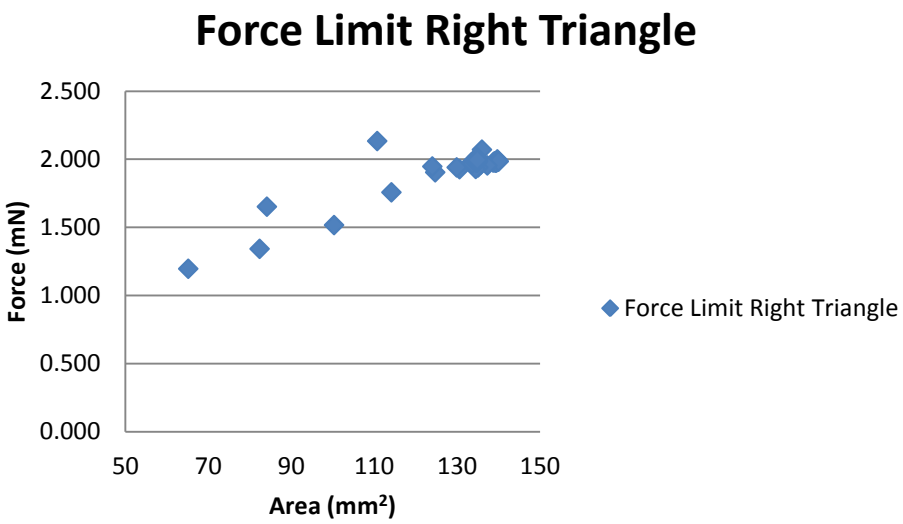


Figure 81. Right Triangle Force Limit

Chapter 5. CONCLUSION

5.1 Results of the Study

An IPMC Force Model was coupled with modeFRONTIER in order to optimize fingers with respect to deflection, force exerted, both force and deflection, and area minimization, which minimizes IPMC cost. The force model was combined into one large model that is able to run many small sub-models that were made to model IPMC fingers. This model is able to model the force and deflection of any finger using one code without going through the entire process of modeling a finger and using Matlab and switching back to modeling. This optimization process is needed as early investigations into size and shape of IPMC fingers led to interesting results, as different shapes were better suited for certain design goals. The optimization process was capable of modeling many different fingers and predicting the best design depending on the desired need, such as force or deflection. It was also capable of minimizing the area of fingers, while still achieving design goals, such as force or deflection. This is important as the waste of material can be minimized and the best design can be achieved. Many different optimization processes were run to change the area, while optimizing for the force or deflection. As seen in Chapter 4, we are able to achieve a desired force or deflection, such as 2 mm or 2 mN, while minimizing area. It can be seen that these different shapes each have advantages for certain design goals, whether a high force or a high deflection is desired. The best way to achieve these design goals without simply cutting out an arbitrary IPMC finger and testing it is to use an optimization package. This optimization package should be able to predict the best way to design a finger, which is what this thesis accomplished.

5.2 Limitations

Although modeFRONTIER is able to optimize these fingers for the best design, there are some limitations to this study. Limits must be set on the width and length of the fingers being modeled, as modeFRONTIER would never converge on an answer without these limits. As seen, a finger with a thin base and a long length is capable of a high degree of deflection. This deflection will always increase when making the finger longer, but there must be a limit to how long the finger can get. As seen with the triangle fingers, the limits have a greater effect than they do with the rectangle, especially when looking at the ratio of the width to the length. Also, a finger with a very wide base and very short length is capable of exerting a high force, but this design is impractical, as the finger will not be able to actuate a great amount. The main objective of IPMC fingers is usually to grasp an object. ModeFRONTIER can make the finger as short as possible and will achieve this high force, but this design is impractical as the finger will not be able to deflect enough to grasp an object. This is also a reason the limits are set in modeFRONTIER.

Meshing is also a concern in this study. The mesh is not changed in the models when modeFRONTIER is predicting the best design. It was not a problem when running the models presented in this thesis, but when making fingers very small or using complex shapes, meshing may become a concern when using modeFRONTIER.

Back relaxation was also not included in these studies, as we were mainly concerned with an instantaneous deflection and force exerted. These studies did not include a transient study that included a back relaxation term. This was not necessary as we wished to optimize for force or deflection for a given finger. Force and deflection are usually instantaneous measurements and back relaxation does not have an effect when these studies are short.

5.3 Future Research

Using modeFRONTIER and the Comsol model, the effects of miniaturization can be studied. This can be done very easily in modeFRONTIER, as the length and base are changed according to the area of the finger. The limits of the base and length may also be changed to determine other shapes. The length limit can be made much larger to see how thin and long a finger may get and how much it can deflect. More complex finger shapes can be created using modeFRONTIER and Comsol. This was the first time modeFRONTIER was used to optimize fingers, so the shapes were kept simple. Transient studies, including back relaxation studies, may also be included in the new Comsol models if longer grasping times are desired.

APPENDICES

Appendix A

A1. Concentration M-File

```
 %This M-File models the Nafion layer in an IPMC finger.  Once a voltage is
%applied, the concentration of cations will be modeled.  This model will
%then be given voltages at every point in the finger and the concentration
%at any point will be determined.

function out = model
%
% concentration.m
%
% Model exported on Jun 27 2012, 14:50 by COMSOL 4.3.0.151.

import com.comsol.model.*
import com.comsol.model.util.*

model = ModelUtil.create('Model');

model.modelPath('C:\Users\Justin\Desktop\Comsol Models');

model.name('concentrationdistribution.mph');

model.modelNode.create('mod1');

model.geom.create('geom1', 1);
model.geom('geom1').feature.create('i1', 'Interval');
model.geom('geom1').feature('i1').set('p2', '180e-6');
model.geom('geom1').run;

model.variable.create('var1');
model.variable('var1').set('Van', '2');
model.variable('var1').set('Vcat', '0');

model.physics.create('chds', 'DilutedSpecies', 'geom1');

model.mesh.create('mesh1', 'geom1');
model.mesh('mesh1').feature.create('edg1', 'Edge');

model.variable('var1').name('Variables 1a');

model.view('view1').axis.set('xmin', '-9.000000318337698E-6');
model.view('view1').axis.set('xmax', '1.8899999849963933E-4');

model.physics('chds').prop('EquationForm').set('form', 'Transient');
model.physics('chds').prop('Migration').set('Migration', '1');
model.physics('chds').prop('Convection').set('Convection', '0');
```

```
model.physics('chds').feature('cdm1').set('V', 'Vcat+((Vcat-Van)/180e-
6)*x[V/m]');
model.physics('chds').feature('cdm1').set('D_0', {'6e-12[m^2/s]' '0' '0'
'0' '6e-12[m^2/s]' '0' '0' '0' '6e-12[m^2/s]'});
model.physics('chds').feature('cdm1').set('z', '1');
model.physics('chds').feature('cdm1').set('um', '2.4630522e-15[s*mol/kg]');
model.physics('chds').feature('init1').set('c', '1250');

model.mesh('mesh1').feature('size').set('hmax', '1e-6');
model.mesh('mesh1').run;

model.frame('material1').sorder(1);

model.study.create('std1');
model.study('std1').feature.create('time', 'Transient');

model.sol.create('sol1');
model.sol('sol1').study('std1');
model.sol('sol1').attach('std1');
model.sol('sol1').feature.create('st1', 'StudyStep');
model.sol('sol1').feature.create('v1', 'Variables');
model.sol('sol1').feature.create('t1', 'Time');
model.sol('sol1').feature('t1').feature.create('fc1', 'FullyCoupled');
model.sol('sol1').feature('t1').feature.create('d1', 'Direct');
model.sol('sol1').feature('t1').feature.remove('fcDef');

model.result.create('pg1', 'PlotGroup1D');
model.result('pg1').set('probetag', 'none');
model.result('pg1').feature.create('lngr1', 'LineGraph');
model.result('pg1').feature('lngr1').selection.all;
model.result('pg1').feature('lngr1').selection.all;

model.study('std1').feature('time').set('tlist', 'range(0,0.1,5)');

model.sol('sol1').attach('std1');
model.sol('sol1').feature('st1').name('Compile Equations: Time Dependent');
model.sol('sol1').feature('st1').set('studystep', 'time');
model.sol('sol1').feature('v1').set('control', 'time');
model.sol('sol1').feature('t1').set('control', 'time');
model.sol('sol1').feature('t1').set('tlist', 'range(0,1,5)');
model.sol('sol1').feature('t1').set('maxorder', '2');
model.sol('sol1').feature('t1').feature('fc1').set('maxiter', '5');
model.sol('sol1').feature('t1').feature('fc1').set('jtech', 'once');
model.sol('sol1').feature('t1').feature('d1').set('linsolver', 'pardiso');
model.sol('sol1').runAll;

model.result('pg1').name('Concentration (chds)');
model.result('pg1').set('looplevelinput', {'manual'});
model.result('pg1').set('showlooplevel', {'on' 'off' 'off'});
model.result('pg1').set('looplevel', {'1,31,51'});
model.result('pg1').set('xlabel', 'x-coordinate (m)');
model.result('pg1').set('ylabel', 'Concentration (mol/m<sup>3</sup>)');
model.result('pg1').set('xlabelactive', false);
model.result('pg1').set('ylabelactive', false);
model.result('pg1').feature('lngr1').set('xdata', 'expr');
```

```
model.result('pg1').feature('lngr1').set('xdataexpr', 'x');
model.result('pg1').feature('lngr1').set('xdataunit', 'm');
model.result('pg1').feature('lngr1').set('xdatadescr', 'x-coordinate');
model.result('pg1').feature('lngr1').selection.all;
model.result('pg1').feature('lngr1').selection.all;

out = model;
```

A2. 7x17 mm Rectangle Fem

```
%This m-file contains the fem structure that is used to model the finger in
%three 7x17 mm blocks.  This model contains the voltage distribution that
%will be used to determine the concentration throughout the IPMC.

function out = model
%
% sevenbyseventeen.m
%
% Model exported on Apr 17 2013, 07:45 by COMSOL 4.3.0.151.

import com.comsol.model.*
import com.comsol.model.util.*

model = ModelUtil.create('Model');

model.modelPath('C:\Users\Justin\Desktop\Comsol Models');

model.modelNode.create('mod1');

model.geom.create('geom1', 3);

model.mesh.create('mesh1', 'geom1');

model.physics.create('ec', 'ConductiveMedia', 'geom1');

model.study.create('std1');
model.study('std1').feature.create('stat', 'Stationary');
model.study('std1').feature('stat').set('sweeptype', 'sparse');
model.study('std1').feature('stat').set('plistarr_vector_start', {});
model.study('std1').feature('stat').set('usesol', 'off');
model.study('std1').feature('stat').set('constraintgroup', {});
model.study('std1').feature('stat').set('plot', 'off');
model.study('std1').feature('stat').set('adaption', 'off');
model.study('std1').feature('stat').set('notstudy', 'zero');
model.study('std1').feature('stat').set('plistarr', {});
model.study('std1').feature('stat').set('notsolnum', '1');
model.study('std1').feature('stat').set('plistarr_vector_numvalues', {});
model.study('std1').feature('stat').set('plist', '');
model.study('std1').feature('stat').set('nottimeinterp', 'off');
model.study('std1').feature('stat').set('useloadcase', 'off');
model.study('std1').feature('stat').set('loadgroup', {});
model.study('std1').feature('stat').set('useparam', 'off');
model.study('std1').feature('stat').set('plistarr_vector_step', {});
model.study('std1').feature('stat').set('plistarr_vector_function', {});
model.study('std1').feature('stat').set('notsolmethod', 'init');
model.study('std1').feature('stat').set('plistarr_vector_method', {});
model.study('std1').feature('stat').set('geometricNonlinearity', false);
model.study('std1').feature('stat').set('nott', '0');
model.study('std1').feature('stat').set('loadgroupweight', {});
model.study('std1').feature('stat').set('probesel', 'all');
model.study('std1').feature('stat').set('notsolvertype', 'none');
model.study('std1').feature('stat').set('loadcase', {});
```

```
model.study('std1').feature('stat').set('geometricNonlinearityActive', true);
model.study('std1').feature('stat').set('plistarr_vector_stop', {});
model.study('std1').feature('stat').set('pname', {});
model.study('std1').feature('stat').set('showGeometricNonlinearity', 'on');
model.study('std1').feature('stat').activate('ec', true);

% IPMC finger is being built
model.geom('geom1').run;
model.geom('geom1').feature.create('blk1', 'Block');
model.geom('geom1').feature('blk1').setIndex('size', '7e-3', 0);
model.geom('geom1').feature('blk1').setIndex('size', '17e-3', 1);
model.geom('geom1').feature('blk1').setIndex('size', '10e-6', 2);
model.geom('geom1').run('blk1');
model.geom('geom1').run('blk1');
model.geom('geom1').feature.create('blk2', 'Block');
model.geom('geom1').feature('blk2').setIndex('size', '7e-3', 0);
model.geom('geom1').feature('blk2').setIndex('size', '17e-3', 1);
model.geom('geom1').feature('blk2').setIndex('size', '180e-6', 2);
model.geom('geom1').feature('blk2').setIndex('pos', '10e-6', 2);
model.geom('geom1').run('blk2');
model.geom('geom1').run('blk2');
model.geom('geom1').feature.create('blk3', 'Block');
model.geom('geom1').feature('blk3').setIndex('size', '7e-3', 0);
model.geom('geom1').feature('blk3').setIndex('size', '17e-3', 1);
model.geom('geom1').feature('blk3').setIndex('size', '10e-6', 2);
model.geom('geom1').feature('blk3').setIndex('pos', '190e-6', 2);
model.geom('geom1').run('blk3');
model.geom('geom1').runAll;
model.geom('geom1').run;

%Materials are being assigned to each domain
model.material.create('mat1');
model.material('mat1').name('Pt');
model.material('mat1').set('family', 'custom');
model.material('mat1').set('lighting', 'cooktorrance');
model.material('mat1').set('specular', 'custom');
model.material('mat1').set('customspecular', [0.7843137254901961 1 1]);
model.material('mat1').set('fresnel', 0.9);
model.material('mat1').set('roughness', 0.1);
model.material('mat1').set('shininess', 200);
model.material('mat1').propertyGroup('def').set('electricconductivity',
'8.9e6[S/m]');
model.material('mat1').propertyGroup('def').set('thermalexpansioncoefficient'
, '8.80e-6[1/K]');
model.material('mat1').propertyGroup('def').set('heatcapacity',
'133[J/(kg*K)]');
model.material('mat1').propertyGroup('def').set('density', '21450[kg/m^3]');
model.material('mat1').propertyGroup('def').set('thermalconductivity',
'71.6[W/(m*K)]');
model.material('mat1').propertyGroup.create('Enu', 'Young''s modulus and
Poisson''s ratio');
model.material('mat1').propertyGroup('Enu').set('poissonsratio', '0.38');
model.material('mat1').propertyGroup('Enu').set('youngsmodulus',
'168e9[Pa]');
model.material('mat1').set('family', 'custom');
model.material('mat1').set('lighting', 'cooktorrance');
```

```
model.material('mat1').set('specular', 'custom');
model.material('mat1').set('customspecular', [0.7843137254901961 1 1]);
model.material('mat1').set('fresnel', 0.9);
model.material('mat1').set('roughness', 0.1);
model.material('mat1').set('shininess', 200);
model.material.create('mat2');
model.material('mat2').name('Nylon');
model.material('mat2').set('family', 'custom');
model.material('mat2').set('lighting', 'phong');
model.material('mat2').set('fresnel', 0.5);
model.material('mat2').set('roughness', 0.1);
model.material('mat2').set('specular', 'custom');
model.material('mat2').set('customspecular', [0.7843137254901961
0.7843137254901961 0.7843137254901961]);
model.material('mat2').set('diffuse', 'custom');
model.material('mat2').set('customdiffuse', [0.39215686274509803
0.39215686274509803 0.9803921568627451]);
model.material('mat2').set('ambient', 'custom');
model.material('mat2').set('customambient', [0.39215686274509803
0.39215686274509803 0.7843137254901961]);
model.material('mat2').set('shininess', 500);
model.material('mat2').propertyGroup('def').set('heatcapacity',
'1700[J/(kg*K)]');
model.material('mat2').propertyGroup('def').set('relpermittivity', '4');
model.material('mat2').propertyGroup('def').set('thermalexpansioncoefficient'
, '280e-6[1/K]');
model.material('mat2').propertyGroup('def').set('density', '1150[kg/m^3]');
model.material('mat2').propertyGroup('def').set('thermalconductivity',
'0.26[W/(m*K)]');
model.material('mat2').propertyGroup.create('Enu', 'Young''s modulus and
Poisson''s ratio');
model.material('mat2').propertyGroup('Enu').set('poissonsratio', '0.4');
model.material('mat2').propertyGroup('Enu').set('youngsmodulus', '2e9[Pa]');
model.material('mat2').set('family', 'custom');
model.material('mat2').set('lighting', 'phong');
model.material('mat2').set('fresnel', 0.5);
model.material('mat2').set('roughness', 0.1);
model.material('mat2').set('specular', 'custom');
model.material('mat2').set('customspecular', [0.7843137254901961
0.7843137254901961 0.7843137254901961]);
model.material('mat2').set('diffuse', 'custom');
model.material('mat2').set('customdiffuse', [0.39215686274509803
0.39215686274509803 0.9803921568627451]);
model.material('mat2').set('ambient', 'custom');
model.material('mat2').set('customambient', [0.39215686274509803
0.39215686274509803 0.7843137254901961]);
model.material('mat2').set('shininess', 500);
model.material('mat1').selection.set([1 3]);
model.material('mat2').selection.set([2]);
model.material('mat1').propertyGroup('def').set('electricconductivity',
{'1e6[S/m]'});
model.material('mat1').propertyGroup('def').set('relpermittivity',
{'1.000265'});
model.material('mat2').propertyGroup('def').set('electricconductivity',
{'10'});

%Voltage and ground being assigned.  The voltage can be changed to any
```

```
%value in the electric potential line.
model.physics('ec').feature.create('gnd1', 'Ground', 2);
model.physics('ec').feature('gnd1').selection.set([2]);
model.physics('ec').feature.create('pot1', 'ElectricPotential', 2);
model.physics('ec').feature('pot1').selection.set([8]);
model.physics('ec').feature('pot1').set('V0', 1, '2');

model.mesh('mesh1').autoMeshSize(6);
model.mesh('mesh1').run;
model.mesh('mesh1').autoMeshSize(5);
model.mesh('mesh1').run;
model.mesh('mesh1').autoMeshSize(7);
model.mesh('mesh1').run;
model.mesh('mesh1').autoMeshSize(5);
model.mesh('mesh1').run;
model.mesh('mesh1').autoMeshSize(6);
model.mesh('mesh1').run;

model.sol.create('sol1');
model.sol('sol1').study('std1');
model.sol('sol1').feature.create('st1', 'StudyStep');
model.sol('sol1').feature('st1').set('study', 'std1');
model.sol('sol1').feature('st1').set('studystep', 'stat');
model.sol('sol1').feature.create('v1', 'Variables');
model.sol('sol1').feature('v1').set('control', 'stat');
model.sol('sol1').feature.create('s1', 'Stationary');
model.sol('sol1').feature('s1').feature.create('fc1', 'FullyCoupled');
model.sol('sol1').feature('s1').feature.create('i1', 'Iterative');
model.sol('sol1').feature('s1').feature('i1').set('prefuntype', 'left');
model.sol('sol1').feature('s1').feature('i1').set('maxlinit', 10000);
model.sol('sol1').feature('s1').feature('i1').set('linsolver', 'cg');
model.sol('sol1').feature('s1').feature('i1').set('rhob', 400);
model.sol('sol1').feature('s1').feature('fc1').set('linsolver', 'i1');
model.sol('sol1').feature('s1').feature('i1').feature.create('mg1',
'Multigrid');
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('prefun',
'amg');
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('iter', 2);
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('mgcycle',
'v');
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('mcasegen',
'any');
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('gmglevels',
1);
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('scale', 2);
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('massem',
true);
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('mkeep',
false);
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('rmethod',
'longest');
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('mglevels',
5);
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('maxcoarsedo
f', 5000);
```

```
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('amgauto',
3);
model.sol('sol1').feature('s1').feature.remove('fcDef');
model.sol('sol1').attach('std1');

% Plot of the voltage distribution.
model.result.create('pg1', 'PlotGroup3D');
model.result('pg1').name('Electric potential');
model.result('pg1').set('data', 'dset1');
model.result('pg1').set('solrepresentation', 'solnum');
model.result('pg1').set('oldanalysistype', 'noneavailable');
model.result('pg1').set('data', 'dset1');
model.result('pg1').feature.create('mslc1', 'Multislice');
model.result('pg1').feature('mslc1').name('Multislice');
model.result('pg1').feature('mslc1').set('data', 'parent');
model.result('pg1').feature('mslc1').set('solrepresentation', 'solnum');
model.result('pg1').feature('mslc1').set('expr', 'V');
model.result('pg1').feature('mslc1').set('unit', 'V');
model.result('pg1').feature('mslc1').set('descr', 'V');
model.result('pg1').feature('mslc1').set('inheritplot', 'none');
model.result('pg1').feature('mslc1').set('data', 'parent');
model.result('pg1').feature('mslc1').set('expr', 'V');
model.result('pg1').feature('mslc1').set('unit', 'V');
model.result('pg1').feature('mslc1').set('inheritplot', 'none');
model.result('pg1').feature('mslc1').set('data', 'parent');
model.result('pg1').feature('mslc1').set('expr', 'V');
model.result('pg1').feature('mslc1').set('unit', 'V');
model.result('pg1').feature('mslc1').set('inheritplot', 'none');
model.result('pg1').feature('mslc1').set('data', 'parent');
model.result('pg1').feature('mslc1').set('expr', 'V');
model.result('pg1').feature('mslc1').set('unit', 'V');
model.result('pg1').feature('mslc1').set('inheritplot', 'none');
model.result('pg1').feature('mslc1').set('data', 'parent');
model.result('pg1').feature('mslc1').set('expr', 'V');
model.result('pg1').feature('mslc1').set('unit', 'V');
model.result('pg1').feature('mslc1').set('inheritplot', 'none');
model.result('pg1').feature('mslc1').set('data', 'parent');

model.sol('sol1').runAll;

model.result('pg1').run;
model.result('pg1').feature.create('vol1', 'Volume');
model.result('pg1').run;

model.name('7x17voltage.mph');

model.result('pg1').run;

model.mesh('mesh1').automatic(false);
model.mesh('mesh1').feature('size').set('hauto', '2');
model.mesh('mesh1').run('size');
model.mesh('mesh1').feature('size').set('hauto', '4');
model.mesh('mesh1').run;

model.sol('sol1').study('std1');
```

```
model.sol('sol1').feature.remove('s1');
model.sol('sol1').feature.remove('v1');
model.sol('sol1').feature.remove('st1');
model.sol('sol1').feature.create('st1', 'StudyStep');
model.sol('sol1').feature('st1').set('study', 'std1');
model.sol('sol1').feature('st1').set('studystep', 'stat');
model.sol('sol1').feature.create('v1', 'Variables');
model.sol('sol1').feature('v1').set('control', 'stat');
model.sol('sol1').feature.create('s1', 'Stationary');
model.sol('sol1').feature('s1').feature.create('fc1', 'FullyCoupled');
model.sol('sol1').feature('s1').feature.create('i1', 'Iterative');
model.sol('sol1').feature('s1').feature('i1').set('prefuntype', 'left');
model.sol('sol1').feature('s1').feature('i1').set('maxlinit', 10000);
model.sol('sol1').feature('s1').feature('i1').set('linsolver', 'cg');
model.sol('sol1').feature('s1').feature('i1').set('rhob', 400);
model.sol('sol1').feature('s1').feature('fc1').set('linsolver', 'i1');
model.sol('sol1').feature('s1').feature('i1').feature.create('mg1',
'Multigrid');
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('prefun',
'amg');
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('iter', 2);
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('mgcycle',
'v');
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('mcasegen',
'any');
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('gmglevels',
1);
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('scale', 2);
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('massem',
true);
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('mkeep',
false);
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('rmethod',
'longest');
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('mglevels',
5);
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('maxcoarsedo
f', 5000);
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('amgauto',
3);
model.sol('sol1').feature('s1').feature.remove('fcDef');
model.sol('sol1').attach('std1');
model.sol('sol1').runAll;

model.result('pg1').run;
model.result.export.create('tbl1', 'Table');
model.result.export.remove('tbl1');
model.result.export.create('data1', 'Data');
model.result.export.remove('data1');
model.result.numerical.create('gev1', 'EvalGlobal');
model.result.numerical('gev1').set('expr', 'ec.zref');
model.result.numerical('gev1').set('descr', 'Reference impedance');
model.result.numerical.remove('gev1');
model.result.numerical.create('int1', 'IntVolume');
model.result.numerical('int1').selection.all;
model.result.numerical('int1').selection.set([1 3]);
model.result.numerical('int1').set('expr', 'ec.Ey');
```

```
model.result.numerical('int1').set('descr', 'Electric field, y component');
model.result.table.create('tbl1', 'Table');
model.result.table('tbl1').comments('Volume Integration 1 (ec.Ey)');
model.result.numerical('int1').set('table', 'tbl1');
model.result.numerical('int1').setResult;
model.result.numerical.create('gev1', 'EvalGlobal');
model.result.numerical.create('av1', 'AvSurface');
model.result.numerical('av1').selection.all;
model.result.table.create('tbl2', 'Table');
model.result.table('tbl2').comments('Surface Average 1 (V)');
model.result.numerical('av1').set('table', 'tbl2');
model.result.numerical('av1').setResult;

out = model;
```

A3. Extract Voltage

```
function [Voltage_anode,Voltage_cathode] = Extract_voltage_sample(model)



IPMC_width = 20;
IPMC_length = 20;

%Create an empty square matrix at z=0 (the anode) that contains the points
%in the xy plane where the voltage will be sampled.

[x,y] = meshgrid(0:IPMC_width, 0:IPMC_length);
pz = zeros(1,(IPMC_width+1)*(IPMC_length+1));
p = [x(:)'; y(:)'; pz]*(1e-3);
p_cat = [x(:)'; y(:)'; pz+(190e-3)]*(1e-3);

%Extract voltage information from electrical model

V_cathode = mphinterp(model,'V','coord',p);
V_anode = mphinterp(model,'V','coord',p_cat);

Voltage_anode = [x(:) y(:) V_anode'];
Voltage_cathode = [x(:) y(:) V_cathode'];

Voltage_anode(isnan(Voltage_anode)) = 0;
Voltage_cathode(isnan(Voltage_cathode)) = 0;

end
```

A4. Force and Concentration M-File

```
function [Force,Concentration] =
Echem1Dfinal(Voltage_anode,Voltage_cathode,tf)
%
% concentration.m
%
% Model exported on Jun 27 2012, 14:50 by COMSOL 4.3.0.151.

import com.comsol.model.*
import com.comsol.model.util.*

model = ModelUtil.create('Model');

model.modelPath('C:\Users\Justin\Desktop\Comsol Models');

model.name('concentrationdistributionclean.mph');

model.modelNode.create('mod1');

% length of voltage vector
[n,~] = size(Voltage_anode);

% vector of z sample points
z = ((0:180)'*1e-6);
Concentration = [];


model.geom.create('geom1', 1);
model.geom('geom1').feature.create('i1', 'Interval');


model.variable.create('var1');

model.physics.create('chds', 'DilutedSpecies', 'geom1');

model.mesh.create('mesh1', 'geom1');
model.mesh('mesh1').feature.create('edg1', 'Edge');


model.study.create('std1');
model.study('std1').feature.create('time', 'Transient');

model.sol.create('sol1');
model.sol('sol1').feature.create('st1', 'StudyStep');
model.sol('sol1').feature.create('v1', 'Variables');
model.sol('sol1').feature.create('t1', 'Time');
model.sol('sol1').feature('t1').feature.create('fc1', 'FullyCoupled');
model.sol('sol1').feature('t1').feature.create('d1', 'Direct');
model.sol('sol1').feature('t1').feature.remove('fcDef');
model.result.create('pg1', 'PlotGroup1D');
model.result('pg1').feature.create('lngr1', 'LineGraph');
```

```
for i = 1:n

% extract anode and cathode voltage for element n.
V_a = Voltage_anode(i,3);
V_c = Voltage_cathode(i,3);

% vector of x and y sample points
x = Voltage_anode(i,1)'*ones(181,1);
y = Voltage_anode(i,2)'*ones(181,1);



model.geom('geom1').feature('i1').set('p2', '180e-6');
model.geom('geom1').run;


model.variable('var1').set('Van',strcat(num2str(V_a)),'[V]'), ...);
model.variable('var1').set('Vcat',strcat(num2str(V_c)),'[V]');




% model.variable('var1').name('Variables 1a');

model.view('view1').axis.set('xmin', '-9.000000318337698E-6');
model.view('view1').axis.set('xmax', '1.8899999849963933E-4');

model.physics('chds').prop('EquationForm').set('form', 'Transient');
model.physics('chds').prop('Migration').set('Migration', '1');
model.physics('chds').prop('Convection').set('Convection', '0');
model.physics('chds').feature('cdm1').set('V', 'Vcat+((Vcat-Van)/180e-
6)*x[V/m]');
model.physics('chds').feature('cdm1').set('D_0', {'6e-12[m^2/s]'; '0'; '0';
'0'; '6e-12[m^2/s]'; '0'; '0'; '0'; '6e-12[m^2/s]'});
model.physics('chds').feature('cdm1').set('z', '1');
model.physics('chds').feature('cdm1').set('um', '2.4630522e-15[s*mol/kg]');
model.physics('chds').feature('init1').set('c', '1250');

model.mesh('mesh1').feature('size').set('hmax', '1e-6');
model.mesh('mesh1').run;

model.frame('material1').sorder(1);


model.sol('sol1').study('std1');
model.sol('sol1').attach('std1');



model.result('pg1').set('probetag', 'none');
```

```
model.study('std1').feature('time').set('tlist', 'range(0,1,5)');

model.result('pg1').feature('lngr1').selection.all;
model.result('pg1').feature('lngr1').selection.all;

model.sol('sol1').attach('std1');
model.sol('sol1').feature('st1').name('Compile Equations: Time Dependent');
model.sol('sol1').feature('st1').set('studystep', 'time');
model.sol('sol1').feature('v1').set('control', 'time');
model.sol('sol1').feature('t1').set('control', 'time');
model.sol('sol1').feature('t1').set('tlist', 'range(0,1,5)');
model.sol('sol1').feature('t1').set('maxorder', '2');
model.sol('sol1').feature('t1').feature('fc1').set('maxiter', '5');
model.sol('sol1').feature('t1').feature('fc1').set('jtech', 'once');
model.sol('sol1').feature('t1').feature('d1').set('linsolver', 'pardiso');
model.sol('sol1').runAll;

model.result('pg1').name('Concentration (chds)');
model.result('pg1').set('looplevelinput', {'manual'});
model.result('pg1').set('showlooplevel', {'on' 'off' 'off'});
model.result('pg1').set('xlabel', 'x-coordinate (m)');
model.result('pg1').set('ylabel', 'Concentration (mol/m<sup>3</sup>)');
model.result('pg1').set('xlabelactive', false);
model.result('pg1').set('ylabelactive', false);
model.result('pg1').feature('lngr1').set('xdata', 'expr');
model.result('pg1').feature('lngr1').set('xdataexpr', 'x');
model.result('pg1').feature('lngr1').set('xdataunit', 'm');
model.result('pg1').feature('lngr1').set('xdatadescr', 'x-coordinate');
model.result('pg1').feature('lngr1').selection.all;
model.result('pg1').feature('lngr1').selection.all;

% Save current fem structure for restart purposes
model0=model;

% extract concentration values at points in vector z
con = mphinterp(model0,'c','coord',z','T',tf)';

% Concentration = [x y z c]

Concentration = [Concentration; x y z con];

end
% Force equation

Force = [Concentration(:,1:3) (9e2/1.2)*((Concentration(:,4)-1250).^2)];
name = input('enter a file name or number = ','s');%prompt user for input
fname = strcat(name,'.txt');

% fname = 'temp.csv' ;
fid = fopen(fname,'w');
fprintf(fid,'%%x\ty\tz\tForce\n');
fclose(fid);
```

```
dlmwrite(fname,Force,'precision','%2.6f','delimiter','\t','newline','pc','-
append');



end
```

A5. Complete Code used in modeFRONTIER

```
function out = model
%
% sevenbyseventeenvoltage.m
%
% Model exported on Jul 6 2012, 15:12 by COMSOL 4.3.0.151.

import com.comsol.model.*
import com.comsol.model.util.*

model = ModelUtil.create('Model');

model.modelPath('C:\Users\Justin\Desktop\Comsol Models');

model.name('7x17voltagedistribution.mph');

model.modelNode.create('mod1');

model.geom.create('geom1', 3);
model.geom('geom1').feature.create('blk1', 'Block');
model.geom('geom1').feature.create('blk2', 'Block');
model.geom('geom1').feature.create('blk3', 'Block');
model.geom('geom1').feature('blk1').set('size', {'7e-3' '17e-3' '10e-6'});
model.geom('geom1').feature('blk2').set('pos', {'0' '0' '10e-6'});
model.geom('geom1').feature('blk2').set('size', {'7e-3' '17e-3' '180e-6'});
model.geom('geom1').feature('blk3').set('pos', {'0' '0' '190e-6'});
model.geom('geom1').feature('blk3').set('size', {'7e-3' '17e-3' '10e-6'});
model.geom('geom1').run;

model.material.create('mat1');
model.material('mat1').propertyGroup.create('Enu', 'Young''s modulus and
Poisson''s ratio');
model.material('mat1').selection.set([2]);
model.material.create('mat2');
model.material('mat2').propertyGroup.create('Enu', 'Young''s modulus and
Poisson''s ratio');
model.material('mat2').selection.set([1 3]);

model.physics.create('ec', 'ConductiveMedia', 'geom1');
model.physics('ec').feature.create('gnd1', 'Ground', 2);
model.physics('ec').feature('gnd1').selection.set([2]);
model.physics('ec').feature.create('pot1', 'ElectricPotential', 2);
model.physics('ec').feature('pot1').selection.set([8]);

model.mesh.create('mesh1', 'geom1');
model.mesh('mesh1').feature.create('ftet1', 'FreeTet');

model.result.table.create('evl3', 'Table');
model.result.table.create('tbl1', 'Table');


model.material('mat1').name('Nylon');
```

```
model.material('mat1').propertyGroup('def').set('heatcapacity',
'1700[J/(kg*K)]');
model.material('mat1').propertyGroup('def').set('relpermittivity', {'4' '0'
'0' '0' '4' '0' '0' '0' '4'});
model.material('mat1').propertyGroup('def').set('thermalexpansioncoefficient'
, {'280e-6[1/K]' '0' '0' '0' '280e-6[1/K]' '0' '0' '0' '280e-6[1/K]'});
model.material('mat1').propertyGroup('def').set('density', '1150[kg/m^3]');
model.material('mat1').propertyGroup('def').set('thermalconductivity',
{'0.26[W/(m*K)]' '0' '0' '0' '0.26[W/(m*K)]' '0' '0' '0' '0.26[W/(m*K)]'});
model.material('mat1').propertyGroup('def').set('electricconductivity', {'10'
'0' '0' '0' '10' '0' '0' '0' '10'});
model.material('mat1').propertyGroup('Enu').set('youngsmodulus', '2e9[Pa]');
model.material('mat1').propertyGroup('Enu').set('poissonsratio', '0.4');
model.material('mat2').name('Pt');
model.material('mat2').propertyGroup('def').set('electricconductivity',
{'1e6[S/m]' '0' '0' '0' '1e6[S/m]' '0' '0' '0' '1e6[S/m]'});
model.material('mat2').propertyGroup('def').set('thermalexpansioncoefficient'
, {'8.80e-6[1/K]' '0' '0' '0' '8.80e-6[1/K]' '0' '0' '0' '8.80e-6[1/K]'});
model.material('mat2').propertyGroup('def').set('heatcapacity',
'133[J/(kg*K)]');
model.material('mat2').propertyGroup('def').set('density', '21450[kg/m^3]');
model.material('mat2').propertyGroup('def').set('thermalconductivity',
{'71.6[W/(m*K)]' '0' '0' '0' '71.6[W/(m*K)]' '0' '0' '0' '71.6[W/(m*K)]'});
model.material('mat2').propertyGroup('def').set('relpermittivity',
{'1.000265' '0' '0' '0' '1.000265' '0' '0' '0' '1.000265'});
model.material('mat2').propertyGroup('Enu').set('youngsmodulus',
'168e9[Pa]');
model.material('mat2').propertyGroup('Enu').set('poissonsratio', '0.38');

model.physics('ec').feature('pot1').set('V0', '2');

model.mesh('mesh1').feature('size').set('hauto', 6);
model.mesh('mesh1').run;

model.result.table('evl3').name('Evaluation 3D');
model.result.table('evl3').comments('Interactive 3D values');
model.result.table('tbl1').comments('Surface Integration 1 (1)');

model.study.create('std1');
model.study('std1').feature.create('stat', 'Stationary');

model.sol.create('sol1');
model.sol('sol1').study('std1');
model.sol('sol1').attach('std1');
model.sol('sol1').feature.create('st1', 'StudyStep');
model.sol('sol1').feature.create('v1', 'Variables');
model.sol('sol1').feature.create('s1', 'Stationary');
model.sol('sol1').feature('s1').feature.create('fc1', 'FullyCoupled');
model.sol('sol1').feature('s1').feature.create('i1', 'Iterative');
model.sol('sol1').feature('s1').feature('i1').feature.create('mg1',
'Multigrid');
model.sol('sol1').feature('s1').feature.remove('fcDef');

model.study('std1').feature('stat').set('initstudyhide', 'on');
model.study('std1').feature('stat').set('initsolhide', 'on');
model.study('std1').feature('stat').set('notstudyhide', 'on');
```

```
model.study('std1').feature('stat').set('notsolhide', 'on');

model.result.numerical.create('int1', 'IntSurface');
model.result.numerical('int1').selection.set([3]);
model.result.numerical('int1').set('probetag', 'none');
model.result.create('pg1', 'PlotGroup3D');
model.result('pg1').feature.create('mslc1', 'Multislice');
model.result('pg1').feature.create('vol1', 'Volume');
model.result.export.create('tbl1', 'Table');

model.sol('sol1').attach('std1');
model.sol('sol1').feature('st1').name('Compile Equations: Stationary');
model.sol('sol1').feature('st1').set('studystep', 'stat');
model.sol('sol1').feature('v1').set('control', 'stat');
model.sol('sol1').feature('v1').feature('mod1_V').name('mod1.V');
model.sol('sol1').feature('s1').set('control', 'stat');
model.sol('sol1').feature('s1').feature('fc1').set('termonres', 'off');
model.sol('sol1').feature('s1').feature('i1').set('linsolver', 'cg');
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('prefun',
'amg');
model.sol('sol1').runAll;

model.result.numerical('int1').set('unit', 'm^2');
model.result.numerical('int1').set('table', 'tbl1');
model.result.numerical('int1').set('descr', '1');
model.result.numerical('int1').set('expr', '1');
model.result.numerical('int1').setResult;
model.result('pg1').name('Electric potential');
model.result('pg1').feature('mslc1').name('Multislice');
model.result('pg1').feature('mslc1').set('solrepresentation', 'solnum');
model.result.export('tbl1').set('table', 'tbl1');
model.result.export('tbl1').set('filename',
'C:\Users\jsimpson\Desktop\area.txt');

out = model;

IPMC_width = 20;
IPMC_length = 20;

%Create an empty square matrix at z=0 (the anode) that contains the points
%in the xy plane where the voltage will be sampled.

[x,y] = meshgrid(0:IPMC_width, 0:IPMC_length);
pz = zeros(1,(IPMC_width+1)*(IPMC_length+1));
p = [x(:)'; y(:)'; pz]*(1e-3);
p_cat = [x(:)'; y(:)'; pz+(190e-3)]*(1e-3);

%Extract voltage information from electrical model

V_cathode = mphinterp(model,'V','coord',p);
V_anode = mphinterp(model,'V','coord',p_cat);

Voltage_anode = [x(:) y(:) V_anode'];
Voltage_cathode = [x(:) y(:) V_cathode'];
```

```
Voltage_anode(isnan(Voltage_anode)) = 0;
Voltage_cathode(isnan(Voltage_cathode)) = 0;

fid = fopen('anode1.txt','w');
fprintf(fid,'%%x\ty\tVoltage_anode\r\n');
fclose(fid);
dlmwrite('anode1.txt',Voltage_anode,'precision','%2.6f','delimiter','\t','new
line','pc','-append');

fid = fopen('cathode1.txt','w');
fprintf(fid,'%%x\ty\tVoltage_cathode\r\n');
fclose(fid);
dlmwrite('cathode1.txt',Voltage_cathode,'precision','%2.6f','delimiter','\t',
'newline','pc','-append');

model.result.export('tbl1').run;

import com.comsol.model.*
import com.comsol.model.util.*

model = ModelUtil.create('Model');

model.modelPath('C:\Users\Justin\Desktop\Comsol Models');

model.name('concentrationdistributionclean.mph');

model.modelNode.create('mod1');

% length of voltage vector
[n,~] = size(Voltage_anode);

% vector of z sample points
z = ((0:180)'*1e-6);
Concentration = [];


model.geom.create('geom1', 1);
model.geom('geom1').feature.create('i1', 'Interval');


model.variable.create('var1');

model.physics.create('chds', 'DilutedSpecies', 'geom1');

model.mesh.create('mesh1', 'geom1');
model.mesh('mesh1').feature.create('edg1', 'Edge');


model.study.create('std1');
model.study('std1').feature.create('time', 'Transient');

model.sol.create('sol1');
model.sol('sol1').feature.create('st1', 'StudyStep');
```

```
model.sol('sol1').feature.create('v1', 'Variables');
model.sol('sol1').feature.create('t1', 'Time');
model.sol('sol1').feature('t1').feature.create('fc1', 'FullyCoupled');
model.sol('sol1').feature('t1').feature.create('d1', 'Direct');
model.sol('sol1').feature('t1').feature.remove('fcDef');
model.result.create('pg1', 'PlotGroup1D');
model.result('pg1').feature.create('lngr1', 'LineGraph');


for i = 1:n

% extract anode and cathode voltage for element n.
V_a = Voltage_anode(i,3);
V_c = Voltage_cathode(i,3);

% vector of x and y sample points
x = Voltage_anode(i,1)'*ones(181,1);
y = Voltage_anode(i,2)'*ones(181,1);



model.geom('geom1').feature('i1').set('p2', '180e-6');
model.geom('geom1').run;


model.variable('var1').set('Va',strcat(num2str(V_a)),'[V]'), ...);
model.variable('var1').set('Vc',strcat(num2str(V_c)),'[V]');




% model.variable('var1').name('Variables 1a');

model.view('view1').axis.set('xmin', '-9.000000318337698E-6');
model.view('view1').axis.set('xmax', '1.8899999849963933E-4');

model.physics('chds').prop('EquationForm').set('form', 'Transient');
model.physics('chds').prop('Migration').set('Migration', '1');
model.physics('chds').prop('Convection').set('Convection', '0');
model.physics('chds').feature('cdm1').set('V', 'Vc+((Vc-Va)/180e-6)*x[V/m]');
model.physics('chds').feature('cdm1').set('D_0', {'6e-12[m^2/s]'; '0'; '0';
'0'; '6e-12[m^2/s]'; '0'; '0'; '0'; '6e-12[m^2/s]'});
model.physics('chds').feature('cdm1').set('z', '1');
model.physics('chds').feature('cdm1').set('um', '2.4630522e-15[s*mol/kg]');
model.physics('chds').feature('init1').set('c', '1250');

model.mesh('mesh1').feature('size').set('hmax', '1e-6');
model.mesh('mesh1').run;

model.frame('material1').sorder(1);


model.sol('sol1').study('std1');
```

```
model.sol('sol1').attach('std1');



model.result('pg1').set('probetag', 'none');


model.study('std1').feature('time').set('tlist', 'range(0,1,5)');

model.result('pg1').feature('lngr1').selection.all;
model.result('pg1').feature('lngr1').selection.all;

model.sol('sol1').attach('std1');
model.sol('sol1').feature('st1').name('Compile Equations: Time Dependent');
model.sol('sol1').feature('st1').set('studystep', 'time');
model.sol('sol1').feature('v1').set('control', 'time');
model.sol('sol1').feature('t1').set('control', 'time');
model.sol('sol1').feature('t1').set('tlist', 'range(0,1,5)');
model.sol('sol1').feature('t1').set('maxorder', '2');
model.sol('sol1').feature('t1').feature('fc1').set('maxiter', '5');
model.sol('sol1').feature('t1').feature('fc1').set('jtech', 'once');
model.sol('sol1').feature('t1').feature('d1').set('linsolver', 'pardiso');
model.sol('sol1').runAll;

model.result('pg1').name('Concentration (chds)');
model.result('pg1').set('looplevelinput', {'manual'});
model.result('pg1').set('showlooplevel', {'on' 'off' 'off'});
model.result('pg1').set('xlabel', 'x-coordinate (m)');
model.result('pg1').set('ylabel', 'Concentration (mol/m<sup>3</sup>)');
model.result('pg1').set('xlabelactive', false);
model.result('pg1').set('ylabelactive', false);
model.result('pg1').feature('lngr1').set('xdata', 'expr');
model.result('pg1').feature('lngr1').set('xdataexpr', 'x');
model.result('pg1').feature('lngr1').set('xdataunit', 'm');
model.result('pg1').feature('lngr1').set('xdatadescr', 'x-coordinate');
model.result('pg1').feature('lngr1').selection.all;
model.result('pg1').feature('lngr1').selection.all;

% Save current fem structure for restart purposes
model0=model;

% extract concentration values at points in vector z
con = mphinterp(model0,'c','coord',z','T',5)';

% Concentration = [x y z c]

Concentration = [Concentration; x y z con];

end


% Force equation

Force = [Concentration(:,1:3) (9e2/1.2)*((Concentration(:,4)-1250).^2)];
```

```
fid = fopen('rectangleforce.txt','w');
fprintf(fid,'%%x\ty\tz\tForce\r\n');
fclose(fid);
dlmwrite('rectangleforce.txt',Force,'precision','%2.6f','delimiter','\t','new
line','pc','-append');

%
% rectangleoptimized1.m
%
% Model exported on Jul 12 2013, 13:23 by COMSOL 4.3.2.164.

import com.comsol.model.*
import com.comsol.model.util.*

model = ModelUtil.create('Model');

model.modelPath('.\');

model.name('rectangleoptimizeddeflection.mph');

model.modelNode.create('mod1');

model.file.create('res3');

model.func.create('int1', 'Interpolation');
model.func('int1').set('sourcetype', 'model');
model.func('int1').set('importedstruct', 'Spreadsheet');
model.func('int1').set('funcs', {'Force' '4'});
model.func('int1').set('importedname', 'rectangleforce.txt');
model.func('int1').set('importeddim', '3D');
model.func('int1').set('modelres', 'res3');

model.file('res3').resource('C:\Users\jsimpson\Desktop\rectangleforce.txt');

model.func('int1').set('nargs', '3');
model.func('int1').set('struct', 'spreadsheet');

model.geom.create('geom1', 3);
model.geom('geom1').feature.create('BLK1', 'Block');
model.geom('geom1').feature('BLK1').set('pos', '0.0,0.0,0.0');
model.geom('geom1').feature('BLK1').set('size', {'0.0070' '0.017' '1.8E-4'});
model.geom('geom1').feature('BLK1').set('axis', {'0' '0' '1'});
model.geom('geom1').run;

model.material.create('mat1');
model.material('mat1').propertyGroup.create('Enu', 'Young''s modulus and
Poisson''s ratio');

model.physics.create('smsld', 'SolidMechanics', 'geom1');
model.physics('smsld').identifier('smsld');
model.physics('smsld').feature.create('lemm2', 'LinearElasticModel', 3);
model.physics('smsld').feature('lemm2').selection.set([1]);
model.physics('smsld').feature.create('bl1', 'BodyLoad', 3);
model.physics('smsld').feature('bl1').selection.set([1]);
```

```
model.physics('smsld').feature.create('fix1', 'Fixed', 2);
model.physics('smsld').feature('fix1').selection.set([2]);

model.mesh.create('mesh1', 'geom1');
model.mesh('mesh1').feature.create('ftet1', 'FreeTet');
model.mesh('mesh1').feature('ftet1').selection.geom('geom1', 3);
model.mesh('mesh1').feature('ftet1').selection.set([1]);

model.result.table.create('tbl1', 'Table');
model.result.table.create('tbl2', 'Table');


model.material('mat1').name('Nylon');
model.material('mat1').propertyGroup('def').set('heatcapacity',
'1700[J/(kg*K)]');
model.material('mat1').propertyGroup('def').set('relpermittivity', {'4' '0'
'0' '0' '4' '0' '0' '0' '4'});
model.material('mat1').propertyGroup('def').set('thermalexpansioncoefficient'
, {'280e-6[1/K]' '0' '0' '0' '280e-6[1/K]' '0' '0' '0' '280e-6[1/K]'});
model.material('mat1').propertyGroup('def').set('density', '2000[kg/m^3]');
model.material('mat1').propertyGroup('def').set('thermalconductivity',
{'0.26[W/(m*K)]' '0' '0' '0' '0.26[W/(m*K)]' '0' '0' '0' '0.26[W/(m*K)]'});
model.material('mat1').propertyGroup('Enu').set('youngsmodulus', '5e8[Pa]');
model.material('mat1').propertyGroup('Enu').set('poissonsratio', '0.49');

model.physics('smsld').feature('lemm1').set('Evector_mat', 'userdef');
model.physics('smsld').feature('lemm1').set('Evector', {'2.0e11'; '2.0e11';
'2.0e11'});
model.physics('smsld').feature('lemm1').set('nuvector_mat', 'userdef');
model.physics('smsld').feature('lemm1').set('nuvector', {'0.33'; '0.33';
'0.33'});
model.physics('smsld').feature('lemm1').set('Gvector_mat', 'userdef');
model.physics('smsld').feature('lemm1').set('Gvector', {'7.52e10'; '7.52e10';
'7.52e10'});
model.physics('smsld').feature('lemm1').set('D_mat', 'userdef');
model.physics('smsld').feature('lemm1').set('D', {'2.0e11/((1+0.33)*(1-
2*0.33))*(1-0.33)'; '2.0e11/((1+0.33)*(1-2*0.33))*0.33';
'2.0e11/((1+0.33)*(1-2*0.33))*0.33'; '0'; '0'; '0'; '2.0e11/((1+0.33)*(1-
2*0.33))*0.33'; '2.0e11/((1+0.33)*(1-2*0.33))*(1-0.33)';
'2.0e11/((1+0.33)*(1-2*0.33))*0.33'; '0';  ...
'0'; '0'; '2.0e11/((1+0.33)*(1-2*0.33))*0.33'; '2.0e11/((1+0.33)*(1-
2*0.33))*0.33'; '2.0e11/((1+0.33)*(1-2*0.33))*(1-0.33)'; '0'; '0'; '0'; '0';
'0';  ...
'0'; '2.0e11/((1+0.33)*2)'; '0'; '0'; '0'; '0'; '0'; '0';
'2.0e11/((1+0.33)*2)'; '0';  ...
'0'; '0'; '0'; '0'; '0'; '2.0e11/((1+0.33)*2)'});
model.physics('smsld').feature('lemm2').set('E_mat', 'userdef');
model.physics('smsld').feature('lemm2').set('E', '5e8');
model.physics('smsld').feature('lemm2').set('nu_mat', 'userdef');
model.physics('smsld').feature('lemm2').set('nu', '.48');
model.physics('smsld').feature('lemm2').set('Evector_mat', 'userdef');
model.physics('smsld').feature('lemm2').set('Evector', {'2.0e11'; '2.0e11';
'2.0e11'});
model.physics('smsld').feature('lemm2').set('nuvector_mat', 'userdef');
model.physics('smsld').feature('lemm2').set('nuvector', {'0.33'; '0.33';
'0.33'});
```

```
model.physics('smsld').feature('lemm2').set('Gvector_mat', 'userdef');
model.physics('smsld').feature('lemm2').set('Gvector', {'7.52e10'; '7.52e10';
'7.52e10'});
model.physics('smsld').feature('lemm2').set('D_mat', 'userdef');
model.physics('smsld').feature('lemm2').set('D', {'2.0e11/((1+0.33)*(1-
2*0.33))*(1-0.33)'; '2.0e11/((1+0.33)*(1-2*0.33))*0.33';
'2.0e11/((1+0.33)*(1-2*0.33))*0.33'; '0'; '0'; '0'; '2.0e11/((1+0.33)*(1-
2*0.33))*0.33'; '2.0e11/((1+0.33)*(1-2*0.33))*(1-0.33)';
'2.0e11/((1+0.33)*(1-2*0.33))*0.33'; '0';  ...
'0'; '0'; '2.0e11/((1+0.33)*(1-2*0.33))*0.33'; '2.0e11/((1+0.33)*(1-
2*0.33))*0.33'; '2.0e11/((1+0.33)*(1-2*0.33))*(1-0.33)'; '0'; '0'; '0'; '0';
'0';  ...
'0'; '2.0e11/((1+0.33)*2)'; '0'; '0'; '0'; '0'; '0'; '0';
'2.0e11/((1+0.33)*2)'; '0';  ...
'0'; '0'; '0'; '0'; '0'; '2.0e11/((1+0.33)*2)'});
model.physics('smsld').feature('lemm2').set('rho_mat', 'userdef');
model.physics('smsld').feature('lemm2').set('rho', '2000');
model.physics('smsld').feature('bl1').set('FperVol', {'0';
'Force(x[1/m],y[1/m],z[1/m])'; '0'});

model.mesh('mesh1').feature('size').set('hauto', 2);
model.mesh('mesh1').feature('ftet1').set('zscale', '1');
model.mesh('mesh1').run;

model.result.table('tbl1').comments('Point Evaluation 1 (smsld.disp)');
model.result.table('tbl2').comments('Line Maximum 1 (smsld.disp)');

model.coordSystem('sys1').set('mastercoordsystcomp', 'manual');

model.study.create('std1');
model.study('std1').feature.create('stat', 'Stationary');

model.sol.create('sol1');
model.sol('sol1').study('std1');
model.sol('sol1').attach('std1');
model.sol('sol1').feature.create('st1', 'StudyStep');
model.sol('sol1').feature.create('v1', 'Variables');
model.sol('sol1').feature.create('s1', 'Stationary');
model.sol('sol1').feature('s1').feature.create('fc1', 'FullyCoupled');
model.sol('sol1').feature('s1').feature.create('d1', 'Direct');

model.study('std1').feature('stat').set('initstudyhide', 'on');
model.study('std1').feature('stat').set('initsolhide', 'on');
model.study('std1').feature('stat').set('notstudyhide', 'on');
model.study('std1').feature('stat').set('notsolhide', 'on');

model.result.numerical.create('max1', 'MaxLine');
model.result.numerical('max1').selection.set([6]);
model.result.numerical('max1').set('probetag', 'none');
model.result.create('pg1', 'PlotGroup3D');
model.result('pg1').feature.create('vol1', 'Volume');
model.result('pg1').feature('vol1').feature.create('def1', 'Deform');
model.result.export.create('tbl2', 'Table');

model.sol('sol1').attach('std1');
```

```
model.sol('sol1').feature('st1').name('Compile Equations: Stationary');
model.sol('sol1').feature('st1').set('studystep', 'stat');
model.sol('sol1').feature('v1').set('control', 'stat');
model.sol('sol1').feature('s1').set('control', 'stat');
model.sol('sol1').feature('s1').set('stol', '1.0E-6');
model.sol('sol1').feature('s1').feature('fc1').set('initstep', '1.0');
model.sol('sol1').feature('s1').feature('fc1').set('rstep', '10.0');
model.sol('sol1').feature('s1').feature('fc1').set('minstep', '1.0e-4');
model.sol('sol1').feature('s1').feature('fc1').set('termonres', 'off');
model.sol('sol1').feature('s1').feature('d1').set('errorchk', 'off');
model.sol('sol1').feature('s1').feature('d1').set('linsolver', 'spooles');
model.sol('sol1').runAll;

model.result.numerical('max1').set('unit', 'mm');
model.result.numerical('max1').set('table', 'tbl2');
model.result.numerical('max1').setResult;
model.result('pg1').name('Stress (smsld)');
model.result('pg1').feature('vol1').set('unit', 'mm');
model.result('pg1').feature('vol1').feature('def1').set('scaleactive', true);
model.result.export('tbl2').set('filename', '.\deflection.txt');
model.result.export('tbl2').set('table', 'tbl2');

model.result.export('tbl2').run;



quit
```

A6. Combined Deflection and Force

```
function out = model
%
% sevenbyseventeenvoltage.m
%
% Model exported on Jul 6 2012, 15:12 by COMSOL 4.3.0.151.

import com.comsol.model.*
import com.comsol.model.util.*

model = ModelUtil.create('Model');

model.modelPath('C:\Users\Justin\Desktop\Comsol Models');

model.name('7x17voltagedistribution.mph');

model.modelNode.create('mod1');

model.geom.create('geom1', 3);
model.geom('geom1').feature.create('blk1', 'Block');
model.geom('geom1').feature.create('blk2', 'Block');
model.geom('geom1').feature.create('blk3', 'Block');
model.geom('geom1').feature('blk1').set('size', {'7e-3' '17e-3' '10e-6'});
model.geom('geom1').feature('blk2').set('pos', {'0' '0' '10e-6'});
model.geom('geom1').feature('blk2').set('size', {'7e-3' '17e-3' '180e-6'});
model.geom('geom1').feature('blk3').set('pos', {'0' '0' '190e-6'});
model.geom('geom1').feature('blk3').set('size', {'7e-3' '17e-3' '10e-6'});
model.geom('geom1').run;

model.material.create('mat1');
model.material('mat1').propertyGroup.create('Enu', 'Young''s modulus and
Poisson''s ratio');
model.material('mat1').selection.set([2]);
model.material.create('mat2');
model.material('mat2').propertyGroup.create('Enu', 'Young''s modulus and
Poisson''s ratio');
model.material('mat2').selection.set([1 3]);

model.physics.create('ec', 'ConductiveMedia', 'geom1');
model.physics('ec').feature.create('gnd1', 'Ground', 2);
model.physics('ec').feature('gnd1').selection.set([2]);
model.physics('ec').feature.create('pot1', 'ElectricPotential', 2);
model.physics('ec').feature('pot1').selection.set([8]);

model.mesh.create('mesh1', 'geom1');
model.mesh('mesh1').feature.create('ftet1', 'FreeTet');

model.result.table.create('evl3', 'Table');
model.result.table.create('tbl1', 'Table');


model.material('mat1').name('Nylon');
```

```
model.material('mat1').propertyGroup('def').set('heatcapacity',
'1700[J/(kg*K)]');
model.material('mat1').propertyGroup('def').set('relpermittivity', {'4' '0'
'0' '0' '4' '0' '0' '0' '4'});
model.material('mat1').propertyGroup('def').set('thermalexpansioncoefficient'
, {'280e-6[1/K]' '0' '0' '0' '280e-6[1/K]' '0' '0' '0' '280e-6[1/K]'});
model.material('mat1').propertyGroup('def').set('density', '1150[kg/m^3]');
model.material('mat1').propertyGroup('def').set('thermalconductivity',
{'0.26[W/(m*K)]' '0' '0' '0' '0.26[W/(m*K)]' '0' '0' '0' '0.26[W/(m*K)]'});
model.material('mat1').propertyGroup('def').set('electricconductivity', {'10'
'0' '0' '0' '10' '0' '0' '0' '10'});
model.material('mat1').propertyGroup('Enu').set('youngsmodulus', '2e9[Pa]');
model.material('mat1').propertyGroup('Enu').set('poissonsratio', '0.4');
model.material('mat2').name('Pt');
model.material('mat2').propertyGroup('def').set('electricconductivity',
{'1e6[S/m]' '0' '0' '0' '1e6[S/m]' '0' '0' '0' '1e6[S/m]'});
model.material('mat2').propertyGroup('def').set('thermalexpansioncoefficient'
, {'8.80e-6[1/K]' '0' '0' '0' '8.80e-6[1/K]' '0' '0' '0' '8.80e-6[1/K]'});
model.material('mat2').propertyGroup('def').set('heatcapacity',
'133[J/(kg*K)]');
model.material('mat2').propertyGroup('def').set('density', '21450[kg/m^3]');
model.material('mat2').propertyGroup('def').set('thermalconductivity',
{'71.6[W/(m*K)]' '0' '0' '0' '71.6[W/(m*K)]' '0' '0' '0' '71.6[W/(m*K)]'});
model.material('mat2').propertyGroup('def').set('relpermittivity',
{'1.000265' '0' '0' '0' '1.000265' '0' '0' '0' '1.000265'});
model.material('mat2').propertyGroup('Enu').set('youngsmodulus',
'168e9[Pa]');
model.material('mat2').propertyGroup('Enu').set('poissonsratio', '0.38');

model.physics('ec').feature('pot1').set('V0', '2');

model.mesh('mesh1').feature('size').set('hauto', 6);
model.mesh('mesh1').run;

model.result.table('evl3').name('Evaluation 3D');
model.result.table('evl3').comments('Interactive 3D values');
model.result.table('tbl1').comments('Surface Integration 1 (1)');

model.study.create('std1');
model.study('std1').feature.create('stat', 'Stationary');

model.sol.create('sol1');
model.sol('sol1').study('std1');
model.sol('sol1').attach('std1');
model.sol('sol1').feature.create('st1', 'StudyStep');
model.sol('sol1').feature.create('v1', 'Variables');
model.sol('sol1').feature.create('s1', 'Stationary');
model.sol('sol1').feature('s1').feature.create('fc1', 'FullyCoupled');
model.sol('sol1').feature('s1').feature.create('i1', 'Iterative');
model.sol('sol1').feature('s1').feature('i1').feature.create('mg1',
'Multigrid');
model.sol('sol1').feature('s1').feature.remove('fcDef');

model.study('std1').feature('stat').set('initstudyhide', 'on');
model.study('std1').feature('stat').set('initsolhide', 'on');
model.study('std1').feature('stat').set('notstudyhide', 'on');
```

```
model.study('std1').feature('stat').set('notsolhide', 'on');

model.result.numerical.create('int1', 'IntSurface');
model.result.numerical('int1').selection.set([3]);
model.result.numerical('int1').set('probetag', 'none');
model.result.create('pg1', 'PlotGroup3D');
model.result('pg1').feature.create('mslc1', 'Multislice');
model.result('pg1').feature.create('vol1', 'Volume');
model.result.export.create('tbl1', 'Table');

model.sol('sol1').attach('std1');
model.sol('sol1').feature('st1').name('Compile Equations: Stationary');
model.sol('sol1').feature('st1').set('studystep', 'stat');
model.sol('sol1').feature('v1').set('control', 'stat');
model.sol('sol1').feature('v1').feature('mod1_V').name('mod1.V');
model.sol('sol1').feature('s1').set('control', 'stat');
model.sol('sol1').feature('s1').feature('fc1').set('termonres', 'off');
model.sol('sol1').feature('s1').feature('i1').set('linsolver', 'cg');
model.sol('sol1').feature('s1').feature('i1').feature('mg1').set('prefun',
'amg');
model.sol('sol1').runAll;

model.result.numerical('int1').set('unit', 'm^2');
model.result.numerical('int1').set('table', 'tbl1');
model.result.numerical('int1').set('descr', '1');
model.result.numerical('int1').set('expr', '1');
model.result.numerical('int1').setResult;
model.result('pg1').name('Electric potential');
model.result('pg1').feature('mslc1').name('Multislice');
model.result('pg1').feature('mslc1').set('solrepresentation', 'solnum');
model.result.export('tbl1').set('table', 'tbl1');
model.result.export('tbl1').set('filename',
'C:\Users\jsimpson\Desktop\area.txt');

out = model;

IPMC_width = 20;
IPMC_length = 20;

%Create an empty square matrix at z=0 (the anode) that contains the points
%in the xy plane where the voltage will be sampled.

[x,y] = meshgrid(0:IPMC_width, 0:IPMC_length);
pz = zeros(1,(IPMC_width+1)*(IPMC_length+1));
p = [x(:)'; y(:)'; pz]*(1e-3);
p_cat = [x(:)'; y(:)'; pz+(190e-3)]*(1e-3);

%Extract voltage information from electrical model

V_cathode = mphinterp(model,'V','coord',p);
V_anode = mphinterp(model,'V','coord',p_cat);

Voltage_anode = [x(:) y(:) V_anode'];
Voltage_cathode = [x(:) y(:) V_cathode'];
```

```
Voltage_anode(isnan(Voltage_anode)) = 0;
Voltage_cathode(isnan(Voltage_cathode)) = 0;

fid = fopen('anode1.txt','w');
fprintf(fid,'%%x\ty\tVoltage_anode\r\n');
fclose(fid);
dlmwrite('anode1.txt',Voltage_anode,'precision','%2.6f','delimiter','\t','new
line','pc','-append');

fid = fopen('cathode1.txt','w');
fprintf(fid,'%%x\ty\tVoltage_cathode\r\n');
fclose(fid);
dlmwrite('cathode1.txt',Voltage_cathode,'precision','%2.6f','delimiter','\t',
'newline','pc','-append');

model.result.export('tbl1').run;

import com.comsol.model.*
import com.comsol.model.util.*

model = ModelUtil.create('Model');

model.modelPath('C:\Users\Justin\Desktop\Comsol Models');

model.name('concentrationdistributionclean.mph');

model.modelNode.create('mod1');

% length of voltage vector
[n,~] = size(Voltage_anode);

% vector of z sample points
z = ((0:180)'*1e-6);
Concentration = [];


model.geom.create('geom1', 1);
model.geom('geom1').feature.create('i1', 'Interval');


model.variable.create('var1');

model.physics.create('chds', 'DilutedSpecies', 'geom1');

model.mesh.create('mesh1', 'geom1');
model.mesh('mesh1').feature.create('edg1', 'Edge');


model.study.create('std1');
model.study('std1').feature.create('time', 'Transient');

model.sol.create('sol1');
model.sol('sol1').feature.create('st1', 'StudyStep');
```

```
model.sol('sol1').feature.create('v1', 'Variables');
model.sol('sol1').feature.create('t1', 'Time');
model.sol('sol1').feature('t1').feature.create('fc1', 'FullyCoupled');
model.sol('sol1').feature('t1').feature.create('d1', 'Direct');
model.sol('sol1').feature('t1').feature.remove('fcDef');
model.result.create('pg1', 'PlotGroup1D');
model.result('pg1').feature.create('lngr1', 'LineGraph');


for i = 1:n

% extract anode and cathode voltage for element n.
V_a = Voltage_anode(i,3);
V_c = Voltage_cathode(i,3);

% vector of x and y sample points
x = Voltage_anode(i,1)'*ones(181,1);
y = Voltage_anode(i,2)'*ones(181,1);



model.geom('geom1').feature('i1').set('p2', '180e-6');
model.geom('geom1').run;


model.variable('var1').set('Va',strcat(num2str(V_a)),'[V]'), ...);
model.variable('var1').set('Vc',strcat(num2str(V_c)),'[V]');




% model.variable('var1').name('Variables 1a');

model.view('view1').axis.set('xmin', '-9.000000318337698E-6');
model.view('view1').axis.set('xmax', '1.8899999849963933E-4');

model.physics('chds').prop('EquationForm').set('form', 'Transient');
model.physics('chds').prop('Migration').set('Migration', '1');
model.physics('chds').prop('Convection').set('Convection', '0');
model.physics('chds').feature('cdm1').set('V', 'Vc+((Vc-Va)/180e-6)*x[V/m]');
model.physics('chds').feature('cdm1').set('D_0', {'6e-12[m^2/s]'; '0'; '0';
'0'; '6e-12[m^2/s]'; '0'; '0'; '0'; '6e-12[m^2/s]'});
model.physics('chds').feature('cdm1').set('z', '1');
model.physics('chds').feature('cdm1').set('um', '2.4630522e-15[s*mol/kg]');
model.physics('chds').feature('init1').set('c', '1250');

model.mesh('mesh1').feature('size').set('hmax', '1e-6');
model.mesh('mesh1').run;

model.frame('material1').sorder(1);


model.sol('sol1').study('std1');
```

```
model.sol('sol1').attach('std1');



model.result('pg1').set('probetag', 'none');


model.study('std1').feature('time').set('tlist', 'range(0,1,5)');

model.result('pg1').feature('lngr1').selection.all;
model.result('pg1').feature('lngr1').selection.all;

model.sol('sol1').attach('std1');
model.sol('sol1').feature('st1').name('Compile Equations: Time Dependent');
model.sol('sol1').feature('st1').set('studystep', 'time');
model.sol('sol1').feature('v1').set('control', 'time');
model.sol('sol1').feature('t1').set('control', 'time');
model.sol('sol1').feature('t1').set('tlist', 'range(0,1,5)');
model.sol('sol1').feature('t1').set('maxorder', '2');
model.sol('sol1').feature('t1').feature('fc1').set('maxiter', '5');
model.sol('sol1').feature('t1').feature('fc1').set('jtech', 'once');
model.sol('sol1').feature('t1').feature('d1').set('linsolver', 'pardiso');
model.sol('sol1').runAll;

model.result('pg1').name('Concentration (chds)');
model.result('pg1').set('looplevelinput', {'manual'});
model.result('pg1').set('showlooplevel', {'on' 'off' 'off'});
model.result('pg1').set('xlabel', 'x-coordinate (m)');
model.result('pg1').set('ylabel', 'Concentration (mol/m<sup>3</sup>)');
model.result('pg1').set('xlabelactive', false);
model.result('pg1').set('ylabelactive', false);
model.result('pg1').feature('lngr1').set('xdata', 'expr');
model.result('pg1').feature('lngr1').set('xdataexpr', 'x');
model.result('pg1').feature('lngr1').set('xdataunit', 'm');
model.result('pg1').feature('lngr1').set('xdatadescr', 'x-coordinate');
model.result('pg1').feature('lngr1').selection.all;
model.result('pg1').feature('lngr1').selection.all;

% Save current fem structure for restart purposes
model0=model;

% extract concentration values at points in vector z
con = mphinterp(model0,'c','coord',z','T',5)';

% Concentration = [x y z c]

Concentration = [Concentration; x y z con];

end


% Force equation

Force = [Concentration(:,1:3) (9e2/1.2)*((Concentration(:,4)-1250).^2)];
```

```
fid = fopen('rectangleforce.txt','w');
fprintf(fid,'%%x\ty\tz\tForce\r\n');
fclose(fid);
dlmwrite('rectangleforce.txt',Force,'precision','%2.6f','delimiter','\t','new
line','pc','-append');

%
% rectangleoptimized1.m
%
% Model exported on Jul 12 2013, 13:23 by COMSOL 4.3.2.164.

import com.comsol.model.*
import com.comsol.model.util.*

model = ModelUtil.create('Model');

model.modelPath('.\');

model.name('rectangleoptimizeddeflection.mph');

model.modelNode.create('mod1');

model.file.create('res3');

model.func.create('int1', 'Interpolation');
model.func('int1').set('sourcetype', 'model');
model.func('int1').set('importedstruct', 'Spreadsheet');
model.func('int1').set('funcs', {'Force' '4'});
model.func('int1').set('importedname', 'rectangleforce.txt');
model.func('int1').set('importeddim', '3D');
model.func('int1').set('modelres', 'res3');

model.file('res3').resource('C:\Users\jsimpson\Desktop\rectangleforce.txt');

model.func('int1').set('nargs', '3');
model.func('int1').set('struct', 'spreadsheet');

model.geom.create('geom1', 3);
model.geom('geom1').feature.create('BLK1', 'Block');
model.geom('geom1').feature('BLK1').set('pos', '0.0,0.0,0.0');
model.geom('geom1').feature('BLK1').set('size', {'0.0070' '0.017' '1.8E-4'});
model.geom('geom1').feature('BLK1').set('axis', {'0' '0' '1'});
model.geom('geom1').run;

model.material.create('mat1');
model.material('mat1').propertyGroup.create('Enu', 'Young''s modulus and
Poisson''s ratio');

model.physics.create('smsld', 'SolidMechanics', 'geom1');
model.physics('smsld').identifier('smsld');
model.physics('smsld').feature.create('lemm2', 'LinearElasticModel', 3);
model.physics('smsld').feature('lemm2').selection.set([1]);
model.physics('smsld').feature.create('bl1', 'BodyLoad', 3);
model.physics('smsld').feature('bl1').selection.set([1]);
```

```
model.physics('smsld').feature.create('fix1', 'Fixed', 2);
model.physics('smsld').feature('fix1').selection.set([2]);

model.mesh.create('mesh1', 'geom1');
model.mesh('mesh1').feature.create('ftet1', 'FreeTet');
model.mesh('mesh1').feature('ftet1').selection.geom('geom1', 3);
model.mesh('mesh1').feature('ftet1').selection.set([1]);

model.result.table.create('tbl1', 'Table');
model.result.table.create('tbl2', 'Table');


model.material('mat1').name('Nylon');
model.material('mat1').propertyGroup('def').set('heatcapacity',
'1700[J/(kg*K)]');
model.material('mat1').propertyGroup('def').set('relpermittivity', {'4' '0'
'0' '0' '4' '0' '0' '0' '4'});
model.material('mat1').propertyGroup('def').set('thermalexpansioncoefficient'
, {'280e-6[1/K]' '0' '0' '0' '280e-6[1/K]' '0' '0' '0' '280e-6[1/K]'});
model.material('mat1').propertyGroup('def').set('density', '2000[kg/m^3]');
model.material('mat1').propertyGroup('def').set('thermalconductivity',
{'0.26[W/(m*K)]' '0' '0' '0' '0.26[W/(m*K)]' '0' '0' '0' '0.26[W/(m*K)]'});
model.material('mat1').propertyGroup('Enu').set('youngsmodulus', '5e8[Pa]');
model.material('mat1').propertyGroup('Enu').set('poissonsratio', '0.49');

model.physics('smsld').feature('lemm1').set('Evector_mat', 'userdef');
model.physics('smsld').feature('lemm1').set('Evector', {'2.0e11'; '2.0e11';
'2.0e11'});
model.physics('smsld').feature('lemm1').set('nuvector_mat', 'userdef');
model.physics('smsld').feature('lemm1').set('nuvector', {'0.33'; '0.33';
'0.33'});
model.physics('smsld').feature('lemm1').set('Gvector_mat', 'userdef');
model.physics('smsld').feature('lemm1').set('Gvector', {'7.52e10'; '7.52e10';
'7.52e10'});
model.physics('smsld').feature('lemm1').set('D_mat', 'userdef');
model.physics('smsld').feature('lemm1').set('D', {'2.0e11/((1+0.33)*(1-
2*0.33))*(1-0.33)'; '2.0e11/((1+0.33)*(1-2*0.33))*0.33';
'2.0e11/((1+0.33)*(1-2*0.33))*0.33'; '0'; '0'; '0'; '2.0e11/((1+0.33)*(1-
2*0.33))*0.33'; '2.0e11/((1+0.33)*(1-2*0.33))*(1-0.33)';
'2.0e11/((1+0.33)*(1-2*0.33))*0.33'; '0';  ...
'0'; '0'; '2.0e11/((1+0.33)*(1-2*0.33))*0.33'; '2.0e11/((1+0.33)*(1-
2*0.33))*0.33'; '2.0e11/((1+0.33)*(1-2*0.33))*(1-0.33)'; '0'; '0'; '0'; '0';
'0';  ...
'0'; '2.0e11/((1+0.33)*2)'; '0'; '0'; '0'; '0'; '0'; '0';
'2.0e11/((1+0.33)*2)'; '0';  ...
'0'; '0'; '0'; '0'; '0'; '2.0e11/((1+0.33)*2)'});
model.physics('smsld').feature('lemm2').set('E_mat', 'userdef');
model.physics('smsld').feature('lemm2').set('E', '5e8');
model.physics('smsld').feature('lemm2').set('nu_mat', 'userdef');
model.physics('smsld').feature('lemm2').set('nu', '.48');
model.physics('smsld').feature('lemm2').set('Evector_mat', 'userdef');
model.physics('smsld').feature('lemm2').set('Evector', {'2.0e11'; '2.0e11';
'2.0e11'});
model.physics('smsld').feature('lemm2').set('nuvector_mat', 'userdef');
model.physics('smsld').feature('lemm2').set('nuvector', {'0.33'; '0.33';
'0.33'});
```

```
model.physics('smsld').feature('lemm2').set('Gvector_mat', 'userdef');
model.physics('smsld').feature('lemm2').set('Gvector', {'7.52e10'; '7.52e10';
'7.52e10'});
model.physics('smsld').feature('lemm2').set('D_mat', 'userdef');
model.physics('smsld').feature('lemm2').set('D', {'2.0e11/((1+0.33)*(1-
2*0.33))*(1-0.33)'; '2.0e11/((1+0.33)*(1-2*0.33))*0.33';
'2.0e11/((1+0.33)*(1-2*0.33))*0.33'; '0'; '0'; '0'; '2.0e11/((1+0.33)*(1-
2*0.33))*0.33'; '2.0e11/((1+0.33)*(1-2*0.33))*(1-0.33)';
'2.0e11/((1+0.33)*(1-2*0.33))*0.33'; '0';  ...
'0'; '0'; '2.0e11/((1+0.33)*(1-2*0.33))*0.33'; '2.0e11/((1+0.33)*(1-
2*0.33))*0.33'; '2.0e11/((1+0.33)*(1-2*0.33))*(1-0.33)'; '0'; '0'; '0'; '0';
'0';  ...
'0'; '2.0e11/((1+0.33)*2)'; '0'; '0'; '0'; '0'; '0'; '0';
'2.0e11/((1+0.33)*2)'; '0';  ...
'0'; '0'; '0'; '0'; '0'; '2.0e11/((1+0.33)*2)'});
model.physics('smsld').feature('lemm2').set('rho_mat', 'userdef');
model.physics('smsld').feature('lemm2').set('rho', '2000');
model.physics('smsld').feature('bl1').set('FperVol', {'0';
'Force(x[1/m],y[1/m],z[1/m])'; '0'});

model.mesh('mesh1').feature('size').set('hauto', 2);
model.mesh('mesh1').feature('ftet1').set('zscale', '1');
model.mesh('mesh1').run;

model.result.table('tbl1').comments('Point Evaluation 1 (smsld.disp)');
model.result.table('tbl2').comments('Line Maximum 1 (smsld.disp)');

model.coordSystem('sys1').set('mastercoordsystcomp', 'manual');

model.study.create('std1');
model.study('std1').feature.create('stat', 'Stationary');

model.sol.create('sol1');
model.sol('sol1').study('std1');
model.sol('sol1').attach('std1');
model.sol('sol1').feature.create('st1', 'StudyStep');
model.sol('sol1').feature.create('v1', 'Variables');
model.sol('sol1').feature.create('s1', 'Stationary');
model.sol('sol1').feature('s1').feature.create('fc1', 'FullyCoupled');
model.sol('sol1').feature('s1').feature.create('d1', 'Direct');

model.study('std1').feature('stat').set('initstudyhide', 'on');
model.study('std1').feature('stat').set('initsolhide', 'on');
model.study('std1').feature('stat').set('notstudyhide', 'on');
model.study('std1').feature('stat').set('notsolhide', 'on');

model.result.numerical.create('max1', 'MaxLine');
model.result.numerical('max1').selection.set([6]);
model.result.numerical('max1').set('probetag', 'none');
model.result.create('pg1', 'PlotGroup3D');
model.result('pg1').feature.create('vol1', 'Volume');
model.result('pg1').feature('vol1').feature.create('def1', 'Deform');
model.result.export.create('tbl2', 'Table');

model.sol('sol1').attach('std1');
```

```
model.sol('sol1').feature('st1').name('Compile Equations: Stationary');
model.sol('sol1').feature('st1').set('studystep', 'stat');
model.sol('sol1').feature('v1').set('control', 'stat');
model.sol('sol1').feature('s1').set('control', 'stat');
model.sol('sol1').feature('s1').set('stol', '1.0E-6');
model.sol('sol1').feature('s1').feature('fc1').set('initstep', '1.0');
model.sol('sol1').feature('s1').feature('fc1').set('rstep', '10.0');
model.sol('sol1').feature('s1').feature('fc1').set('minstep', '1.0e-4');
model.sol('sol1').feature('s1').feature('fc1').set('termonres', 'off');
model.sol('sol1').feature('s1').feature('d1').set('errorchk', 'off');
model.sol('sol1').feature('s1').feature('d1').set('linsolver', 'spooles');
model.sol('sol1').runAll;

model.result.numerical('max1').set('unit', 'mm');
model.result.numerical('max1').set('table', 'tbl2');
model.result.numerical('max1').setResult;
model.result('pg1').name('Stress (smsld)');
model.result('pg1').feature('vol1').set('unit', 'mm');
model.result('pg1').feature('vol1').feature('def1').set('scaleactive', true);
model.result.export('tbl2').set('filename', '.\deflection.txt');
model.result.export('tbl2').set('table', 'tbl2');

model.result.export('tbl2').run;



import com.comsol.model.*
import com.comsol.model.util.*

model = ModelUtil.create('Model');

model.modelPath('.\');

model.name('1.2included7x17newest.mph');

model.modelNode.create('mod1');

model.file.create('res2');

model.func.create('int1', 'Interpolation');
model.func('int1').set('importedname', 'rectangleforce.txt');
model.func('int1').set('funcs', {'Force' '4'});
model.func('int1').set('modelres', 'res2');
model.func('int1').set('importedstruct', 'Spreadsheet');
model.func('int1').set('importeddim', '3D');
model.func('int1').set('sourcetype', 'model');

model.file('res2').resource('C:\Users\jsimpson\Desktop\rectangleforce.txt');

model.func('int1').set('nargs', '3');
model.func('int1').set('struct', 'spreadsheet');

model.geom.create('geom1', 3);
model.geom('geom1').feature.create('BLK1', 'Block');
```

```
model.geom('geom1').feature.create('CYL1', 'Cylinder');
model.geom('geom1').feature('BLK1').set('pos', '0.0,0.0,0.0');
model.geom('geom1').feature('BLK1').set('size', {'0.0070' '0.017' '1.8E-4'});
model.geom('geom1').feature('BLK1').set('axis', {'0' '0' '1'});
model.geom('geom1').feature('CYL1').set('axis', {'0' '0' '1'});
model.geom('geom1').feature('CYL1').set('r', '5.0E-4');
model.geom('geom1').feature('CYL1').set('pos', {'0.0035' '0.015' '1.8e-4'});
model.geom('geom1').feature('CYL1').set('h', '0.0010');
model.geom('geom1').run;

model.material.create('mat1');
model.material('mat1').propertyGroup.create('Enu', 'Young''s modulus and
Poisson''s ratio');
model.material('mat1').propertyGroup.create('RefractiveIndex', 'Refractive
index');
model.material('mat1').selection.set([2]);
model.material.create('mat2');
model.material('mat2').propertyGroup.create('Enu', 'Young''s modulus and
Poisson''s ratio');
model.material('mat2').selection.set([1]);

model.physics.create('smsld', 'SolidMechanics', 'geom1');
model.physics('smsld').identifier('smsld');
model.physics('smsld').feature.create('lemm2', 'LinearElasticModel', 3);
model.physics('smsld').feature('lemm2').selection.set([1]);
model.physics('smsld').feature.create('bl1', 'BodyLoad', 3);
model.physics('smsld').feature('bl1').selection.set([1]);
model.physics('smsld').feature.create('fix1', 'Fixed', 2);
model.physics('smsld').feature('fix1').selection.set([2 9]);

model.mesh.create('mesh1', 'geom1');
model.mesh('mesh1').feature.create('ftet1', 'FreeTet');
model.mesh('mesh1').feature.create('ftet2', 'FreeTet');
model.mesh('mesh1').feature('ftet1').selection.geom('geom1', 3);
model.mesh('mesh1').feature('ftet1').selection.set([1]);
model.mesh('mesh1').feature('ftet2').selection.geom('geom1', 3);
model.mesh('mesh1').feature('ftet2').selection.set([2]);
model.mesh('mesh1').feature('ftet2').feature.create('size1', 'Size');

model.result.table.create('tbl1', 'Table');

model.material('mat1').name('Silica glass');
model.material('mat1').propertyGroup('def').set('heatcapacity',
'703[J/(kg*K)]');
model.material('mat1').propertyGroup('def').set('thermalexpansioncoefficient'
, {'0.55e-6[1/K]' '0' '0' '0' '0.55e-6[1/K]' '0' '0' '0' '0.55e-6[1/K]'});
model.material('mat1').propertyGroup('def').set('relpermittivity', {'2.09'
'0' '0' '0' '2.09' '0' '0' '0' '2.09'});
model.material('mat1').propertyGroup('def').set('thermalconductivity',
{'1.38[W/(m*K)]' '0' '0' '0' '1.38[W/(m*K)]' '0' '0' '0' '1.38[W/(m*K)]'});
model.material('mat1').propertyGroup('def').set('relpermeability', {'1' '0'
'0' '0' '1' '0' '0' '0' '1'});
model.material('mat1').propertyGroup('def').set('density', '2203[kg/m^3]');
model.material('mat1').propertyGroup('def').set('electricconductivity', {'1e-
14[S/m]' '0' '0' '0' '1e-14[S/m]' '0' '0' '0' '1e-14[S/m]'});
```

```
model.material('mat1').propertyGroup('Enu').set('youngsmodulus',
'73.1e9[Pa]');
model.material('mat1').propertyGroup('Enu').set('poissonsratio', '0.17');
model.material('mat1').propertyGroup('RefractiveIndex').set('n', '');
model.material('mat1').propertyGroup('RefractiveIndex').set('ki', '');
model.material('mat1').propertyGroup('RefractiveIndex').set('n', {'1.45' '0'
'0' '0' '1.45' '0' '0' '0' '1.45'});
model.material('mat1').propertyGroup('RefractiveIndex').set('ki', {'0' '0'
'0' '0' '0' '0' '0' '0' '0'});
model.material('mat2').name('Nylon');
model.material('mat2').propertyGroup('def').set('heatcapacity',
'1700[J/(kg*K)]');
model.material('mat2').propertyGroup('def').set('relpermittivity', {'4' '0'
'0' '0' '4' '0' '0' '0' '4'});
model.material('mat2').propertyGroup('def').set('thermalexpansioncoefficient'
, {'280e-6[1/K]' '0' '0' '0' '280e-6[1/K]' '0' '0' '0' '280e-6[1/K]'});
model.material('mat2').propertyGroup('def').set('density', '2000[kg/m^3]');
model.material('mat2').propertyGroup('def').set('thermalconductivity',
{'0.26[W/(m*K)]' '0' '0' '0' '0.26[W/(m*K)]' '0' '0' '0' '0.26[W/(m*K)]'});
model.material('mat2').propertyGroup('Enu').set('youngsmodulus', '5e8[Pa]');
model.material('mat2').propertyGroup('Enu').set('poissonsratio', '0.49');

model.physics('smsld').feature('lemm1').set('Evector_mat', 'userdef');
model.physics('smsld').feature('lemm1').set('Evector', {'2.0e11'; '2.0e11';
'2.0e11'});
model.physics('smsld').feature('lemm1').set('nuvector_mat', 'userdef');
model.physics('smsld').feature('lemm1').set('nuvector', {'0.33'; '0.33';
'0.33'});
model.physics('smsld').feature('lemm1').set('Gvector_mat', 'userdef');
model.physics('smsld').feature('lemm1').set('Gvector', {'7.52e10'; '7.52e10';
'7.52e10'});
model.physics('smsld').feature('lemm1').set('D_mat', 'userdef');
model.physics('smsld').feature('lemm1').set('D', {'2.0e11/((1+0.33)*(1-
2*0.33))*(1-0.33)'; '2.0e11/((1+0.33)*(1-2*0.33))*0.33';
'2.0e11/((1+0.33)*(1-2*0.33))*0.33'; '0'; '0'; '0'; '2.0e11/((1+0.33)*(1-
2*0.33))*0.33'; '2.0e11/((1+0.33)*(1-2*0.33))*(1-0.33)';
'2.0e11/((1+0.33)*(1-2*0.33))*0.33'; '0';  ...
'0'; '0'; '2.0e11/((1+0.33)*(1-2*0.33))*0.33'; '2.0e11/((1+0.33)*(1-
2*0.33))*0.33'; '2.0e11/((1+0.33)*(1-2*0.33))*(1-0.33)'; '0'; '0'; '0'; '0';
'0';  ...
'0'; '2.0e11/((1+0.33)*2)'; '0'; '0'; '0'; '0'; '0'; '0';
'2.0e11/((1+0.33)*2)'; '0';  ...
'0'; '0'; '0'; '0'; '0'; '2.0e11/((1+0.33)*2)'});
model.physics('smsld').feature('lemm2').set('E_mat', 'userdef');
model.physics('smsld').feature('lemm2').set('E', '5e8');
model.physics('smsld').feature('lemm2').set('nu_mat', 'userdef');
model.physics('smsld').feature('lemm2').set('nu', '.48');
model.physics('smsld').feature('lemm2').set('Evector_mat', 'userdef');
model.physics('smsld').feature('lemm2').set('Evector', {'2.0e11'; '2.0e11';
'2.0e11'});
model.physics('smsld').feature('lemm2').set('nuvector_mat', 'userdef');
model.physics('smsld').feature('lemm2').set('nuvector', {'0.33'; '0.33';
'0.33'});
model.physics('smsld').feature('lemm2').set('Gvector_mat', 'userdef');
model.physics('smsld').feature('lemm2').set('Gvector', {'7.52e10'; '7.52e10';
'7.52e10'});
model.physics('smsld').feature('lemm2').set('D_mat', 'userdef');
```

```
model.physics('smsld').feature('lemm2').set('D', {'2.0e11/((1+0.33)*(1-
2*0.33))*(1-0.33)'; '2.0e11/((1+0.33)*(1-2*0.33))*0.33';
'2.0e11/((1+0.33)*(1-2*0.33))*0.33'; '0'; '0'; '0'; '2.0e11/((1+0.33)*(1-
2*0.33))*0.33'; '2.0e11/((1+0.33)*(1-2*0.33))*(1-0.33)';
'2.0e11/((1+0.33)*(1-2*0.33))*0.33'; '0';  ...
'0'; '0'; '2.0e11/((1+0.33)*(1-2*0.33))*0.33'; '2.0e11/((1+0.33)*(1-
2*0.33))*0.33'; '2.0e11/((1+0.33)*(1-2*0.33))*(1-0.33)'; '0'; '0'; '0'; '0';
'0';  ...
'0'; '2.0e11/((1+0.33)*2)'; '0'; '0'; '0'; '0'; '0'; '0';
'2.0e11/((1+0.33)*2)'; '0';  ...
'0'; '0'; '0'; '0'; '0'; '2.0e11/((1+0.33)*2)'});
model.physics('smsld').feature('lemm2').set('rho_mat', 'userdef');
model.physics('smsld').feature('lemm2').set('rho', '2000');
model.physics('smsld').feature('bl1').set('FperVol', {'0';
'Force(x[1/m],y[1/m],z[1/m])'; '0'});

model.mesh('mesh1').feature('size').set('hauto', 2);
model.mesh('mesh1').feature('ftet2').feature('size1').set('hauto', 4);
model.mesh('mesh1').run;

model.result.table('tbl1').comments('Volume Integration 1 (smsld.RFz)');

model.coordSystem('sys1').set('mastercoordsystcomp', 'manual');

model.study.create('std1');
model.study('std1').feature.create('stat', 'Stationary');

model.sol.create('sol1');
model.sol('sol1').study('std1');
model.sol('sol1').attach('std1');
model.sol('sol1').feature.create('st1', 'StudyStep');
model.sol('sol1').feature.create('v1', 'Variables');
model.sol('sol1').feature.create('s1', 'Stationary');
model.sol('sol1').feature('s1').feature.create('fc1', 'FullyCoupled');
model.sol('sol1').feature('s1').feature.create('d1', 'Direct');

model.study('std1').feature('stat').set('initstudyhide', 'on');
model.study('std1').feature('stat').set('initsolhide', 'on');
model.study('std1').feature('stat').set('notstudyhide', 'on');
model.study('std1').feature('stat').set('notsolhide', 'on');

model.result.numerical.create('int1', 'IntVolume');
model.result.numerical('int1').selection.set([2]);
model.result.numerical('int1').set('probetag', 'none');
model.result.create('pg1', 'PlotGroup3D');
model.result('pg1').feature.create('surf1', 'Surface');
model.result('pg1').feature('surf1').feature.create('def', 'Deform');
model.result.export.create('tbl1', 'Table');

model.sol('sol1').attach('std1');
model.sol('sol1').feature('st1').name('Compile Equations: Stationary');
model.sol('sol1').feature('st1').set('studystep', 'stat');
model.sol('sol1').feature('v1').set('control', 'stat');
model.sol('sol1').feature('s1').set('control', 'stat');
model.sol('sol1').feature('s1').set('stol', '1.0E-6');
```

```
model.sol('sol1').feature('s1').feature('fc1').set('termonres', 'off');
model.sol('sol1').feature('s1').feature('fc1').set('rstep', '10.0');
model.sol('sol1').feature('s1').feature('fc1').set('initstep', '1.0');
model.sol('sol1').feature('s1').feature('fc1').set('minstep', '1.0e-4');
model.sol('sol1').feature('s1').feature('d1').set('linsolver', 'spooles');
model.sol('sol1').feature('s1').feature('d1').set('errorchk', 'off');
model.sol('sol1').runAll;

model.result.numerical('int1').set('expr', 'smsld.RFz');
model.result.numerical('int1').set('unit', 'mN');
model.result.numerical('int1').set('descr', 'Reaction force, z component');
model.result.numerical('int1').set('table', 'tbl1');
model.result.numerical('int1').setResult;
model.result('pg1').name('Stress (smsld)');
model.result('pg1').feature('surf1').set('expr', 'smsld.mises');
model.result('pg1').feature('surf1').set('unit', 'N/m^2');
model.result('pg1').feature('surf1').set('descr', 'von Mises stress');
model.result('pg1').feature('surf1').feature('def').set('scale',
'75.6551305019355');
model.result('pg1').feature('surf1').feature('def').set('scaleactive',
false);
model.result.export('tbl1').set('filename', '.\rectangleforcefinal.txt');
model.result.export('tbl1').set('table', 'tbl1');

model.result.export('tbl1').run;



quit
```

REFERENCES

[1] Bar-Cohen,Y., *Electric flex.* IEEE Spectrum, Vol. 41, pp. 29-33, 2004.

[2] Bonomo, C., Fortuna, L., Graziani, S., Mazza, D., *A circuit to model an Ionic Polymer-Metal Composite as actuator*. 2004 IEEE International Symposium on Circuits and Systems, 2004, IEEE: Vancouver, BC, Canada. p. IV-864-7.

[3] Shahinpoor, M., Kim, K., *Ionic Polymer Metal Composites-I. Fundamentals.* Smart Materials and Structures, 2001. 10: p. 819-33.

[4] Gierke, T., Munn, G., Wilson, F., *The morphology in nafion perfluorinated membrane products, as determined by wide- and small-angle x-ray studies.* Journal of Polymer Science, 1981.19(11): p. 1687-1704.

[5] Nemat-Nasser, S., and Shahram, Z., *Modeling of the electrochemomechanical response of ionic polymer-metal composites with various solvents.* Journal of Applied Physics, 2006. p. 064310.

[6] Nemat-Nasser, S., Li, J.Y., *Electromechanical response of ionic polymer-metal composites.* Journal of Applied Physics, 2000.87(7): p. 3321-31.

[7] Sadeghipour, K., Salomon R., and Neogi, S., *Development of a Novel Electrochemically Active Membrane and 'Smart' Material Based Vibration Sensor/Damper*. Smart Materials and Structures, (1992) 172-179.

[8] Bonomo, C., Fortuna, L., Giannone, P., Graziani, S., Strazzeri, S., *A model for ionic polymer metal composites as sensors.* Smart Materials and Structures, 2006.15(3): p. 749-58.

[9] Shahinpoor, M., Bar-Cohen, Y., Simpson, J.O., Smith, J., *Ionic polymer-metal composites (IPMCs) as biomimetic sensors, actuators and artificial muscles-a review.* Smart Materials and Structures, 1998.7(6): p. R15-30.

[10] Oguro, K., Kawami, Y. andTakenaka, H., *Bending of an Ion-Conducting Polymer Film-Electrode Composite by an Electric Stimulus at Low Voltage.* Trans. Journal of MicromachineSociety, Vol. 5, (1992) pp. 27-30.

[11] Kanno, R., Kurata, A., Hattori, M., Tadokoro, S., Takamori, T., Oguro, K., *Characteristics and modeling of ICPF actuator*. Proceedings of Japan-USA Symposium on Flexible Automation, 1994, Inst. Syst. Control & Inf. Eng: Kobe, Japan. p. 691-8.

[12] Bar-Cohen, Y., *Electroactive Polymers as Artificial Muscles – Realities and Challenges.* Proceedings of the 42nd AIAA, 2001. 2001-1492.

[13] Shahinpoor, M., Kim, K., *Ionic polymer-metal composites: III. Modeling and Simulation as biomimetic sensors, actuators, transducers, and artificial muscles.* Smart Materials and Structures, Vol. 13**,** 2004. 1362

[14] Nemat-Nasser, S., Wu, Y., *Tailoring the actuation of ionic polymer-metal composites.* Smart Materials and Structures, 2006.15(4): p. 909-23.

[15] Kim, K.J., Shahinpoor, M., *Ionic Polymer-Metal Composites: II. Manufacturing Techniques.* Smart Materials and Structures, 2003.12(2003): p. 65-79.

[16] Shahinpoor, M., Kim, K.J., *The effect of surface-electrode resistance on the performance of ionic polymer-metal composite (IPMC) artificial muscles.* Smart Materials and Structures, 2000.9(4): p. 543-51.

[17] Bar-Cohen,Y., *Electroactive Polymer (EAP) Actuators as Artificial Muscles*. SPIE Press, Washington, 2001.

[18] Brunetto, P., Fortuna, L., Giannone, P., Graziani, S., Strazzeri, S., *IPMCs as Vibration Sensors*. IEEE International Instrumentation and Measurement Technology Conference, 2008.

[19] Bonomo, C., Brunetto, P., Fortuna, L., Giannone, P., Graziani, S., Strazzeri, S., *A Tactile Sensor for Biomedical Applications Based on IPMCs*. IEEE Sensors Journal Vol. 8, No 8. August 2008

[20] Kruusamae, K., Brunetto, P., Graziani, S., Punning, A., Di Pasquale, G., Aabloo, A., *Self-sensing ionic polymer–metal composite actuating device with patterned surface electrodes.* Polymer International, Vol. 59, 2010.300–4.

[21] Nemat-Nasser, S., *Micromechanics of actuation of ionic polymer-metal composites.* Journal of Applied Physics, 2002.92(5): p. 2899-915.

[22] Branco, PJ, Dente, JA., *Derivation of a continuum model and its electric equivalent-circuit representation for ionic polymer-metal composite (IPMC) electromechanics.* Smart Materials and Structures, 2006. 15: (2006) 378-392.

[23] Bhat, N., Kim, W.J., *Precision force and position control of an ionic polymer metal composite.* Proceedings of IMechE, 2004. Vol. 218: p. 421-31.

[24] Lopes, B., Branco, P., *Electromechanical characterization of non-uniform charged IPMC devices*. Journal of Physics: Conference Series 127*,* 2008.

[25] Pugal, D., *Model of self-oscillating ionic polymer-metal composite bending actuator.* Thesis. Tartu University*,* 2009.

[26] Pugal, D., Aabloo, A., Kim, K., *Modeling IPMC material with dynamic surface characteristics.* Proceedings of Smart Materials, Adaptive Structures and Intelligent Systems, 2009.

[27] Martinez, M., Lumia, R., *Distributed force simulation for arbitrarily shaped IPMC actuators.* Smart Materials and Structures, Vol. 22**.**