Modified Maxwell Model for Hysteresis Compensation of Piezoelectric Stack Actuators

by

Xiaoyue Xie

Submitted to the
Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in Mechanical Engineering

at the

Massachusetts Institute of Technology

June 2015

Signature of Author: Signature redacted

Department of Mechanical Engineering
May 14, 2015

Certified by: Signature redacted

Kamal Youcef-Toumi
Professor of Mechanical Engineering
Thesis Supervisor

Accepted by: Signature redacted

Anette Hosoi
Professor of Mechanical Engineering
Undergraduate Officer

Modified Maxwell Model for Hysteresis Compensation of Piezoelectric Stack Actuators

by

Xiaoyue Xie

Submitted to the Department of Mechanical Engineering

on May 20, 2015 in Partial Fulfillment of the

Requirements for the Degree of

Bachelor of Science in Mechanical Engineering

ABSTRACT

This thesis presents new observations of the hysteresis behavior of piezoelectric stack actuators and proposes an Input-Range Dependent Maxwell Model for more accurate hysteresis compensation. Experimental studies show that the assumptions of the classical Maxwell model do not fully hold: the actuator behaves differently in the initiation stage compared to the later cycles, and the parameters of the Maxwell model are dependent on the input history. Two most prominent factors are the input range of the most recent half loop and the local extremum input at the beginning of the current half loop. To accommodate for these variations, two types of modified Maxwell model are presented: the Input-Range Dependent Maxwell Model and the Local-Extremum Dependent Maxwell Model. We further propose parameter estimation schemes for each modified model. In both models, one set of parameters is obtained for the initiation stage and another set for later cycles, and the first Maxwell spring constant is related to the input history – input range or local extremum, respectively. Further studies suggested that the linear dependence of the first spring constant on the input range is much stronger than on the local extremum. Simulations with the identified Input-Range Dependent Maxwell Model gave a maximum percentage error of 2.71%, as compared with a percentage error of 8.29% using the classical Maxwell model. This suggests that the model can accurately predict the response of a piezoelectric stack actuator and is promising for hysteresis compensation in nano-positioning applications.

Thesis Supervisor: Kamal Youcef-Toumi

Title: Professor of Mechanical Engineering

Table of Contents

List of Figures

1. Introduction

Piezoelectric stack actuators, especially unipolar piezoelectric stack actuators, with their high resolution and fast frequency response, are widely used in nanopositioning applications, such as scanning probe microscopy (SPM). However, a major limitation to piezo actuators is their nonlinear response to an applied electrical field. Two major forms of nonlinearity are hysteresis and drift[1]. If left unmodeled, hysteresis can cause poor control performance in nano-positioning applications.

Hence, it is of great importance to develop a precise model for the hysteresis behavior of a piezoelectric actuator. Ideally, the developed model should reflect the underlying physics and feature a simple structure to enable straightforward implementation of the corresponding inverse in the positioning control system. A number of studies have been published on methods of hysteresis modeling. Two prominent models are the Preisach model[2, 3, 4] and the Maxwell resistive capacitor model[1, 5, 6]. The Preisach model have been rigorously researched since its introduction. The system identification methodology has been established, and its integration into the controller has been developed[3, 4]. However, the Preisach model has several drawbacks: the two assumptions of this model, namely, the wiping-out assumption and the congruency assumption, do not fully hold[7]. Due to the nature of a double integration, error could build up in constructing the model. Deriving the inverse Preisach model is performed through numerical approximation which adds to errors and makes control implementation rather complex[8]. Furthermore, the Preisach model is a pure mathematical representation of the piezo response curve and is not based on the underlying physics of the phenomenon.

In the Maxwell resistive capacitor model, the underlying energy consumption mechanism is represented by Coulomb's friction. The method is very intuitive and is built upon the underlying physics related to the orientation and activation of various dipole domains. Although the Maxwell model shows advantages in its simplicity in parameter estimation and control implementation[9], it has not received the proper attention, especially compared to the Preisach model.

In this work we aim to expand the work of Goldfarb and present a more generalized extension of the Maxwell resistive capacitor model of hysteresis. The generalization of the model is based on our observations regarding the relationship between the input range and frictional losses. More specifically, it is shown that the elastic constants associated with the Maxwell elasto-slide elements in a half hysteresis loop are dependent on both the location of the element at the beginning of the current half loop as well as the distance traveled by the element in the most recent half loop. The phenomenon is observed to be more prominent in the initial stages of piezo displacement for both expansion and compression. This is potentially caused by the switching of voltage-specific dipole regions and delay in the response of piezo

to the change of applied voltages. Simulation of the modified Maxwell model of a piezoelectric actuator is also investigated.

2. Hysteresis in Piezoelectric Stack Actuator

2.1 Hysteresis and Its Underlying Physics

Previous research shows that the hysteresis of piezoelectric actuators is rate-independent and exhibits nonlocal memory[1]. That is, the output displacement depends on the current value and the history of the input. A physical explanation for the phenomenon was proposed by Chen and Montgomery[10]. They proposed that as the dipole domains in the piezoceramics switch under an external electrical field, the effective number of dipoles aligned in the electrical field direction changes. The delay between the domain switching and the change of electrical field, and the energy loss in the switching process, may be the primary cause of hysteresis. In addition, the number of dipoles switching directions depend on the strength of electrical field, leading to the nonlocal memory property of hysteresis.

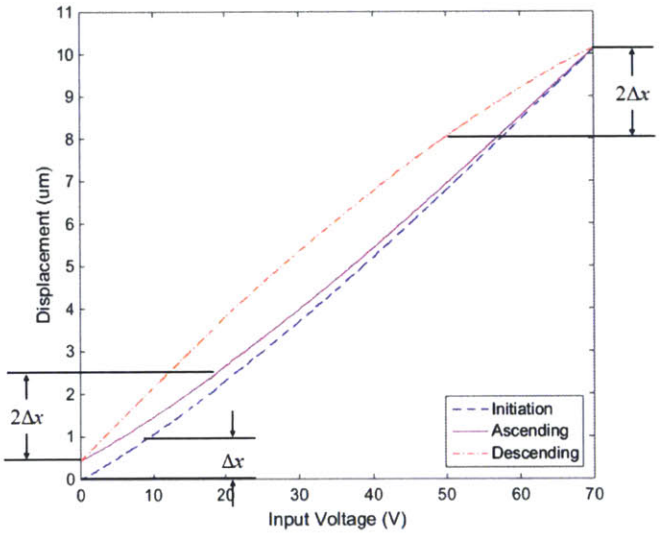


Figure 2-1: A characteristic hysteresis loop divided into three categories: the initiation stage, ascending curves, and descending curves. Classical Maxwell model assumes that the slope within a displacement of Δx in the initiation stage is the same as the slopes within a displacement of $2\Delta x$ in the corresponding sections in later cycles.

2.2 Classical Maxwell Resistive Capacitor Model

The classical Maxwell resistive capacitor model bears the above theoretical explanation of hysteresis. If the hysteresis curve is divided into three categories: the initiation stage between the start of input to the

first local maximum, the ascending curves in later cycles when the input voltage increases, and the descending curves when the input voltage decreases, as shown in Figure 2-1, the classical Maxwell model assumes that the slope within a displacement of Δx in the initiation stage is the same as the slopes within a displacement of $2\Delta x$ in the corresponding sections in later cycles; under a cyclic input voltage function of range V_{in}, the ascending and descending curves are symmetrical about the average displacement of both directions under the input of $V_{in}/2$; and for a certain change between the current input and the beginning of the current ascending or descending curve, the shape of the hysteresis loop remains the same regardless of the input history or the current input.

2.3 New Observations of the Hysteresis Behavior

We studied the hysteresis behavior of a PI P-885.51 unipolar piezoelectric stack actuator under different input voltage conditions, and make the following observations. Our data show that these three assumptions of the classical Maxwell model described above do not fully hold.

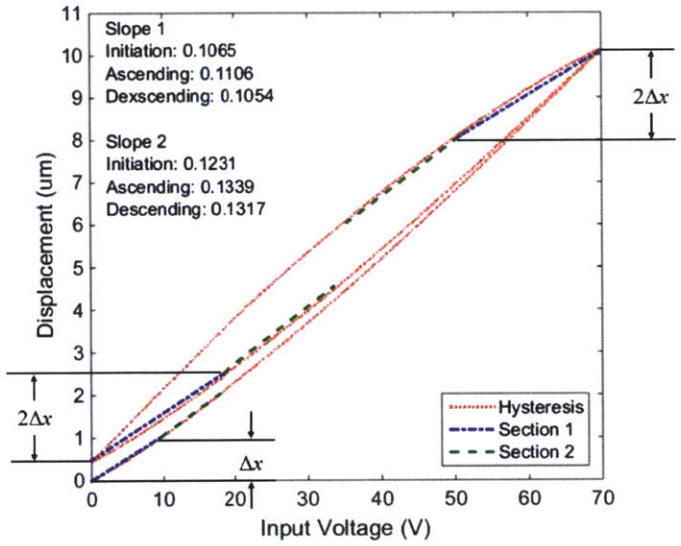


Figure 2-2: Slopes in the initiation stage and later cycles under an input of triangular wave between 0-70V. The slopes in the initiation stage are different from corresponding sections in the ascending and descending curves.

Firstly, hysteresis behavior varies in the three stages of the input described above. As shown in Figure 2-2, under a triangular input, if we plot the displacement of the piezoelectric actuator versus the input voltage, the hysteresis curve within Δx in the initiation stage has a different slope from the corresponding sections within $2\Delta x$ in later cycles.

Secondly, while the hysteresis behavior under a constant input range is the same within either the ascending or descending curves, the two sets of curves are asymmetrical. To better visualize the difference in the piezo response to input changes, we created plots by rotating the descending curves by 180° and shifting both curves to the origin, as shown in Figure 2-3, which allows us to compare the absolute change in displacement in response to the same absolute change in voltage. It is observed that the beginning of the ascending curves is steeper than the beginning of the descending curves, while the end of the ascending curves has a smaller slope than the end of the descending curves. In other words, when the input starts decreasing from its maximum, it takes more change in voltage to achieve the same change in displacement as when the input starts increasing from its minimum; whereas at the end of the descending curves, as the decreasing input approaches the minimum value, the same change in voltage results in a larger change in displacement in comparison with the end of increasing voltage.

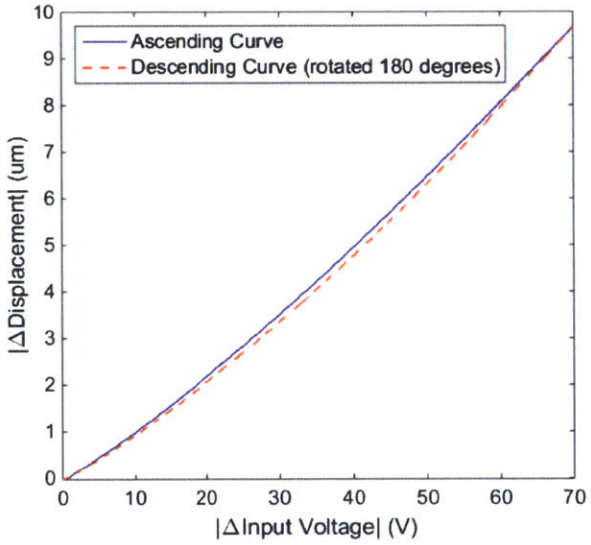
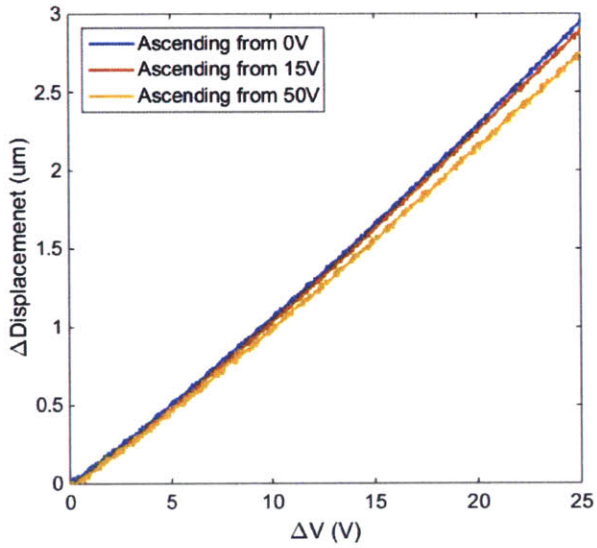
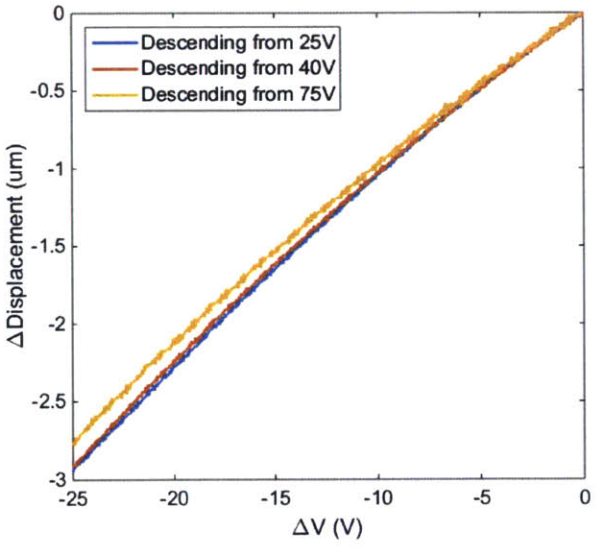


Figure 2-3: The absolute change in displacement versus the absolute change in voltage in the ascending and descending curves. The descending curves are rotated by 180° and both sets of curves are shifted to the origin.

Finally, the shape of the hysteresis loop in later cycles are not independent of the input history. On the one hand, the local extremum at which the input voltage starts increasing or decreasing also affects the shape of the half loop. Figure 2-4(a) and (b) compare the change in displacement when the input voltage increases or decreases from different initial points, while the input range remains at 25V. When the increase of input voltage starts at a smaller local minimum, the hysteresis curve is steeper, and the same change in input voltage leads to a smaller change in displacement. Similarly, a descending curve starting at a higher local maximum needs less change in input to achieve the same output displacement.



(a)



(b)

Figure 2-4: Change in displacement in response to change in the input voltage when the input range is 25V for (a) ascending curves with the initial inputs of 0V, 15V, and 50V; (b) descending curves when the input starts decreasing at 25V, 40V, and 75V.

On the other hand, the slopes and, therefore, the Maxwell spring constants of the current ascending or descending curve are affected by the input range of the most recent descending or ascending curve, respectively. We calculated the Maxwell parameters of hysteresis loops under triangular inputs of different ranges: 10V, 15V, 20V, 25V, 30V, 35V, and 40V, with the starting points at 0V for all ascending curves and at 75V for all descending curves, as shown in Figure 2-5. The Δx in all cases were maintained at 0.1366μm, or 0.9% of the maximum displacement, such that 20 Maxwell elements were activated in the ascending curves from 0-40V. The Maxwell slopes of the first three elements are shown

in Figure 2-6(a) to (c), and the calculated Maxwell spring constants are shown in Figure 2-6(d) to (f). For either the ascending or descending curves, the impact of the input range on the curve slopes is the most prominent in the initial stage, as both the Maxwell slope and spring constant increases linearly with the input range. The dependence of later Maxwell slopes spring constants is significantly weaker. For the outlier in Figure 2-6(f), we speculate that it is because only three Maxwell elements moved in the case and the fourth section was relatively short, thus the Maxwell slope for the fourth section was not accurate. Since the third spring constant is the difference between the third and the fourth slopes, the spring constant was also not representative of the actual value.

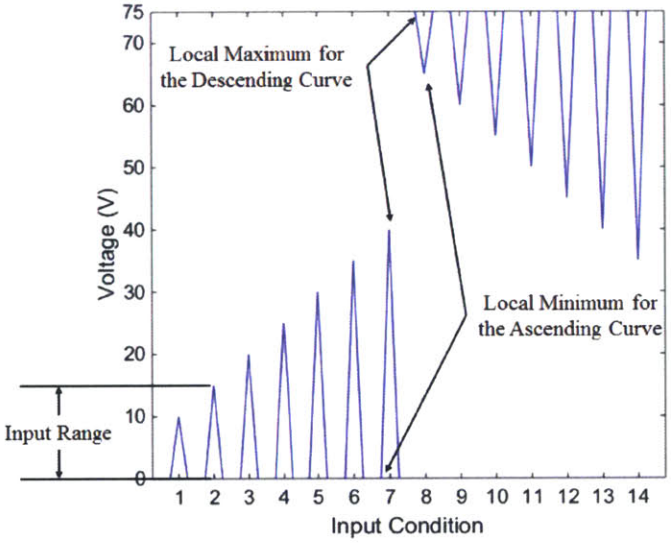


Figure 2-5: Input signals with different ranges of 10V, 15V, 20V, 25V, 30V, 35V, and 40V with the initial voltage of each cycle at 0V and 75V. The input frequency was 2Hz for all conditions.

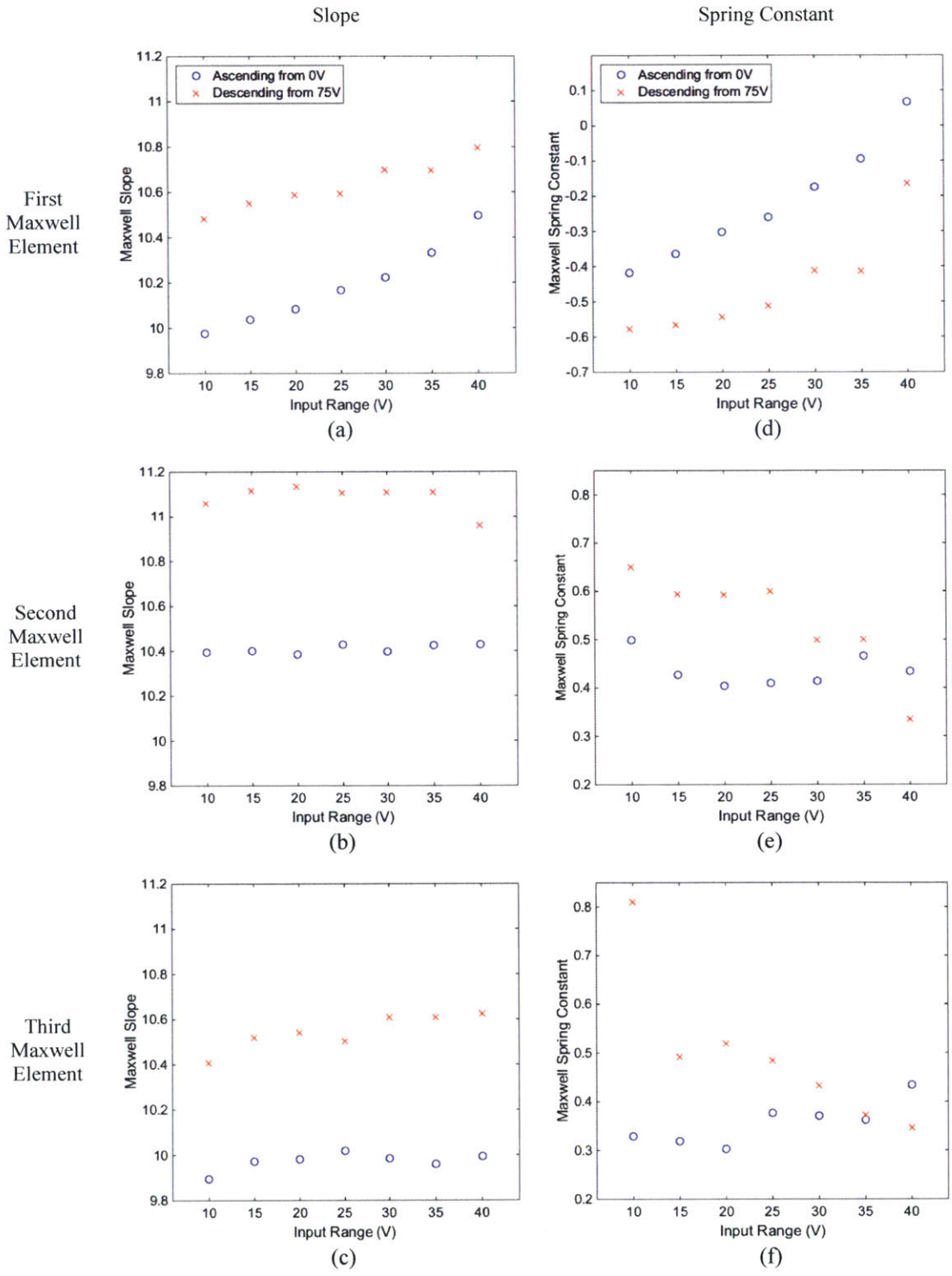


Figure 2-6: Maxwell parameters of ascending curves starting at 0V and descending curves starting at 75V under different input ranges. (a)-(c): Maxwell slopes of the first three elements; (d)-(f): Spring constants of the first three elements.

3. Modified Maxwell Model and Identification

3.1 Modified Maxwell Model

To accommodate for the new observations, the Maxwell Model described in Goldfarb's paper[1] is generalized as described in the following. As shown in the mechanical schematic in Figure 3-1, the modified Maxwell model consists of n elasto-slide elements connected in parallel, each of the Maxwell element analogous to a Maxwell resistive capacitor in the electrical domain. The total force, F, is analogous to the input voltage, and the displacement, x, corresponds to the total displacement of the piezoelectric stack actuator.

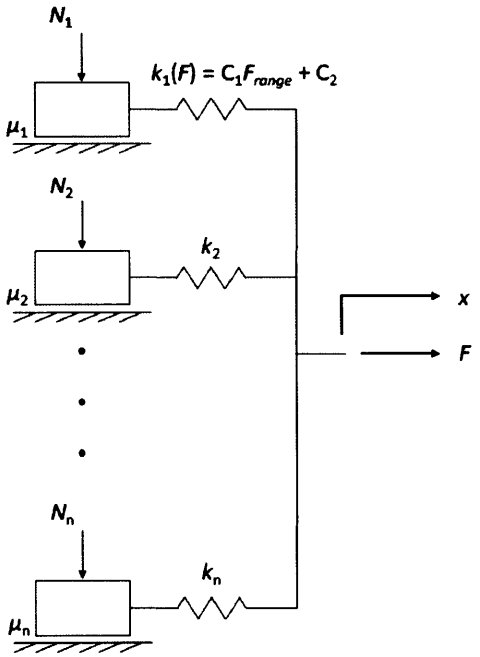


Figure 3-1: Schematic of a modified Maxwell model with N elasto-slide elements in parallel. The spring constant of the first Maxwell element is linear to the input force, which is analogous to the input voltage in the electrical domain.

Each Maxwell element is composed of a massless block connected in series with a spring. For the i^{th} Maxwell element,, as shown in Figure 3-2, a normal force, N_i, is applied to the block, the coefficient of static friction between the block and the surface is μ_i, the location of the block is x_{bi}, and the spring constant is k_i. The constitutive relationship of each Maxwell element can be expressed as:

$$f_i = \mu_i N_i \tag{1}$$

$$x_{si} = x - x_{bi} \tag{2}$$

$$x_{ei} = f_i/k_i \tag{3}$$

$$F_i = \begin{cases} k_i x_{si} & \text{if } |x_{si}| < x_{ei} \\ f_i sgn(\dot{x}) \text{ and } |x_{si}| = x_{ei} & \text{else} \end{cases}, \tag{4}$$

where f_i is the breakaway force, x_{si} is the spring displacement, x_{ei} is elongation of the spring when the breakaway force is reached. Therefore, under an input force of F, the functions governing the Maxwell model is given by:

$$F = \sum_{i=1}^{n} F_i\,, \tag{5}$$

where F_i is the output force of the ith Maxwell element.

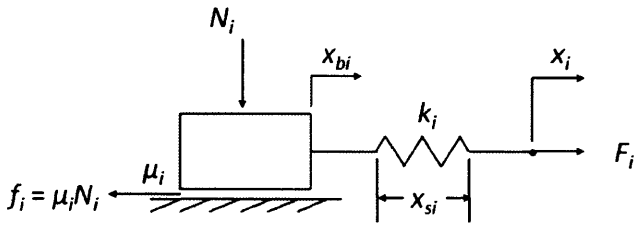


Figure 3-2: Schematic of the i^{th} element in the Maxwell model.

3.1.1 Two Sets of Maxwell Parameters

Our first modification to the classical Maxwell model is an introduction of two separate sets of Maxwell parameters, the breakaway forces and spring constants, for each element: in the initiation stage, $\{f, k\}_{initiation}$, and in later cycles, $\{f, k\}_{cycle}$. This is based on our observation that the piezoelectric actuator behaves differently between the initiation stage and the later cycles. For the initiation stage, since the input voltage always starts from 0, the hysteresis curve for a piezoelectric actuator would always follow the same path, thus $\{f, k\}_{\text{initiation}}$ are constants, giving

$$\{f, k\}_{initiation,i} = \{f_{0,i}, k_{0,i}\}, \qquad i = 1,2 \ldots n_0, \tag{6}$$

where $f_{0,i}$, $k_{0,i}$ and n_0 are, respectively, the breakaway force and spring constant of the ith Maxwell element, and the number of blocks moved in the initiation stage.

3.1.2 Influence of the Input History

Secondly, as shown in Figure 2-4 and Figure 2-6, the shape of the later cycles are affected by the input history, especially the input range of the most recent half loop and the voltage at the beginning of the

curve. Therefore, we propose and compare two methods to account for the influence of the input history on the hysteresis behavior.

(a) Model I: Input-Range Dependent Maxwell Model

In the first method, we select the spring constant of the first elasto-slide element to be a linear function of the change in input in the most recent half loop. Comparing Figure 2-6(d) with (e) and (f), the effect of the input range on Maxwell parameters is the most prominent in the beginning stages of the ascending or descending curves, in which the spring constants increases linearly with the input range. Therefore, we define F_{range} as absolute change in force in the most recent section prior to the current travel section, then the spring constant of the first Maxwell element, $k_{c,1}$, can be expressed as

$$k_{c,1}(F_{range}) = C_1 F_{Range} + C_2 \,, \tag{7}$$

where C_1 and C_2 are constants to be determined in the model construction. For the construction of the model, we assume that the deformation of the spring at which the static friction is balanced, Δx, remains the same regardless of input conditions. Therefore, the static friction coefficient of the first Maxwell element, $\mu_{c,1}$, is also a dependent on the most recent change of force, F_{range}. It follows that the breakaway force of the first block, $f_{c,1}$, is also a function of F_{range}, giving

$$f_{c,1}(F_{range}) = k_{c,1}\Delta x = (C_1 F_{range} + C_2)\Delta x \,, \tag{8}$$

where Δx is the displacement of the first spring at which the spring force balances the breakaway force.

The parameters of the remaining Maxwell elements are constants, similar to the classical Maxwell model.

(b) Model II: Local-Extremum Dependent Maxwell Model

The second method relates the Maxwell parameters of the first elasto-slide element to the local minimum or maximum input at the beginning of the current travel section. Define F_{extre} as the local extremum input at the beginning of the current half loop, the spring constant of the first elasto-slide element can be expressed as

$$k_{c,1}(F_{extre}) = D_1 F_{extre} + D_2 \,, \tag{9}$$

where D_1 and D_2 are constants to be determined in the model construction. Similar to the method in part (a), we assume that Δx remains constant. Thus, the relationship between the breakaway force, $f_{c,1}$, and the local extremum, F_{extre}, is given by

$$f_{c,1}(F_{extre}) = k_{c,1}\Delta x = (D_1 F_{extre} + D_2)\Delta x \,, \tag{10}$$

where Δx is the displacement of the first spring at which the spring force balances the breakaway force.

For either methods of accounting for the input history, the Maxwell parameters in later cycles can be expressed as

$$\{f,k\}_{cycle,i} = \begin{cases} \{f_{c,1}(F), k_{c,1}(F)\} & i = 1 \\ \{f_{c,i}, k_{c,i}\} & i = 2,3 \ldots n_c \end{cases}, \quad (11)$$

where $f_{c,i}$ and $k_{c,i}$ are the breakaway force and spring constant of the ith elasto-slide element in later cycles, F is either the range or local extremum of the input, and n_c is the number of blocks moved in later cycles.

It is worth noting that n_0 and n_c are not necessarily the same. In fact, if we assume the spring displacement, x_{ei} (i = 1, 2, …, min(n_i, n_c)), at which point the breakaway force of each block is balanced by the spring force, remains constant for all Maxwell elements over time, then $n_i = n_c$ only if $F_{max,i} = 1/2$ F_{max}, where $F_{max,i}$ is the maximum input in the initiation stage and F_{max} is the absolute maximum input. However, since Maxwell parameters in the initiation stage and later cycles are obtained separately, this will not be a problem in model construction.

3.2 Model Identification

Methods to identify the parameters in the two types of modified Maxwell resistive capacitor model for a piezoelectric stack actuator with a maximum input voltage of V_{max} and a maximum travel of X_{max} is described as the following.

3.2.1 Experimental Setup

The experimental procedure for both types of Maxwell model is the same.

- Set up the piezoelectric stack actuator with one end fixed and the other end free to move, with a laser interferometer pointing at the free end. A sample setup is shown in Figure 4-1.
- Excite the piezoelectric actuator with M triangular input voltage functions, $v(t)$, with the same frequency $2\pi/T_0$, where T_0 is the duration of one cycle, and ranges of 0V-v_i (i = 1, 2, ..., M), where $0 < v_1 < v_2 < \ldots < v_M < V_{max}$.
- Measure the displacement of the free end under each input condition with the laser interferometer.

3.2.2 Parameters in the Initiation Stage

To obtain the Maxwell parameters in the initiation stage, $\{f, k\}_{initiation}$, we create a *v*-*x* plot combining the hysteresis curves in the initiation stage from all tests, with the input voltage as the vertical axis and displacement as the horizontal axis, as shown in Figure 3-2.

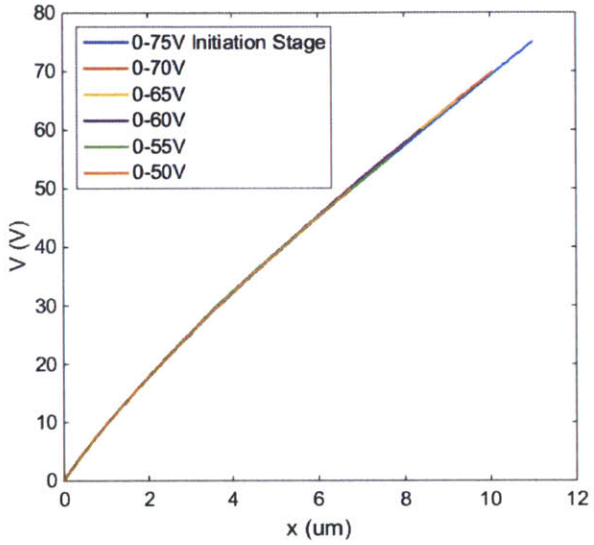


Figure 3-3: *v*-*x* plots for obtaining Maxwell parameters in the initiation stage.

Divide the interval between origin and maximum x to n_0 equally-spaced segments, giving the length of each segment $\Delta x = [x(V_{max}) - x(0)]/n_0$. Identify the slope of each segment, $s_{0,i}$ (i = 1, 2, …, n_0), by finding the best linear fit lines to the curve. Use the identified slopes to calculate the spring constants, $k_{0,i}$ (i=1,2,…, n_0), and the breakaway forces, $f_{0,i}$ (i = 1, 2, …, n_0), in the initiation stage as

$$\begin{bmatrix} k_{0,1} \\ k_{0,2} \\ \vdots \\ k_{0,n_0} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & & 1 & 1 \\ 0 & 1 & 1 & \cdots & 1 & 1 \\ 0 & 0 & 1 & & 1 & 1 \\ & \vdots & & \ddots & \vdots & \\ 0 & 0 & 0 & & 1 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} s_{0,1} \\ s_{0,2} \\ \vdots \\ s_{0,n_0} \end{bmatrix} \tag{12}$$

$$\text{and} \begin{bmatrix} f_{0,1} \\ f_{0,2} \\ \vdots \\ f_{0,n_0} \end{bmatrix} = \Delta x \begin{bmatrix} 1 \\ 2 \\ \vdots \\ n_0 \end{bmatrix}^T \begin{bmatrix} k_{0,1} \\ k_{0,2} \\ \vdots \\ k_{0,n_0} \end{bmatrix}. \tag{13}$$

3.2.3 Parameters in Later Cycles

(a) Input-Range Dependent Maxwell Model

(1) To find the Maxwell parameters in later cycles using the Input-Range Dependent Maxwell Model, we create a *v*-*x* plot combining the ascending and descending curves in each test, giving a

total of M plots, as shown in Figure 3-4. The descending curves are rotated by 180° and both the ascending and descending curves are shifted such that their starting points align at the origin.

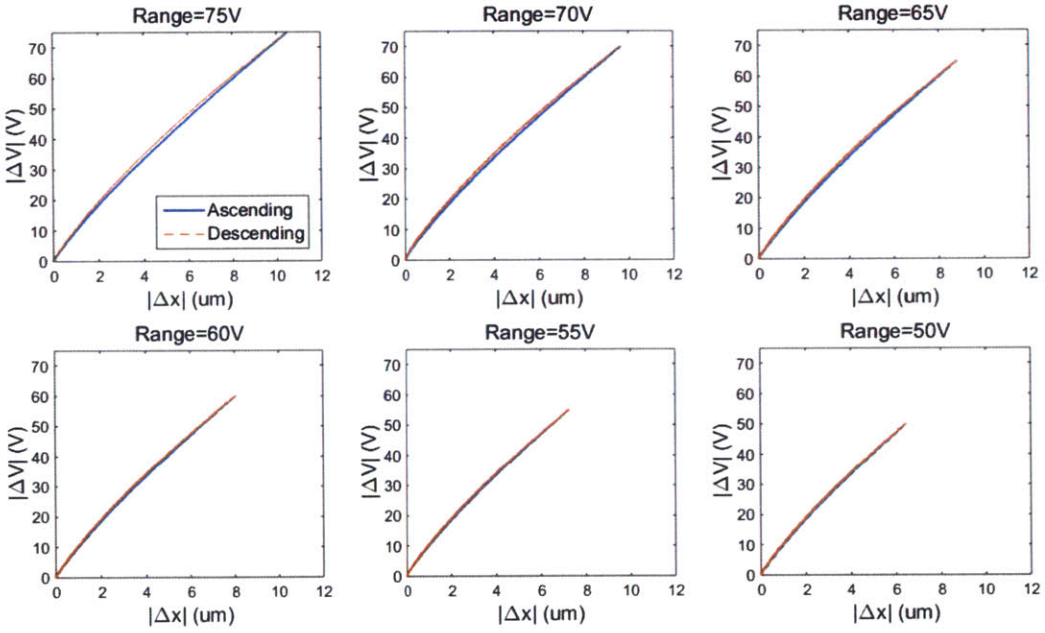


Figure 3-4: v-x plots for construction of the Input-Range Dependent Maxwell Model. Each plot includes the ascending and descending curves in each test.

(2) Divide the interval between origin and maximum x of each plot into segments of $2\Delta x$. Note that the number of segments for each plot, n_i' (i = 1, 2… M), may be different for each plot, the largest number being n_c. Identify the slope of each segment, $s_{j,i}$ (j = 1, 2…n_i', i = 1, 2… M), by finding the best linear fit lines to the data from both the ascending and descending curves. Since the spring constant of the first Maxwell element only affects the slope of the first section, all other slopes are constant regardless of the travel range. Define the slope of the first segment as a function of input voltage,

$$s_{c,1}(F_{range,i}) = s_{1,i} \qquad i = 1,2 \dots M\,, \tag{14}$$

with $$F_{range,i} = v_i \qquad i = 1,2 \dots M\,, \tag{15}$$

where v_i is the range of the ith input. Therefore, the slopes of the hysteresis curve in later cycles can be expressed as

$$s_{c,i} = \begin{cases} s_{c,1}(F_{range}) \\ \frac{\sum_{j=2}^{n_i'} s_{j,i}}{n_i'} \end{cases} \quad i = 1,2 \dots M \quad . \tag{16}$$

The spring constants in later cycles can then be calculated by

$$\begin{bmatrix} k_{c,1}(v_i) \\ k_{c,2} \\ k_{c,3} \\ \vdots \\ k_{c,n_c} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & & 1 & 1 \\ 0 & 1 & 1 & \cdots & 1 & 1 \\ 0 & 0 & 1 & & 1 & 1 \\ & \vdots & & \ddots & \vdots & \\ 0 & 0 & 0 & \cdots & 1 & 1 \\ 0 & 0 & 0 & & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} s_{c,1}(v_i) \\ s_{c,2} \\ s_{c,3} \\ \vdots \\ s_{c,n_c} \end{bmatrix}, \quad i = 1,2, \dots M. \tag{17}$$

Recall that we select the first spring constant as a linear function of the input range of the most recent half loop, $k_{c,1}(F_{range}) = C_1 F_{range} + C_2$. Therefore, we can find C_1 and C_2 by plotting $k_{c,1}$ versus V and obtaining the best linear fit line,

$$k_{c,1}(F_{range}) = K_{0,range} + C F_{range} \, , \tag{18}$$

where K_0 is the offset and C is the slope of the line. The breakaway forces in later cycles can then be calculated by

$$\begin{bmatrix} f_{c,1}(F_{range}) \\ f_{c,2} \\ \vdots \\ f_{c,n_c} \end{bmatrix} = \Delta x \begin{bmatrix} 1 \\ 2 \\ \vdots \\ n_c \end{bmatrix}^T \begin{bmatrix} k_{c,1}(F_{range}) \\ k_{c,2} \\ \vdots \\ k_{c,n_c} \end{bmatrix}. \tag{19}$$

(b) Local-Extremum Dependent Maxwell Model

(1) To identify the Local-Extremum Dependent Maxwell Model, we divide the hysteresis loops in later cycles into ascending and descending curves. Create a plot containing the ascending curves from all tests, while keeping descending curves from each test in a separate graph, as shown in Figure 3-5, giving a total of M+1 plots. The descending curves are rotated by 180° and both ascending and descending curves are shifted to the origin for better visualization.

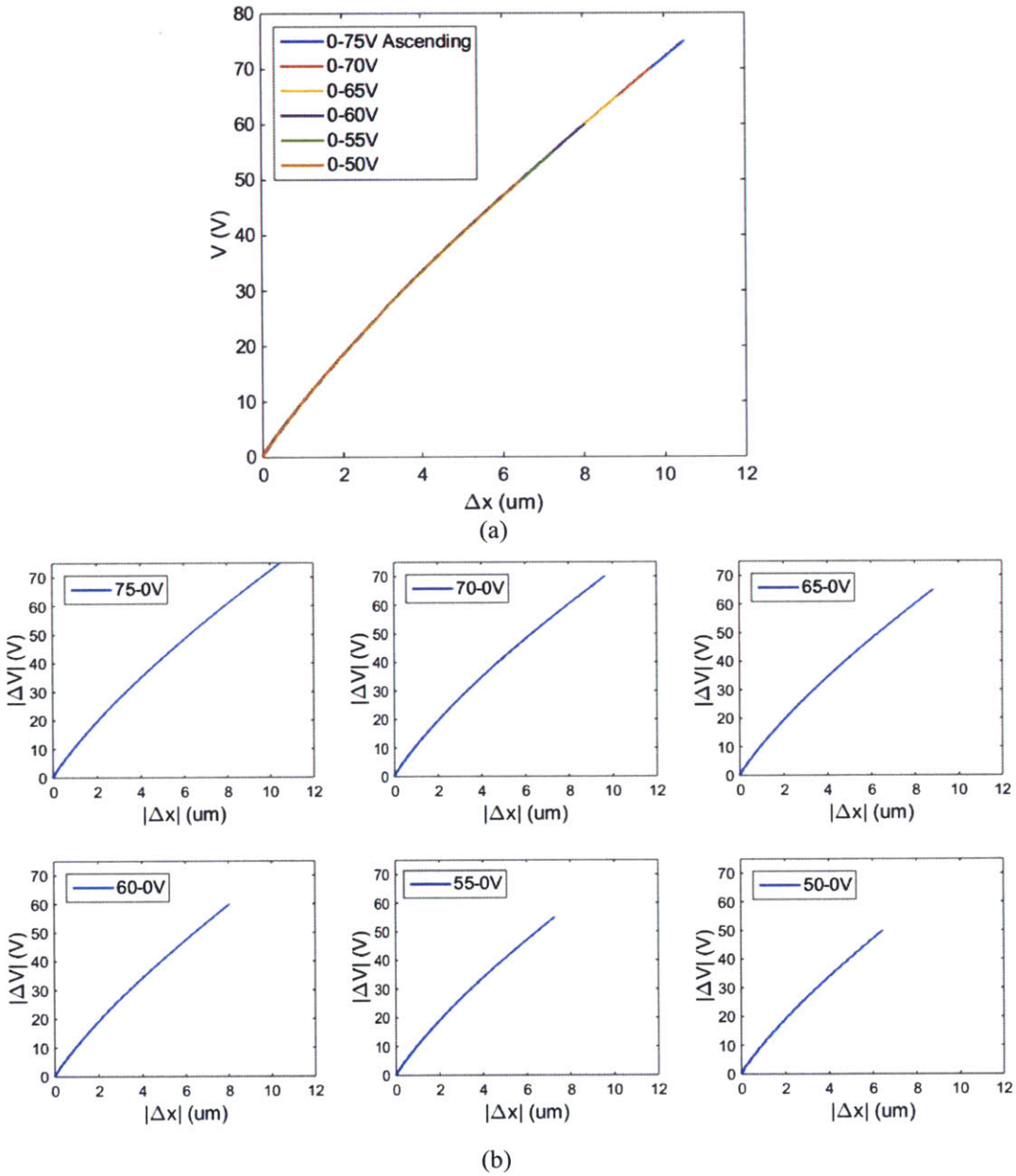


Figure 3-5: *v-x* plots for construction of the Local-Extremum Dependent Maxwell Model. Divide the hysteresis curves in later cycles into two categories: (a) one plot combining the ascending curves from all tests, and (b) a set of plots of descending curves from each test.

(2) Divide the interval between origin and maximum x of each plot into segments of $2\Delta x$. Note that the number of segments for each plot, n_i' (i = 1, 2… M+1), may be different for each plot, the largest number being n_c. Identify the slope of each segment, $s_{j,i}$ (j = 1, 2…n_i', i = 1, 2… M+1), by finding the best linear fit lines to the data from all the curves in the plot. Define the slope of the first segment as a function of input voltage,

$$s_{c,1}(F_{extre}) = \begin{cases} s_{1,1} & F_{extre} = 0 \\ s_{1,i} & F_{extre} = v_i,\ i = 1,2 \dots M \end{cases}, \tag{20}$$

where v_i is the maximum input of the ith test. Therefore, the slopes of the hysteresis curve in later cycles can be expressed as

$$s_{c,i} = \begin{cases} s_{c,1}(F_{extre}) \\ \frac{\sum_{j=2}^{n_i'} s_{j,i}}{n_i'} \end{cases} \quad i = 1,2 \dots M+1 \quad . \tag{21}$$

The spring constants in later cycles can then be calculated by

$$\begin{bmatrix} k_{c,1}(v_i) \\ k_{c,2} \\ k_{c,3} \\ \vdots \\ k_{c,n_c} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & & 1 & 1 \\ 0 & 1 & 1 & \cdots & 1 & 1 \\ 0 & 0 & 1 & & 1 & 1 \\ & \vdots & & \ddots & \vdots & \\ 0 & 0 & 0 & \cdots & 1 & 1 \\ 0 & 0 & 0 & & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} s_{c,1}(v_i) \\ s_{c,2} \\ s_{c,3} \\ \vdots \\ s_{c,n_c} \end{bmatrix}, \qquad i = 1,2, \dots M. \tag{22}$$

Recall that we select the first spring constant as a linear function of the local extremum at the beginning of the current ascending or descending curve, $k_{c,1}(F_{extre}) = D_1 F_{extre} + D_2$. Therefore, we can find D_1 and D_2 by plotting $k_{c,1}$ versus V and obtaining the best linear fit line,

$$k_{c,1}(F_{extre}) = K_{0,extre} + C_{extre} F_{extre}\ , \tag{23}$$

where K_0 is the offset and C is the slope of the line. The breakaway forces in later cycles can then be calculated by

$$\begin{bmatrix} f_{c,1}(F_{extre}) \\ f_{c,2} \\ \vdots \\ f_{c,n_c} \end{bmatrix} = \Delta x \begin{bmatrix} 1 \\ 2 \\ \vdots \\ n_c \end{bmatrix}^T \begin{bmatrix} k_{c,1}(F_{extre}) \\ k_{c,2} \\ \vdots \\ k_{c,n_c} \end{bmatrix}. \tag{24}$$

4. Experimental Study of A Piezoelectric Actuator

To validate and compare the Input-Range Dependent and the Local-Extremum Dependent Maxwell Models, a P-885.51 piezoelectric stack actuator manufactured by Physik Instrument (PI) was used for experimental studies by examining the simulation results from the modified Maxwell models. The manufacturer specifications give a nominal displacement of the piezoelectric actuator is 15um under an input of 100V with an electrical capacitance of 1.5uF and a resonance frequency of 70kHz[11].

4.1 Experimental Setup

The piezoelectric stack actuator was attached to a fixed stage on one end with the other end free to move, as shown in Figure 4-1. The laser interferometer (SIOS Meßtechnik GmbH SP-S 120 with a working distance of 50mm) pointed at a piece of aluminum foil attached to the free end such that its high reflectivity could ensure precise measurements of the displacement. The displacement of the free end was measured at a sampling rate of 5000Hz. The actuator was excited with a set of triangular voltage functions, as shown in Figure 4-2, with the input ranging from 0-10V, 0-15V, ... 0-75V, and 75-65V, 75-60V, ... 75-5V, giving 4 sets of data for each of the input range of 10V, 15V, ... 75V. The input frequency was 2Hz with 10 cycles per input voltage. The experiment under each input condition was repeated 3 times.

The data from all input conditions were used for the construction of the two types of modified Maxwell model using the methods described above. The data under the input of 0-65V were compared with the simulation results from the obtained parameters in order to validate and evaluate the models.

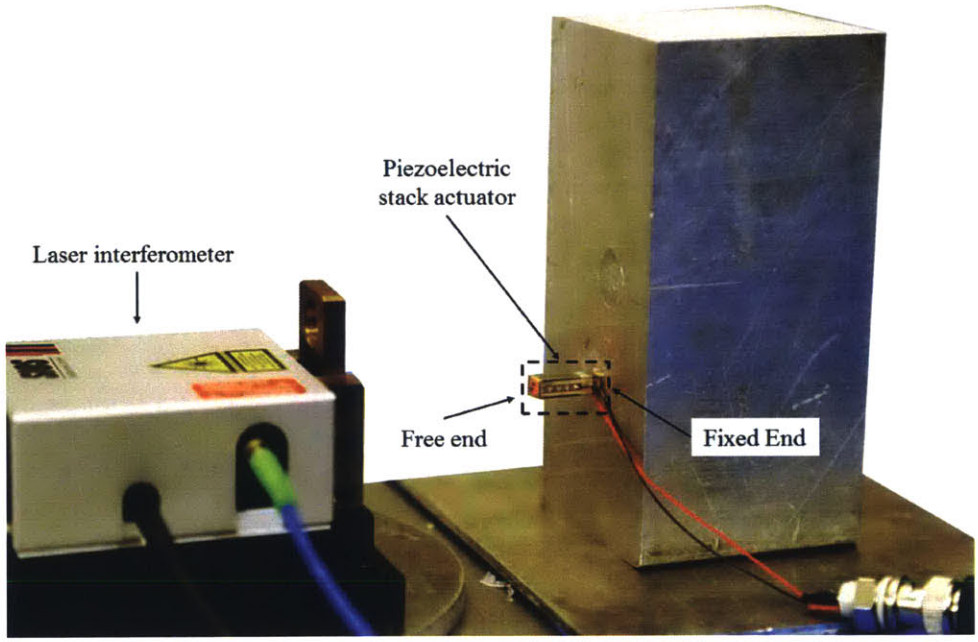


Figure 4-1: Experimental setup. The piezoelectric actuator was fixed on one end with the other end free to move. A laser interferometer pointed at a piece of aluminum foil attached at the free end to measure the displacement.

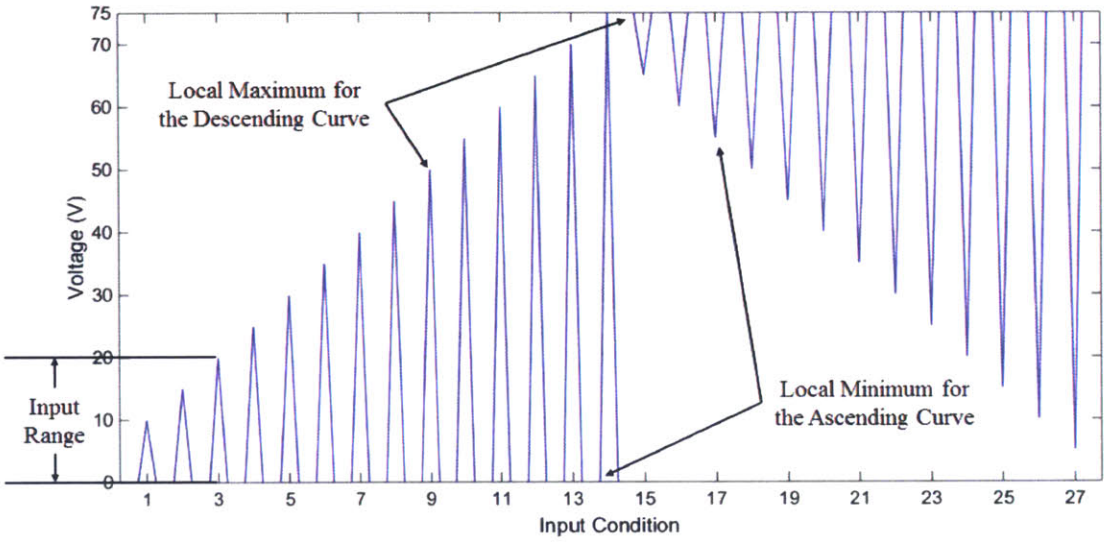


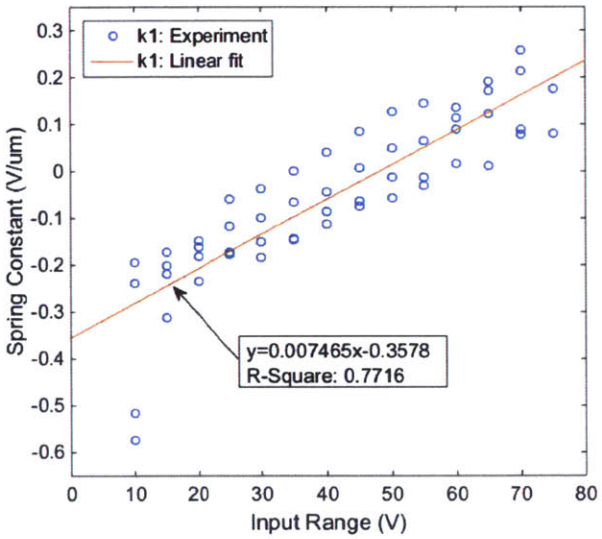
Figure 4-2: Input functions for model construction. The input ranges varied from 10V to 75V, with the initial input point at 0V or 75V. Each test consisted of 10 cycles of the same input condition and was repeated 3 times in the experiment.

4.2 Model Construction and Results

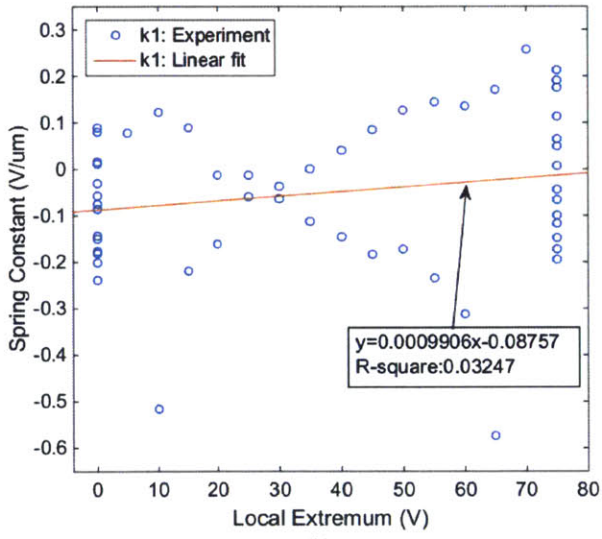
For each type of modified Maxwell model, the model parameters were obtained with the methods described above with the numbers of elements in the initiation stages, $n_0 = 40$, with corresponding $\Delta x = 0.2947\mu m$, or 1.96% of the nominal maximum travel. The spring constants are listed in Appendix A and B. Note that the Maxwell parameters in the initiation stage as well as all but the first element in later cycles are the same for both models.

To validate the influence of the input history on the hysteresis behavior, we compare the Maxwell parameters of the first elasto-slide element obtained through the Input-Range Dependent Maxwell Model and the Local-Extremum Dependent Maxwell Model, shown in Figure 4-3(a) and (b). Relating the first spring constant to the input range of the most recent half loop, we got the function $k_{c,1}(F_{range}) = 7.465 \times 10^{-3} F_{range} - 0.3578$, and the r-square between the linear fit and the experimental data is 0.7716, whereas the dependence of the first spring constant on the local extremum at the beginning of the current half loop gives $k_{c,1}(F_{extre}) = 9.906 \times 10^{-4} F_{extre} - 0.08757$, with an r-square of 0.02972.

With a much higher r-square value, the linear dependence of $k_{c,1}$ on the input range of the most recent half loop, F_{range}, is much stronger than the linear dependence on the local extremum, F_{extre}. Therefore, the Input-Range Dependent Maxwell Model is a promising choice of the modified Maxwell model and was applied in simulation for validation in the following.



(a)



(b)

Figure 4-3: Spring constants of the first elasto-slide element and their dependence on the input history. (a) Applying the Input-Range Dependent Maxwell Model, the first spring constant is a function of the input range of the most recent half loop, $k_{c,1}(F_{range}) = 7.465 \times 10^{-3} F_{range} - 0.3578$, and the r-square of the linear fit is 0.7716. (b) Applying the Local Extremum Dependent Maxwell Model, the first spring constant is given by $k_{c,1}(F_{extre}) = 9.526 \times 10^{-4} F_{extre} - 0.08523$, with an r-square of 0.02972.

4.3 Simulation Results

Since the initial sections of the hysteresis curve show a much stronger dependence on the input range, the simulations in the following sections were constructed with the Input-Range Dependent Maxwell Model. Maintaining the same $\Delta x = 0.2947 \mu m$ as in the model construction, we constructed simulations of the actuator displacements in response to triangular inputs of 0-65V, as shown in Figure X, with the

parameters obtained through the Input-Range Dependent Maxwell Model. The comparison between simulation results and the experimental data, the absolute error, and the percentage error are shown in Figure 4-5(a)-(d). The largest absolute error is 0.25μm, and the largest percentage error, which is defined as the ratio between the absolute error and the maximum travel under the input, is 2.71%. Both the largest percentage error occurred at the input voltage of 37.7V, when the actual displacement of the piezoelectric actuator was 6.28μm.

4.4 Comparison with the Classical Maxwell Model

To validate that the Input-Range Dependent Maxwell Model indeed improves the accuracy of hysteresis modeling, a classical Maxwell model was constructed and the simulation results were compared. The classical Maxwell model with 40 elasto-slide elements was identified from the hysteresis response under the triangular input of 0-75V. The Maxwell parameters, as listed in Appendix C, were obtained with data from both the initiation stage and later cycles, assuming that they remain constant through time.

With the identified classical Maxwell mode, a simulation of the hysteresis response under a triangular input of 0-65V was constructed. The simulation results and errors are shown in Figure 4-5(e)-(h). Comparing the simulation results with the modified model, or Figure 4-4(a) and (e), it is obvious that simulation with the Input-Range Dependent Maxwell Model follows the actual hysteresis loop more accurately. The maximum absolute error using the classical Maxwell model is 0.77μm and a maximum percentage error is 8.29%, as compared with 0.25μm and 2.71% using the modified Maxwell model.

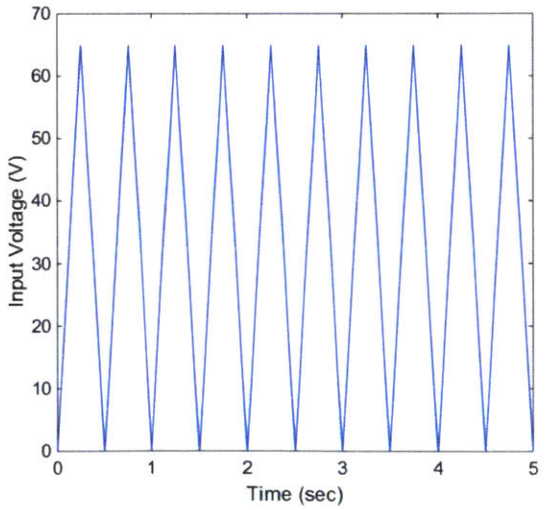
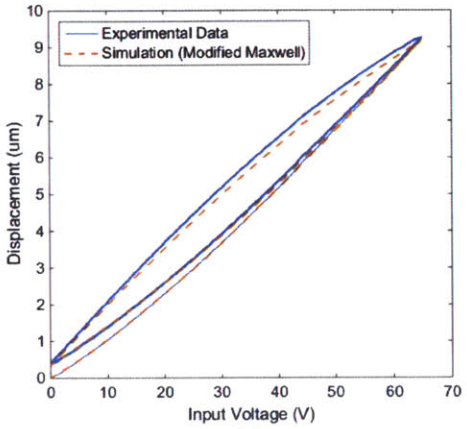


Figure 4-4: The triangular input of 0-65V at 2Hz used for model validation.

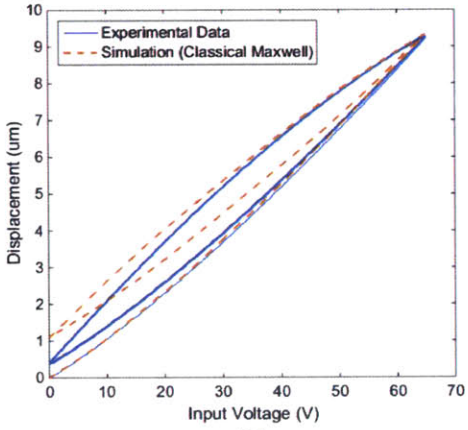
Input-Range Dependent Maxwell Model

Classical Maxwell Model

x-v

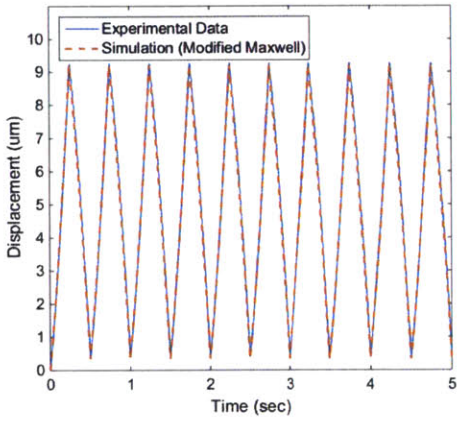


(a)

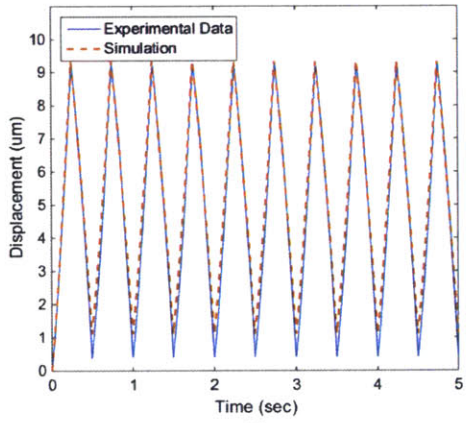


(e)

x-t

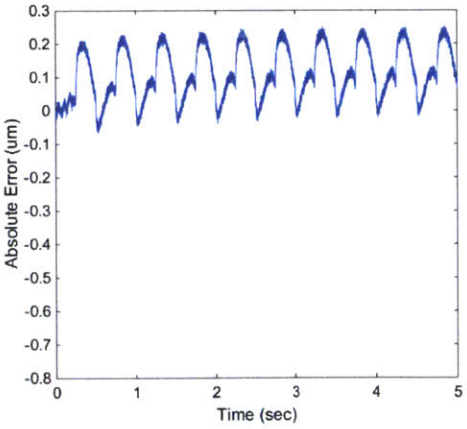


(b)

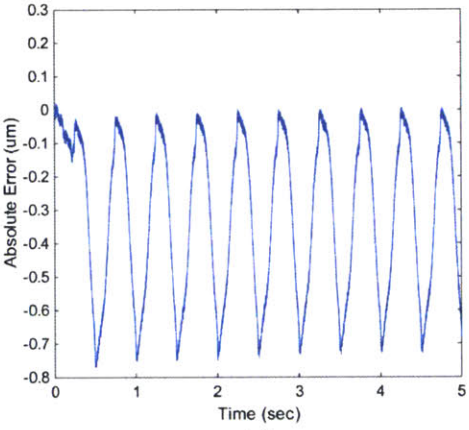


(f)

Absolute Error

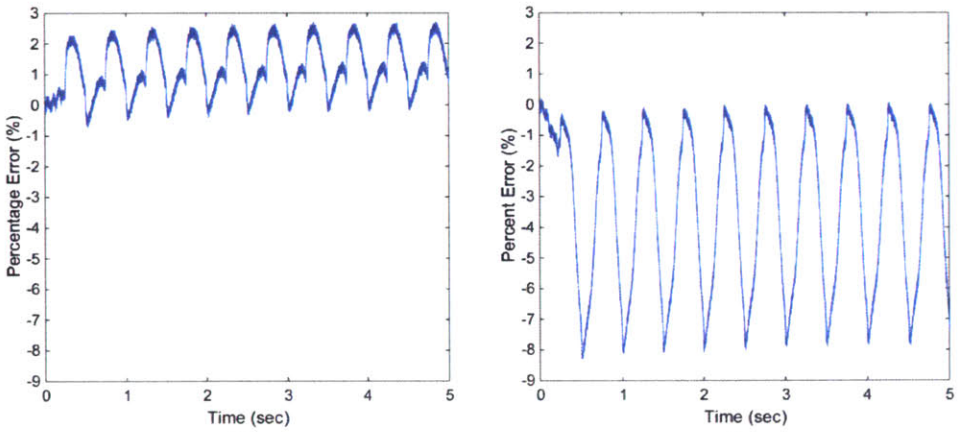


(c)



(g)

Percentage Error



(d) (h)

Figure 4-5: Simulation results of hysteresis loop, displacement over time, absolute error, and percentage error under a triangular input of 0-65V at a frequency of 2Hz using (a)-(d): the Input-Range Dependent Maxwell Model, and (e)-(h): the classical Maxwell model. Both models were constructed with 40 elasto-slide elements.

Reducing the percentage simulation error by 67% from 8.29% to 2.71%, the Input-Range Dependent Maxwell Model is indeed more accurate than the classical model in modeling hysteresis behavior of the piezoelectric actuator, and, therefore, is more ideal for hysteresis compensation in nano-positioning applications.

5. Conclusions

New observations of the hysteresis behavior of piezoelectric stack actuators suggest that the actuator behaves differently between the initiation stage and later cycles, and the hysteresis loop in later cycles is affected by the input history, specifically the input range of the most recent half loop and the local extremum input at the beginning of the current half cycle. We propose two types of modified Maxwell model to relieve the limits of the classical Maxwell model and account for the new observations. Further investigation of these two modified models suggests that the shape of the initial portion of a half loop has a much stronger linear dependence on the input range than on the local extremum. By obtaining two sets of parameters for the initiation stage and later cycles of the hysteresis behavior of piezoelectric stack actuators, and selecting the spring constant of the first Maxwell elasto-slide element as a linear function of the input range of the most recent half loop, the Input-Range Dependent Maxwell Model provides an accurate model for simulating the displacement response of a piezoelectric actuator to input voltages. An inverse of the modified model can be easily obtained for nano-positioning applications. Future work

should include exploring a simple method to account for the effect of both the input range and the local extremum, as well as investigating the influence of travel direction on the hysteresis behavior.

Acknowledgement

I would like to give special thanks to Prof. Youcef-Toumi and Dr. Bozchalooi for their guidance and advice on completing the project, Andrew Houck for his help on data collection, and MIT Mechatronics Research Lab for providing the equipment and technical support.

Reference

1. Goldfarb, M. and Celanovic, N., "Modeling piezoelectric stack actuators for control of micromanipulation," *IEEE Control Systems Magazine*, Vol. 17, pp. 69–79, 1997
2. Ge, P. and Jouaneh, M., "Modeling hysteresis in piezoceramic actuators," *Precision Engineering*, Vol. 17, pp. 211–221, 1995
3. Ge, P. and Jouaneh, M., "Tracking control of a piezoceramic actuator," *IEEE Transactions on Control Systems Technology*, Vol. 4, pp. 209–216, 1996
4. Ge, P. and Jouaneh, M., "Generalized Preisach model for hysteresis nonlinearity of piezoceramic actuators," *Precision Engineering*, Vol. 20, pp. 99–111, 1997
5. Goldfarb, M. and Celanovic, N., "A lumped parameter electromechanical model for describing the nonlinear behavior of piezoelectric actuators," *ASME Journal of Dynamic Systems, Measurement and Control*, Vol. 119, pp. 478–485, 1997
6. Lee, S.-H. and Royston, T. J., "Modeling piezoceramic transducer hysteresis in the structural vibration control problem," *Journal of the Acoustical Society of America*, Vol. 108, pp. 2843–2855, 2000
7. Song, G., Zhao, J., Zhou, X. and Abreu-Garcia, J. A., "Tracking control of a piezoceramic actuator with hysteresis compensation using inverse Preisach model," *IEEE/ASME Transactions on Mechatronics*, Vol. 10, No. 2, pp. 198-209, 2005
8. Adly, A. A. and Mayergoyz, I. D., "Preisach modeling of magnetostrictive hysteresis," Journal of Applied Physics, Vol. 69, pp. 5777–5779, 1991
9. Royston, T. J. and Houston, B. H., "Modeling and measurement of nonlinear dynamic behavior in piezoelectric ceramics with application to 1-3 composites," Journal of the Acoustical Society of America, Vol. 104, pp. 2814–2827, 1998
10. Chen, P.J. and Montgomery, T., "A macroscopic theory for the existence of the hysteresis and butterfly loops in ferroelectricity," *Ferroelectrics*, Vol. 23, Issue 1, pp. 199-207, 1980
11. "Piezoelectric Actuators: Components, Technologies, Operation," Physik Instrument (PI), available online at http://www.piezo.ws/pdf/Piezo.pdf

Appendices

Appendix A: Input-Range Dependent Maxwell Model - Parameters in the Initiation Stage

Maxwell Element No.	Spring Constant (V/μm)	Breakaway Force (V)	Maxwell Element No.	Spring Constant (V/μm)	Breakaway Force (V)
1	0.6775	0.1997	21	0.0207	0.1281
2	0.7979	0.4703	22	0.2673	1.7330
3	0.559	0.4942	23	-0.0807	-0.5470
4	0.4258	0.5019	24	-0.3461	-2.4479
5	0.1909	0.2813	25	0.0755	0.5562
6	0.1529	0.2704	26	0.1876	1.4374
7	0.1104	0.2277	27	0.1591	1.2659
8	-0.0689	-0.1624	28	0.2395	1.9763
9	0.1312	0.3480	29	-0.091	-0.7777
10	0.1518	0.4474	30	-0.065	-0.5747
11	0.4707	1.5259	31	-0.0706	-0.6450
12	0.1381	0.4884	32	-0.196	-1.8484
13	0.1177	0.4509	33	0.07	0.6808
14	0.1752	0.7228	34	0.2916	2.9218
15	-0.2121	-0.9376	35	0.1027	1.0593
16	-0.1538	-0.7252	36	-0.1082	-1.1479
17	0.0953	0.4774	37	0.0593	0.6466
18	0.1328	0.7045	38	-0.0687	-0.7693
19	0.2242	1.2554	39	-0.2964	-3.4066
20	0.1453	0.8564	40	6.1268	72.2227

Appendix B: Input-Range Dependent Maxwell Model - Parameters in Later Cycles

Maxwell Element No.	Spring Constant (V/ μm)		Breakaway Force (V)
1	Input-Range Dependent Maxwell Model*	$k_{c,1}(F_{range})$	$f_{c,1}(F_{range})$
	Local-Extremum Dependent Maxwell Model**	$k_{c,1}(F_{extre})$	$f_{c,1}(F_{extre})$
2	0.7053		0.4157
3	0.5323		0.4706
4	0.4077		0.4806
5	0.3362		0.4954
6	0.2647		0.4680
7	0.2355		0.4858
8	0.1922		0.4531
9	0.1792		0.4753
10	0.1325		0.3905
11	0.1170		0.3793
12	0.0958		0.3388
13	0.1068		0.4092
14	0.0440		0.1815
15	0.0459		0.2029
16	0.0753		0.3551
17	0.0380		0.1904
18	2.3520		12.4764
19	4.0409		22.6262

*Input-Range Dependent Maxwell Model:

$$k_{c,1}(F_{range}) = 7.465 \times 10^{-3} F_{range} - 0.3578$$

$$f_{c,1}(F_{range}) = 2.200 \times 10^{-3} F_{range} - 0.1054$$

**Local -Extremum Dependent Maxwell Model:

$$k_{c,1}(F_{extre}) = 9.526 \times 10^{-4} F_{extre} - 0.08523$$

$$f_{c,1}(F_{extre}) = 2.807 \times 10^{-4} F_{extre} - 0.02512$$

Appendix C: Classical Maxwell Model Parameters

Maxwell Element No.	Spring Constant (V/μm)	Breakaway Force (V)	Maxwell Element No.	Spring Constant (V/μm)	Breakaway Force (V)
1	1.3911	0.3823	21	0.1588	0.9164
2	0.6169	0.3390	22	0.3665	2.2157
3	0.4905	0.4044	23	-0.1828	-1.1554
4	0.4142	0.4553	24	-0.2071	-1.3659
5	0.2836	0.3897	25	0.2208	1.5169
6	0.2311	0.3810	26	0.0921	0.6580
7	0.1927	0.3707	27	-0.0656	-0.4867
8	0.2591	0.5696	28	0.3067	2.3599
9	0.1852	0.4580	29	0.064	0.5100
10	0.0897	0.2465	30	-0.285	-2.3495
11	0.1573	0.4755	31	-0.1419	-1.2088
12	0.1842	0.6074	32	0.0255	0.2242
13	0.0871	0.3112	33	0.3002	2.7223
14	0.0771	0.2966	34	0.183	1.7098
15	0.0938	0.3866	35	-0.2391	-2.2997
16	0.011	0.0484	36	-0.0591	-0.5847
17	0.0408	0.1906	37	0.1432	1.4560
18	-0.0271	-0.1340	38	-0.2757	-2.8790
19	0.0854	0.4459	39	-0.0761	-0.8156
20	-0.69	-3.7922	40	6.2385	68.5736

Appendix D: Matlab Code for Obtaining Data

```
function [experiment_setup,time,V,Disp]=Read_data_Disp(file_name,gain,sense)
%experiment_setup = the setup of the experiment, indicated in the file name
%time = time stamp
%V = input voltage as a function of time
%Disp = displacement as a function of time
%gain = gain of the amplifier
%file_name = the name of the .lvm file, including ".lvm" in the name
%sense = sensitivity of the laser interferometer

%first get the set up by truncating .lvm from the file name
experiment_setup=strtok(file_name,'.');

%this step reads .lvm file and put all data with colunm names into
"data.Segment1"
%Data are stored in "data.Segment1.data"
data=lvm_import(file_name,0);

%The first column is time
time=data.Segment1.data(:,1);

%The second column is the voltage input before going through amplifier
%The default output of the amplifier is 75V
%To get the voltage applied to the piezo, the second column needs to be
%multiplied by the amplifier gain, and add 75V
V=data.Segment1.data(:,2)*gain+75;

%The third column is the voltage measure by the laser interferometer
Vout=data.Segment1.data(:,3);

%Now convert output voltage into positive values and start at 0
Vout=-(Vout-Vout(1));
Vout=Vout-min(Vout);

%translate output voltage to displacement, ratio unit is um/V
switch sense
    case 0
        ratio=0.24;
    case 1
        ratio=0.97;
    case 2
        ratio=3.87;
    case 3
        ratio=15.48;
    case 4
        ratio=61.9;
    case 5
        ratio=247.61;
    case 6
        ratio=990.44;
end
Disp=Vout*ratio;
end
```

Appendix E: Matlab Code for Dividing Hysteresis Loop into Sections

```
function
[V_p2p,Disp_p2p,V1,Disp1,Vup,Disp_up,Vdown,Disp_down,cycle]=directional2(file
_name,gain,sense)
%V_p2p and Disp_p2p includes all the sections
%V1 and Disp1 are voltage and displacement in the initiation stage
%Vup and Disp_up are voltage and displacement in the ascending curves
%Vdown and Disp_down are voltage and displacement in the descening curves

%% The first step is to read data from the file
[setup,time,V,Disp]=Read_data_Disp(file_name, gain, sense);

%% Now find the number of peaks and the peaks' locations
%To find peaks of input, check the sign of
%(Vin(i)-Vin(i-1))*(Vin(i)-Vin(i+1)), if sign=+, then i is a local peak
%Notice that the first and last input are also local peaks

%find number of data points
n=length(V);

%the first peak is the first input
cycle_number=1;
peak_index(1)=1;
V_peak(1)=V(1);
Disp_peak(1)=Disp(1);

%now check all data except for the last one
for i=2:n-1
    if -1<sign((V(i)-V(i-1))*(V(i)-V(i+1)));
        cycle_number=cycle_number+1;
        peak_index(cycle_number)=i;
    end
end

%the last peak is the last input
peak_number=cycle_number+1;
peak_index(peak_number)=n;

%% Now find each peak to peak sections of Vin and Vout
%the first index of (i)th cell is the (i)th peak
%the last index of (i)th cell is the (i+1)th peak
Vup={};
Disp_up={};
Vdown={};
Disp_down={};
up=0;
down=0;
for i=1:cycle_number
    V_p2p{i}=V(peak_index(i):peak_index(i+1));
    Disp_p2p{i}=Disp(peak_index(i):peak_index(i+1));
    if i==1
        V1=V_p2p{1};
        Disp1=Disp_p2p{1};
    else if V_p2p{i}(1)<V_p2p{i}(end)
```

```
            up=up+1;
            Vup{up}=V_p2p{i};
            Disp_up{up}=Disp_p2p{i};
        else
            down=down+1;
            Vdown{down}=V_p2p{i};
            Disp_down{down}=Disp_p2p{i};
        end
    end
end

%% Output how many forward and backward curves
cycle=[cycle_number,up,down];

end
```

Appendix F: Matlab Code for Identifying Slopes of Sections

```
function [setup,dx,s]=Maxwell_const_multi_input(file_name,spring_N, gain,
sense)
%s is the cells containing of slopes from all sections
%dx is the displacement of each section
%file_name is the data gathered in LabView
%spring_N is the number of springs used to construct the model
%gain is the gain of amplifier
%sense is the sensitivity of the laser interferometer

%% The first step is to read data from the file
[setup,time,V_raw,Disp_raw]=Read_data_Disp(file_name, gain,sense);
[V_p2p,Disp_p2p,V1, Disp1, V_up, Disp_up, V_down,
Disp_down,cycle]=directional2(file_name,gain,sense);

cycle_number=cycle(1);
up_number=cycle(2);
down_number=cycle(3);
%% second, define the each sub-sections used to construct model
dx=range(Disp_raw)/spring_N;
x=cell(1,cycle_number); %the segment points for each curve
N=zeros(cycle_number,1); %number of springs for each curve

%define the first set to be the initiation stage, the 2~up_number+1 sets to
%be the ascending stages, and the rest to be the descending stages.
Disp=cell(1,cycle_number);
V=cell(1,cycle_number);
Disp{1}=Disp1;
V{1}=V1;
x{1}=min(Disp{1}):dx:max(Disp{1});
N(1)=length(x{1})-isinteger(range(Disp{1})/dx);

for i=2:up_number+1
    Disp{i}=Disp_up{i-1};
    V{i}=V_up{i-1};
    x{i}=min(Disp{i}):2*dx:max(Disp{i});
    N(i)=length(x{i})-isinteger(range(Disp{i})/(2*dx));
end
for i=up_number+2:cycle_number
    Disp{i}=Disp_down{i-up_number-1};
    V{i}=V_down{i-up_number-1};
    x{i}=max(Disp{i}):-2*dx:min(Disp{i});
    N(i)=length(x{i})-isinteger(range(Disp{i})/(2*dx));
end

%% Now find the slope for each section
s=cell(1,cycle_number); %s includes slopes for all the sections

for i=1:cycle_number
    %first cut each curve into sections
    j=1;
    point=length(Disp{i});
    x_model=cell(1,N(i));
    F_model=cell(1,N(i));
```

```
    for m=1:point
        if i<=up_number+1&&j<N(i) %the first curve and all the forward curves
            j=j+(Disp{i}(m)>x{i}(j+1));
        elseif j<N(i)    %downward curves
            j=j+(Disp{i}(m)<x{i}(j+1));
        end

        x_model{j}=[x_model{j};Disp{i}(m)];
        F_model{j}=[F_model{j};V{i}(m)];
    end
    s{i}=zeros(N(i),1);
    for j=1:N(i)
        p=polyfit(x_model{j},F_model{j},1);
        s{i}(j)=p(1);
    end
end

end
```

Appendix G: Matlab Code for Identifying Spring Constants under Different Input Conditions

```
function [k,f,s_avg,N]=Minor_loop_k(s,dx,repeat,range_number)
%k is a cell of spring constants for each range
%f is a cell of breakaway forces
%s_avg is the average slope of each segment
%N is the number of elements
%s is the slope of all sections
%dx is the displacement of one segment
%range_number is the number of input ranges

%% find average slopes
section_number=2*range_number+1;
s_avg=cell(1,section_number);

s_avg{1}=s{1};
for i=2:section_number
    s_sum=0;
    for j=2:repeat-1
        m=(i-2)*10+j;
        delement=length(s_sum)-length(s{m});
        if ismatrix(s_sum) && delement<0
            s_sum=[s_sum;s{m}(end+delement+1:end)];
        elseif delement>0
            s{m}=[s{m};s_sum(end-delement+1:end)];
        end
        s_sum=s_sum+s{m};
    end
    s_avg{i}=s_sum/(repeat-2);
end

%% find k and f
N=zeros(section_number,1);
k=cell(1,section_number);
f=cell(1,section_number);

for i=1:section_number
    N(i)=length(s_avg{i});
end
N_max=max(N);

for i=1:section_number
    if N(i)==N_max||N(i)==1
        A=triu(ones(N(i)));
        k{i}=A\s_avg{i};
        f{i}=k{i}*i*dx;
    elseif N(i)>=N_max-2 || N(i)<=3
        A=triu(ones(N(i)));
        k{i}=A\s_avg{i};
        k{i}=k{i}(1:end-1); %ignore the last one since it could be the sum of
all other springs
        f{i}=k{i}*i*dx;
    elseif N(i)>=4
```

```
        A=triu(ones(N(i)));
        k{i}=A\s_avg{i};
        k{i}=k{i}(1:end-2); %ignore the last two since they could be the sum
of all other springs
        f{i}=k{i}*i*dx;
    end
end

end
```

Appendix H: Matlab Code for Finding Relationships between Spring Constants and the Input History and Creating Plots

```
file={'5-75-5 Step5 sense3 test 1.lvm','5-75-5 Step5 sense3 test 2.lvm','5-
75-5 Step5 sense3 test 3.lvm'};
test_number=3;
sense=3;
gain=7.5;
repeat=10;
step_up_number=15;
range_number=29;
section_number=2*range_number+1;
Vstart_75=zeros(section_number,2);
Vstart_75(:,2)=[ones(1+range_number,1);-1*ones(range_number,1)];
Vstart_75(:,1)=[zeros(step_up_number+1,1);[70:-
5:5]';[5:5:75]';75*ones(step_up_number-1,1)];

V_raw=cell(1,test_number);
Disp_raw=cell(1,test_number);
setup=cell(1,test_number);
s_sec=cell(1,test_number);
k=cell(1,test_number);
f=cell(1,test_number);
s=cell(1,test_number);
N=zeros(section_number,test_number);
k_avg=cell(1,section_number);
s_avg=cell(1,section_number);
f_avg=cell(1,section_number);

Disp_range=zeros(test_number,1);
spring_N=zeros(test_number,1);

%% Make sure all dx are the same for all tests
spring_N(1)=40;
for i=1:test_number
    [setup{i},time,V_raw{i},Disp_raw{i}]=Read_data_Disp(file{i},gain,sense);
    Disp_range(i)=range(Disp_raw{i});
end
dx=Disp_range(1)/spring_N(1);
spring_N(2:end)=Disp_range(2:end)/dx;

%% find the parameters for each test
for i=1:test_number
    [setup{i},dx,s_sec{i}]=Maxwell_const_multi_input(file{i},spring_N(i),
gain, sense);
    for j=152:582
        s_sec{i}{j-1}=s_sec{i}{j};
    end
    [k{i},f{i},s{i},N(:,i)]=Minor_loop_k(s_sec{i},dx,repeat,range_number);
end

%% Use s_avg to calculate k
for i=1:3
    s_new{i}={s{i}{3:16} s{i}{18:30} s{i}{32:45} s{i}{47:59}};
end
```

```
s_avg=zeros(19:1);
s_sum=zeros(19,1);
s_number=zeros(19,1);

for i=1:19
    for j=1:3
        for p=1:54
            if length(s_new{j}{p})>i ||length(s_new{j}{p})==19
                s_sum(i)=s_sum(i)+s_new{j}{p}(i);
                s_number(i)=s_number(i)+1;
            end
        end
    end
    s_avg(i,1)=s_sum(i)/s_number(i);
end
A=triu(ones(19));
k_avg=A\s_avg;

%% Find the spring constant function k=AV(range)+B
k1=zeros(54,1);
for i=1:54
    k1(i)=(k{1}{i}(1)+k{2}{i}(1)+k{3}{i}(1))/3;
end
Vrange=[[10:5:75]';[10:5:70]';[10:5:75]';[10:5:70]'];
[Vrange_sort,Vorder]=sort(Vrange);
k1_sort_range=k1(Vorder);
p=polyfit(Vrange_sort,k1_sort_range,1);
A_range=p(1)
B_range=p(2)

figure
plot(Vrange_sort,k1_sort_range,'o',[-4 80],[A_range*(-4)+B_range
A_range*80+B_range]);
xlabel('Input Range (V)','fontsize',14)
ylabel('Spring Constant (V/um)','fontsize',14)
xlim([0 80])
ylim([-0.65 0.35])
legend('k1: Experiment','k1: Linear fit','location','northwest')

%% Find the spring constant function k=AV(extremum)+B
Vextre=[zeros(14,1);[65:-5:5]';[10:5:75]';75*ones(13,1)];
[Vextre_sort,Vorder]=sort(Vextre);
k1_sort_extre=k1(Vorder);
p=polyfit(Vextre_sort,k1_sort_extre,1);
A_extre=p(1)
B_extre=p(2)

figure
plot(Vextre_sort,k1_sort_extre,'o',[-4 80],[A_extre*(-4)+B_extre
A_extre*80+B_extre]);
xlabel('Local Extremum (V)','fontsize',14)
ylabel('Spring Constant (V/um)','fontsize',14)
xlim([-4 80])
ylim([-0.65 0.35])
legend('k1: Experiment','k1: Linear fit','location','northwest')
```

Appendix I: Matlab Code for Simulation with Input-Range Dependent Maxwell Model

```
function [V_model,
D_model]=Maxwell_simulate_range(dx,k_ini,k_avg,A_range,B_range,Vin)
V_model=Vin;
Vrange=range(Vin);
k_avg(1)=A_range*Vrange+B_range;
x1=dx*[1:length(k_ini)]';
x_cycle=2*dx*[1:length(k_avg)]';
A1=triu(ones(length(k_ini)));
s1=A1*k_ini;
A_cycle=triu(ones(length(k_avg)));
s_cycle=A_cycle*k_avg;

%% Divide Vin into sections
%To find peaks of input, check the sign of
%(Vin(i)-Vin(i-1))*(Vin(i)-Vin(i+1)), if sign=+, then i is a local peak
%Notice that the first and last input are also local peaks

%find number of data points
n_point=length(Vin);

%the first peak is the first input
peak_number=1;
peak_index(1)=1;
Vin_peak(1)=Vin(1);

%now check all data except for the last one
for i=2:length(Vin)-1
    if -1<sign((Vin(i)-Vin(i-1))*(Vin(i)-Vin(i+1)));
        peak_number=peak_number+1;
        peak_index(peak_number)=i;
        Vin_peak(peak_number)=Vin(i);
    end
end

%the last peak is the last input
peak_number=peak_number+1;
peak_index(peak_number)=n_point;
Vin_peak(peak_number)=Vin(n_point);

%number of cycles is peak_number-1
cycle_number=peak_number-1;

%% First simulate the initiation section
% Calculate the total force at which a block moves
N1=length(k_ini);      %total number of blocks
F1=zeros(N1,1);
F1(1)=x1(1)*s1(1);
for i=2:N1
    F1(i)=F1(i-1)+s1(i)*(x1(i)-x1(i-1));
end
```

```
%now find how many blocks move
block_index1=find(F1-(Vin_peak(2)-Vin_peak(1))>0,1); %the 1st block that
doens't move
if isempty(block_index1)
    block_index1=N1;     %if all blocks move, treat the last part as linear
    sprintf('Error: Exceeds model range');
end

if block_index1==1  %no block moves
    D_model=Vin/s1(1);
    return    %when no block moves, the function stops here
else
    for i=1:block_index1-1
        breakpt1(i)=find(Vin-F1(i)>0,1);
    end

    D_model(1:breakpt1(1))=Vin(1:breakpt1(1))/s1(1);

    if block_index1>=3
        for i=2:block_index1-1
            D_model(breakpt1(i-1)+1:breakpt1(i))=D_model(breakpt1(i-
1))+(Vin(breakpt1(i-1)+1:breakpt1(i))-Vin(breakpt1(i-1)))/s1(i,1);
        end
    end

D_model(breakpt1(end)+1:peak_index(2))=D_model(breakpt1(end))+(Vin(breakpt1(e
nd)+1:peak_index(2))-Vin(breakpt1(end)))/s1(block_index1,1);
end
D_model=D_model';

%% Now simulate other cycles
% We need to treat the forward and backward sections differently
% first calculate how much change in input force will make a block move
% notice that x is the same for both directions
N_cycle=length(k_avg);
F_cycle=zeros(N_cycle,1);
F_cycle(1)=x_cycle(1)*s_cycle(1);
for i=2:N_cycle
    F_cycle(i)=F_cycle(i-1)+s_cycle(i)*2*dx;
end

%% Then we find how many blocks move in each section and find Vout
if cycle_number>1
    for cycle=2:cycle_number
        points=peak_index(cycle)+1:peak_index(cycle+1); %index of the points
in this section
        dVin=Vin(points)-Vin(peak_index(cycle));

        %ascending sections
        if mod(cycle,2)==1
            %find how many blocks move
            indexup=find(F_cycle-max(dVin)>0,1); %the 1st block that doens't
move
            if isempty(indexup)
                indexup=N_cycle;     %if all blocks move, treat the last part
as linear
```

```
            sprintf('Error: Exceeds model range');
        end

        %now we can simulate the displacement
        if indexup==1  %no block moves
            D_model(points)=dVin/s_cycle(1)+D_model(points(1)-1);
        else
            for i=1:indexup-1
                breakup(i)=find(dVin-F_cycle(i)>0,1)+peak_index(cycle);
                if isempty(nonzeros(dVin-F_cycle(i+1)>0))
                    break
                end
            end
            lastblock=i;        %the last block to move

            %Divide the section into 3 subsections, first block, other
            %blocks, and after the last moving block
            D_model(points(1):breakup(1))=D_model(points(1)-
1)+dVin(1:(breakup(1)-points(1)+1))/s_cycle(1);
            if indexup>=3
                for i=2:lastblock
                    D_model(breakup(i-1)+1:breakup(i))=D_model(breakup(i-
1))+(Vin(breakup(i-1)+1:breakup(i))-Vin(breakup(i-1)))/s_cycle(i);
                end
            end

D_model(breakup(end)+1:points(end))=D_model(breakup(end))+(Vin(breakup(end)+1
:points(end))-Vin(breakup(end)))/s_cycle(lastblock);
        end

        %descending sections
    else
        %find how many blocks move
        indexdown=find(F_cycle-max(abs((dVin)))>0,1); %the 1st block that
doens't move
        if isempty(indexdown)
            indexdown=N_cycle;     %if all blocks move, treat the last
part as linear
            sprintf('Error: Exceeds model range');
        end

        %now we can simulate the displacement
        if indexdown==1  %no block moves
            D_model(points)=dVin/s_cycle(1)+D_model(points(1)-1);
        else
            for i=1:indexdown-1
                breakdown(i)=find(abs(dVin)-
F_cycle(i)>0,1)+peak_index(cycle);
                if isempty(nonzeros(abs(dVin)-F_cycle(i+1)>0))
                    break
                end
            end
            lastblock=i;        %the last block to move

            %Divide the section into 3 subsections, first block, other
            %blocks, and after the last moving bl
```

```
                D_model(points(1):breakdown(1))=D_model(points(1)-
1)+dVin(1:(breakdown(1)-points(1)+1))/s_cycle(1);
                if indexdown>=3
                    for i=2:lastblock
                        D_model(breakdown(i-
1)+1:breakdown(i))=D_model(breakdown(i-1))+(Vin(breakdown(i-
1)+1:breakdown(i))-Vin(breakdown(i-1)))/s_cycle(i);
                    end
                end

D_model(breakdown(end)+1:points(end))=D_model(breakdown(end))+(Vin(breakdown(
end)+1:points(end))-Vin(breakdown(end)))/s_cycle(lastblock);
            end
        end
    end
end
end
```

Appendix J: Matlab Code for Construction of the Classical Maxwell Model

```
function
[experiment_setup,s,x,k,F]=Maxwell_construct(file_name,spring_N,gain,sense)
%experiment_setup includes min and max Vin, frequency of Vin & cycle number
%peak_index is the index of peaks
%s is the column vector of slopes
%x is the column vector of Disp locations where one spring gives up
%k is the column vector of stiffness of each spring
%F is the column vector of the breakout force of each spring
%file_name is the data gathered in LabView
%spring_N is the number of springs used to construct the model
%gain is the gain of amplifier
%sense is the sensitivity of the laser interferometer

%% The first step is to read data from the file
[experiment_setup,time,Vin,Disp]=Read_data_Disp(file_name, gain,sense);

%% Now find the number of peaks and the peaks' locations
%To find peaks of input, check the sign of
%(Vin(i)-Vin(i-1))*(Vin(i)-Vin(i+1)), if sign=+, then i is a local peak
%Notice that the first and last input are also local peaks

%find number of data points
n=length(Vin);
%the first peak is the first input
peak_number=1;
peak_index(1)=1;
Vin_peak(1)=Vin(1);
Disp_peak(1)=Disp(1);
%now check all data except for the last one
for i=2:length(Vin)-1
    if -1<sign((Vin(i)-Vin(i-1))*(Vin(i)-Vin(i+1)));
        peak_number=peak_number+1;
        peak_index(peak_number)=i;
        Vin_peak(peak_number)=Vin(i);
        Disp_peak(peak_number)=Disp(i);
    end
end
%the last peak is the last input
peak_number=peak_number+1;
peak_index(peak_number)=n;
Vin_peak(peak_number)=Vin(n);
Disp_peak(peak_number)=Disp(n);

%% Now create cells arrays of Vin and Vout
%The dimension of the cell arrays is (spring number N) by (peak_number-1)
%such that each row contains the period between two peaks
%and the (i)th cell in a row corresponds to the (i)th section used in the
%model
Maxwell_force=cell([peak_number-1,spring_N]);
Maxwell_disp=cell([peak_number-1, spring_N]);
```

```
%first, find each peak to peak sections of Vin and Vout
%the first index of (i)th cell is the (i)th peak
%the last index of (i)th cell is the (i+1)th peak
for i=1:peak_number-1
    Vin_pk_pk{i}=Vin(peak_index(i):peak_index(i+1));
    Vout_pk_pk{i}=Disp(peak_index(i):peak_index(i+1));
end

%% second, define the each sub-sections used to construct model
Vout_range=max(Disp)-min(Disp);
section_length=Vout_range/spring_N;       %find the length of each section
section_limit=min(Disp):section_length:max(Disp);

%the Vout location where each spring gives up is the same as section limit
x=section_limit(2:end)';

%% now divide each section according to the section limits
%the first pk-pk will be divided into N sections, with Disp divided
%according to the section limits
%the rest pk-pk sections will be divided into sub sections whose section
%lengths are 2*section_length

%% the first pk-pk needs to be divided into N sections
%if Vout(i)>=section_limit(j) and Vout(i)<section_limit(j)+1, then Vout(i)
%belongs to the jth cell of the first row of Maxwell_out, and the
%corresponding Vin belongs to the jth cell of the first row of Maxwell_in
j=1;
for i=1:length(Vout_pk_pk{1})
    if (Vout_pk_pk{1}(i)-section_limit(j)>0)&&(Vout_pk_pk{1}(i)-
section_limit(j+1)>0)
        j=j+1;
    end
    Maxwell_disp{1,j}=[Maxwell_disp{1,j} Vout_pk_pk{1}(i)];
    Maxwell_force{1,j}=[Maxwell_force{1,j} Vin_pk_pk{1}(i)];
end

%% Now divide the rest pk-pk sections
%for even pk-pk sections, the direction of Vin and Vout is the reverse of
%the first section, therefore the first cell corresponds to the last
%section_limit
%for odd pk-pk sections, the direction of Vin and Vout is the same as the
%first section, therefore the first cell corresponds to the first
%section_limit

for i=2:peak_number-1
    input_number=length(Vout_pk_pk{i});         %find how many input points
are in this section
    j=1;

    if mod(i,2)==0
        %for even pk-pk sections, compare the kth point of Vout_pk_pk{i}
        %with section limits in the reverse order
        %i.e. if the kth point falls between section_limit(end) and
        %section_limit(end-2), it goes to the 1st cell in the ith row
        for k=1:input_number
```

```
            if ((Vout_pk_pk{i}(1)-
Vout_pk_pk{i}(k))>2*j*section_length)&&((Vout_pk_pk{i}(1)-
Vout_pk_pk{i}(k))<=2*(j+1)*section_length)
                j=j+1;
            end
            Maxwell_disp{i,j}=[Maxwell_disp{i,j} Vout_pk_pk{i}(k)];
            Maxwell_force{i,j}=[Maxwell_force{i,j} Vin_pk_pk{i}(k)];
        end
    else
        %for odd pkk-pk sections, compare the kth point of Vout_pk_pk{i}
        %with section limits in the same order
        for k=1:input_number
            if ((Vout_pk_pk{i}(k)-
Vout_pk_pk{i}(1))>2*j*section_length)&&((Vout_pk_pk{i}(k)-
Vout_pk_pk{i}(1))<=2*(j+1)*section_length)
                j=j+1;
            end
            Maxwell_disp{i,j}=[Maxwell_disp{i,j} Vout_pk_pk{i}(k)];
            Maxwell_force{i,j}=[Maxwell_force{i,j} Vin_pk_pk{i}(k)];
        end
    end
end

%% now we need to find the slope of each cell
%create a matrix of slopes such that slope(i,j) is the slope corresponding
%to the jth cell of ith spring
%if the cell is empty, then no slope is added to the vector
slope=zeros(spring_N,peak_number-1);
for i=1:spring_N
    for j=1:peak_number-1
        if isempty(Maxwell_force{j,i})~=1
            p=polyfit(Maxwell_disp{j,i},Maxwell_force{j,i},1);
            slope(i,j)=p(1);
        end
    end
end

%% now we need to find the matrix k for springs
%first find the slope for the ith section such that the mean square root
%error is the smallest for slope(:,i)
%therefore, the slope s(i) would be then mean of nonzero slope(:,i)
s=[];
for i=1:spring_N
    s(i)=mean(nonzeros(slope(i,:)));
end

% S=A*K where A is a upper matrix of ones
A=triu(ones(spring_N));
s=s';
k=A\s;
%% now we need to fine the breakout forces F
F=diag(k)*x;

end
```

Appendix K: Matlab Code for Simulation with the Classical Maxwell Model

```
function [model_check, Vin_model, Disp_model]=Maxwell_simulate(F, s, k, x,
gain, Vin, spring_N)
%spring_check outputs if enough springs are used to generate the model
%Vin_model is the same as Vin, aka Maxwell force, used to create the model
%Vout_model is the simulated Vout, indicating displacement
%s is the slope of each section
%F is the column vector of breakout forces
%k is the column vector of Maxwell stiffnesses
%x is the column vector of Vin where a spring gives up
%spring_N is number of spring used in the model
Vin_model=Vin;

%% First find how many springs give up
%First I find the total force of each breaking point
Ftotal(1)=sum(k)*x(1);
for i=2:spring_N
    Ftotal(i)=Ftotal(i-1)+sum(k(i:end))*(x(i)-x(i-1));
end
Ftotal=Ftotal';

%now I compare Vin with Ftotal and find the last spring to give up
spring_index=find(Ftotal-max(Vin)>0,1);

%% Before simulating the hysteresis, we check if the model is sufficient for
modeling Vin
%1st case: Vin max is smaller than the first Ftotal
%in this case, no spring breaks
if spring_index==1
    model_case=1;
    %2nd case: more than one block move
else if spring_index<=spring_N
            model_case=2;
            %finally, Vin max is bigger than Ftotal max.
            %in this case, all spring break in the end.
        else if isempty(spring_index)==1
                model_case=3;
                spring_index=spring_N;
            end
        end
    end
end

%% Now we find how many peaks, aka cycles, are in the input
%To find peaks of input, check the sign of
%(Vin(i)-Vin(i-1))*(Vin(i)-Vin(i+1)), if sign=+, then i is a local peak
%Notice that the first and last input are also local peaks

%find number of data points
n=length(Vin);

%the first peak is the first input
```

```
peak_number=1;
peak_index(1)=1;
Vin_peak(1)=Vin(1);

%now check all data except for the last one
for i=2:length(Vin)-1
    if -1<sign((Vin(i)-Vin(i-1))*(Vin(i)-Vin(i+1)));
        peak_number=peak_number+1;
        peak_index(peak_number)=i;
        Vin_peak(peak_number)=Vin(i);
    end
end

%the last peak is the last input
peak_number=peak_number+1;
peak_index(peak_number)=n;
Vin_peak(peak_number)=Vin(n);

%number of cycles is peak_number-1
cycle_number=peak_number-1;

%% Now we find the Vout associated with each model case
%note that f=kx, therefore x=f/k for each spring
switch model_case
    case 1      %no spring breaks
        Disp_model=Vin/s(1);

    case {2,3}      %some or all springs break
        %in the first cycle, it takes Ftotal for each spring to break
        cycle=1;

        for i=1:spring_index-1
            breakpoint(i)=find(Vin-Ftotal(i)>0,1);
        end
        %note: in the case where all springs break according to the model,
        %we treat it as if the last spring doesnt break till the end

        Disp_model(1:breakpoint(1))=Vin(1:breakpoint(1))/s(1);

        for i=2:spring_index-1
            Disp_model(breakpoint(i-
1)+1:breakpoint(i))=Disp_model(breakpoint(i-1))+(Vin(breakpoint(i-
1)+1:breakpoint(i))-Vin(breakpoint(i-1)))/s(i);
        end

Disp_model(breakpoint(end)+1:peak_index(2))=Disp_model(breakpoint(end))+(Vin(
breakpoint(end)+1:peak_index(2))-Vin(breakpoint(end)))/s(spring_index);

        %after the first cycle, it takes 2*F for each spring to break
        %and the first spring to break is always the first spring
        if cycle_number>1
            breakpoint=0;
            for cycle=2:cycle_number
                %the direction of input matters in determining Vout
                %therefore we need to check the direction of input.
```

```
                points=peak_index(cycle)+1:peak_index(cycle+1);
                delta_Vin=Vin(points)-Vin(peak_index(cycle));

                %an odd cycle's direction is the same as the first cycle
                if mod(cycle,2)==1
                    for i=1:spring_index-1
                        breakpoint(i)=find(delta_Vin-
2*Ftotal(i)>0,1)+peak_index(cycle);
                        if isempty(nonzeros(delta_Vin-2*Ftotal(i+1)>0))
                            break
                        end
                    end

                %an even cycle's direction is the opposite of the first
                else
                    for i=1:spring_index-1
                        breakpoint(i)=find(abs(delta_Vin)-
2*Ftotal(i)>0,1)+peak_index(cycle);
                        if isempty(nonzeros(abs(delta_Vin)-2*Ftotal(i+1)>0))
                            break
                        end
                    end
                end
                spring_last=i;          %the last spring to give up

                %now that we have the breakpoints, we can calculate Vout
                Disp_model(points(1):breakpoint(1))=Disp_model(points(1)-
1)+delta_Vin(1:(breakpoint(1)-points(1)+1))/s(1);
                for i=2:spring_last
                    Disp_model(breakpoint(i-
1)+1:breakpoint(i))=Disp_model(breakpoint(i-1))+(Vin(breakpoint(i-
1)+1:breakpoint(i))-Vin(breakpoint(i-1)))/s(i);
                end

Disp_model(breakpoint(end)+1:points(end))=Disp_model(breakpoint(end))+(Vin(br
eakpoint(end)+1:points(end))-Vin(breakpoint(end)))/s(spring_last+1);

            end
        end
        Disp_model=Disp_model';
end
```