LMS-BASED METHOD FOR DAMAGE DETECTION APPLIED TO PHASE II OF THE STRUCTURAL HEALTH MONITORING BENCHMARK PROBLEM

A Thesis

by

ROBIN HUCKABY PRESTON

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2006

Major Subject: Civil Engineering

LMS-BASED METHOD FOR DAMAGE DETECTION APPLIED TO PHASE II OF THE STRUCTURAL HEALTH MONITORING BENCHMARK PROBLEM

A Thesis

by

ROBIN HUCKABY PRESTON

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2006

Major Subject: Civil Engineering

ABSTRACT

LMS-Based Method for Damage Detection Applied to Phase II of

the Structural Health Monitoring Benchmark Problem. (May 2006)

Robin Huckaby Preston, B.S., Texas Tech University

Chair of Advisory Committee: Dr. Luciana Barroso

Structural Health Monitoring (SHM) is the process of monitoring the state of a structure to determine the existence, location, and degree of damage that may exist within the entire structure. A structure's health or level of damage can be monitored by identifying changes in structural or modal parameters. In this research, the structure's health is monitored by identifying changes in structural stiffness. The Adaptive Least Mean Square (LMS) filtering approach is used to directly identify changes in structural stiffness for the IASC-ASCE Structural Health Monitoring Task Group Benchmark problem for both Phase I and II. The research focuses primarily on Phase II of the benchmark problem. In Phase II, modeling error and noise is introduced to the problem making the problem more realistic. The research found that the LMS filter approach can be used to detect damage and distinguish relative severity of the damage in Phase II of the benchmark problem in real time. Even though the LMS filter approach identified damage, a threshold below which damage is hard to identify exists. If the overall stiffness changes less than 10%, then identifying the presence and location of damage is difficult. But if the time of damage is known, then the presence and location can be determined. The research is of great interest to those in the structural health monitoring community, structural engineers, and inspection practitioners who deal with structural damage identification problems.

ACKNOWLEDGEMENTS

I would like to thank my committee chair, Dr. Barroso, and my committee members, Dr. Roesset and Dr. Langari, for their guidance and support throughout the course of this research.

I also want to thank my husband for all his encouragement and support. Without him, I would not have been able to receive my M.S. I would also like to thank my parents whose love and support have allowed me to be where I am in life.

NOMENCLATURE

ASCE	American Society of Civil Engineers
ERA	Eigensystem Realization Algorithm
FEM	Finite Element Method
IASC	International Association for Structural Control
LMS	Least Means Square
MSE	Mean Square Error
MIMO	Multi-input/ Multi-output
RLS	Recursive Least Squares
SHM	Structural Health Monitoring

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

The ability to monitor the integrity of a structure at the earliest possible time is of great importance for engineers. The knowledge that a structure is safe for use allows the structure to be optimally used and can prevent possible catastrophic events from occurring. Throughout the service life of engineering structures, various types of damage, from cracking in an individual member to complete failure of several members, are often observed. The ability to detect this damage quickly, in real time, is of great importance so that the safety and integrity of the structure can be ensured. Current damage detection methods are based on knowing the general area of damage so various localized experiments can be performed in that area. These local methods include visual inspection, ultrasonic, x-ray, and magnet field methods but localized methods are only used on small sections due to cost. A need exists for a method that is able to detect damage on a global scale of the engineering system and that can be implemented in real time which will assist in long term maintenance and immediate analysis of the structure in catastrophic events.

Structural Health Monitoring (SHM) is the process of monitoring the state of a structure to determine the existence, location, and degree of damage that may exist within the entire structure. Vibration-based SHM methods are based on the fact that damage can be determined from changes in dynamic properties of a structure. A change in the structure's physical properties, mass, damping or stiffness, corresponds to a change in its modal parameters (Doebling et al. 1996). Modal parameters or physical properties from the undamaged and damaged structure are compared to establish the presence, location and severity of damage. The problem lies in that modal parameters can be insensitive to localized damage and sensitive to variations in the physical properties of the structure and added noise in the input signals. And most importantly,

---

This thesis follows the style of the *Journal of Structural Engineering.*

SHM methods based on modal parameters can not be implemented in real time, on a sample to sample basis as the event occurs, because the entire measured response needs to be processed to identify damage (Chase et al 2003a). The ability to access damage in a structure without using modal parameters is needed.

Research has been completed on a technique that applies the Adaptive Least Mean Square (LMS) filtering theory directly to identify changes in the structure's stiffness parameters in real time. The method based on the adaptive LMS filter takes advantage of the filter's ability to adaptively model noisy signals to identify changes in structural parameters in comparison to a base structural model. The algorithm presented in this thesis is computationally simple and can be run in real time to detect damage in the structure. The LMS method has been applied to a 4 degree-of-freedom and a 12 degree-of-freedom SHM benchmark problem with successful results (Chase et al 2003a).

The SHM benchmark problem was developed by a SHM Task group in order to study the efficacy of various SHM methods. The benchmark problem comprises of two phases. Chase et al's work focused on Phase I which was a simple model intended to test various concepts (2003a). The second phase is more complex because a 120 degree-of-freedom system with modeling error is included. On a real structure, both the actual mass and stiffness values will not match the design mass and stiffness values so the addition of modeling error replicates a more realistic situation. In addition while gathering data from accelerometers, noise will indirectly enter in the data record and need to be added to Phase II of the benchmark problem. In order to expand the range of application of this method to more realistic applications, the adaptive filter will need to be expanded to Phase II of the benchmark problem.

Although the primary focus of this thesis is Phase II of the benchmark problem, the concept was used on simple systems to replicate and validate the previous work completed by Chase et al (2003a). The results show that it is easy to distinguish the presence and severity of damage when the damage is large and noise is small. The LMS method was then tested in Phase II where all cases contained noise and error to test the robustness of the method. The LMS method was able to identify the presence and location of damage in real time even with the presence of noise and modeling data. If the overall stiffness of the structures changes below a certain percentage then the damage can not be identified with the presence of noise and error. The threshold with the given model error and noise contained in the benchmark problem is 10%. Below 10%, the change due to damage and change due to error are hard to distinguish. If each degree of freedom is studied individually, then identifying damage below the 10% threshold if the time of damage is known becomes easier. When looking at specific degrees of freedom that were damaged, sharp changes in stiffness were seen at the time of damage. In degrees-of-freedom that didn't undergo damage, sharp changes were seen at random times indicative of modeling error or noise. In the cases where full failure of members and alpha changes are more than 10%, the sharp changes at random times are only seen as noise and not possible damage. So the tests on Phase II prove that the method does indicate the presence of damage but is limited in when it can detect the exact severity of the damage. The severity is detected when damage is above a threshold or the time of damage is known.

This research shows that the LMS method can identify damage when applied to Phase II of the SHM benchmark problem. Even with the addition of modeling error and noise, the LMS method is robust enough to filter out the noise. But the most important aspect of the LMS method is that it is able to identify presence and severity of damage in real time. The filter can be used to quickly asses the structural health of a system during a catastrophic event.

REVIEW OF TECHNICAL STATE OF ART

Structural Health Monitoring

Structural Health Monitoring (SHM) is the process of monitoring the state of a structure to determine the existence, location, and degree of damage that may exist within the entire structure. Structural Health Monitoring can be divided into two categories: local and global methods. Local methods focus on individual parts of the system in great detail while global methods focus on analyzing the general health of the entire system. Currently, most methods used are local methods, like visual inspection, magnetic methods and strain measurements. These methods are limited in their applications because they require on prior knowledge of the general location of damage in the structure and rely on human expertise. Unfortunately, damage that occurs in civil structures usually is obscured from view. For example after the 1995 Hyogo-Ken Nanbu (Kobe) earthquake, damage in buildings was only found after intensive one-by-one visual inspections were conducted (Mita 1999). The earthquake also brought to light to the problem of determining damage in foundations, piles, and other parts of the structure that can not be seen in visual inspections. Also, even if human inspection of the structures located all forms of damage, the integrity of a structure between inspections cannot be evaluated. The need to be able to effectively and efficiently inspect civil structures has brought about a development in global monitoring techniques.

Global techniques consisting of vibrational analysis can be implemented on an entire structure to evaluate the integrity of the entire structure (Doebling et al 1996). Vibration analysis methods are categorized based on four levels of damage identification (Rytter 1993):

- Level 1: Determination that damage is present in the structure
- Level 2: Determination of the location of the damage
- Level 3: Quantification of the severity of the damage
- Level 4: Prediction of the remaining service life the structure

Currently most methods predict level 1 and level 2 damage. As SHM becomes more developed, the ability to continually asses a structure's health will be achieved. The methods will be able to quickly assess the integrity of the structure so plans of remediation and possibly evacuation can be developed.

Structural Health Monitoring Benchmark Problem

Numerous global SHM methods have been studied by various researchers and documented in summary reviews (Doebling et al 1996). The techniques could not be compared to one another because the different methods were tested on different structures with varying loading conditions. A common problem that all the techniques could be applied to would provide a platform for consistent evaluation of the SHM methods (Johnson et at 2000). The International Association for Structural Control (IASC) and the Dynamics committee of the American Society of Civil Engineers (ASCE) Engineering Mechanics Division formed the SHM committee to develop a series of benchmark problems (Johnson et al 2000). The benchmark problems consist of simple examples to more realistic and difficult situations where noise and modeling error are included.

The purpose of the benchmark problem was to create a common problem on an existing structural model where the input and response of the structure was known. A picture of the model is shown in Figure 2.1 (Dyke et al 2000). The existing structural model consists of a 4-story, 2-bay, by 2-bay steel frame scale-model and is located at the University of British Columbia (Black and Ventura 1998). The model was built to scale size of one third. The model was excited by an electromagnetic shaker and added mass on the top floor (Dyke et al 2000). Accelerometers were placed along each perimeter frames at each floor to accurately capture measurements of the structure's responses. Two finite element models, a 12 degree-of-freedom model, and a 120 degree-of-freedom model were created of the structure to analyze the simulated data. The 12 degree-of-freedom model assumes that the structure acts like a shear building with three degree-of-freedom per floor: translation in x and y direction and rotation. Based on the assumption that no constraints on the rotation of the floor nodes and horizontal translation are present, the 120 degree-of-freedom model was created.

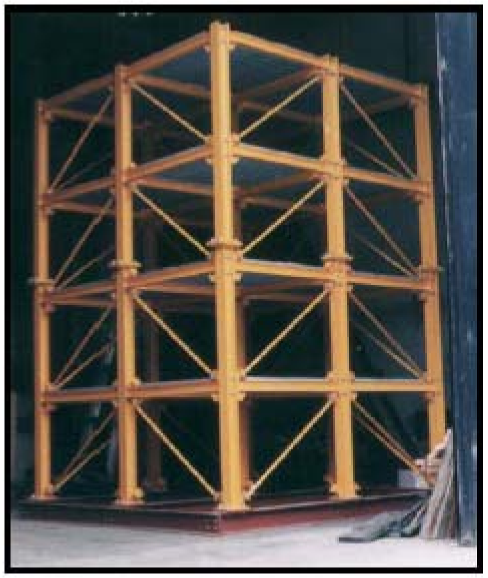


Figure 2.1. Benchmark Problem Model (Dyke et al 2000)

The benchmark problem was divided into two phases with different levels of complexity. Both phases consist of simulated acceleration data from the undamaged structure and several damage scenarios. Each damage scenario was simulated by the removal of varying bracing in the structure that caused reduction in stiffness and the loss of rotational stiffness (Dyke et al 2000). Data for each scenario and case is generated with a MATLAB (MathWorks 2002) program, *datagen.m*, which can be downloaded form the Structural Health Monitoring website, http://wusceel.cive.wustl.edu/asce/shm . The first phase of the benchmark problem consisted of five simulation cases and five damage patterns. A summary of the five cases is shown in Table 2.1. Case 1, 3, and 4 dealt with the 12 degree-of-freedom model while Case 2 and 5 dealt with the 120 degree-of-freedom model. Case 1 and 2 are more complex than Case 3, 4, and 5 because ambient vibration which is small in comparison to vibration due to excitation is included in the simulated data and the input force from the ambient vibrations is unknown. The small vibrations result in data that is hard to process. Case 1 and 2 consisted of loading at each floor in the y direction and the ambient vibration. In Case 3, 4, and 5, the structure was excited by a shaker positioned on the roof in a diagonal direction to excite the structure in both the x and y direction. In addition, Case 1, 2, and 3 have symmetric loading on each floor while Case 4 and 5 have asymmetric loading on the top floor. A summary of the five damage scenarios is shown in Table 2.2. In addition, noise is included in all acceleration data mimicking realistic test conditions.

Table 2.1. Five Case Simulations from Phase 1

Case (1)	# DOF (2)	Excitation (3)	Mass Distributions (4)
1	12	All Stories	Symmetric
2	120	All Stories	Symmetric
3	12	Only At Roof	Symmetric
4	12	Only At Roof	Asymmetric
5	120	Only At Roof	Asymmetric

Table 2.2. Five Damage Scenarios from Phase 1

Pattern (1)	Damage Description (2)
1	All Braces of 1st Story are Removed
2	All Braces of 1st and 3rd Stories are Removed
3	1 Brace of 1st Story is Removed
4	1 Brace of Each 1st and 3rd Stories are Removed
5	Pattern 4 and Unscrew the Left End of Element 18

Phase II of the Benchmark Problem only deals with the 120 degree-of-freedom model and includes more complexities than Phase I. Modeling error of up to 10% of mass values and 5% of stiffness values is added to the problem and the modeling error within the data set varies. The same degree of noise in the acceleration data from Phase

I is included. This allows the benchmark problem to resemble more realistic test situations. The fully braced case has four damage scenarios, the partially braced case has three damage scenarios, and the blind case has two damage scenarios. The damage scenarios are summarized in Table 2.3. No information is given regarding the two blind cases.

Table 2.3. Damage Scenarios from Phase II

Pattern (1)	Bracing (2)	Damage Description (3)
DP1B	Braced	50% Loss of Stiffness on 2 Braces on 1st Floor
DP2B	Braced	25% Loss of Stiffness on 2 Braces on 1st Floor
DP3B	Braced	DP1B and 25 % Loss of Stiffness on 2 Braces on 3rd Floor
DP1U	Unbraced	Loss of Rotational Stiffness in 5 Connections on Level 1 and 2
DP2U	Unbraced	Loss of Rotational Stiffness in 2 Connections in Level 1
DP3Bu	Braced	DP1B and 25 % Loss of Stiffness on 2 Braces on 3rd Floor
DP1Uu	Unbraced	Loss of Rotational Stiffness in 5 Connections on Level 1 and 2

SHM Methods Based on Change in Frequency

The earliest work in SHM focused on methods based on changes in frequency. Natural frequencies of structures are shown in peaks of the dynamic response spectrum. Shifts in the spectrum are changes in frequency and are directly correlated to the existence of damage. Identifying damage from frequency alone has shown to have many limitations. It is only effective when large damage is present and only verifies the existence of that damage (Chase et al 2003a). Also apparent damage can be mistaken for modeling error or noise. Also the entire response data must be known before analysis can be completed making real time implementation impossible. Although these methods have been shown to be impractical, they have laid the foundation for more advanced methods.

The change in frequency methods were developed from studies completed in the 1970s by the offshore oil industry. Vandiver (1975) analyzed change in the frequency of the first bending modes of an offshore pile supported tower to identify damage after a ship impact. Measurements of the two fundamental bending mode frequencies and the first torsional mode frequency had been obtained before the collision. A lumped mass model of the tower was created to predict the lowest natural frequencies of both undamaged and damaged cases of the tower. The differences in frequency between undamaged and damaged cases were compared. Because only the lowest frequencies were analyzed, damage could not be easily identified. In addition, changes in mass, marine growth, modeling error, and sea floor shifts were shown to mask inherent damage.

In 1976, Loland and Dodds monitored three platforms in the North Sea over a period of nine months to record changes in frequencies, mode shapes, and response spectra (Loland and Dodds 1976). The main downfall with this method was that of data collection. Data was only collected above the water line. And as with Vandiver's work, only the lowest frequencies were predicted so the actual location of damage could not be determined.

In 1983, Nataraja also conducted research on the responses of three offshore platforms in the North Sea for two years (Nataraja 1983). New developments in technology allowed Nataraja to monitor ambient vibration thus allowing measurements to be collected continually. Results showed that only the lowest frequencies could be collected with accuracy. Changes in deck mass were seen in the frequency shifts so it was concluded that changes in mass would have to be monitored. Again as with the previous methods, determine presence and location of damage with accuracy solely from frequency shifts proved to be impossible.

In 1994, Friswell et al. cataloged frequency shifts based on known damage scenarios. Damage scenarios were selected from the most likely damage cases. The ratio between the undamaged frequency for several modes and those corresponding to the damage scenarios were compiled. They assumed no error in the model and no noise in the data. Both assumptions make the method hard to implement in real situations.

SHM Methods Based on Change in Mode Shape

Damage is also characterized by shifts in a structure's eigenvalues and eigenvetors. Research has been conducted on the correlation between a structure's mode shape and the location and severity of damage. The location of damage was shown to occur at the greatest change in the mode shape. The combination of change in frequency and mode shape was used to determine the existence of damage. Like methods based on frequency, methods based on mode shape requires the entire finite response data for analysis limiting real time implementation.

In 1985, Yuen examined changes in the mode shape of a cantilever beam undergoing various damage cases (Yuen 1985). The modulus of elasticity was varied to simulate the damage cases. The cantilever beam had two degrees of freedom, one translation and one rotation. The two degrees-of-freedom were separated and the mode shapes analyzed. When the modulus of elasticity was varied, the first modes changed shape which indicated damage while changes in higher modes were harder to characterize.

In 1994, Salawu and Williams compare the results of various techniques based on changes in mode shapes to determine damage (Salawu and Williams 1994). They calculate a ratio between the undamaged mode shape and the damaged mode shape. They concluded that the correct selection of the modes analyzed determined how successful the analysis was. Again, the location of damage was not able to be determined from this method.

Modal shapes can be used to detect damage if used in conjunction with other methods. In 2000, Dyke et al used the modal parameters to determine the optimal stiffness which can be an indicator of damage (Dyke et al 2000). The method was tested on a reduced 4 degree-of-freedom version of the SHM benchmark problem. The cross correlation function, R_{xx}, was calculated from the acceleration data and has the same form as the response data. This approach was effective for determining modal parameters because it can be averaged over a number of samples. The eigensystem realization algorithm (ERA) was used to identify the modal parameters from the free response data because it can be applied to multi-input/multi-output (MIMO) systems. The optimal stiffness coefficients are then determined by using a nonlinear constrained optimization. Changes in the stiffness matrix are correlated to damage in the structure. Although this method was able to identify damage and the degree-of-freedom in which the damage was located, the method could not be implemented in real time.

SHM Methods Based on Change in Flexibility Matrix

The flexibility matrix is the inverse of the static stiffness matrix and it relates applied force to the response of the structure (Doebling 1996). Each column of the flexibility matrix represents the displacement pattern of the structure when a unit force is applied at each degree-of-freedom. The flexibility matrix can be estimated by using the mass-normalized mode shapes. Because usually only the lowest modes are recorded, this method produces only an approximation of the flexibility matrix. By comparing the flexibility matrix of an undamaged structure to that of a flexibility of a damaged structure, damage can be detected. Like the previous method, analysis of the flexibility matrix is done on the complete response data limiting real time implementation.

Pandey and Biswas studied the flexibility matrices of simple analytical beam models and an actual wide-flange steel beam (Pandey and Biswas 1994). The flexibility matrix can be estimated from the lower frequency modes of the structure to allow for quick calculation time. Once the presence of damage has been established, full modal data can be collected and analyzed so the location of the damage can be determined. In both the analytical and experimental cases, changes in flexibility matrix corresponded to damage in the structure. Because both test cases were simple, the method needed to be tested on more complex models.

In 2000, Bernal and Gunes analyze the flexibility matrix to detect damage on the SHM benchmark problem (Bernal and Gunes 2000). First, they used the ERA with a Kalman Observer to identify eigenvalues and eigenvectors when input was known. In cases where the input was not measured, a Subspace Identification algorithm was used. The flexibility matrix was then computed from the modal parameters. The level of the structure where the damage occurred can be determined by computing a singular value decomposition of the change in the flexibility matrix. Although the results of this study match those of the benchmark study, the method can not be applied in real time.

SHM Methods Based on Using the Wavelet Method

Damage and the moment when damage occurred can be determined by spikes from the wavelet decomposition of the acceleration response data (Corbin et al 2000). The location of damage can be determined by patterns in the spatial distribution of the spikes. Wavelet analysis is an extension of the Fourier transform but with the ability to focus in on signal at particular locations. Because wavelet analysis can determine the presence of damage and the time of damage, the information can be used to give a better understanding of the cause of the damage. This method is limited in that it can not be implemented in real time.

Corbin et al applied a wavelet method to a 3 degree-of-freedom system, a cantilever beam, and the SHM benchmark problem (Corbin et al 2000). Wavelet analysis was applied to the simulation data from all three cases. In Case 1, random excitation with added noise was applied to the system and damage was modeled as completed loss of stiffness in certain parts of the system. Because the damage was large, it was easily detectable as spikes. In Case 2, harmonic excitation was applied to the beam and damage was simulated by decreasing rotation constraints between two elements of the Finite Element Method, FEM. Examination of the spikes of the acceleration data demonstrated the existence of damage. Case 3 used the acceleration data in the x direction of Case 3 with damage scenario 2 of the SHM benchmark problem. Spikes occurred at floors 1, 2, and 3 at the time of damage. The method used in the research identified the presence and location of damage. It still can not be implemented in real time because it needs the entire finite response to process and identify damage.

In 2000, Hou et al examined how noise in the acceleration data affected the ability of the wavelet method to detect damage (Hou et al 2000). The method is applied to simulation data from a single degree-of-freedom model subjected to a harmonic excitation and to real acceleration data from the 1971 San Fernando earthquake. A detectability map is proposed to study the effects of noise and damage severity on the ability to detect damage. It is shown that this method is not robust in the presence of strong noise and can be insensitive to small amounts of damage.

SHM Methods Based on Adaptive Identification

A SHM method that can continually monitor the integrity of a structure in real time is needed. Many techniques require that the entire response data be completed before analysis can begin. Adaptive methods are used to achieve real time results by continually monitoring the various states of a structure. In 2000, Loh et al used a fading Kalman filter technique to achieve real-time capability. Although the methods were able to detect damage, they were computationally costly.

The Least Mean Squares (LMS) and Recursive Least Squares (RLS) algorithms can be used to detect damage because they are both simple to implement and not computationally costly (Haykin 1991). The Adaptive Filters are digital filters that aid in signal processing. The filter's algorithm consists of two basic processes: a filtering process, which estimates the output given an input signal and generates the error in this estimation, and an adaptive process which adjusts the parameters in the filter depending on the error calculated (Haykin 1991).

Chase et al applied the LMS filter on the SHM benchmark problem to detect damage (Chase et al 2003a). The LMS algorithm is one of the most widely used of all the adaptive filtering algorithms because it is simple to implement (Haykin 1991). It is an approximation of the Steepest Descent Method using an estimator of the gradient instead and is used to track changes in the stiffness of the structure. When damage occurs, it is assumed that a member of the system has fractured or completely failed. This failure of a member directly correlates with a reduction is the overall stiffness of the system. In addition, damage will greatly change the stiffness of steel structures and can easily be modeled (Zimmerman and Kauk 1994).

The methods were tested on a simplified 4 degree-of-freedom version of the SHM benchmark problem, and the Case 1,3, and 4 with damage scenarios 1 thru 4 of Phase I of the SHM benchmark problem. Descriptions of the damage scenarios and cases are shown in Table 2.1 and 2.2. Noise was included in the data, but no modeling error was present. It was shown that in all cases that the method was able to identify the presence and location of damage and converge quickly.

In addition, Chase et al present a procedure for using the adaptive RLS filter on the SHM benchmark problem to detect damage (Chase et al 2003b). Although the RLS is more computationally complex then the LMS filter, it is a faster method. The filter contains self-adjusting coefficients that are used to model a noisy signal. The objective is to minimize the Mean Square Error (MSE) between the original noisy signal and the modeled signal.

The RLS method is applied to a simplified 4 degree-of-freedom version of the SHM benchmark problem, and the Case 1, 3, and 4 with damage scenarios 1 thru 4 of Phase I of the SHM benchmark problem. Noise is included in the data but no modeling error is present. The 4 degree-of-freedom model was used to verify the ability of the method to detect damage. All cases proved that the RLS method was able to identify presence and location of damage in real time. Although both the RLS method and the LMS method both identify damage in real time, the LMS method is less complex and easier to implement.

The LMS method has been shown to identify damage in Phase I of the SHM benchmark problem in real time. The method needs to be applied to a more complex system that closer resembles an actual scenario. On a real structure, both the actual mass and stiffness values will not match the design mass and stiffness values so the addition of modeling error replicates a more realistic situation. In addition while gathering data from accelerometers, noise will indirectly enter in the data record and need to be added to Phase II of the benchmark problem. In addition, the method needs to be expanded to find both the existence and location of damage.

PROBLEM STATEMENT

The main objective of the proposed research is to further evaluate the Adaptive LMS method's ability to model changes in stiffness parameters of a system so presence and location damage can be identified as applied to the SHM benchmark problem. The Adaptive LMS method is being studied because of its ability to model changes in stiffness in real time. The ability to detect this damage quickly, in real time, is of great importance so one can ensure the safety and integrity of the structure. Specific issues to be investigated are:

1. The evaluation of the effectiveness of the LMS method on a 12 and 120 degree-of-freedom systems and the comparison of these results to those of the 4 degree-of-freedom system.
2. The LMS method will be expanded to Phase II of the SHM benchmark problem. The effectiveness of the LMS method to identify damage in real time with the introduction of modeling error in the form of noisy data and uncertain structural parameters will be evaluated.

The LMS method has been applied to Phase I of the benchmark problem where it was successfully shown to be an accurate method that can be applied in real time (Chase et al 2003a). The method needs to be tested on more complex situations so it can eventually be used on real buildings to locate real damage. In order to test the method on more complex situations, the LMS method will need to be expanded to Phase II of the SHM Benchmark Problem where modeling error in the form of noisy data and uncertain structural parameters is introduced.

As the LMS method is tested on more complex situations, its ability to be used in real situations effectively and efficiently will be realized. Because the method is computationally simple and can be conducted in real time, the method will be of great use by the SHM professional community. Then structures can be properly instrumented to detect damage in real time thus helping in long term maintenance of buildings and identifying damage in catastrophic events.

RESEARCH STRATEGY

This study will focus on the use of the Adaptive LMS method as an efficient methodology for predicting changes in stiffness parameters in order to detect damage in steel frame structures. In order to validate the method, it was first tested on simple cases and then expanded to specific cases of the SHM benchmark problem. All model testing was implemented in MATLAB (MathWorks 2002). In addition, all tests use a sampling rate of 100 Hz. 100 Hz is used throughout the SHM benchmark problem because of the problem's setup. Data acquisition systems were used to record the structural responses. The system had anti-aliasing filters set to 100 Hz (Dyke et al. 2001).

Tracking Changes in Stiffness

Changes in the stiffness of structures have also been used to identify damage (Zimmerman and Kauk 1994). When damage occurs, it is assumed that a member of the system has fractured or completely failed. This failure of a member directly correlates with a reduction is the overall stiffness of the system. In addition, damage will greatly change the stiffness of steel structures and can easily be modeled. The equation of motion can be refined to include a change is stiffness as:

$$M \cdot \{\ddot{x}\} + C \cdot \{\dot{x}\} + (K + \Delta K) \cdot \{x\} = -M \cdot \ddot{x}_g \tag{4.1}$$

where *M*, *C*, and *K* are the mass, damping, stiffness matrices of the model, respectively, ΔK is the change of stiffness of the structure, $\{x\}$,{x}, and $\{x\}$ are the actual displacement, velocity, and acceleration vectors, respectively, and x_g is the ground motion acceleration (Chase et al 2003a).

In order to model the change in stiffness per degree of freedom, ΔK is defined with time varying scalar parameters, α_i. In a three story example, is defined as:

$$\Lambda K = \alpha_1 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \alpha_2 \begin{bmatrix} 1 & -1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \alpha_3 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix} \tag{4.2}$$

So the change in stiffness becomes:

$$\Delta \mathbf{K} = \sum_{i=1}^{n} \alpha_\iota \tag{4.3}$$

hence,

$$y_k = \sum_{i=1}^{n} \alpha_\iota x = \mathbf{F} - \mathbf{M} \cdot \ddot{x} - \mathbf{C} \cdot \dot{x} - \mathbf{K} \cdot x \tag{4.4}$$

which can be termed as y_k at each discrete time step, k (Chase et al 2003a).

Development of Adaptive Filter

The adaptive filter has been used to model the individual, scalar elements of the signal, y_k at each discrete time step, k (Chase et al 2003a). The filter estimates y_k by calculating n_k. The modeled value, n_k, is calculated from:

$$n_k = w_k^T x_k \tag{4.5}$$

where w_k is the adjustable filter coefficient vector or weight vector at time k, and x_k is the vector of the current and previous filter outputs. The modeled value, n_k, is compared to the y_k to calculate the Mean Square Error (MSE):

$$e_k = y_k - n_k \tag{4.6}$$

The adjustable filter coefficient vector of weight is updated each time step with the Widrow-Hopf LMS algorithm to minimize the error, e_k, and is defined as (Ifeachor and Jervis 1993):

$$w_{k+1} = w_k + 2\mu \cdot e_k \cdot x_k \tag{4.7}$$

The equation is solved at each time step to create a number of degree-of-freedoms by length of time matrix. If the stiffness of the system does not change, then y_k should be zero. It should be noted that any modeling error or noise in the data will be complied and will show up as change in stiffness. It is necessary to be able to separate the change in stiffness from damage and that from modeling error.

The LMS filter ability to model the noisy signal, y_k, depends on two variables, μ and *m*. Even though the various cases tested could be further refined by varying these variables, the parameters need to be fixed in a practical application. The variable, μ, is a positive scalar that controls the stability and rate of convergence of the filter. It was selected for each degree-of-freedom and was based on the range of displacement of the corresponding degree-of-freedom. For each damage scenario and degree of freedom, μ was selected based on trial and error methods. The individual μ used for each case can be seen in the MATLAB code in Appendix (MathWorks 2002). The number of taps, *m*, is the number of previous time steps' displacement that are included in calculating *n*. Even though by varying *m* the problem would be optimized, the number was fixed for practical application. Chase et al chose the number of taps, *m*, as 5 (Chase et al 2003a). In order to improve the results from Chase et al's study, the number of taps, *m*, was chosen to be 6.

A two-step and an one-step approach are available to model the output signal. The two-step method models n_k with multiple LMS filters for each degree-of-freedom. The two-step approach is fast, simple and robust to noise but can be computationally complex because it requires matrix solutions each time step. As the number of degree-of-freedom rises, the two-step method becomes difficult to solve so a method without a matrix solution is required. The one-step approach combines the two steps of modeling the noisy signal and solving for alpha without a matrix solution.

Sample Rate Issues

All tests use a sampling rate of 100 Hz. 100 Hz is used throughout the SHM benchmark problem because of the problem's setup. Data acquisition systems were used to record the structural responses. The system had anti-aliasing filters set to 100 Hz (Dyke et al. 2001). It should be noted that the convergence rate in adaptive LMS depends significantly on the sampling rate. Previous work has been done on analyzing the effect of the sample rate on the convergence rate (Chase et al 2003a). The 100 Hz sampling rate provides an adequate convergence rate in the SHM benchmark problem.

In addition, the effects of the frequency of the forcing function on capturing the acceleration data was addressed (Dyke et al 2001). Density spectrums of acceleration of the shaker mass and the acceleration responses from the SHM benchmark problem were analyzed. The spectrum of the shaker mass shows that the shaker is unable to input significant forces at low frequencies due to stroke limitation. Low Frequency noise is found in the density spectrums of the acceleration responses. The low frequency noise is due to the lack of significant force input in this region.

Single Degree-of-Freedom Model

As a proof of concept, the LMS algorithm was first tested on a one degree-of-freedom system. The model consisted of one floor of the four story model used in the SHM Benchmark problem. The first floor was further simplified to act as a lump mass model. Many excitation forces varying in complexity and modeling error were introduced to the problem to ensure the method could accurately predict damage regardless of complexity of input. The following input forces were used on the single degree-of-freedom system:

- Constant force of 100 kN
- Harmonic force of 100 sin (5t) kN
- Random generated force that ranged from 50 kN to 150 kN

The response of the model to each force was simulated using the MATLAB function ode45 with a sample rate of 100Hz (MathWorks 2002). Damage scenario1 in

Phase I of the benchmark problem was used and the scenario involves complete removal of all braces in the first floor. The overall stiffness of the single degree-of-freedom was reduced from 106.6 MN/m to 58.4 MN/m, a 45% reduction. The response data of the undamaged system was spliced with the response data from the damaged system. The combined simulation data was then processed by the filter to determine the presence of damage. In addition, modeling error and noise was added to each model. The modeling error was consistent with Phase II of the benchmark problem and consisted of:

- 10 % error in mass
- 5 % error in stiffness
- 10 % noised added to the acceleration data

The data with the error was then processed through the filter to determine the presence of damage.

The calculation of alpha on the single degree-of-freedom case was relatively simple to implement. From equation 4.4, *n* is substituted for *y* and the new equation is solved for alpha:

$$\alpha_k = n_k / \; x_k \tag{4.8}$$

Because the system is a single degree-of-freedom, the calculation is simple leaving alpha as column vector. More complicated situations will be discussed later.

Because alpha is calculated by dividing the filter signal, *n*, by the displacement, *x*, alpha is sensitive to near zero displacements that occur during sign changes. Where displacement oscillates around zero, it is necessary to filter out small displacements to ensure that alpha converges. The absolute maximum value of x_k was found and *g* becomes 70% of this maximum value.

$$g = 0.70 * \max(x(1,:)) \tag{4.9}$$

For the 3, 4, 12, and 120 degree-of-freedom model, the percentage was determined through a trial and error process. The percentage of the maximum value of x_k depends on the damage scenario and degree-of-freedom. For each time step, if x_k is less than g, then alpha for that time step is equal to the alpha from the previous time step. This method filters out the spikes in the alpha versus time graph so changes in alpha due to damage can be identified.

Four Degree-of-Freedom Model

To further refine the process of using the LMS adaptive filter, it was applied to a four degree-of-freedom model of the benchmark problem. The four degree-of-freedom model was used in Chase et al's work and comparison between the results of both studies could be made (Chase et al 2003a). The assumption was made that the structure in the benchmark problem was acting as a lump mass model thus each degree-of-freedom modeled consisted of an individual floor. So the four degree-of-freedom is a 2-D model of the building. The four degree-of-freedom model was tested to ensure that the interaction of the four floors was being captured in the model. The interaction of the four degrees-of-freedom could be seen in that damage to one story was seen in the adjacent degrees-of-freedom. Again, many excitation forces varying in complexity and modeling error were tested on the model. The same three forces from the single degree-of-freedom model:

- Constant force of 100 kN
- Harmonic force of 100 sin (5t) kN
- Random generated force that ranged from 50 kN to 150 kN

were applied to the system. The response of the model to each force was simulated using the sample rate of 100Hz, using MATLAB function ode45, the damage scenario 1 of Phase I, and the combination of the undamaged simulation data with the damaged simulation data were identical to that of the single degree-of-freedom model (MathWorks 2002). The combined simulation data was then processed by the same filter to determine the presence of damage. Also, the addition of the same modeling error and noise was added to the simulation data to test the ability of the filter to predict damage in the presence of error.

The calculation of alpha was more challenging to implement because of the interaction between degree-of-freedoms in the system. Equation 4.4 with the substitution of *n* for *y* in a four degree-of-freedom system at each time step becomes:

$$n_k = \begin{bmatrix} \alpha_1 + \alpha_2 & -\alpha_2 & 0 & 0 \\ -\alpha_2 & \alpha_2 + \alpha_3 & -\alpha_3 & 0 \\ 0 & -\alpha_3 & \alpha_3 + \alpha_4 & -\alpha_4 \\ 0 & 0 & -\alpha_4 & \alpha_4 \end{bmatrix} \cdot \begin{bmatrix} x_{1k} \\ x_{2k} \\ x_{3k} \\ x_{4k} \end{bmatrix} \tag{4.10}$$

Solving for alpha for each time step results in:

$$\alpha_{1k} = \frac{n_{1k} - \alpha_{2k}(x_{1k} - x_{2k})}{x_{1k}} \tag{4.11}$$

$$\alpha_{2k} = \frac{n_{2k} - \alpha_{3k}(x_{2k} - x_{3k})}{(-x_{1k} + x_{2k})} \tag{4.12}$$

$$\alpha_{3k} = \frac{n_{3k} - \alpha_{4k}(x_{3k} - x_{4k})}{(-x_{2k} + x_{3k})} \tag{4.13}$$

$$\alpha_{4k} = \frac{n_{4k}}{(-x_{3k} + x_{4k})} \tag{4.14}$$

Since alpha is calculated by various combinations of displacements, near zero denominators of alpha cause spikes to appear in alpha. To help alpha converge, near zero values were filtered out as discuss in Equation 4.9. The maximum value of the denominator of each alpha over the length of time is found. As discussed previously, individual alphas with corresponding denominators below a certain percentage below the maximum value are filtered out. The percentage was determined through a trial and error process and depended on the damage scenario and degree-of-freedom. The percentage ranged from 70 % to 85 %. For each time step, if the denominator of alpha at a particular time is less than the certain percentage of the maximum value of *x*, then the alpha for that time step is equal to the alpha from the previous time step. This method filters out the spikes in the alpha versus time graph so changes in alpha due to damage can be identified.

Three Degree-of-Freedom System

In order to investigate the effects of translational and rotational degrees-of-freedom on the filter, a three degree-of-freedom model was constructed. The model is a 3-D portal frame and consisted of the translation in the x and y directions and the rotation of the first floor of the SHM benchmark problem. Again, many excitation forces varying in complexity and modeling error were tested on the model. The same three forces as before were used but the forces had to be applied to all degrees-of-freedom in order to simulate response. The third degree-of-freedom is in the rotational direction and is measure in radians. Because the output of the rotational degree-of-freedom is small in comparison to the response of the other two degrees-of-freedom, the force applied to the rotational degree-of-freedom needed to be reduced by an order of magnitude. Damage scenario 1 from the benchmark problem consists of removal of all bracing so the stiffness in the x, y, and rotational directions will be reduced. The stiffness in the x-direction is reduced by 45%, the stiffness in the y-direction is reduced by 71%, and the stiffness in the rotational direction is reduced by 65% (Johnson et al 2000). The other variables including the sample rate of 100Hz, using MATLAB function ode45, and the combination of the undamaged simulation data with the damaged simulation data were identical to that of the single degree-of-freedom model (MathWorks 2002). The combined simulation data with and without the addition of error was then processed by the same filter to determine the presence of damage. The calculation of alpha was similar to that on the single degree-of-freedom system.

Phase I: 12 Degree-of-Freedom Model

In order to compare results with Chase et al, a 12 degree-of-freedom model, a 3-D four story portal frame, was constructed and tested using simulation data from Phase I of the SHM benchmark problem (Chase et al 2003a). The process for performing structural health monitoring is illustrated in the flowchart in Figure 4.1. Acceleration data from the various damage scenarios was simulated by running the MATLAB program, datagen.m, which was downloaded from the IASC-ASCE SHM task group website (IASC-ASCE SHM Task group 2004). Acceleration data is gathered from 16 accelerometers, two in each the x and y directions per floor. Each accelerometer is located along the perimeter of the frame in the middle of bay it is located in. Per floor, the x-translational acceleration in the middle of the floor is found by averaging the two x-translational accelerations along the perimeter of the frame.

$$\ddot{x}_x = \frac{\ddot{x}_1 + \ddot{x}_2}{2} \tag{4.15}$$

The y-translational acceleration in the middle of the floor is calculated in the same manner. The rotational acceleration of each floor is computed as the difference between the two x-translational accelerations on that floor divided by the distance between the two x-translational accelerations which is 2.5 m (Chopra 2001).

$$\ddot{x}_\theta = \frac{\ddot{x}_1 - \ddot{x}_2}{2.5} \tag{4.16}$$

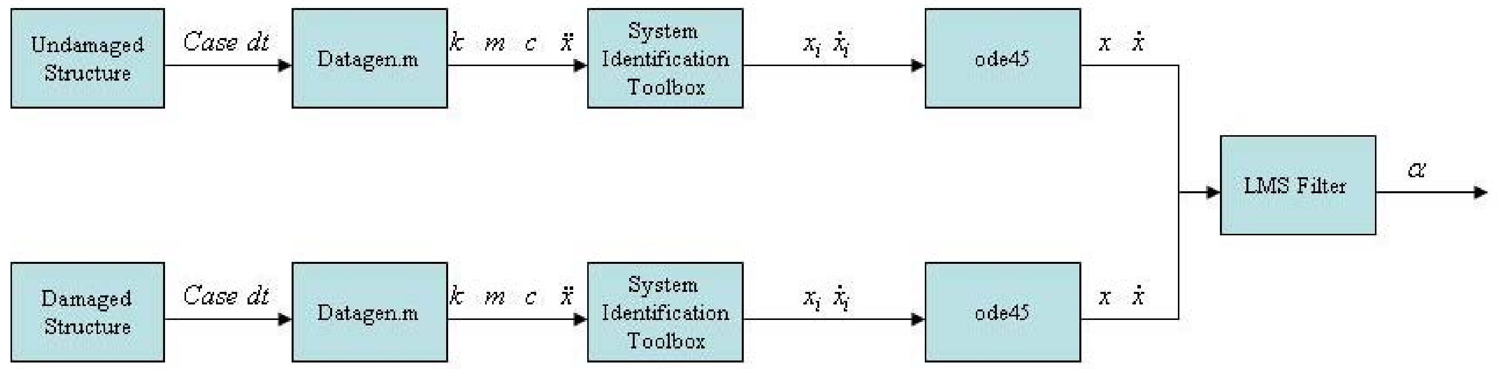


Figure 4.1. Flowchart of SHM of 12 Degree-of-Freedom System

The initial values of velocity and displacement of the simulated data are unknown because the acceleration record was started in the middle of the disturbance. Even with simple numerical methods, displacements without known initial values can be difficult to calculate from the acceleration data. In order to generate velocity and displacement data from the given acceleration data, the System Identification Toolbox from MATLAB was used to estimate the velocity and displacement data (MathWorks 2002). The Toolbox allows you to build and evaluate linear models of dynamic systems from measured input-output data.

With the initial values of displacement and velocity known, a similar method as used in the single, three, and four degrees-of-freedom systems was used to determine the entire displacement and velocity responses. In the previous cases, the forcing function was designated, but in this case the forcing function is given in the benchmark problem. The velocity and displacement of the undamaged and damaged model was simulated using the MATLAB function ode45 with a sample rate of 100Hz. Because the displacement data was generated with so many unknowns, additional error was introduced into the system. With the advancement of GPS sensors, the ability to gather acceleration and displacement data is possible thus eliminating the need to simulate data. With a GPS sensor both the acceleration and displacement data is know so simple numerical methods can be used to solve for the velocity data.

The response data for both the undamaged case and the damaged cases were then combined so the LMS filter could process the data. The alphas for each degree-of-freedom are calculated in a similar manner as the alphas in the four degree-of-freedom model. The individual equations for alpha are solved and can be seen in the MATLAB code in Appendix A. The process for

This process was completed on Case 1, 3, and 4 with damage scenarios 1 thru 4. The data for the self-generated data with and without data was also processed through the filter. The data was then processed to identify the presence and location of the damage scenarios.

Phase II: 120 Degree-of-Freedom Model

Phase II of the SHM benchmark problem introduces modeling error to the problem. Because only the 120 degree-of-freedom model is used, the model needs to be simplified to the 12 degree-of-freedom case. This is a relatively easy progress because the acceleration data is gathered from the same geometry of accelerometers as before. In addition, the mass and stiffness matrices for the reduced 12 degree-of-freedom system were simply the first 12 by 12 matrix from the full 120 degree-of-freedom model. So similar methods from Phase I were used to process the data in Phase II. Self generated response data was imported into the System Identification Toolbox and the models generated were used to obtain velocity data from the benchmark's acceleration data. The displacement data was again estimated by setting the equation of motion to zero. The filter was then used to process the data to identify the presence, severity and location of damage.

RESULTS

Single Degree-of-Freedom Results

The single degree-of-freedom results show that the LMS approach accurately and quickly tracks the change in alpha. The results for the constant force without error are shown in Figure 5.1. This basic case is the benchmark for the results in all the phases and cases. The approximation converges in 2.5 seconds to the actual change and shows a 45 % change in alpha.

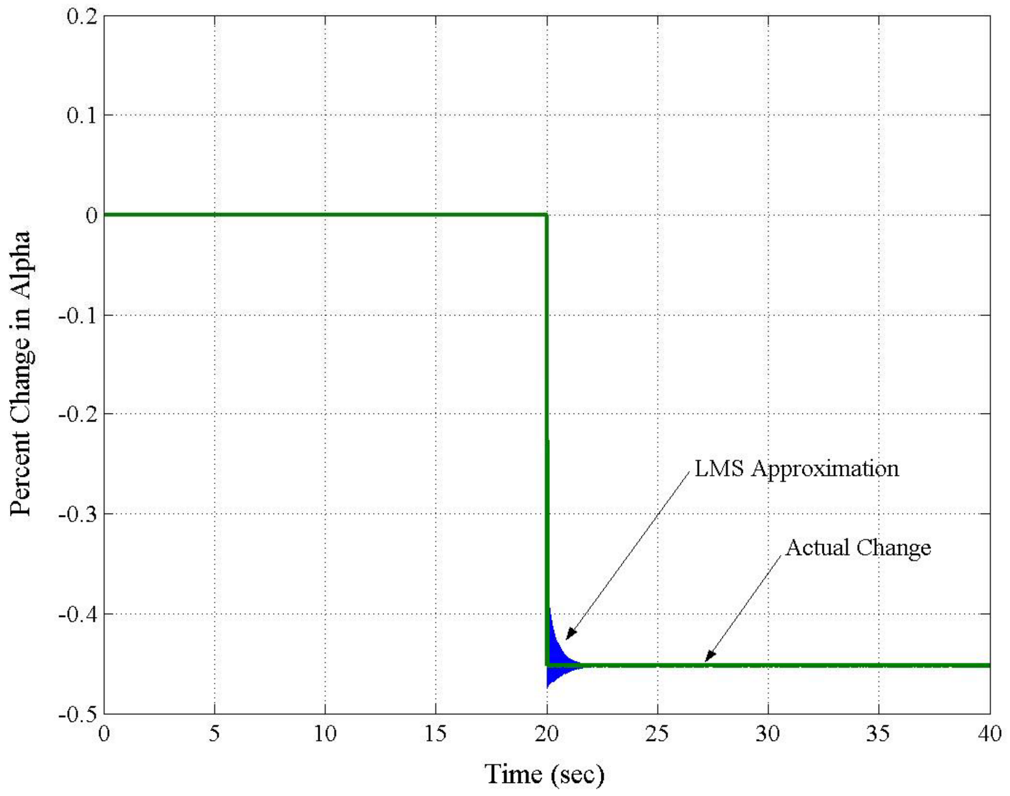


Figure 5.1. Single Degree-of-Freedom System with Constant Force and No Error

Phase II of the benchmark problem introduces modeling error into the problem of with a 10% error in mass, a 5% error in stiffness, and 10% noised added to the acceleration data. In order to test the effects of the method in the presence of error, error was added to the simple case of the single-degree-of-freedom system as shown in Figure 5.2.

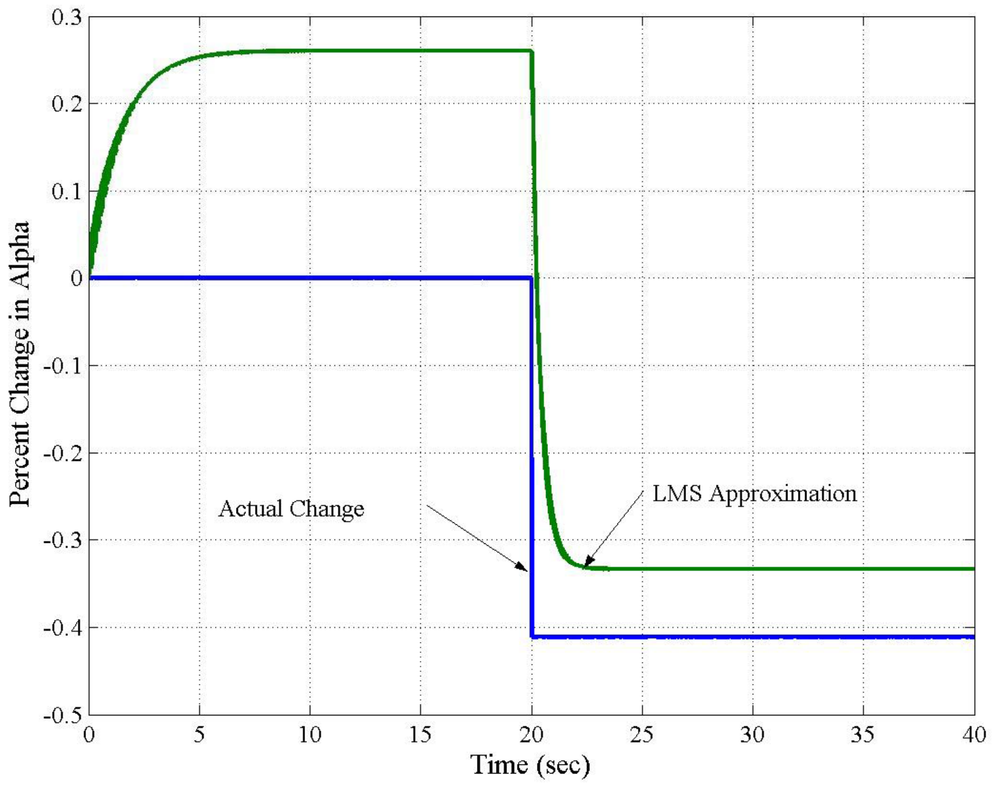


Figure 5.2. Single Degree-of-Freedom System with Constant Force and Added Error

As shown in the graph, the approximation does not match the actual change in alpha. The baselines developed with no damage present are different between the case with and without modeling error and noise. The overall step change of each line needs to be compared instead of the final percent change in alpha. A sharp change, which can be seen in each line, indicates change in stiffness. The LMS approximation shows a net change of 59 % in alpha compared with the actual change being 42 %. The error in the system adds to the overall change in alpha but what is important to note is that there is a change in alpha. This principle will be used in the more complicated cases in Phase II. Even though there is modeling error added, the LMS filter is able to identify damage in real time.

A random force was used to excite the single degree-of-freedom system to test the Adaptive Filter's application to random excitations. The results from the random force with no error are shown in Figure 5.3.

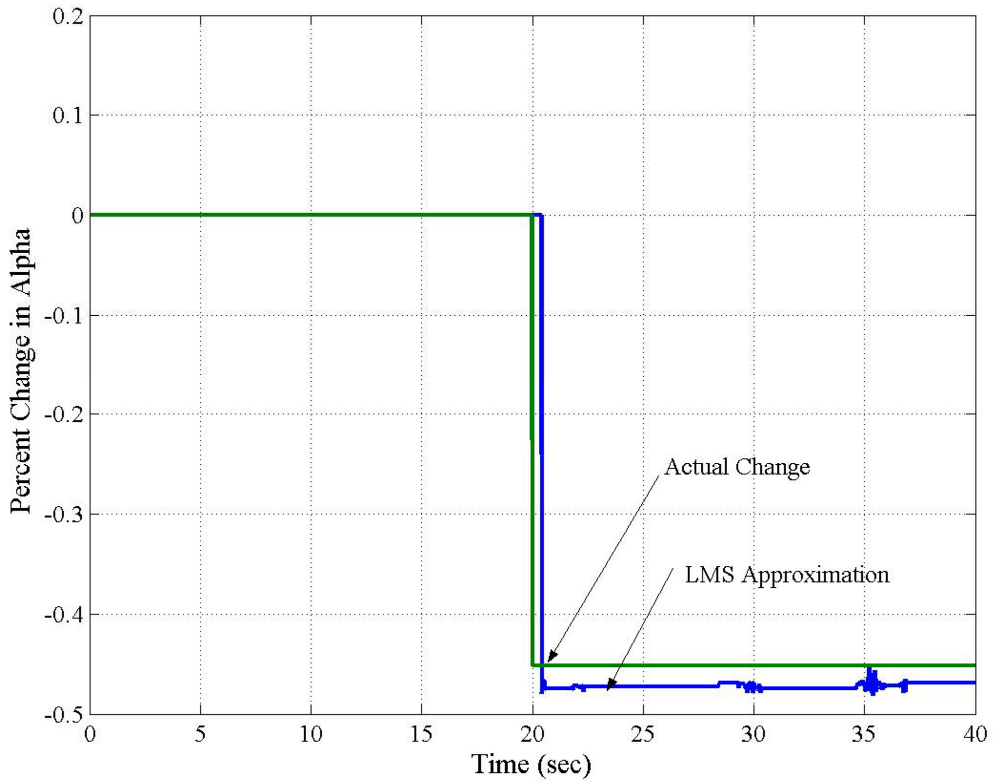


Figure 5.3. Single Degree-of-Freedom System with Random Force and No Error

The approximation does not converge to the same number as the actual change although it does converge. The change in alpha in the LMS approximation is 47 % compared with the actual change of 45 % in alpha. This slight difference can be accounted by the variability in the random force used to excite the system. These results confirm that the Adaptive Filter application does accurately and quickly tracks changes in alpha and thus stiffness. The remainder of the results can be seen in Appendix B.

Four Degree-of-Freedom Results

The results from the four degree-of-freedom system show how damage to one floor is reflected. The results from the case with constant force and no error are shown in Figure 5.4. Only α_1 changes because the damage only occurred to the first floor. The results converge to 45 % change in alpha in 2.6 seconds and have the same pattern as shown in Figure 5.1. The results confirm that if damage is confined to a story then only the alpha representative of the story should change. The remainder of the results is shown in Appendix B. This principle will be shown in more complicated cases in Phase II.

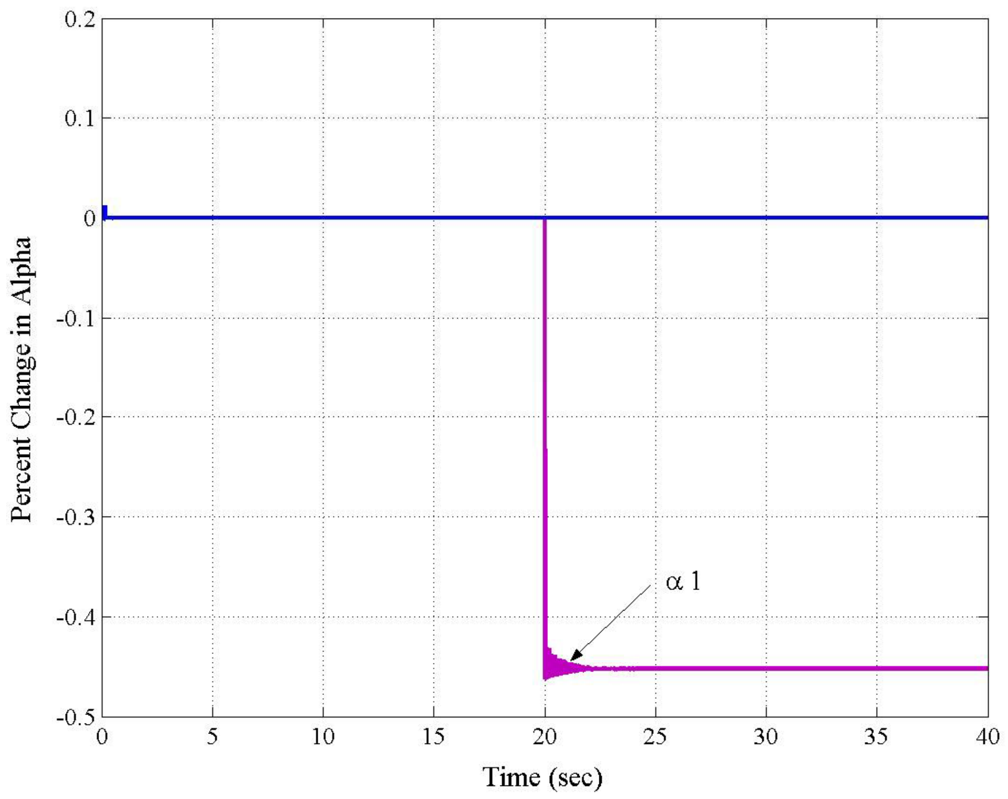


Figure 5.4. Four Degree-of-Freedom System with Constant Force and No Error

Three Degree-of-Freedom Results

The results for the three degree-of-freedom system show that change in the three directions, x, y, and translational, can be tracked by the adaptive filter method. Figure 5.5 shows the result of using constant force with added error. All three alphas correspond with the percent change in stiffness in the given direction. As discussed earlier, the added error in mass, stiffness, and signal causes the baseline of the approximated results to differ from the actual results. The net change in alpha is the important number to look at. The alpha in the first degree-of-freedom changes 47 %, the second alpha changes 74 %, and the third alpha changes 69 %.

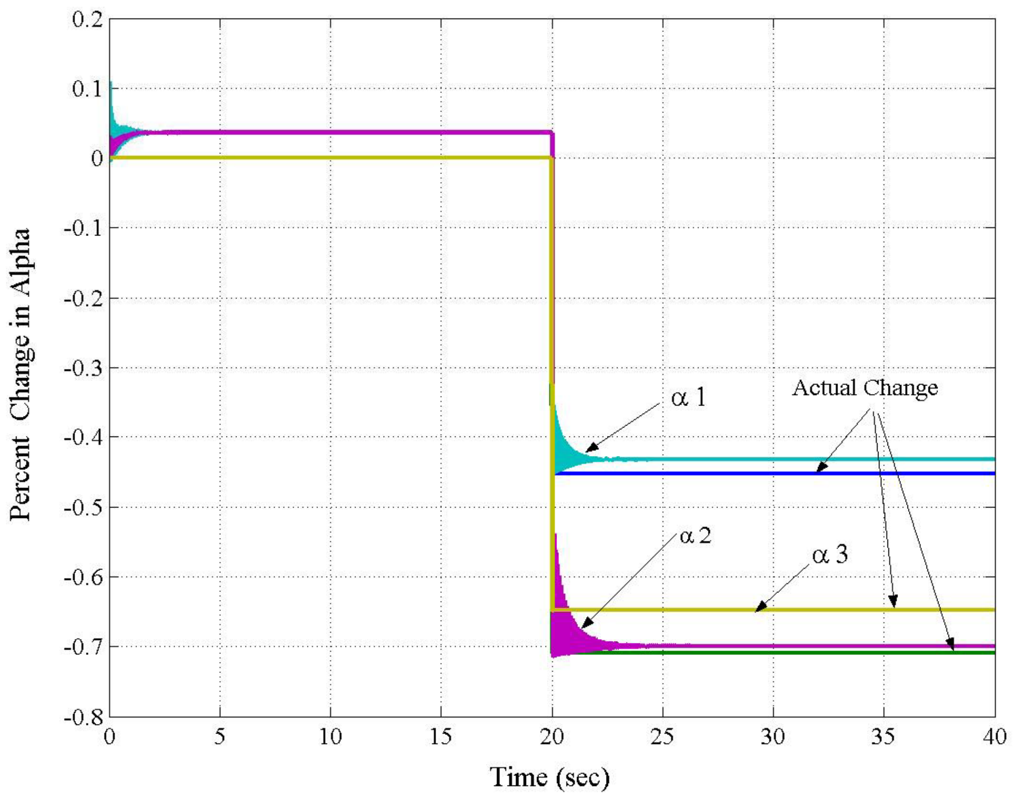


Figure 5.5. Three Degree-of-Freedom System with Constant Force and Added Error

Figure 5.6 shows the results from a random force with no error. It should be noted that as the system becomes more and more complicated and more noise is added, the approximation will not converge to a single number but varies around a single number. By filtering out the near zero response, the band of variation of the approximation narrows but it still does not converge. The main thing that is determined from the graph in the percent change in alpha that can be estimated even with variance in the line. The alpha in the first degree-of-freedom changes 48 %, the second alpha changes 72 %, and the third alpha changes 65 %. Again the LMS filter is predicting the change in stiffness in the three degree-of-freedom system in real time and with accurate results. The important trend to watch for is a sharp change in alpha that predicts the presence of damage to the structure. The remainder of the results can be seen in Appendix B.

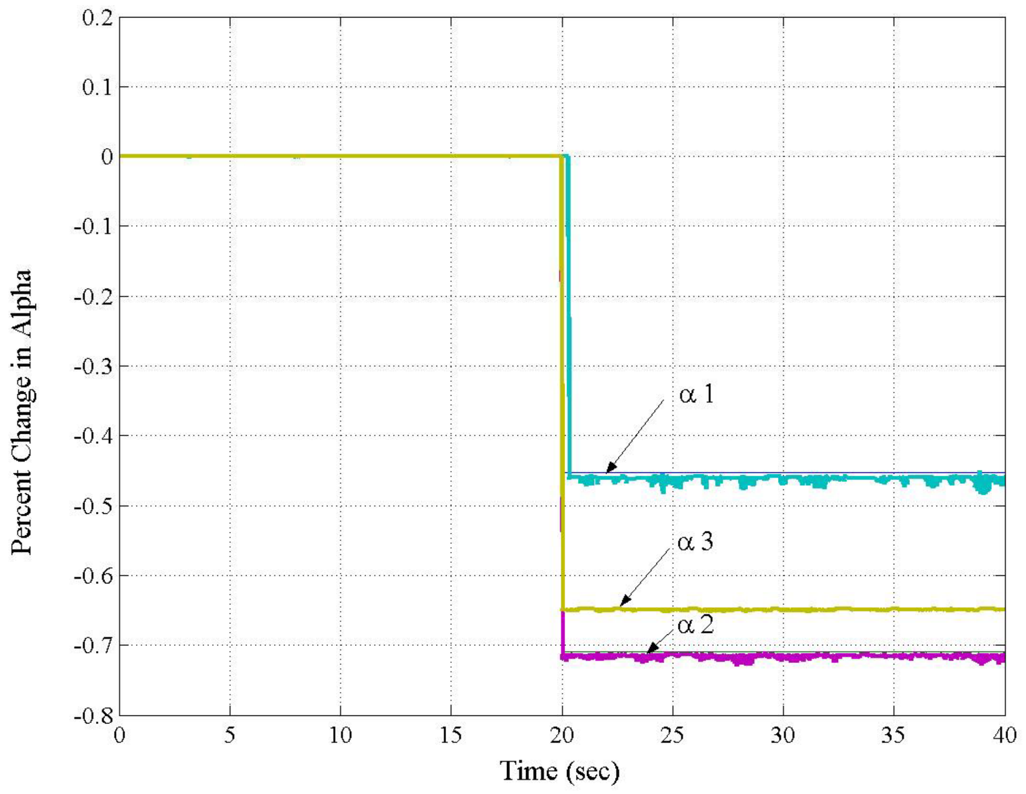


Figure 5.6. Three Degree-of-Freedom System with Random Force and No Error

Phase I: 12 Degree-of-Freedom Results for Case 1

Case 1 involves excitation on all stories in one direction. Damage Pattern 1 is the complete removal of bracing on the first floor and the results are shown in Figure 5.7. It was expected that damage would be shown in the change in alpha 1, 2 and 3 because they correspond to the first floor of the model. The alpha in the first degree-of-freedom changes 45 %, the second alpha changes 48 %, and the third alpha changes 65 %. Because the bracing is completely removed, the percent change in alpha is significant and can be seen even with the inclusion of error. Damage Pattern 2 involves the complete removal of bracing on the first and third floor and the results are shown in Figure 5.8. The alpha in the first degree-of-freedom changes 45 %, the second alpha changes 51 %, the third alpha changes 65 %, the seventh alpha changes 45 %, the eighth alpha changes 60 %, and the ninth alpha changes 65 %. The results show that the filter can indicate damage in the form of changes in α for multiple stories even though they are not adjacent stories. All results converge within 2 seconds and give indication of where damage has occurred.

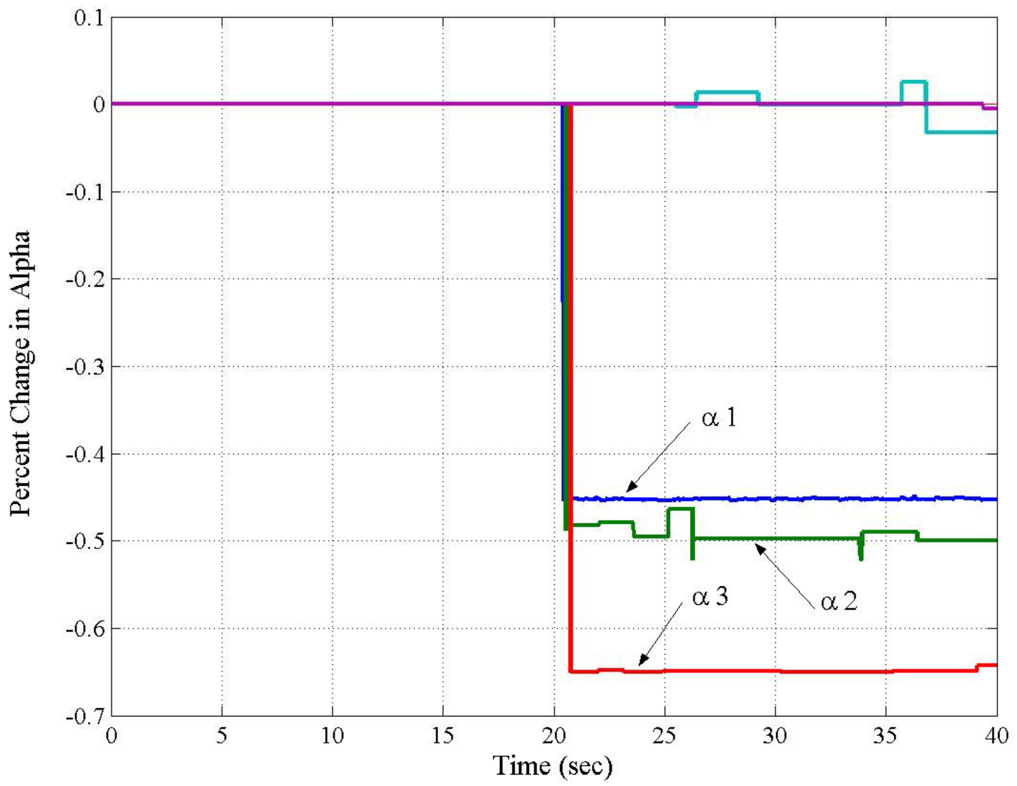


Figure 5.7. Phase I: Case 1 with Damage Pattern 1

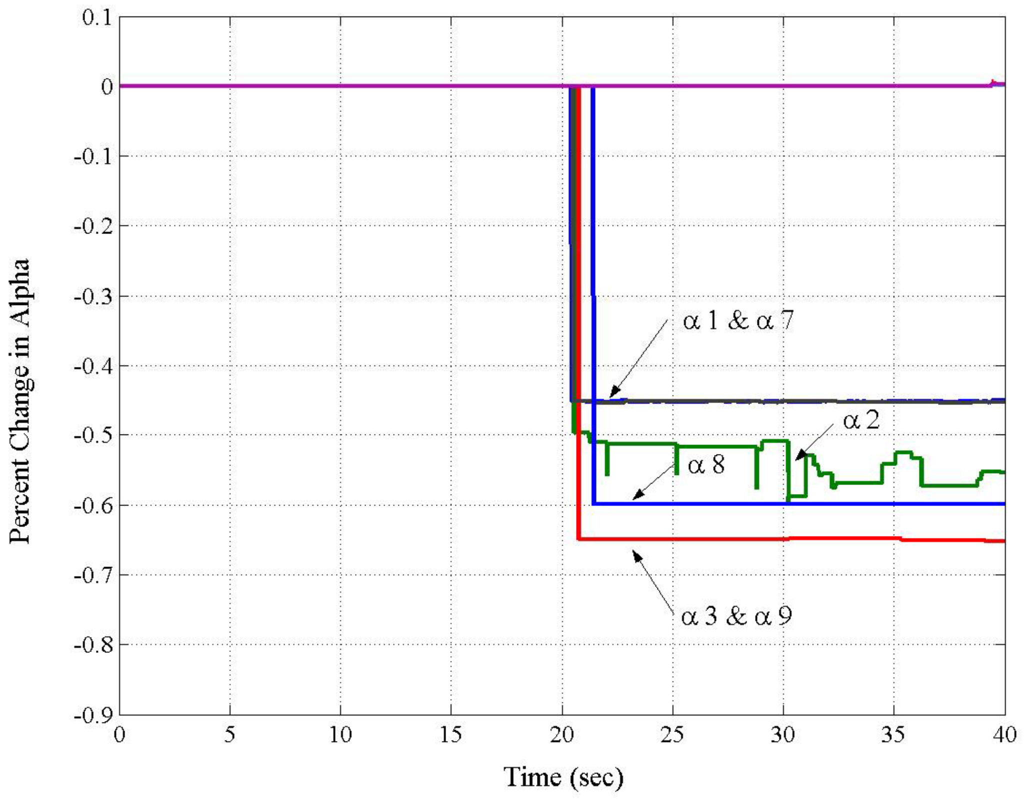


Figure 5.8. Phase I: Case 1 with Damage Pattern 2

Damage Pattern 3 and 4 involve the partial removal of bracing on the first and first and third floors respectively. Because the bracing is only partially removed, the percent change in alpha is significantly less than the change in alpha of damage pattern 2. The results for Damage Pattern 3 are shown in Figure 5.9. Because the loading in Case 1 occurred in one direction, the response in the other two directions is small in comparison. In Figure 5.9, only the change in the y direction can be seen. Also in comparison to the y response, the rotational response is incredibly small. In the presence of large error, damage in the rotational direction is hard to estimate. Damage is hard to find in this case but the filter does shown a reduction in overall stiffness of the 2nd degree-of-freedom to be 12 %. The remainder of the results can be seen in Appendix B.

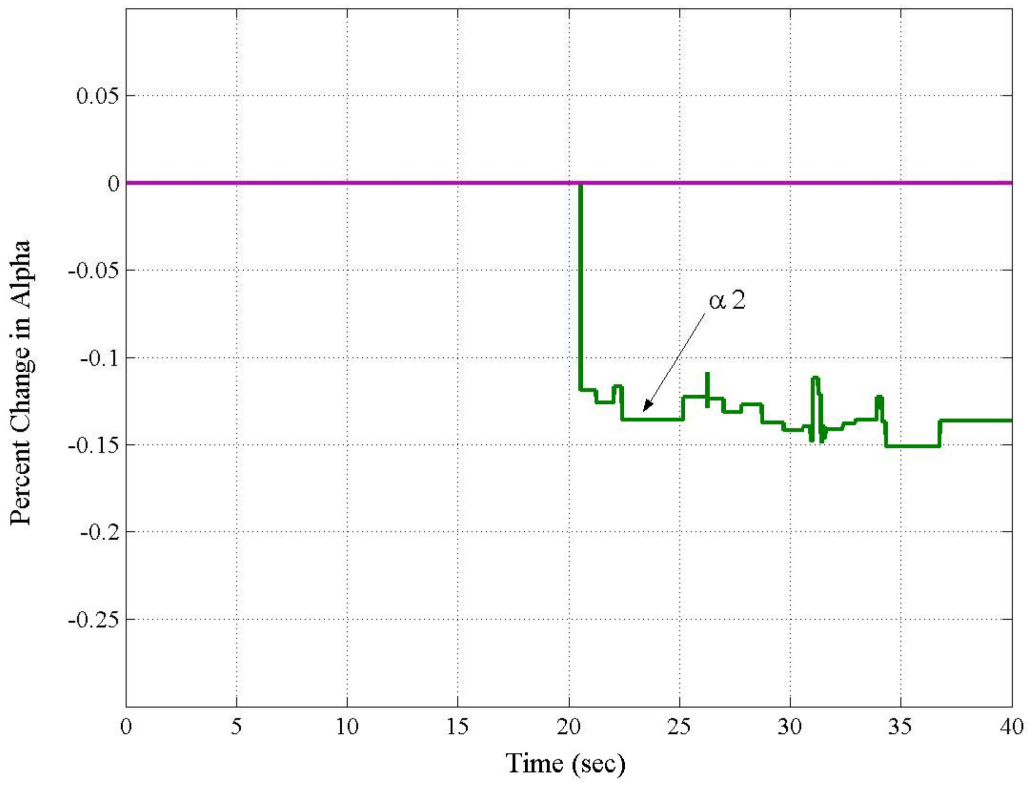


Figure 5.9. Phase I: Case 1 with Damage Pattern 3

Phase I: 12 Degree-of-Freedom Results for Case 3 and Case 4

Case 3 data is generated by exciting the structure only on the roof in the diagonal direction. In addition, the loading in Case 3 is symmetric where the loading in Case 4 is asymmetric. Because all degrees-of-freedom are excited by the diagonal shaker, the responses in all the degrees-of-freedom for Case 3 and 4 are larger then those in Case 1. Figure 5.10 shows the result from Case 3 with Damage Pattern 2. The change is alpha corresponds to the damage on the 1st and 3rd floors with the removal of all braces on those floors. With response data that is larger in magnitude, changes in α are easier to identify because near zero response are less frequent.

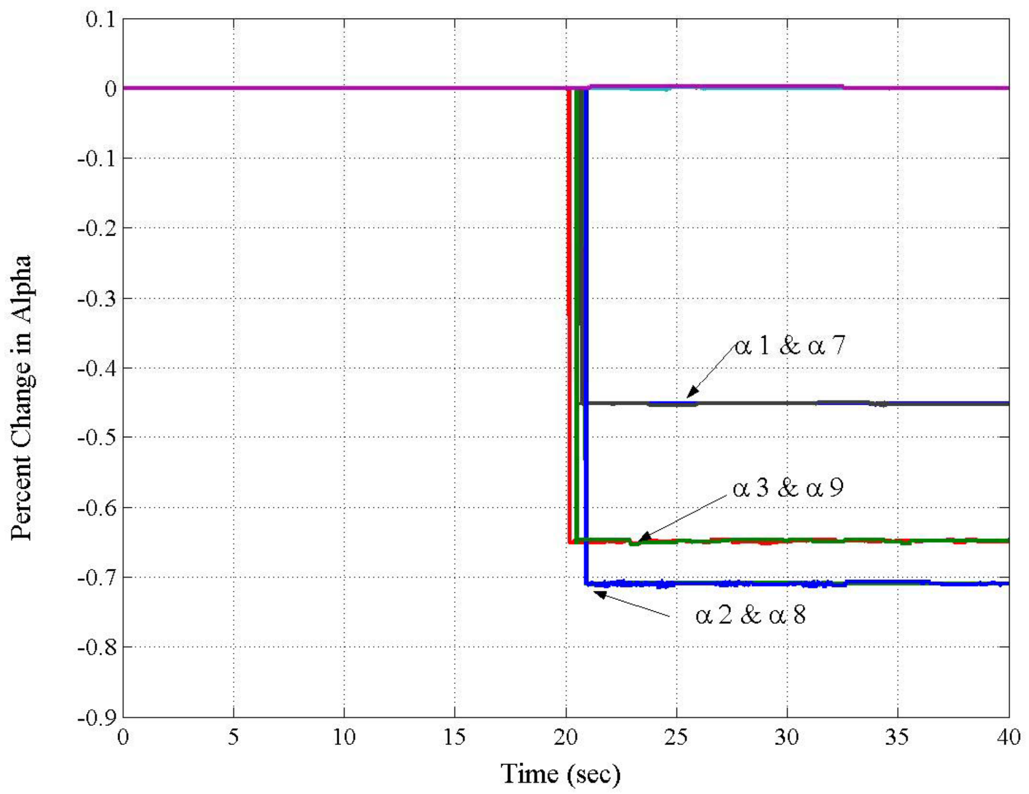


Figure 5.10. Phase I: Case 3 with Damage Pattern 2

The alpha in the first degree-of-freedom changes 45 %, the second alpha changes 71 %, the third alpha changes 65 %, the seventh alpha changes 45 %, the eighth alpha changes 71 %, and the ninth alpha changes 65 %. Figure 5.10 shows such a distinct change because of the form of excitation and the large amount of damage occurring. In all the six degrees-of-freedom where damage has occurred, the predicted results match and converge in less than a second to the actual results. The damage is harder to identify in damage patterns where partial damage occurs. Figure 5.11 shows the results for Case 3 with Damage Pattern 4 where there is partial removal of the bracing on the 1st and 3rd floor in the y direction. Because of the small change in alpha, change from damage is harder to identify than the change from error. Even though the approximations do not converge to a single number, the presence of damage in the y direction can be detected from the graph. The change in the alpha in the second degree-of-freedom is 19 % and change in alpha in the eighth degree-of-freedom is 12 %. Results for Case 4 are similar to those of Case 3 because of the similar force being applied to the system. The results can be seen in Appendix B.

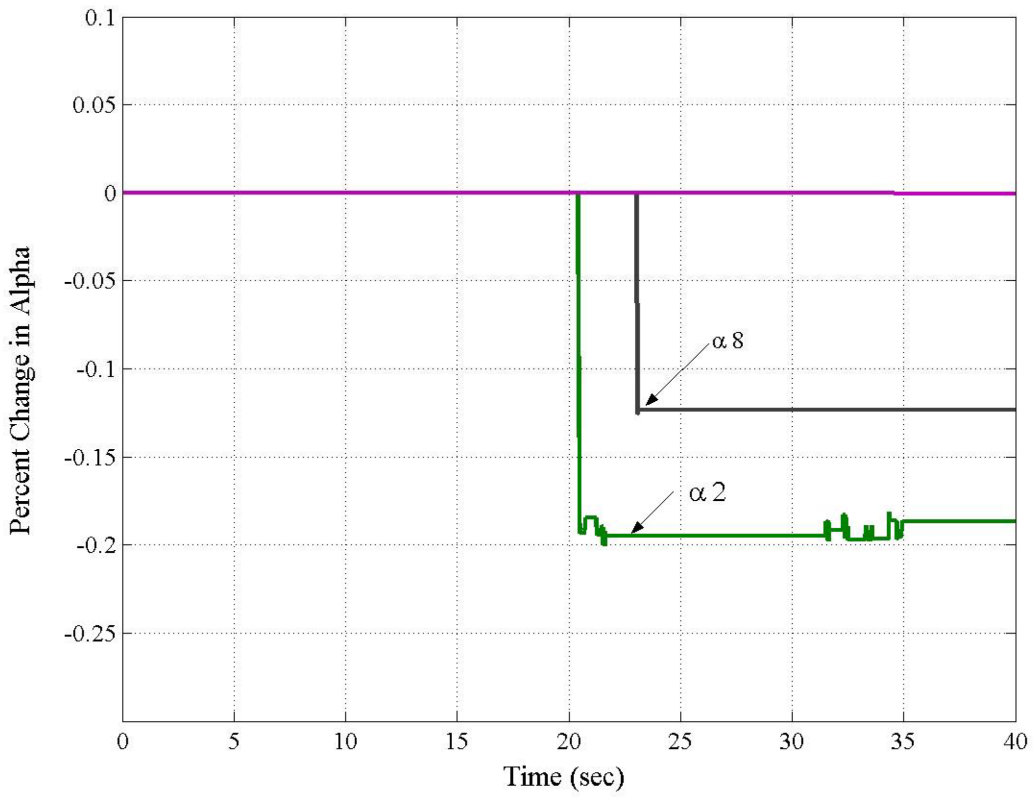


Figure 5.11. Phase I: Case 3 with Damage Pattern 4

Phase II: 120 Degrees-of-Freedom Results

Phase II of the Benchmark Problem only deals with the 120 degree-of-freedom model and includes more complexities than Phase I. Also modeling error of up to 10 %

of mass values and 5 % of stiffness values is added to the problem. The modeling error varies within the data set. The same degree of noise in the acceleration data from Phase I is included. This allows the benchmark problem to reassemble more realistic test situations.   There are four damage scenarios for the fully braced case, three for the partially braced case, and two blind cases. The cases for the partially braced case involve loss of rotational stiffness and were not tested.

In all scenarios when the alpha was monitored for all degrees-of-freedom, change due to damage can only be identified if the change in alpha is greater than the noise data. This can be seen in Figure 5.12 for damage pattern Braced 1 with symmetric loading. The damage in the first degree-of-freedom can clearly be seen because it is larger than 10% change in alpha. Below the 10 % change in alpha mark, the change in alpha is due to noise and error in the system It shows that there is a threshold at which it is hard to distinguish between various changes in alpha. This threshold has been determined to be about 10 % in this situation. In this benchmark problem, the 10 % change in alpha correlates roughly with 50 % loss of stiffness on two braces on a floor. When looking at the graph of change in alphas for the entire system, it is hard to distinguish damage less than 50 % loss of stiffness on two braces.

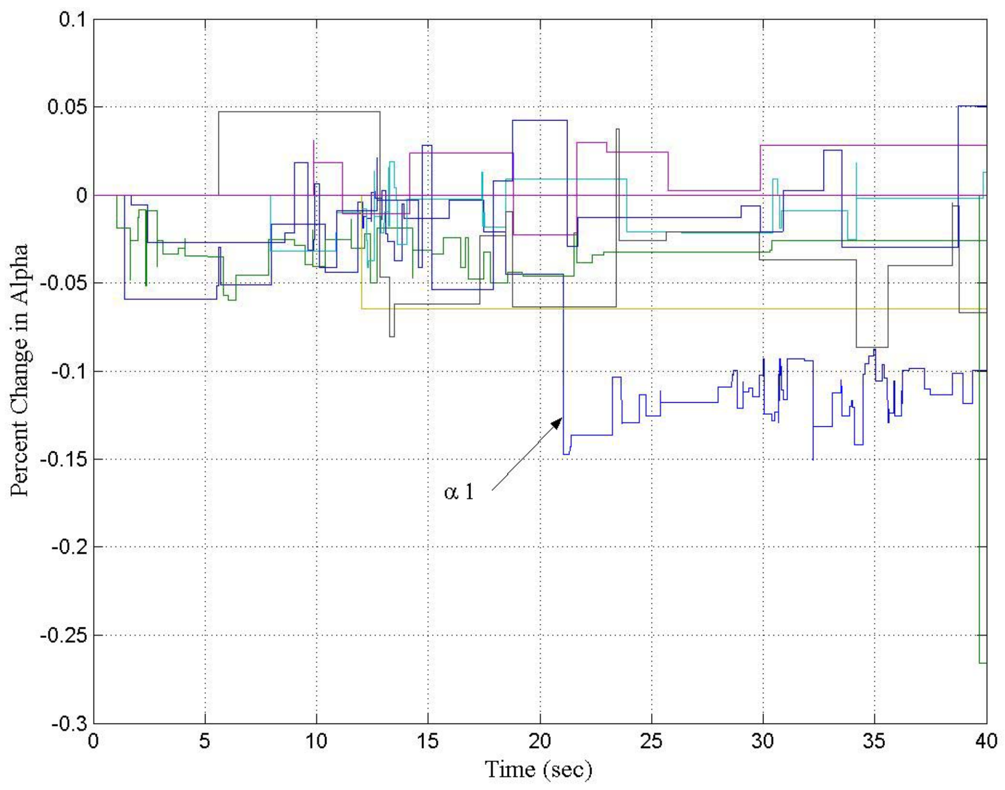


Figure 5.12. Phase II: Damage Pattern for Braced 1 with Symmetric Loading

Now if individual alphas were monitored separately, then changes due to damage that are less than the 10 % threshold could be detected if the time of damage is known. Figure 5.13 demonstrates the 10 % threshold where degrees-of-freedom one through three for the damage pattern of Braced 2 with symmetric loading is monitored. Because the time of damage is known to be at 20 seconds it is easy to identify damage in second degree-of-freedom as 7 % change in alpha because of the sharp change at that time. In Figure 5.14, damage due to noise at a time other than 20 seconds is shown. The change in alpha in Figure 5.14 mirrors that of Figure 5.13, but does not occur at the time of damage so it can be assumed not to be damage. In situations of catastrophic failure, the time of failure is known so sudden failure can be detected. In terms of long term maintenance when the time of failure is uncertain, the ability to identify damage can be more difficult. Complete failure of a member is easy to detect at any time but partial failure can be hard to distinguish. In this benchmark problem, damage less than 50 % of a brace is hard to capture. That specific threshold will vary depending on the system.

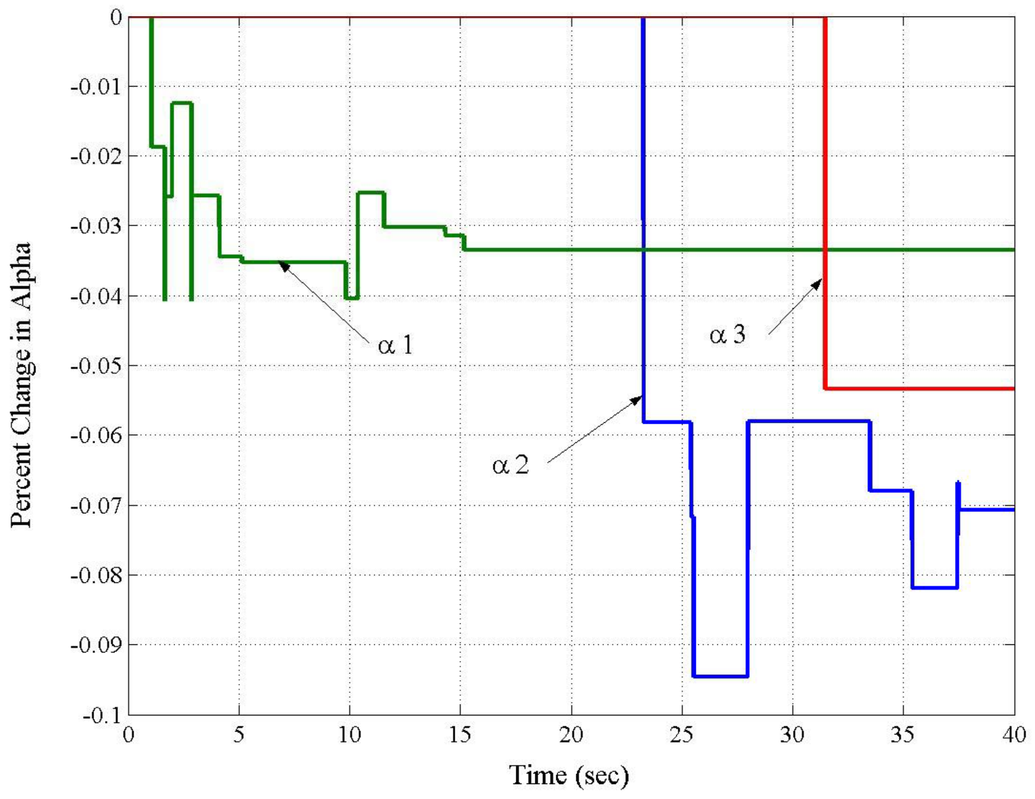


Figure 5.13. Phase II: Damage Pattern for Braced 2 Showing Damage on 1st Floor

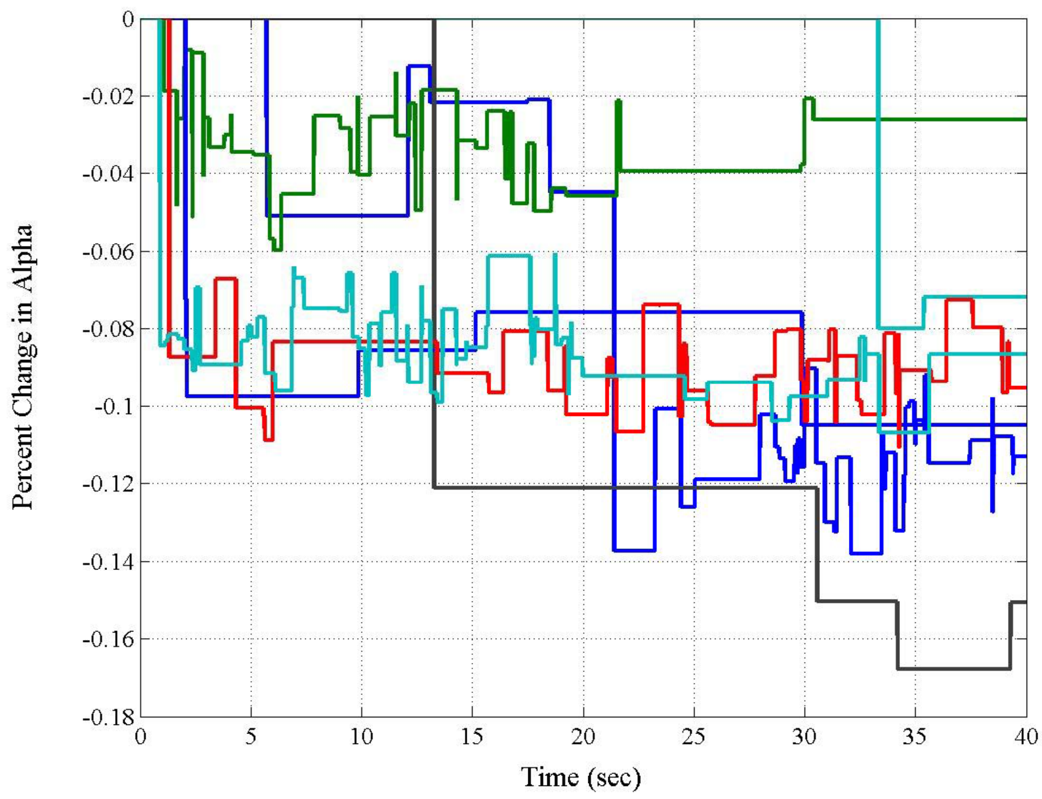


Figure 5.14. Phase II: Damage Pattern for Braced 3 Showing All Alphas

In addition, two blind cases were included in Phase II of the benchmark problem. In both blind cases, neither the loading nor the amount of error or noise was known. The results for the 1st blind case showing just the damage on the 1st floor are shown in Figure 5.15. The graph shows damage in the 2nd degree-of-freedom which looks very similar to the damage seen in Figure 5.12 and has a change of 12 %. It can be assumed that the blind case 1 involves around 50 % loss of 2 braces on the 1st floor in the strong direction. If the filter is tested with various data from various damage patterns, then filter can be used to estimate what type of real damage has occurred to the structure. The remainder of the results is shown in Appendix B.

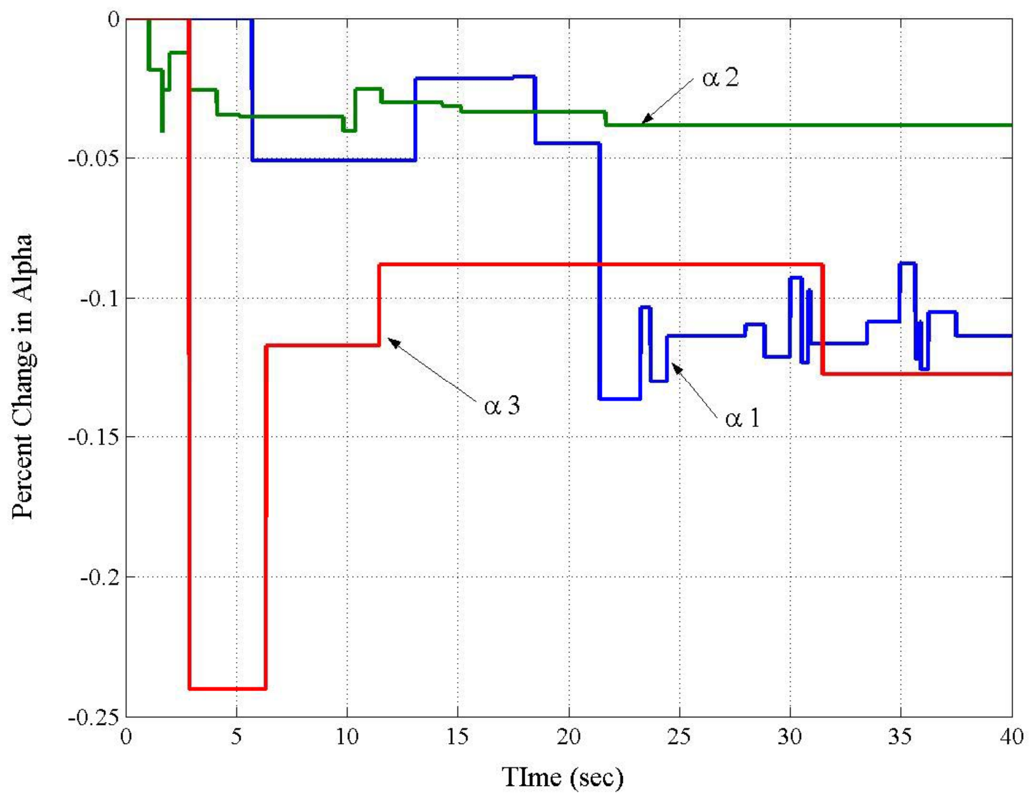


Figure 5.15. Phase II: Blind Damage Pattern 1 Showing Damage on 1st Floor

CONCLUSIONS

Conclusions

This thesis presents a Structural Health Monitoring technique for civil structures using adaptive Least Mean Square filtering theory. The method was applied to Phases I and II of the SHM benchmark problem. The method was applied to Phase I to confirm previous research that had been completed by Chase et al (2003). Phase II was developed in order to make the problem resemble a more realistic situation by introducing modeling error and noise. The robustness and adaptively of the adaptive LMS filtering method was tested by applying the method to Phase II of the benchmark problem. The results show that the adaptive LMS filtering method can be an effective tool in identifying damage in real time depending on the situation.

It was shown that for Phase I the method was able to identify damage in real time effectively. As stated previously, showing the results for Phase I was verification of the work done by Chase et al (2003). Phase I is a simpler case than Phase II because the damage scenarios were only for a 12 degree-of-freedom model and contained no modeling error or noise. In replicating the results from Phase I, a difficultly in monitoring changes in the rotational degree-of-freedoms was found. Monitoring changes in the rotational degree-of-freedoms is difficult because the rotational response is incredibly small in comparison to the *y* response. As a result in the presence of large error, damage in the rotational direction was hard to identify. In order to overcome this difficultly,

Phase II introduced modeling error and noise to the benchmark problem. In addition, the damage scenarios involved partial damage as opposed to complete damage of bracing in the structure found in Phase I. With the combination of noise and small damage, identifying damage became more difficult. This research determined that there is a threshold at which various changes in alpha are hard to distinguish. The change in alpha of a particular degree-of-freedom directly correlates to change in the overall stiffness of that degree-of-freedom. If damage in a degree-of-freedom results in a change of stiffness below 10 % of the overall stiffness of that particular degree-of-freedom, then it hard to determine is the change is due to damage or modeling error. In this benchmark problem, the 10 % change in alpha or stiffness correlates roughly with 50 % loss of stiffness on two braces on a floor. When looking at the graph of change in alphas for the entire system, it is hard to distinguish damage less than 50 % loss of stiffness on two braces. The threshold and its correlated damage level changes depending on the structure being monitored.

If individual alphas were monitored separately, then changes due to damage could be detected if the time of damage is known. Because the time of damage in the benchmark problem is know, it is easy to identity changes in alpha due to damage. In situations of catastrophic failure, the time of failure is known so sudden failure can be detected. In terms of long term maintenance when the time of failure is uncertain, the ability to identify damage can be more difficult. The LMS filter can definitely detect damage in real time when the time of damage is known and the damage results in more than 10% loss of stiffness of the individual degree-of-freedom being monitored.

Two blind cases were also included in Phase II of the SHM benchmark problem. No information regarding type and time of damage was given regarding the blind cases. Because various damage patterns had been run beforehand, the results from the other damage scenarios were compared with the results from the blind test cases to determine location and severity of damage. The results from the blind case matched that of the results from a previous damage case. The LMS filter was used to identify both the location and severity of damage in the blind cases in real time.

Summary of Contributions

The LMS filtering method has the ability to identify presence and location of damage of civil structures in real time with certain constraints. In the case of the benchmark problem with the inclusion of modeling error and noise, identifying changes in alpha due to damage or noise can be difficult. For this case in the presence of modeling error and noise, a limit to which damage can be detected was determined. If there is a greater change than 10 % in the overall stiffness of a particular degree-of-freedom, then damage can be easily identified. In cases where the change in stiffness is less than 10 %, the individual degree-of-freedom can be separated out so each can be monitored for sharp changes which indicate damage. In this situation, the time of damage is needed so one can distinguish between change in alpha due noise or damage. Most importantly, the LMS filter approach can be used to predict location and severity of damage depending on what prior damage cases were run. In the blind case, both location and severity were determined because the graph of alpha matched that of a previously run damage scenario.

Future Research

The results point to the validity of using the LMS filtering technique to detect damage in civil structures. Not only can the technique be used in real time but it is simple to implement. Also it can detect damage in the presence of noise and modeling error and has the ability to predict location and severity of damage. Future research needs to focus on fine tuning the methods ability to predict location and severity. More test cases need to be run on the filter so it can be used as a predicting tool.

REFERENCES

Bernal, D., and Gunes, B. (2000). "Observer/Kalman and Subspace Identification of the UBC Benchmark Structural Model." *Proceedings of the 14th ASCE Engineering Mechanics Conference*, Austin, TX.

Black, C.J., and Ventura, C.E. (1998). "Blind Test on Damage Detection of a Steel Frame Structure." *16th International Modal Analysis Conference,* Santa Barbara, CA, pp 623-629.

Chase, J.G., Hwang, K.L., and Barroso, L.R. (2003a). "A Simple LMS-Based Approach to the Structural Health Monitoring Benchmark Problem." *Earthquake Engineering and Structural Dynamics,* 34, pp 575-594.

Chase, J.G., Begoc, V., and Barroso, L.R. (2003b). "Computationally Efficient Structural Health Monitoring for the Benchmark Structure Using Adaptive RLS Filters." *Computers and Structures,* 83 (8-9), pp 639-647.

Chopra, A.K. (2001). *Dynamics of Structures: Theory and Applications to Earthquake Engineering*, Prentice-Hall, Upper Saddle River, NJ.

Corbin, M., Hera, A., and Hou, Z. (2000). "Locating Damage Regions Using Wavelet Approach." *Proceedings of the 14th ASCE Engineering Mechanics Conference*, Austin, TX.

Doebling, S.W., Farrar, C.R., and Prime, M.B. (1996). "A Summary Review of Vibration-Based Damage Identification Methods." Los Alamos National Laboratory, Los Alamos, NM, LA-13070-MS.

Dyke, S.J., Caicedo, J.M., and Johnson, E.A. (2000). "Monitoring of a Benchmark Structure for Damage Identification." *Proceedings of the 14th ASCE Engineering Mechanics Conference,* Austin , TX.

Friswell, M.I., Penny, J.E.T., and Wilson, D.A.L. (1994). "Using Vibration Data and Statistical Measures to Locate Damage in Structures." *Modal Analysis: The International Journal of Analytical and Experimental Modal Analysis*, 9(4), pp 239-254.

Haykin, S. (1991). *Adaptive Filter Theory,* 2nd Edition, Prentice-Hall, Upper Saddle River, NJ.

Hou, Z., Noori, M., and Amand, R. (2000). "Wavelet-Based Approach for Structural Damage Detection." *Journal of Engineering Mechanics*, 126(7), pp 677-683.

IASC-ASCE SHM Task group website. (1999). " Structural Health Monitoring Committee ASCE" <http://wusceel.cive.wustl.edu/asce/shm> March 6, 2003.

Ifeachor, E.C., and Jervis, B.W. (1993). *Digital Signal Processing: A Practical Approach*, Addison-Wesley, Boston, MA.

Johnson, E.A., Lam, H.F., Katafygiotis, L.S., and Beck, J.L. (2000). "A Benchmark Problem for Structural Health Monitoring and Damage Detection." *Proceedings of the 14th ASCE Engineering Mechanics Conference*, Austin, TX.

Loh, C.H., Lin, C.Y., and Huang, C.C. (2000). "Time Domain Identification of Frames under Earthquake Loadings." *Journal of Engineering Mechanics*, 126(7), pp 693-703.

Loland, O., and Dodds, J.C. (1976). "Experience in Developing and Operating Integrity Monitoring System in North Sea." *Proceedings of the 8th Annual Offshore Technology Conference*, Houston, TX, pp 313-319.

MathWorks. (2002). Matlab (Student Version Release 13) [Computer software]. Mathworks, Natwick, MA.

Mita, A. (1999). "Emerging Needs in Japan for Health Monitoring Technologies in Civil and Building Structures." *Structural Health Monitoring 2000*, pp 56-67.

Nataraja, R. (1983). "Structural Integrity Monitoring in Real Seas." *Proceedings of the 15th Annual Offshore Technology Conference*, Houston, TX, pp 221-228.

Pandey, A.K. and Biswas, M. (1994). "Damage Detection in Structures Using Changes in Flexibility." *Journal of Sound and Vibration*, 169(1), pp 3-17.

Rytter, A. (1993). "Vibration Based Inspection of Civil Engineering Structures." Ph.D. Dissertation, Department of Building Technology and Structural Engineering, Aalborg University, Denmark.

Salawu, O.S. and Williams, C. (1994). "Damage Location Using Vibration Mode Shapes." *Proceedings of the 12th International Modal Analysis Conference*, Demark, pp 933-939.

Vandiver, J.K. (1975). "Detection of Structural Failure on Fixed Platforms by Measurement of Dynamic Response." *7th Annual Offshore Technology Conference*, Houston, TX, pp 243-252.

Yuen, M.M.F. (1985). "A Numerical Study of the Eigenparameters of a Damaged Cantilever." *Journal of Sound and Vibration*, 103, pp 301-310.

Zimmerman, D.C., and Kauk, M. (1994). "Structural Damage Detection Using a Minimum Rank Update Theory." *Journal of Sound and Vibration*, 116(2), pp 221-231.

APPENDIX A

MATLAB COMPUTER CODE

Single Degree-of-Freedom

```
 %==========================================================
%==========================================================
% Determining Response of SDOF to Generated Load Input
% Constant Force with No Error

% Created by: Robin Preston
clear
clc
%==========================================================
% Input Variables
%--------------------------------------------------------------------------
% Stiffness Parameter
K1 = 106600000;           %N/m
K2 =  58400000;           %N/m

% Mass
M = 3200;                 %kg

% Damping
Xil = 0.01;
C = 2*Xil*(K1*M)^(1/2);

% Initial conditions
x0 = 0;
xd0 = 0;

% Filter Variables
u      =5000;
m      = 6;            %Number of Taps

%==========================================================
% Response with K1
%--------------------------------------------------------------------------
%... Force Matrix
t       = 40;
t_1     = 20;                  %seconds  pt where stiffness changes
dt      = 0.001;               %seconds

time   = 0:dt:t;
t_span1 = 0:dt:t_1;
t_span2 = t_1:dt:t;

po     = 100000;              % magnitude of force
F      = po * ones(size(time));
```

```
F2     = F(20001:length(time));

%=============================================================
% Solve for Response
%-------------------------------------------------------------------------
[t,x1] = ode45(@sdof_New,t_span1,[x0 xd0],[],M,C,K1,F,time);

% Generate xdd data
for k=1:length(t_span1)
    x1dd(k,1) =  (F(k)-K1*x1(k,1)-C*x1(k,2))/M;
end

%=============================================================
% Response with K2
%-------------------------------------------------------------------------
% Force Matrix

x20 = x1(length(x1),1);     %... take info at end of first segment and make
x2d0 = x1(length(x1),2);    % it initial condition for next segment

[t,x2] = ode45(@sdof_New,t_span2,[x20 x2d0],[],M,C,K2,F,time);
% Generate xdd data
for k=1:length(t_span2)
    x2dd(k,1) =  (F2(k)-K2*x2(k,1)-C*x2(k,2))/M;
end

%=============================================================
% Combination of Two Responses
%-------------------------------------------------------------------------
x = [x1(:,1); x2(2:length(x2),1)];
xd = [x1(:,2); x2(2:length(x2),2)];
xdd = [x1dd(:,1); x2dd(2:length(x2dd),1)];

%=============================================================
% Adaptive LMS Filter
%-------------------------------------------------------------------------
% Calculate yk
 for k = 1: length(time)
    yk(k) = F(k)-M*xdd(k)-K1*x(k)-C*xd(k);
 end
% Adaptive Filter
w(1) = 1;
for k =1
    n(k)=w(k)*x(k);
    e(k) = yk(k)-n(k);
    w(k+1) = w(k)+2*u*e(k)*x(k);
end
for k =2
    n(k)=w(k)*x(k)+w(k)*x(k-1);
    e(k) = yk(k)-n(k);
    w(k+1) = w(k)+2*u*e(k)*x(k);
end
for k=3
    n(k)=w(k)*x(k)+w(k)*x(k-1)+w(k)*x(k-2);
    e(k) = yk(k)-n(k);
    w(k+1) = w(k)+2*u*e(k)*x(k);
end
for k=4
    n(k)=w(k)*x(k)+w(k)*x(k-1)+w(k)*x(k-2)+w(k)*x(k-3);
    e(k) = yk(k)-n(k);
    w(k+1) = w(k)+2*u*e(k)*x(k);
end
for k=5
    n(k)=w(k)*x(k)+w(k)*x(k-1)+w(k)*x(k-2)+w(k)*x(k-3)+w(k)*x(k-4);
    e(k) = yk(k)-n(k);
    w(k+1) = w(k)+2*u*e(k)*x(k);
end
for k=6:length(time)
    n(k)=w(k)*x(k)+w(k)*x(k-1)+w(k)*x(k-2)+w(k)*x(k-3)+w(k)*x(k-4)+w(k)*x(k-5);
    e(k) = yk(k)-n(k);
    w(k+1) = w(k)+2*u*e(k)*x(k);
end
% Back Calculate Alpha or Change in Stiffness
warning off MATLAB:divideByZero

for k=1:length(time)
    alpha(k) =n(k)/(x(k)*K1);
end
for k=1:length(time)
    test(k) =yk(k)/(x(k)*K1);
end
figure(1)
plot(time,alpha,time,test)
xlabel('Time (sec)')
ylabel('Change in a (N/m)')
grid
%==========================================================
%==========================================================
%============================================================
%============================================================
% Define’s SDOF system in state-space format

 function [y_prime] = sdof(t,y,m,c,k,F,time)
%------------------------------------------------------------------------

% System matrices -- State-Space

A =  [    0       1

       -(k/m)  -(c/m) ];


B =  [    0

       -(1/m)  ];


% Pull out current force vector -- uses linear interpolation for

% values between points available in time history

F_t = interp1(time,F,t);


% Compute first derivative values at this time step.

y_prime = A*y + B*F_t;

%============================================================
%============================================================
```

Four Degrees-of-Freedom

```
%=============================================================
%=============================================================
% **Determining Response of Four-DOF of Benchmark Problem**
% **Constant Force with No Error**

% Created by: Robin Preston
clear
clc
%=============================================================
%Input Variables
%--------------------------------------------------------------------------
% Load Data
load Case1Damage1

K2 = [58400000  0 0 0
      0 106600000 0 0
      0 0 106600000 0
      0 0 0 106600000] ;

K  = [106600000 0 0 0
      0 106600000 0 0
      0 0 106600000 0
      0 0 0 106600000] ;

M  = [3452.4 0 0 0
      0 2652.4 0 0
      0 0 2652.4 0
      0 0 0 1809.9];

% Degree of Freedom
ndof=4;

% Damping
Xil = 0.01;
C = 2*Xil*(K*M)^(1/2);
C2 = 2*Xil*(K2*M)^(1/2);

%=============================================================
% Build K, M, and C matrices for filter.

%All values based on benchmark problem
%--------------------------------------------------------------------------
disp('Building Matrices...');
% Mass
```

```
M1  = 3200;             %kg
M2  = 2400;             %kg
M3  = 2400;             %kg
M4  = 1600;             %kg

% Build  Mass Matrix
MM = [M1  0  0  0
      0  M2  0  0
      0  0  M3  0
      0  0   0 M4];

% Stiffness Parameter
KK = [106600000 0 0 0
     0 106600000 0 0
     0 0 106600000 0
     0 0 0 106600000] ;

% Build Damping Matrix
CC = 2*Xil*(KK*MM)^(1/2);

% Initial Conditions
x0  = zeros(ndof,1);
xd0 = zeros(ndof,1);

%==========================================================
% Time Vector and Force Matrix
%---------------------------------------------------------------------------
disp('Building Force Matrix...')
% Force Matrix
t      = 40;
t_1     = 20;                 %seconds  pt where stiffness changes
dt      = 0.001;                %seconds

time   = 0:dt:t;
t_span1 = 0:dt:t_1;
t_span2 = t_1:dt:t;

FF  =  100000;                  %Force for translational DOF
f   = zeros(1,length(time));
f2  = f(1,20001:length(time));
force      = FF*ones(1,length(time));
force2     = force(1,20001:length(time));

F   = [force; force; force; force];
```

```
FF = F(:,1:20001);
F2  = [force2; force2; force2; force2];

%=================================================================
% RESPONSE IN ORIGINAL COORDINATES
%-----------------------------------------------------------------------
disp('Solving for Response 1...')
% Solve for Response 1-- ode45
% Initial Conditions
x0  = zeros(ndof, 1);
xd0 = zeros(ndof,1);

[t,x1] = ode45(@mdof,t_span1,[x0; xd0],[],M,C,K,FF,t_span1);

%Generate xdd data
xd1 = [x1(:,5) x1(:,6) x1(:,7) x1(:,8)]';
x1  = [x1(:,1) x1(:,2) x1(:,3) x1(:,4)]';

xdd1 = zeros(ndof, length(t_span1));
for k = 1:length(t_span1)
   xdd1(:,k)=inv(M)*[F(:,k)-C*xd1(:,k)-K*x1(:,k)];
end

%=================================================================
% Response with K2
%-----------------------------------------------------------------------
disp('Solving for Response 2...')
% Solve for Response 2 -- ode45

x20 = x1(:,length(x1));     %... take info at end of first segment and make
x2d0 = xd1(:,length(x1));    % it initial condition for next segment

[t,x2] = ode45(@mdof,t_span2,[x20; x2d0],[],M,C2,K2,F2,t_span2);

 %Generate xdd data
xd2 = [x2(:,5) x2(:,6) x2(:,6) x2(:,8)]';
x2  = [x2(:,1) x2(:,2) x2(:,3) x2(:,8)]';

xdd2 = zeros(ndof, length(t_span2));
for k = 1:length(t_span2)
   xdd2(:,k)=inv(M)*[F2(:,k)-C2*xd2(:,k)-K2*x2(:,k)];
end
```

```
%============================================================
% Combination of Two Responses
%------------------------------------------------------------------------
x = [x1(:,1:length(t_span1)) x2(:,2:length(x2))];
xd = [xd1(:,1:length(t_span1)) xd2(:,2:length(xd2))];
xdd = [xdd1(:,1:length(t_span1)) xdd2(:,2:length(xdd2))];

% Adaptive LMS Filter
%------------------------------------------------------------------------
disp('Running Adaptive LMS Filter...');
% Filter Variables
u      = [10000;1000;1000;1000];
m      = 6;            %Number of Taps

% Run Filter Function
yk = zeros(ndof, length(time));
n  = zeros(ndof, length(time));
e  = zeros(ndof, length(time));
w  = zeros(ndof, length(time));

for k = 1:length(time)
   yk(:,k) = F(:,k)-M*xdd(:,k)-KK*x(:,k)-CC*xd(:,k);
end
for m = 1: ndof
   [n,e,w] = ff(time,yk,n,e,w,x,m,u);
end

%============================================================
% Calculate Change in Alpha
%------------------------------------------------------------------------
disp('Solving for Alpha...')
 warning off MATLAB:divideByZero
 warning off MATLAB:singularMatrix

Alpha = zeros(4,length(time));
for k = 2:length(time)
   Alpha(4,k) =n(4,k)/((-x(3,k)+x(4,k))*KK(4,4));
end

for k = 2:length(time)
   Alpha(3,k) = (n(3,k)-[Alpha(4,k)*(x(3,k)-x(4,k))])/((-x(2,k)+x(3,k))*KK(3,3));
end
for k = 2:length(time)
   Alpha(2,k) = (n(2,k)-[Alpha(3,k)*(x(2,k)-x(3,k))])/((-x(1,k)+x(2,k))*KK(2,2));
```

```
end
Alpha(2,:)= 0;

for k = 2:length(time)
   Alpha(1,k) = (n(1,k)-Alpha(2,k)*(x(1,k)-x(2,k)))/((x(1,k))*KK(1,1));
end

g1  = -0.65*min(x(1,:));
g2  = -0.65*min(x(2,:));
g3  = -0.65*min(x(3,:));
g4  = -0.65*min(x(4,:));
for k = 2:length(time)
   if abs(x(1,k)) < g1
      Alpha(1,k) = Alpha(1,k-1);
   end
end
for k = 2:length(time)
   if abs(x(2,k)) < g2
      Alpha(2,k) = Alpha(2,k-1);
   end
end
for k = 3:length(time)
   if abs(x(3,k)) < g3
      Alpha(3,k) = Alpha(3,k-1);
   end
end
for k = 2:length(time)
   if abs(x(4,k)) < g4
      Alpha(4,k) = Alpha(4,k-1);
   end
end

test = zeros(ndof, length(time));
for m = 1:4
   test(m,20001:40001) = -(1-(K2(m,m)/K(m,m)));
end

figure(1)
plot(time,test,time,Alpha)
xlabel('Time (sec)')
ylabel('Percent Change in Alpha')
grid
%=============================================================
%=============================================================
%==========================================================
%==========================================================
% ADAPTIVE LMS FILTER
% created by Robin Preston

function [n,e,w] = ff(time,yk,n,e,w,x,m,u);
%----------------------------------------------------------------------
% Adaptive Filter
w(m,1) = 1;

for k =1
   n(m,k)=w(m,k)*x(m,k);
   e(m,k) = yk(m,k)-n(m,k);
   w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k =2
   n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1);
   e(m,k) = yk(m,k)-n(m,k);
   w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k=3
   n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1)+w(m,k)*x(m,k-2);
   e(m,k) = yk(m,k)-n(m,k);
   w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k=4
   n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1)+w(m,k)*x(m,k-2)+w(m,k)*x(m,k-3);
   e(m,k) = yk(m,k)-n(m,k);
   w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k=5
   n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1)+w(m,k)*x(m,k-2)+w(m,k)*x(m,k-
3)+w(m,k)*x(m,k-4);
   e(m,k) = yk(m,k)-n(m,k);
   w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k=6:(length(time)-1)
```

```
    n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1)+w(m,k)*x(m,k-2)+w(m,k)*x(m,k-
3)+w(m,k)*x(m,k-4)+w(m,k)*x(m,k-5);
    e(m,k) = yk(m,k)-n(m,k);
    w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k=length(time)
    n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1)+w(m,k)*x(m,k-2)+w(m,k)*x(m,k-
3)+w(m,k)*x(m,k-4)+w(m,k)*x(m,k-5);
    e(m,k) = yk(m,k)-n(m,k);
end

%=============================================================
%=============================================================
%==========================================================
%==========================================================
% Defines Four-DOF system in state-space format

% Created by Robin Preston

function [y_prime] = mdof(t,y,M,C,K,F,time)
%----------------------------------------------------------------------
% System matrices -- State-space
inv_mass = inv(M);

A11 = zeros(4,4);
A12 = eye(4,4);
A21 = [-inv_mass*K];
A22 = [-inv_mass*C];
A= [A11  A12
   A21  A22];

E  = eye(4,4);
B1 = zeros(4,4);
B2 = [-inv_mass*E];
B  = [B1
     B2];

% Pull out current force vector -- uses linear interpolation for
% values between points available in time history
F_t = interp1(time,F',t)';

% Compute first derivative values at this time step.
y_prime = A*y + B*F_t;

%==========================================================
%==========================================================
```

Three Degrees-of-Freedom

```
%=========================================================
%=========================================================
% Determining Response of Three-DOF of Benchmark Problem
% Constant Force and No Error

% Created by: Robin Preston
clear
clc
%=========================================================
%Input Variables
%-------------------------------------------------------------------------
% Load Data
load Case1Damage1

K2 = [58400000 0 0
   0 19700000 0
   0 0 81300000];

K  = [106600000 0 0
    0 67900000 0
    0 0 232000000];

M  = [3452.4 0 0
    0 3542.4 0
    0 0 3819.4];
% Degree of Freedom
ndof=3;

% Damping
Xil = 0.01;
C = 2*Xil*(K*M)^(1/2);
C2 = 2*Xil*(K2*M)^(1/2);

%=========================================================
% Build K, M, and C matrices for filter.

%All values based on benchmark problem
%-------------------------------------------------------------------------
disp('Building Matrices...');
% Mass
M1  = 3200;              %kg
Io1 = 3333.33;           %kg-m^2

% Build  Mass Matrix
```

```
MM = [M1  0  0
      0  M1  0
      0  0 Io1];

% Stiffness Parameter
KK = [106600000 0 0
     0 67900000 0
     0 0 232000000];

% Build Damping Matrix
CC = 2*Xil*(KK*MM)^(1/2);

% Initial Conditions
x0  = zeros(ndof,1);
xd0 = zeros(ndof,1);

%===========================================================
% Time Vector and Force Matrix
%-----------------------------------------------------------------------
disp('Building Force Matrix...')
% Force Matrix
t       = 40;
t_1     = 20;                    %seconds  pt where stiffness changes
dt      = 0.001;                   %seconds

time   = 0:dt:t;
t_span1 = 0:dt:t_1;
t_span2 = t_1:dt:t;
FF  =  100000;                    %Force for translational DOF
FFF = -10000;                     %Force for rotational DOF
f   = zeros(1,length(time));
f2  = f(1,20001:length(time));
force      = FF*ones(1,length(time));
fforce     = FFF*ones(1, length(time));
force2     = force(1,20001:length(time));
fforce2    = fforce(1,20001:length(time));

F   = [force; force; fforce];
FF = F(:,1:20001);
F2  = [force2; force2; fforce2];

%===========================================================
% RESPONSE IN ORIGINAL COORDINATES
%-----------------------------------------------------------------------
```

```
disp('Solving for Response 1...')
% Solve for Response 1-- ode45
% Initial Conditions
x0  = zeros(ndof, 1);
xd0 = zeros(ndof,1);

[t,x1] = ode45(@mdof,t_span1,[x0; xd0],[],M,C,K,FF,t_span1);

%Generate xdd data
xd1 = [x1(:,4) x1(:,5) x1(:,6)]';
x1  = [x1(:,1) x1(:,2) x1(:,3)]';

xdd1 = zeros(ndof, length(t_span1));
for k = 1:length(t_span1)
   xdd1(:,k)=inv(M)*[F(:,k)-C*xd1(:,k)-K*x1(:,k)];
end

%=============================================================
% Response with K2
%-------------------------------------------------------------------------
disp('Solving for Response 2...')
% Solve for Response 2 -- ode45

x20 = x1(:,length(x1));     %... take info at end of first segment and make
x2d0 = xd1(:,length(x1));    % it initial condition for next segment

[t,x2] = ode45(@mdof,t_span2,[x20; x2d0],[],M,C2,K2,F2,t_span2);

 %Generate xdd data
xd2 = [x2(:,4) x2(:,5) x2(:,6)]';
x2  = [x2(:,1) x2(:,2) x2(:,3)]';

xdd2 = zeros(ndof, length(t_span2));
for k = 1:length(t_span2)
   xdd2(:,k)=inv(M)*[F2(:,k)-C2*xd2(:,k)-K2*x2(:,k)];
end

%=============================================================
% Combination of Two Responses
%-------------------------------------------------------------------------
x = [x1(:,1:length(t_span1)) x2(:,2:length(x2))];
xd = [xd1(:,1:length(t_span1)) xd2(:,2:length(xd2))];
xdd = [xdd1(:,1:length(t_span1)) xdd2(:,2:length(xdd2))];
```

```
%=========================================================================
% Adaptive LMS Filter
%-------------------------------------------------------------------------
disp('Running Adaptive LMS Filter...');
% Filter Variables
u      = [10000;1000;50000];
m      = 6;             %Number of Taps

% Run Filter Function
yk = zeros(ndof, length(time));
n  = zeros(ndof, length(time));
e  = zeros(ndof, length(time));
w  = zeros(ndof, length(time));

for k = 1:length(time)
    yk(:,k) = F(:,k)-M*xdd(:,k)-KK*x(:,k)-CC*xd(:,k);
end

for m = 1: ndof
    [n,e,w] = ff(time,yk,n,e,w,x,m,u);
end

%=========================================================================
% Calculate Change in Alpha
%-------------------------------------------------------------------------
disp('Solving for Alpha...')
 warning off MATLAB:divideByZero
 warning off MATLAB:singularMatrix

Alpha = zeros(ndof,length(time));

for k=2:length(time)
    for m = 1:2
        Alpha(m,k) =n(m,k)/(x(m,k)*KK(m,m));
    end
end

for k=2:length(time)
    for m = 3
        Alpha(m,k) =yk(m,k)/(x(m,k)*KK(m,m));
    end
end

g1  = 0.90*max(x(1,:));
```

```
g2  = 0.90*max(x(2,:));
g3  = 0.99*max(x(3,:));

for k = 2:length(time)
    if abs(x(1,k)) < g1
        Alpha(1,k) = Alpha(1,k-1);
    end
end
for k = 2:length(time)
    if abs(x(2,k)) < g2
        Alpha(2,k) = Alpha(2,k-1);
    end
end
for k = 2:length(time)
    if abs(x(3,k)) < g3
        Alpha(3,k) = Alpha(3,k-1);
    end
end

test = zeros(ndof, length(time));
for m = 1:ndof
    test(m,20001:40001) = -(1-(K2(m,m)/K(m,m)));
end

figure(1)
plot(time,test,time,Alpha)
xlabel('Time (sec)')
ylabel('Percent Change in Alpha')
grid

%===========================================================
%===========================================================
%============================================================
%============================================================
% ADAPTIVE LMS FILTER
% created by Robin Preston

function [n,e,w] = ff(time,yk,n,e,w,x,m,u);
%------------------------------------------------------------------------
% Adaptive Filter
w(m,1) = 1;

for k =1
   n(m,k)=w(m,k)*x(m,k);
   e(m,k) = yk(m,k)-n(m,k);
   w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k =2
   n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1);
   e(m,k) = yk(m,k)-n(m,k);
   w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k=3
   n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1)+w(m,k)*x(m,k-2);
   e(m,k) = yk(m,k)-n(m,k);
   w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k=4
   n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1)+w(m,k)*x(m,k-2)+w(m,k)*x(m,k-3);
   e(m,k) = yk(m,k)-n(m,k);
   w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k=5
   n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1)+w(m,k)*x(m,k-2)+w(m,k)*x(m,k-
3)+w(m,k)*x(m,k-4);
   e(m,k) = yk(m,k)-n(m,k);
   w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k=6:(length(time)-1)
   n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1)+w(m,k)*x(m,k-2)+w(m,k)*x(m,k-
3)+w(m,k)*x(m,k-4)+w(m,k)*x(m,k-5);
```

```
    e(m,k) = yk(m,k)-n(m,k);
    w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k=length(time)
    n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1)+w(m,k)*x(m,k-2)+w(m,k)*x(m,k-
3)+w(m,k)*x(m,k-4)+w(m,k)*x(m,k-5);
    e(m,k) = yk(m,k)-n(m,k);
end

%==========================================================
%==========================================================
%=============================================================
%=============================================================
% Defines Four-DOF system in state-space format

% Created by Robin Preston

function [y_prime] = mdof(t,y,M,C,K,F,time)
%-----------------------------------------------------------------------
% System matrices -- State-space
inv_mass = inv(M);

A11 = zeros(4,4);
A12 = eye(4,4);
A21 = [-inv_mass*K];
A22 = [-inv_mass*C];
A= [A11  A12
   A21  A22];

E  = eye(4,4);
B1 = zeros(4,4);
B2 = [-inv_mass*E];
B  = [B1
      B2];

% Pull out current force vector -- uses linear interpolation for
% values between points available in time history
F_t = interp1(time,F',t)';

% Compute first derivative values at this time step.
y_prime = A*y + B*F_t;

%=============================================================
%=============================================================
```

Phase I: 12 Degrees-of-Freedom

```
%==================================================================
%==================================================================
% Determining Response of 12-DOF of Benchmark Problem
% Case 1 Damage1: Web Simulated Data

% Created by: Robin Preston

%==================================================================
% Input Variables
%-----------------------------------------------------------------------
% Load Variable
load Case1Undamaged
force1 = force;
K1 = K;
M1 = M;
load Case1Damage1
force2 = force;
K2 = K;
M2 = M;

% Degree of Freedom
[ndof,ndof] = size(M);

% Damping
Xil = 0.01;
C1 = 2*Xil*(K1*M1)^(1/2);
C2 = 2*Xil*(K2*M2)^(1/2);

%==================================================================
% Build K, M, and C matrices for filter.
% All values based on benchmark problem
%-----------------------------------------------------------------------
disp('Building Matrices...');
% Mass
MM1 = 3200;         %kg
MM2 = 2400;         %kg
MM3 = 2400;         %kg
MM4 = 1600;         %kg

Io1 = 3333.33;      %kg-m^2
Io2 = 2500.00;      %kg-m^2
Io3 = 2500.00;      %kg-m^2
Io4 = 1666.67;      %kg-m^2
```

```
% Build  Mass Matrix
MM = [MM1  0  0  0  0  0  0  0  0  0  0  0
     0  MM1  0  0  0  0  0  0  0  0  0  0
     0  0 Io1  0  0  0  0  0  0  0  0  0
     0  0  0  MM2  0  0  0  0  0  0  0  0
     0  0  0  0  MM2  0  0  0  0  0  0  0
     0  0  0  0  0 Io2  0  0  0  0  0  0
     0  0  0  0  0  0  MM3  0  0  0  0  0
     0  0  0  0  0  0  0  MM3  0  0  0  0
     0  0  0  0  0  0  0   0 Io3 0  0  0
     0  0  0  0  0  0  0   0  0 MM4  0  0
     0  0  0  0  0  0  0   0  0  0 MM4  0
     0  0  0  0  0  0  0   0  0  0 0 Io4];

% Stiffness Parameter
KX1 = 106600000;       %N/m
KX2 = 106600000;       %N/m
KX3 = 106600000;       %N/m
KX4 = 106600000;       %N/m

KY1 = 67900000;       %N/m
KY2 = 67900000;       %N/m
KY3 = 67900000;       %N/m
KY4 = 67900000;       %N/m

Ko1 = 232000000;      %N/m
Ko2 = 232000000;      %N/m
Ko3 = 232000000;      %N/m
Ko4 = 232000000;      %N/m

% Build Stiffness Matrix Components
KK = zeros(ndof, ndof);

for m = 1
   KK(m,m) = KX1+KX2;
   KK(m+3,m)=-KX1;
   KK(m,m+3)=-KX1;
end
for m = 4
   KK(m,m) = KX1+KX2;
   KK(m+3,m)=-KX1;
   KK(m,m+3)=-KX1;
end
```

```
for m = 7
   KK(m,m) = KX1+KX2;
   KK(m+3,m)=-KX1;
   KK(m,m+3)=-KX1;
end
for m=10
   KK(m,m)=KX4;
end
for m = 2
   KK(m,m) = KY1+KY2;
   KK(m+3,m)=-KY1;
   KK(m,m+3)=-KY1;
end
for m = 5
   KK(m,m) = KY1+KY2;
   KK(m+3,m)=-KY1;
   KK(m,m+3)=-KY1;
end
for m = 8
   KK(m,m) = KY1+KY2;
   KK(m+3,m)=-KY1;
   KK(m,m+3)=-KY1;
end
for m=11
   KK(m,m) = KY4;
end
for m = 3
   KK(m,m) = Ko1+Ko2;
   KK(m+3,m)=-Ko1;
   KK(m,m+3)=-Ko1;
end
for m = 6
   KK(m,m) = Ko1+Ko2;
   KK(m+3,m)=-Ko1;
   KK(m,m+3)=-Ko1;
end
for m = 9
   KK(m,m) = Ko1+Ko2;
   KK(m+3,m)=-Ko1;
   KK(m,m+3)=-Ko1;
end
for m = 12
   KK(m,m) = Ko4;
end
```

```
% Build Damping Matrix
CC = 2*Xil*(KK*MM)^(1/2);

%=========================================================================
% Time Vector and Force Matrix
%-------------------------------------------------------------------------
time = 0:dt:40;

force = [force1(1:length(force1)-1,:); force2];
f = zeros(1,40001);
FF = [f; force(:,1)'; f; f; force(:,2)'; f; f; force(:,3)'; f; f; force(:,4)'; f];

%=========================================================================
% Load Data
%-------------------------------------------------------------------------
% Manipulate Acc data to generate displacement, velocity and
% acceleration matrices
disp('Loading Data');

load XXun1;
load XXdun1;
load XXddun1;

%=========================================================================
% Adaptive LMS Filter
%-------------------------------------------------------------------------
disp('Running Adaptive LMS Filter');
% Filter Variables
u      = [10;1;100;10;1;100;10;1;100;10;1;100];
m      = 6;                    %Number of Taps

% Run Filter Function
yk = zeros(ndof, length(time));
n  = zeros(ndof, length(time));
e  = zeros(ndof, length(time));
w  = zeros(ndof, length(time));
for k = 1:length(20001)
   yk(:,k) = FF(:,k)-M1(:,:)*xdd(:,k)-K1(:,:)*x(:,k)-C1(:,:)*xd(:,k);
end
for k = 20002:length(time)
   yk(:,k) = FF(:,k)-M1(:,:)*xdd(:,k)-K1(:,:)*x(:,k)-C1(:,:)*xd(:,k);
end
```

```
for m = 1:ndof
    [n,e,w] = ff(time,yk,n,e,w,x,m,u);
end

%=====================================================================
% Calculate Change in Alpha
%-----------------------------------------------------------------------
disp('Solving for Alpha...')
 warning off MATLAB:divideByZero
 warning off MATLAB:singularMatrix

KKK = [KX1 KY1 Ko1 KX2 KY2 Ko2 KX3 KY3 Ko3 KX4 KY4 Ko4];

    g1  = 0.75*max(x(1,:));
    g2  = 0.75*max(x(2,:));
    g3  = 0.90*max(x(3,:));
    g4  = 0.75*max(-x(1,:)+x(7,:));
    g5  = 0.75*max(-x(2,:)+x(8,:));
    g6  = 0.95*max(-x(3,:)+x(9,:));
    g7  = 0.75*max(-x(4,:)+x(10,:));
    g8  = 0.75*max(-x(5,:)+x(11,:));
    g9  = 0.95*max(-x(6,:)+x(12,:));
    g10  = 0.75*max(x(10,:)-x(7,:));
    g11  = 0.75*max(x(11,:)-x(8,:));
    g12  = 0.95*max(x(12,:)-x(9,:));


Alpha = zeros(ndof,length(time));
for k = 2:length(time)
    for m = 10:12
        Alpha(m,k) =n(m,k)/((x(m,k)-x(m-3,k))*KKK(1,m));
    end
    for m = 7:9
        Alpha(m,k) = (n(m,k)-[Alpha(m+3,k)*(x(m,k)-x(m+3,k))])/((-x(m-
3,k)+x(m,k))*KKK(1,m));
    end

    for m = 4:6
        Alpha(m,k) = (n(m,k)-[Alpha(m+3,k)*(x(m,k)-x(m+3,k))])/((-x(m-
3,k)+x(m,k))*KKK(1,m));
    end
    for m = 1:3
        Alpha(m,k) = (n(m,k)-Alpha(m+3,k)*(x(m,k)-x(m+3,k)))/((x(m,k))*KKK(1,m));
    end
```

```
end

for k = 2:length(time)
    for m = 1
        if abs(x (m,k)) < g1
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
    for m = 2
        if abs(x (m,k)) < g2
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
   for m = 3
        if abs(x (m,k)) < g3
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
    for m =4
        if abs(-x(m-3,k)+x(m,k)) < g4
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
    for m = 5
        if abs(-x(m-3,k)+x(m,k)) < g5
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
    for m = 6
        if abs(-x(m-3,k)+x(m,k)) < g6
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
    for m = 7
        if abs(-x(m-3,k)+x(m,k)) < g7
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
    for m = 8
        if abs(-x(m-3,k)+x(m,k)) < g8
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
    for m = 9
```

```
        if abs(-x(m-3,k)+x(m,k)) < g9
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
   for m = 10
        if abs(x(m,k)-x(m-3,k)) < g10
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
     for m = 11
        if abs(x(m,k)-x(m-3,k)) < g11
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
     for m = 12
        if abs(x(m,k)-x(m-3,k)) < g12
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
end

plot(time, Alpha)
xlabel('Time (sec)')
ylabel('Percent Change in Alpha  ')
grid

%==========================================================
%==========================================================
%=============================================================
%=============================================================
% MANIPULATION OF ACCELERATION DATA
% Derive Velocity and Displacement of Each Time Step
% Derive Response in rotational DOF

% created by Robin Preston
%-----------------------------------------------------------------------
%  acc      =  simulated acceleration time-history.
%             It is a Nt by Nmdof array:
%             Nt    =  number of time steps
%             Nmdof =  number of measured dof
%             ================================================
%             column number (x-coordinate, y-coordinate):
%             7 (   0, 2.5)  8 (1.25, 2.5)  9 ( 2.5, 2.5)
%             4 (   0,1.25)  5 (1.25,1.25)  6 ( 2.5,1.25)
%             1 (   0,   0)  2 (1.25,   0)  3 ( 2.5,   0)
%             ================================================
%             for each floor, sensors are located at columns
%             2, 6, 8 and 4.
%             acc(:,1)  - floor 1 of column 2 in x-direction
%             acc(:,2)  - floor 1 of column 6 in y-direction
%             acc(:,3)  - floor 1 of column 8 in x-direction
%             acc(:,4)  - floor 1 of column 4 in y-direction
%             acc(:,5)  - floor 2 of column 2 in x-direction
%             acc(:,6)  - floor 2 of column 6 in y-direction
%             acc(:,7)  - floor 2 of column 8 in x-direction
%             acc(:,8)  - floor 2 of column 4 in y-direction
%             acc(:,9)  - floor 3 of column 2 in x-direction
%             acc(:,10) - floor 3 of column 6 in y-direction
%             acc(:,11) - floor 3 of column 8 in x-direction
%             acc(:,12) - floor 3 of column 4 in y-direction
%             acc(:,13) - floor 4 of column 2 in x-direction
%             acc(:,14) - floor 4 of column 4 in y-direction
%             acc(:,15) - floor 4 of column 6 in x-direction
%             acc(:,16) - floor 4 of column 8 in y-direction
%-----------------------------------------------------------------------

% Calculate Accleration for 12 DOF for Undamaged
load Case1Undamaged

acc1  = (acc(:,3)+acc(:,1))/2;
acc2  = (acc(:,4)+acc(:,2))/2;
acc3  = (acc(:,3)-acc(:,1))/2.5;
```

```
acc4  = (acc(:,7)+acc(:,5))/2;
acc5  = (acc(:,8)+acc(:,6))/2;
acc6  = (acc(:,7)-acc(:,5))/2.5;
acc7  = (acc(:,11)+acc(:,9))/2;
acc8  = (acc(:,12)+acc(:,10))/2;
acc9  = (acc(:,11)-acc(:,9))/2.5;
acc10 = (acc(:,15)+acc(:,13))/2;
acc11 = (acc(:,16)+acc(:,14))/2;
acc12 = (acc(:,15)-acc(:,13))/2.5;

% Calculate Accleration for 12 DOF for Damage Case
load Case1Damage1

Acc1  = (acc(:,3)+acc(:,1))/2;
Acc2  = (acc(:,4)+acc(:,2))/2;
Acc3  = (acc(:,3)-acc(:,1))/2.5;
Acc4  = (acc(:,7)+acc(:,5))/2;
Acc5  = (acc(:,8)+acc(:,6))/2;
Acc6  = (acc(:,7)-acc(:,5))/2.5;
Acc7  = (acc(:,11)+acc(:,9))/2;
Acc8  = (acc(:,12)+acc(:,10))/2;
Acc9  = (acc(:,11)-acc(:,9))/2.5;
Acc10 = (acc(:,15)+acc(:,13))/2;
Acc11 = (acc(:,16)+acc(:,14))/2;
Acc12 = (acc(:,15)-acc(:,13))/2.5;

%=============================================================
%=============================================================
%==================================================================
%==================================================================
% Model Simulation Using Models from System ID session vela and Val

% Must be run in Workspace, then dies saved as dis.mat

% created by Robin Preston
%-----------------------------------------------------------------
% Undamaged Case
clear vela
clear Vela
clear xdd1
clear xd1
clear xdd2
clear xd2
clear x
clear xd
clear xdd

vel1  = idsim(acc1,vel_1);
vel2  = idsim(acc2,vel_2);
vel3  = idsim(acc3,vel_3);
vel4  = idsim(acc4,vel_4);
vel5  = idsim(acc5,vel_5);
vel6  = idsim(acc6,vel_6);
vel7  = idsim(acc7,vel_7);
vel8  = idsim(acc8,vel_8);
vel9  = idsim(acc9,vel_9);
vel10 = idsim(acc10,vel_10);
vel11 = idsim(acc11,vel_11);
vel12 = idsim(acc12,vel_12);

xdd1 = [acc1 acc2 acc3 acc4 acc5 acc6 acc7 acc8 acc9 acc10 acc11 acc12]';
xd1  = [vel1 vel2 vel3 vel4 vel5 vel6 vel7 vel8 vel9 vel10 vel11 vel12]';

CC = zeros(12, 20001);
for k = 1:20001
   CC(:,k)=inv(K1)*[FF(:,k)-C1*xd1(:,k)-M1*xdd1(:,k)];
end

% Damage Case
Vel1  = idsim(Acc1,Vel_1);
Vel2  = idsim(Acc2,Vel_2);
Vel3  = idsim(Acc3,Vel_3);
```

```
Vel4  = idsim(Acc4,Vel_4);
Vel5  = idsim(Acc5,Vel_5);
Vel6  = idsim(Acc6,Vel_6);
Vel7  = idsim(Acc7,Vel_7);
Vel8  = idsim(Acc8,Vel_8);
Vel9  = idsim(Acc9,Vel_9);
Vel10 = idsim(Acc10,Vel_10);
Vel11 = idsim(Acc11,Vel_11);
Vel12 = idsim(Acc12,Vel_12);

xdd2 = [Acc1 Acc2 Acc3 Acc4 Acc5 Acc6 Acc7 Acc8 Acc9 Acc10 Acc11 Acc12]';
xd2  = [Vel1 Vel2 Vel3 Vel4 Vel5 Vel6 Vel7 Vel8 Vel9 Vel10 Vel11 Vel12]';

CCC = zeros(12, 20001);
for k = 1:20001
   CCC(:,k)=inv(K2)*[FF(:,k+20000)-C2*xd2(:,k)-M2*xdd2(:,k)];
end

% Combine Data
xdd = [xdd1(:,1:length(xdd1)-1) xdd2(:,:)];

xd = [xd1(:,1:length(xd1)-1) xd2(:,:)];

x= [CC(:,1:length(CC)-1) CCC(:,:)];

%=============================================================
%=============================================================
%============================================================
%============================================================
% Generate Model Data for Simulation Model
% Creates individual displacement and velocity vectors to be inputted into
% System Identification Models

% Created by Robin Preston
%-----------------------------------------------------------------------
xd1=xd(1,1:20001)';
xd2=xd(2,1:20001)';
xd3=xd(3,1:20001)';
xd4=xd(4,1:20001)';
xd5=xd(5,1:20001)';
xd6=xd(6,1:20001)';
xd7=xd(7,1:20001)';
xd8=xd(8,1:20001)';
xd9=xd(9,1:20001)';
xd10=xd(10,1:20001)';
xd11=xd(11,1:20001)';
xd12=xd(12,1:20001)';
xdd1=xdd(1,1:20001)';
xdd2=xdd(2,1:20001)';
xdd3=xdd(3,1:20001)';
xdd4=xdd(4,1:20001)';
xdd5=xdd(5,1:20001)';
xdd6=xdd(6,1:20001)';
xdd7=xdd(7,1:20001)';
xdd8=xdd(8,1:20001)';
xdd9=xdd(9,1:20001)';
xdd10=xdd(10,1:20001)';
xdd11=xdd(11,1:20001)';
xdd12=xdd(12,1:20001)';
Xdd1=xdd(1,20002:40001)';
Xdd2=xdd(2,20002:40001)';
Xdd3=xdd(3,20002:40001)';
Xdd4=xdd(4,20002:40001)';
Xdd5=xdd(5,20002:40001)';
Xdd6=xdd(6,20002:40001)';
Xdd7=xdd(7,20002:40001)';
Xdd8=xdd(8,20002:40001)';
Xdd9=xdd(9,20002:40001)';
Xdd10=xdd(10,20002:40001)';
Xdd11=xdd(11,20002:40001)';
Xdd12=xdd(12,20002:40001)';
```

```
Xd1=xd(1,20002:40001)';
Xd2=xd(2,20002:40001)';
Xd3=xd(3,20002:40001)';
Xd4=xd(4,20002:40001)';
Xd5=xd(5,20002:40001)';
Xd6=xd(6,20002:40001)';
Xd7=xd(7,20002:40001)';
Xd8=xd(8,20002:40001)';
Xd9=xd(9,20002:40001)';
Xd10=xd(10,20002:40001)';
Xd11=xd(11,20002:40001)';
Xd12=xd(12,20002:40001)';

%============================================================
%============================================================
%==========================================================
%==========================================================
% ADAPTIVE LMS FILTER
% created by Robin Preston

function [n,e,w] = ff(time,yk,n,e,w,x,m,u);
%-----------------------------------------------------------------------
% Adaptive Filter
w(m,1) = 1;

for k =1
   n(m,k)=w(m,k)*x(m,k);
   e(m,k) = yk(m,k)-n(m,k);
   w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k =2
   n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1);
   e(m,k) = yk(m,k)-n(m,k);
   w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k=3
   n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1)+w(m,k)*x(m,k-2);
   e(m,k) = yk(m,k)-n(m,k);
   w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k=4
   n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1)+w(m,k)*x(m,k-2)+w(m,k)*x(m,k-3);
   e(m,k) = yk(m,k)-n(m,k);
   w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k=5
   n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1)+w(m,k)*x(m,k-2)+w(m,k)*x(m,k-
3)+w(m,k)*x(m,k-4);
   e(m,k) = yk(m,k)-n(m,k);
   w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k=6:(length(time)-1)
   n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1)+w(m,k)*x(m,k-2)+w(m,k)*x(m,k-
3)+w(m,k)*x(m,k-4)+w(m,k)*x(m,k-5);
```

```
    e(m,k) = yk(m,k)-n(m,k);
    w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k=length(time)
    n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1)+w(m,k)*x(m,k-2)+w(m,k)*x(m,k-
3)+w(m,k)*x(m,k-4)+w(m,k)*x(m,k-5);
    e(m,k) = yk(m,k)-n(m,k);
end

%===========================================================
%===========================================================
%============================================================
%============================================================
% Defines Four-DOF system in state-space format

% Created by Robin Preston

function [y_prime] = mdof(t,y,M,C,K,F,time)
%-----------------------------------------------------------------------
% System matrices -- State-space
inv_mass = inv(M);

A11 = zeros(4,4);
A12 = eye(4,4);
A21 = [-inv_mass*K];
A22 = [-inv_mass*C];
A= [A11  A12
   A21  A22];

E  = eye(4,4);
B1 = zeros(4,4);
B2 = [-inv_mass*E];
B  = [B1
     B2];

% Pull out current force vector -- uses linear interpolation for
% values between points available in time history
F_t = interp1(time,F',t)';

% Compute first derivative values at this time step.
y_prime = A*y + B*F_t;

%============================================================
%============================================================
```

Phase II: 120 Degrees-of-Freedom

```
%=============================================================
%=============================================================
% Determining Response of 120-DOF of Benchmark Problem
% Phase II Damage Pattern 1B: Web Simulated Data

% Created by: Robin Preston

%=============================================================
% Input Variables
%-------------------------------------------------------------------------
% Load Variable
load RB
force1 = force;
K1 = K(1:12,1:12);
M1 = M(1:12,1:12);
accun=acc;
load DP1B
force2 = force;
K2 = K(1:12,1:12);
M2 = M(1:12,1:12);
Acc=acc;

% Degree of Freedom
[ndof,ndof] = size(M1);

% Damping
Xil = 0.01;
C1 = 2*Xil*(K1*M1)^(1/2);
C2 = 2*Xil*(K2*M2)^(1/2);

%=============================================================
% Build K, M, and C matrices for filter.
% All values based on benchmark problem
%-------------------------------------------------------------------------
disp('Building Matrices...');
% Mass
MM1 = 3200;              %kg
MM2 = 2300;              %kg
MM3 = 2300;              %kg
MM4 = 1600;              %kg

Io1 = 3333.33;           %kg-m^2
Io2 = 2500.00;           %kg-m^2
Io3 = 2500.00;           %kg-m^2
```

```
Io4 = 1666.67;              %kg-m^2

% Build  Mass Matrix
MM = [MM1  0  0  0  0  0  0  0  0  0  0  0
      0  MM1  0  0  0  0  0  0  0  0  0  0
      0  0 Io1  0  0  0  0  0  0  0  0  0
      0  0  0  MM2  0  0  0  0  0  0  0  0
      0  0  0  0  MM2  0  0  0  0  0  0  0
      0  0  0  0  0 Io2  0  0  0  0  0  0
      0  0  0  0  0  0  MM3  0  0  0  0  0
      0  0  0  0  0  0  0  MM3  0  0  0  0
      0  0  0  0  0  0  0   0 Io3 0  0  0
      0  0  0  0  0  0  0   0  0 MM4  0  0
      0  0  0  0  0  0  0   0  0  0 MM4  0
      0  0  0  0  0  0  0   0  0  0 0 Io4];

% Stiffness Parameter
KX1 = 106600000;           %N/m
KX2 = 106600000;           %N/m
KX3 = 106600000;           %N/m
KX4 = 106600000;           %N/m

KY1 = 67900000;            %N/m
KY2 = 67900000;            %N/m
KY3 = 67900000;            %N/m
KY4 = 67900000;            %N/m

Ko1 = 232000000;           %N/m
Ko2 = 232000000;           %N/m
Ko3 = 232000000;           %N/m
Ko4 = 232000000;           %N/m

% Build Stiffness Matrix Components
KK = zeros(12, 12);

for m = 1
   KK(m,m) = KX1+KX2;
   KK(m+3,m)=-KX1;
   KK(m,m+3)=-KX1;
end
for m = 4
   KK(m,m) = KX1+KX2;
   KK(m+3,m)=-KX1;
   KK(m,m+3)=-KX1;
```

```
end
for m = 7
   KK(m,m) = KX1+KX2;
   KK(m+3,m)=-KX1;
   KK(m,m+3)=-KX1;
end
for m=10
   KK(m,m)=KX4;
end
for m = 2
   KK(m,m) = KY1+KY2;
   KK(m+3,m)=-KY1;
   KK(m,m+3)=-KY1;
end
for m = 5
   KK(m,m) = KY1+KY2;
   KK(m+3,m)=-KY1;
   KK(m,m+3)=-KY1;
end
for m = 8
   KK(m,m) = KY1+KY2;
   KK(m+3,m)=-KY1;
   KK(m,m+3)=-KY1;
end
for m=11
   KK(m,m) = KY4;
end
for m = 3
   KK(m,m) = Ko1+Ko2;
   KK(m+3,m)=-Ko1;
   KK(m,m+3)=-Ko1;
end
for m = 6
   KK(m,m) = Ko1+Ko2;
   KK(m+3,m)=-Ko1;
   KK(m,m+3)=-Ko1;
end
for m = 9
   KK(m,m) = Ko1+Ko2;
   KK(m+3,m)=-Ko1;
   KK(m,m+3)=-Ko1;
end
for m = 12
   KK(m,m) = Ko4;
```

```
end

% Build Damping Matrix
CC = 2*Xil*(KK*MM)^(1/2);

%==============================================================
% Time Vector and Force Matrix
%--------------------------------------------------------------------------
t       = 40;
t_1     = 20;                    %seconds  pt where stiffness changes
dt      = 0.001;                  %seconds

time   = 0:dt:t;
t_span1 = 0:dt:t_1;
t_span2 = t_1:dt:t;

f = zeros(1,20001);
F  = [force(:,1)'; force(:,2)'; f; force(:,3)'; force(:,4)'; f; force(:,5)'; force(:,6)'; f; force(:,7)';
force(:,8)'; f];
F2 = [force2(:,1)'; force2(:,2)'; f; force2(:,3)'; force2(:,4)'; f; force2(:,5)'; force2(:,6)'; f;
force2(:,7)'; force2(:,8)'; f];

%==============================================================
% Load Data
%--------------------------------------------------------------------------
FF = [F F2(:,2:length(F2))];


% load XXun1;
% load XXdun1;
% load XXddun1;

load XXun1extra;
load XXdun1extra;
load XXddun1extra;

%==============================================================
% Adaptive LMS Filter
%--------------------------------------------------------------------------
disp('Running Adaptive LMS Filter');
% Filter Variables
u      = [.5;.5;1000;10;10;1000;10;10;1000;10;10;1000];

m      = 6;               %Number of Taps
```

```
% Run Filter Function
yk = zeros(12, length(time));
n  = zeros(12, length(time));
e  = zeros(12, length(time));
w  = zeros(12, length(time));
for k = 1:length(t_span1)
    yk(:,k) = FF(:,k)-MM(:,:)*xdd(:,k)-KK(:,:)*x(:,k)-CC(:,:)*xd(:,k);
end
for k = 20002:length(time)
    yk(:,k) = FF(:,k)-MM(:,:)*xdd(:,k)-KK(:,:)*x(:,k)-CC(:,:)*xd(:,k);
end

for m = 1: 12
    [n,e,w] = ff(time,yk,n,e,w,x,m,u);
end

%==========================================================
% Calculate Change in Alpha
%------------------------------------------------------------------------
 warning off MATLAB:divideByZero
 warning off MATLAB:singularMatrix
 disp('Calculating Alpha Values...');


KKK = [KX1 KY1 Ko1 KX2 KY2 Ko2 KX3 KY3 Ko3 KX4 KY4 Ko4];

    g1  = 0.85*max(x(1,:));
    g2  = 0.85*max(x(2,:));
    g3  = 0.95*max(x(3,:));
    g4  = 0.85*max(-x(1,:)+x(7,:));
    g5  = 0.85*max(-x(2,:)+x(8,:));
    g6  = 0.90*max(-x(3,:)+x(9,:));
    g7  = 0.85*max(-x(4,:)+x(10,:));
    g8  = 0.85*max(-x(5,:)+x(11,:));
    g9  = 1.10*max(-x(6,:)+x(12,:));
    g10  = 0.85*max(x(10,:)-x(7,:));
    g11  = 0.85*max(x(11,:)-x(8,:));
    g12  = 0.95*max(x(12,:)-x(9,:));


Alpha = zeros(12,length(time));
for k = 2:length(time)
    for m = 10:12
```

```
        Alpha(m,k) = yk(m,k)/((x(m,k)-x(m-3,k))*KKK(1,m));
    end
    for m = 7:9
        Alpha(m,k) = (yk(m,k)-[Alpha(m+3,k)*(x(m,k)-x(m+3,k))])/((-x(m-
3,k)+x(m,k))*KKK(1,m));
    end

    for m = 4:6
        Alpha(m,k) = (yk(m,k)-[Alpha(m+3,k)*(x(m,k)-x(m+3,k))])/((-x(m-
3,k)+x(m,k))*KKK(1,m));
    end
    for m = 1:3
        Alpha(m,k) = (yk(m,k)-Alpha(m+3,k)*(x(m,k)-x(m+3,k)))/((x(m,k))*KKK(1,m));
    end
end

for k = 2:length(time)
    for m = 1
        if abs(x (m,k)) < g1
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
    for m = 2
        if abs(x (m,k)) < g2
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
   for m = 3
        if abs(x (m,k)) < g3
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
    for m =4
        if abs(-x(m-3,k)+x(m,k)) < g4
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
    for m = 5
        if abs(-x(m-3,k)+x(m,k)) < g5
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
    for m = 6
        if abs(-x(m-3,k)+x(m,k)) < g6
```

```
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
    for m = 7
        if abs(-x(m-3,k)+x(m,k)) < g7
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
    for m = 8
        if abs(-x(m-3,k)+x(m,k)) < g8
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
    for m = 9
        if abs(-x(m-3,k)+x(m,k)) < g9
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
   for m = 10
        if abs(x(m,k)-x(m-3,k)) < g10
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
     for m = 11
        if abs(x(m,k)-x(m-3,k)) < g11
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
     for m = 12
        if abs(x(m,k)-x(m-3,k)) < g12
            Alpha(m,k) = Alpha(m,k-1);
        end
    end
end

plot(time, Alpha)
xlabel('Time (sec)')
ylabel('Percent Change in Alpha')
grid

%============================================================
%============================================================
%================================================================
%================================================================
% MANIPULATION OF ACCELERATION DATA
% Derive Velocity and Displacement of Each Time Step
% Derive Response in rotational DOF

% created by Robin Preston
%----------------------------------------------------------------------
%  acc      =  simulated acceleration time-history.
%            It is a Nt by Nmdof array:
%            Nt    =  number of time steps
%            Nmdof =  number of measured dof
%            ===============================================
%            column number (x-coordinate, y-coordinate):
%            7 (   0, 2.5)  8 (1.25, 2.5)  9 ( 2.5, 2.5)
%            4 (   0,1.25)  5 (1.25,1.25)  6 ( 2.5,1.25)
%            1 (   0,   0)  2 (1.25,   0)  3 ( 2.5,   0)
%            ===============================================
%            for each floor, sensors are located at columns
%            2, 6, 8 and 4.
%            acc(:,1)  - floor 1 of column 2 in x-direction
%            acc(:,2)  - floor 1 of column 6 in y-direction
%            acc(:,3)  - floor 1 of column 8 in x-direction
%            acc(:,4)  - floor 1 of column 4 in y-direction
%            acc(:,5)  - floor 2 of column 2 in x-direction
%            acc(:,6)  - floor 2 of column 6 in y-direction
%            acc(:,7)  - floor 2 of column 8 in x-direction
%            acc(:,8)  - floor 2 of column 4 in y-direction
%            acc(:,9)  - floor 3 of column 2 in x-direction
%            acc(:,10) - floor 3 of column 6 in y-direction
%            acc(:,11) - floor 3 of column 8 in x-direction
%            acc(:,12) - floor 3 of column 4 in y-direction
%            acc(:,13) - floor 4 of column 2 in x-direction
%            acc(:,14) - floor 4 of column 4 in y-direction
%            acc(:,15) - floor 4 of column 6 in x-direction
%            acc(:,16) - floor 4 of column 8 in y-direction
%----------------------------------------------------------------------

% Calculate Acceleration for 12 DOF for Undamaged
load Case1Undamaged

acc1  = (acc(:,3)+acc(:,1))/2;
acc2  = (acc(:,4)+acc(:,2))/2;
acc3  = (acc(:,3)-acc(:,1))/2.5;
```

```
acc4  = (acc(:,7)+acc(:,5))/2;
acc5  = (acc(:,8)+acc(:,6))/2;
acc6  = (acc(:,7)-acc(:,5))/2.5;
acc7  = (acc(:,11)+acc(:,9))/2;
acc8  = (acc(:,12)+acc(:,10))/2;
acc9  = (acc(:,11)-acc(:,9))/2.5;
acc10 = (acc(:,15)+acc(:,13))/2;
acc11 = (acc(:,16)+acc(:,14))/2;
acc12 = (acc(:,15)-acc(:,13))/2.5;

% Calculate Acceleration for 12 DOF for Damage Case
load Case1Damage1

Acc1  = (acc(:,3)+acc(:,1))/2;
Acc2  = (acc(:,4)+acc(:,2))/2;
Acc3  = (acc(:,3)-acc(:,1))/2.5;
Acc4  = (acc(:,7)+acc(:,5))/2;
Acc5  = (acc(:,8)+acc(:,6))/2;
Acc6  = (acc(:,7)-acc(:,5))/2.5;
Acc7  = (acc(:,11)+acc(:,9))/2;
Acc8  = (acc(:,12)+acc(:,10))/2;
Acc9  = (acc(:,11)-acc(:,9))/2.5;
Acc10 = (acc(:,15)+acc(:,13))/2;
Acc11 = (acc(:,16)+acc(:,14))/2;
Acc12 = (acc(:,15)-acc(:,13))/2.5;

%==========================================================
%==========================================================
%============================================================
%============================================================
% Model Simulation Using Models from System ID session vela and Val

% Must be run in Workspace, then dies saved as dis.mat

% created by Robin Preston
%-------------------------------------------------------------------
% Undamaged Case
clear vela
clear Vela
clear xdd1
clear xd1
clear xdd2
clear xd2
clear x
clear xd
clear xdd

vel1  = idsim(acc1,vel_1);
vel2  = idsim(acc2,vel_2);
vel3  = idsim(acc3,vel_3);
vel4  = idsim(acc4,vel_4);
vel5  = idsim(acc5,vel_5);
vel6  = idsim(acc6,vel_6);
vel7  = idsim(acc7,vel_7);
vel8  = idsim(acc8,vel_8);
vel9  = idsim(acc9,vel_9);
vel10 = idsim(acc10,vel_10);
vel11 = idsim(acc11,vel_11);
vel12 = idsim(acc12,vel_12);

xdd1 = [acc1 acc2 acc3 acc4 acc5 acc6 acc7 acc8 acc9 acc10 acc11 acc12]';
xd1  = [vel1 vel2 vel3 vel4 vel5 vel6 vel7 vel8 vel9 vel10 vel11 vel12]';

CC = zeros(12, 20001);
for k = 1:20001
   CC(:,k)=inv(K1)*[FF(:,k)-C1*xd1(:,k)-M1*xdd1(:,k)];
end

% Damage Case
Vel1  = idsim(Acc1,Vel_1);
Vel2  = idsim(Acc2,Vel_2);
Vel3  = idsim(Acc3,Vel_3);
```

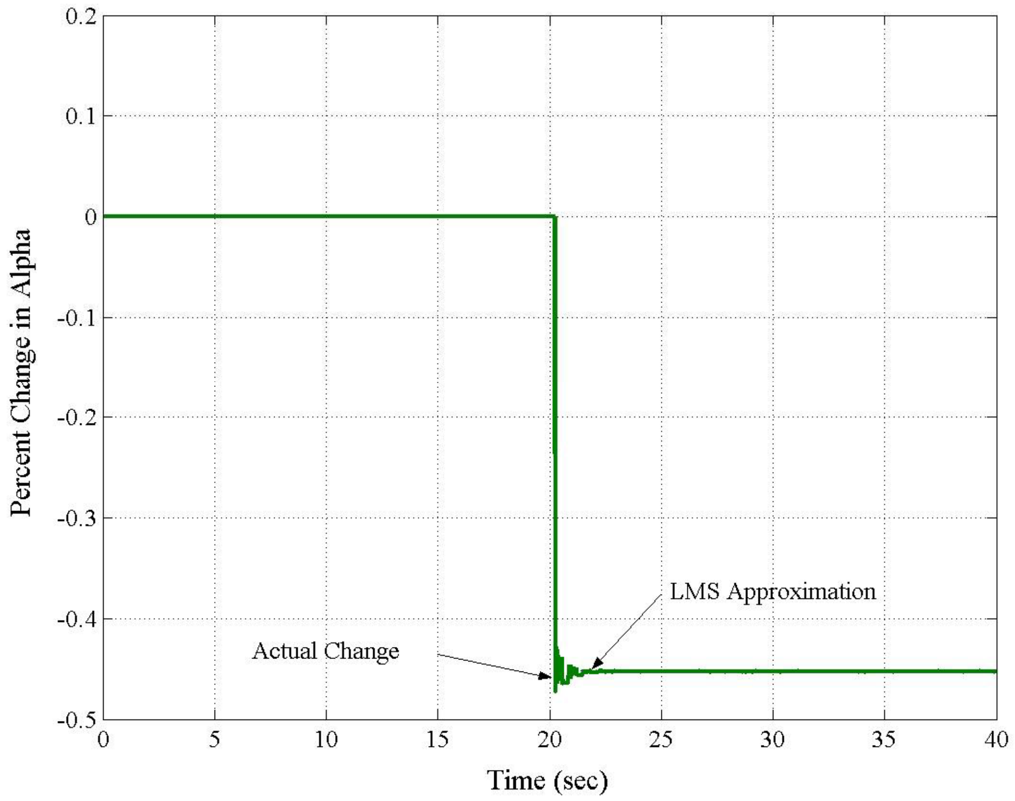
```
Vel4  = idsim(Acc4,Vel_4);
Vel5  = idsim(Acc5,Vel_5);
Vel6  = idsim(Acc6,Vel_6);
Vel7  = idsim(Acc7,Vel_7);
Vel8  = idsim(Acc8,Vel_8);
Vel9  = idsim(Acc9,Vel_9);
Vel10 = idsim(Acc10,Vel_10);
Vel11 = idsim(Acc11,Vel_11);
Vel12 = idsim(Acc12,Vel_12);

xdd2 = [Acc1 Acc2 Acc3 Acc4 Acc5 Acc6 Acc7 Acc8 Acc9 Acc10 Acc11 Acc12]';
xd2  = [Vel1 Vel2 Vel3 Vel4 Vel5 Vel6 Vel7 Vel8 Vel9 Vel10 Vel11 Vel12]';

CCC = zeros(12, 20001);
for k = 1:20001
   CCC(:,k)=inv(K2)*[FF(:,k+20000)-C2*xd2(:,k)-M2*xdd2(:,k)];
end

% Combine Data
xdd = [xdd1(:,1:length(xdd1)-1) xdd2(:,:)];

xd = [xd1(:,1:length(xd1)-1) xd2(:,:)];

x= [CC(:,1:length(CC)-1) CCC(:,:)];

%============================================================
%============================================================
%==========================================================
%==========================================================
% Generate Model Data for Simulation Model
% Creates individual displacement and velocity vectors to be inputted into
% System Identification Models

% Created by Robin Preston
%----------------------------------------------------------------------
xd1=xd(1,1:20001)';
xd2=xd(2,1:20001)';
xd3=xd(3,1:20001)';
xd4=xd(4,1:20001)';
xd5=xd(5,1:20001)';
xd6=xd(6,1:20001)';
xd7=xd(7,1:20001)';
xd8=xd(8,1:20001)';
xd9=xd(9,1:20001)';
xd10=xd(10,1:20001)';
xd11=xd(11,1:20001)';
xd12=xd(12,1:20001)';
xdd1=xdd(1,1:20001)';
xdd2=xdd(2,1:20001)';
xdd3=xdd(3,1:20001)';
xdd4=xdd(4,1:20001)';
xdd5=xdd(5,1:20001)';
xdd6=xdd(6,1:20001)';
xdd7=xdd(7,1:20001)';
xdd8=xdd(8,1:20001)';
xdd9=xdd(9,1:20001)';
xdd10=xdd(10,1:20001)';
xdd11=xdd(11,1:20001)';
xdd12=xdd(12,1:20001)';
Xdd1=xdd(1,20002:40001)';
Xdd2=xdd(2,20002:40001)';
Xdd3=xdd(3,20002:40001)';
Xdd4=xdd(4,20002:40001)';
Xdd5=xdd(5,20002:40001)';
Xdd6=xdd(6,20002:40001)';
Xdd7=xdd(7,20002:40001)';
Xdd8=xdd(8,20002:40001)';
Xdd9=xdd(9,20002:40001)';
Xdd10=xdd(10,20002:40001)';
Xdd11=xdd(11,20002:40001)';
Xdd12=xdd(12,20002:40001)';
```

```
Xd1=xd(1,20002:40001)';
Xd2=xd(2,20002:40001)';
Xd3=xd(3,20002:40001)';
Xd4=xd(4,20002:40001)';
Xd5=xd(5,20002:40001)';
Xd6=xd(6,20002:40001)';
Xd7=xd(7,20002:40001)';
Xd8=xd(8,20002:40001)';
Xd9=xd(9,20002:40001)';
Xd10=xd(10,20002:40001)';
Xd11=xd(11,20002:40001)';
Xd12=xd(12,20002:40001)';

%=============================================================
%=============================================================
%=============================================================
%=============================================================
% ADAPTIVE LMS FILTER
% created by Robin Preston

function [n,e,w] = ff(time,yk,n,e,w,x,m,u);
%-----------------------------------------------------------------------
% Adaptive Filter
w(m,1) = 1;

for k =1
   n(m,k)=w(m,k)*x(m,k);
   e(m,k) = yk(m,k)-n(m,k);
   w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k =2
   n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1);
   e(m,k) = yk(m,k)-n(m,k);
   w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k=3
   n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1)+w(m,k)*x(m,k-2);
   e(m,k) = yk(m,k)-n(m,k);
   w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k=4
   n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1)+w(m,k)*x(m,k-2)+w(m,k)*x(m,k-3);
   e(m,k) = yk(m,k)-n(m,k);
   w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k=5
   n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1)+w(m,k)*x(m,k-2)+w(m,k)*x(m,k-
3)+w(m,k)*x(m,k-4);
   e(m,k) = yk(m,k)-n(m,k);
   w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k=6:(length(time)-1)
   n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1)+w(m,k)*x(m,k-2)+w(m,k)*x(m,k-
3)+w(m,k)*x(m,k-4)+w(m,k)*x(m,k-5);
```

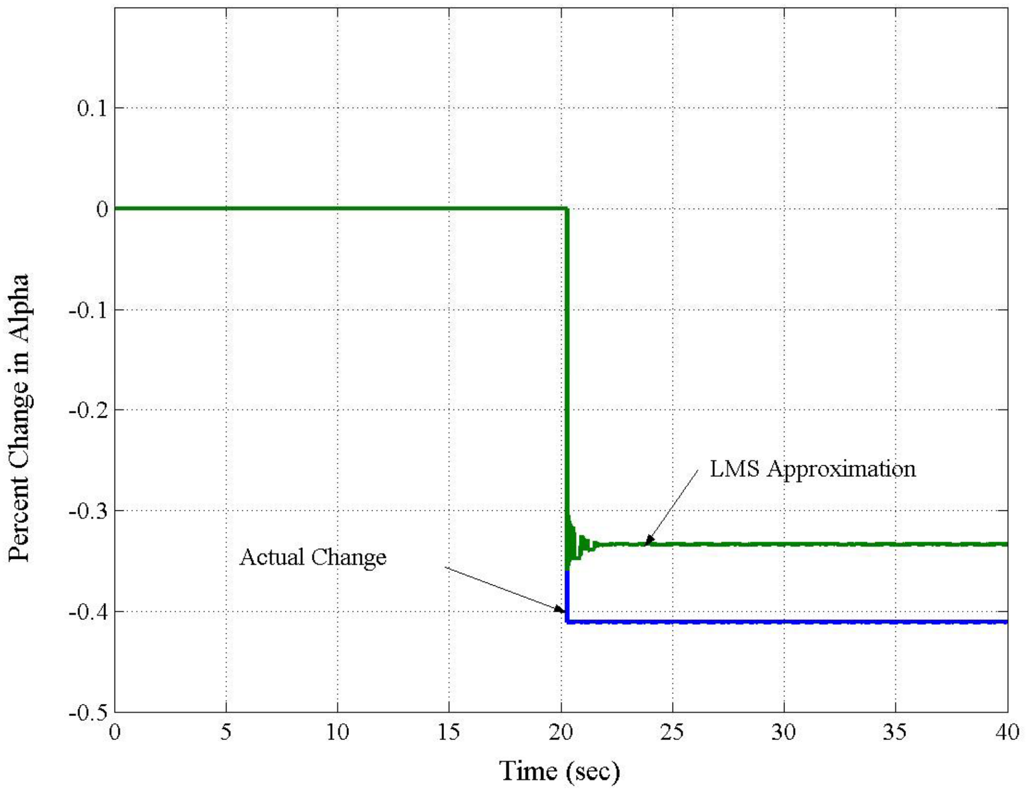
```
    e(m,k) = yk(m,k)-n(m,k);
    w(m,k+1) = w(m,k)+2*u(m,1)*e(m,k)*x(m,k);
end

for k=length(time)
    n(m,k)=w(m,k)*x(m,k)+w(m,k)*x(m,k-1)+w(m,k)*x(m,k-2)+w(m,k)*x(m,k-
3)+w(m,k)*x(m,k-4)+w(m,k)*x(m,k-5);
    e(m,k) = yk(m,k)-n(m,k);
end

%===========================================================
%===========================================================
%=========================================================
%=========================================================
% Defines Four-DOF system in state-space format

% Created by Robin Preston

function [y_prime] = mdof(t,y,M,C,K,F,time)
%-----------------------------------------------------------------------
% System matrices -- State-space
inv_mass = inv(M);

A11 = zeros(4,4);
A12 = eye(4,4);
A21 = [-inv_mass*K];
A22 = [-inv_mass*C];
A= [A11  A12
   A21  A22];

E  = eye(4,4);
B1 = zeros(4,4);
B2 = [-inv_mass*E];
B  = [B1
      B2];

% Pull out current force vector -- uses linear interpolation for
% values between points available in time history
F_t = interp1(time,F',t)';

% Compute first derivative values at this time step.
y_prime = A*y + B*F_t;

%=========================================================
%=========================================================
```

APPENDIX B

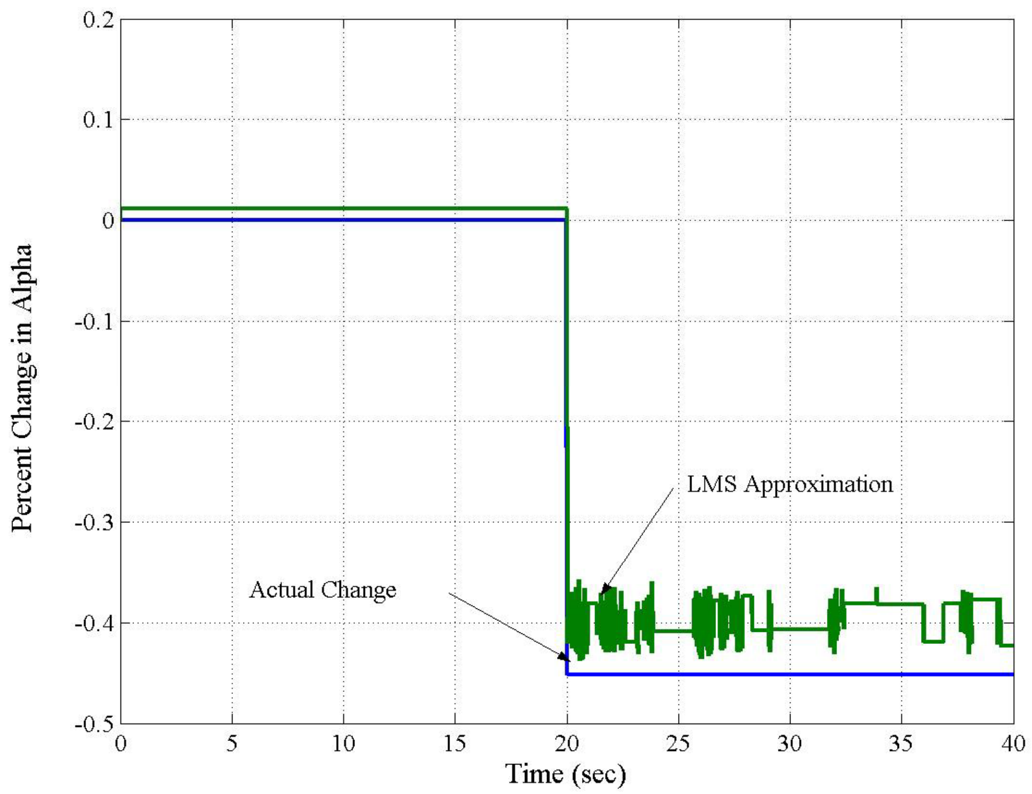
GRAPHICAL RESULTS OF ALL SCENARIOS



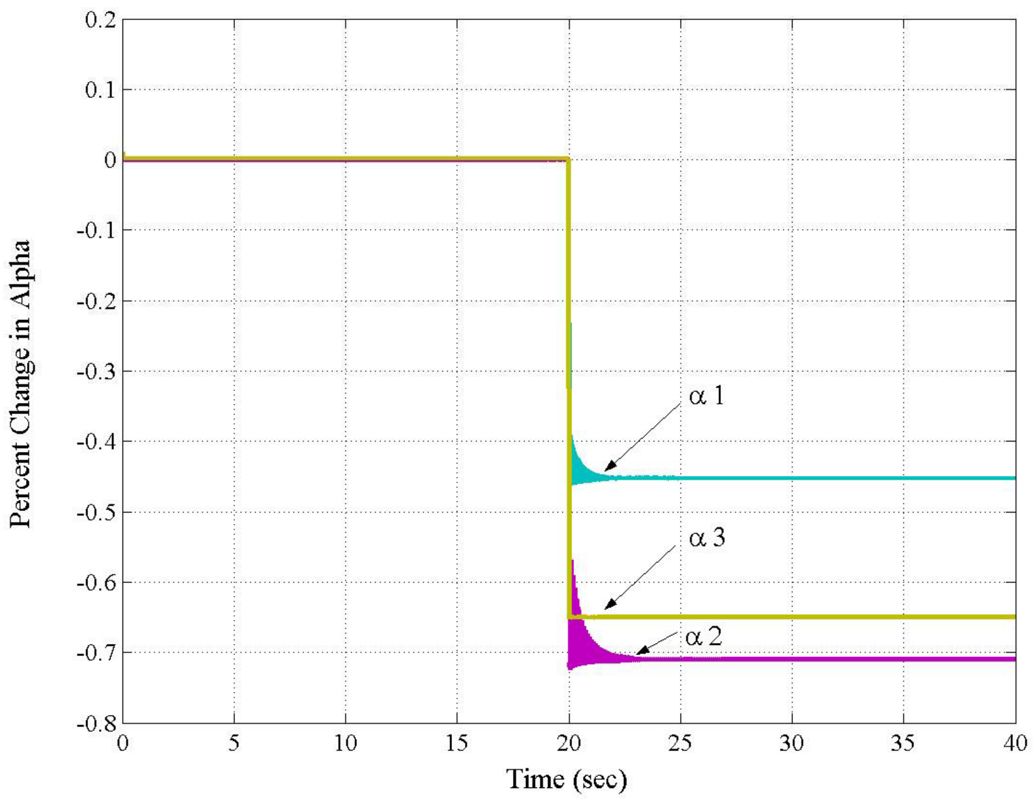
Single Degree of Freedom with Harmonic Force (No Error)



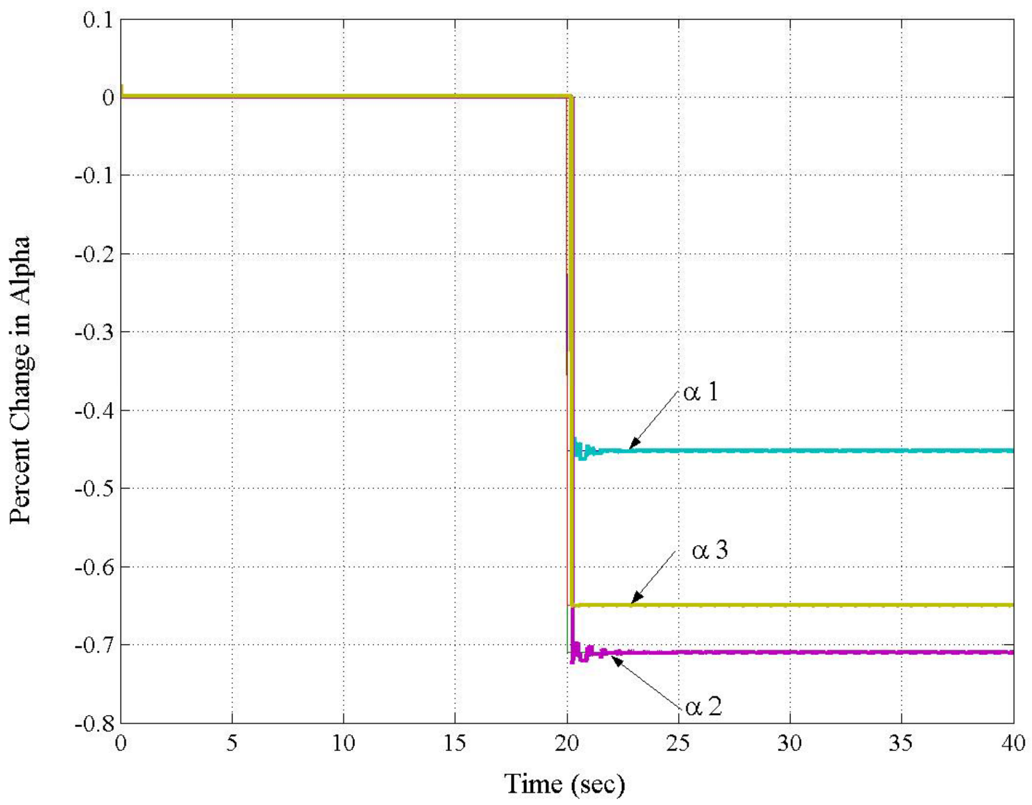
Single Degree of Freedom with Harmonic Force and Added Error



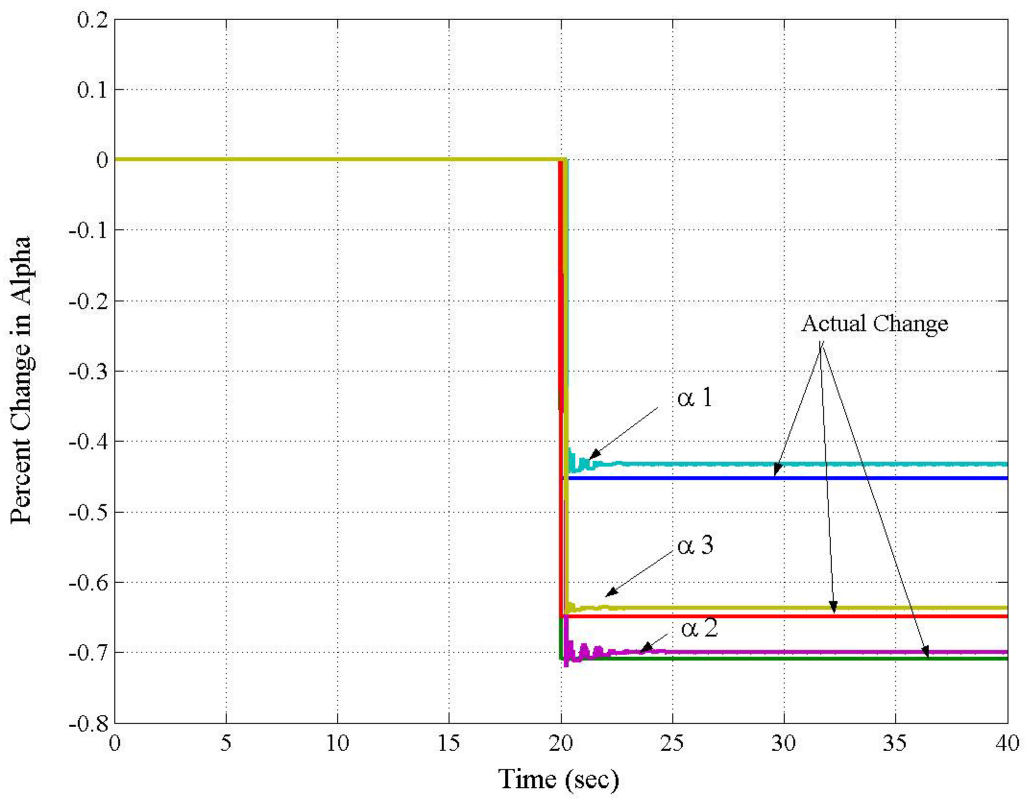
Single Degree of Freedom with Random Force and Added Error



Three Degree of Freedom with Constant Force (No Error)

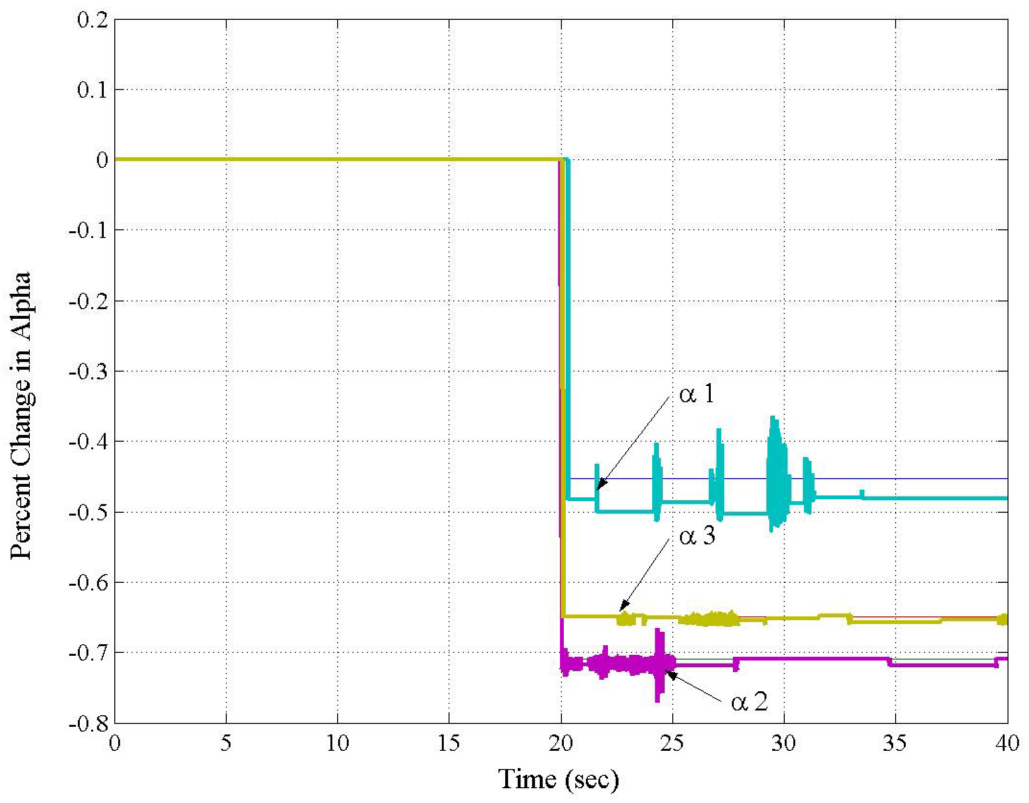


Three Degree of Freedom with Harmonic Force (No Error)

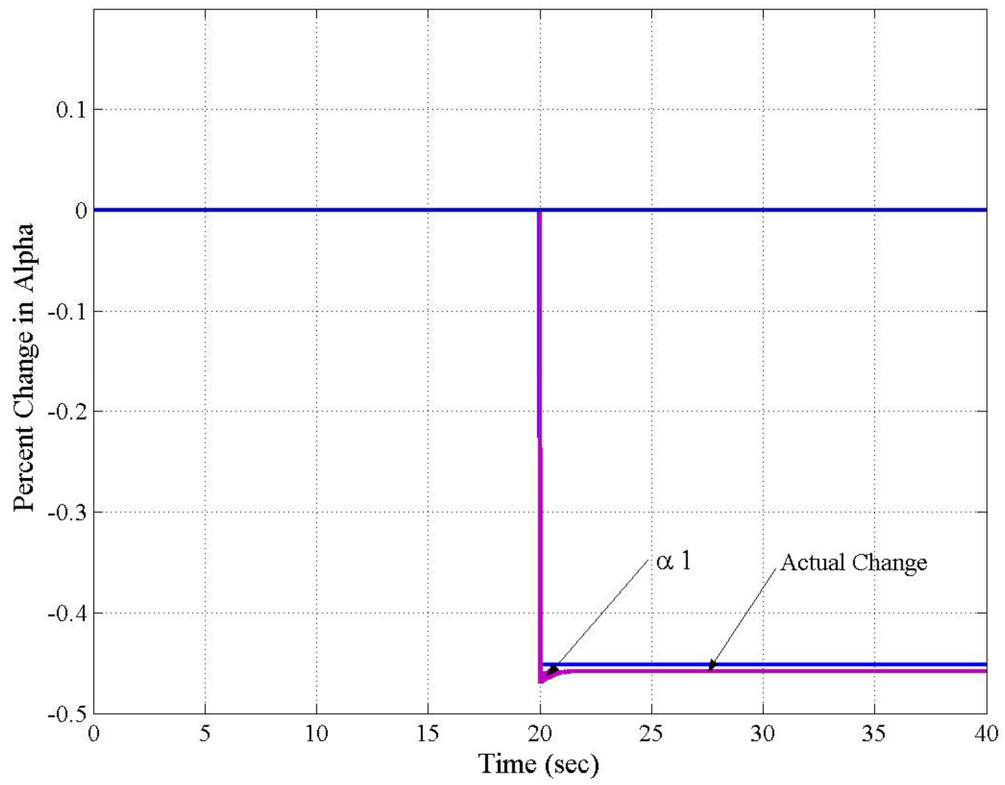


Three Degree Freedom with Harmonic Force and Added Error

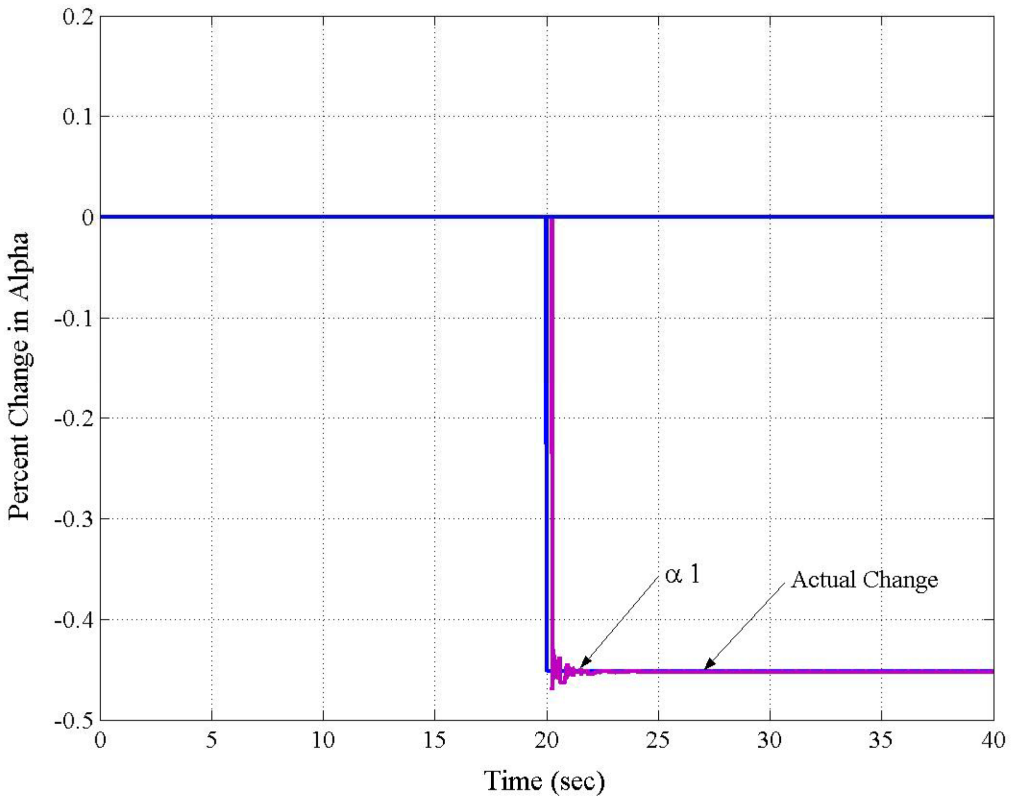
Three Degree of Freedom with Random Force (No Error)



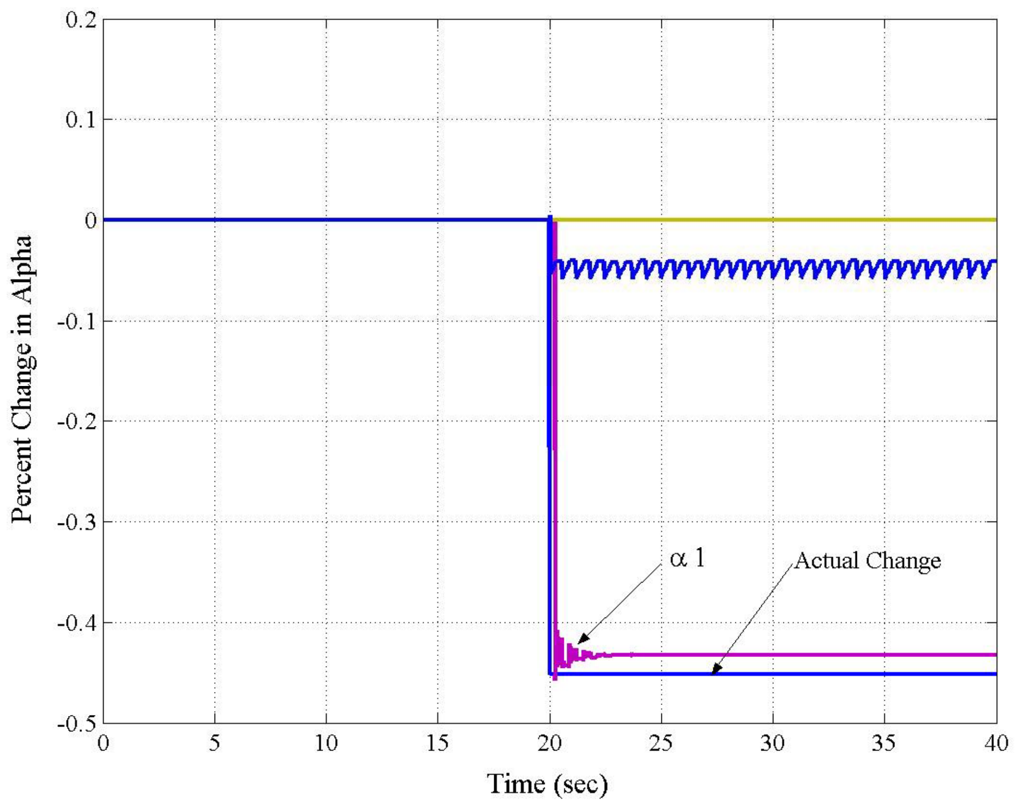
Three Degree of Freedom with Random Error and Added Error



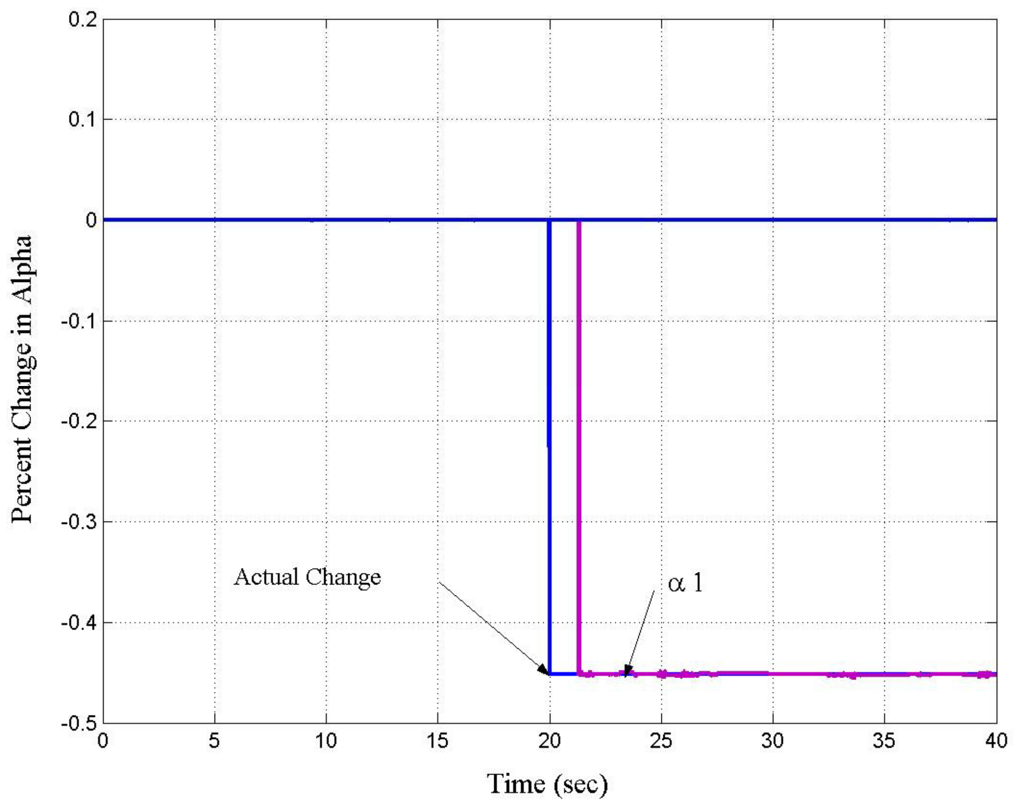
Four Degree of Freedom with Constant Force and Added Error



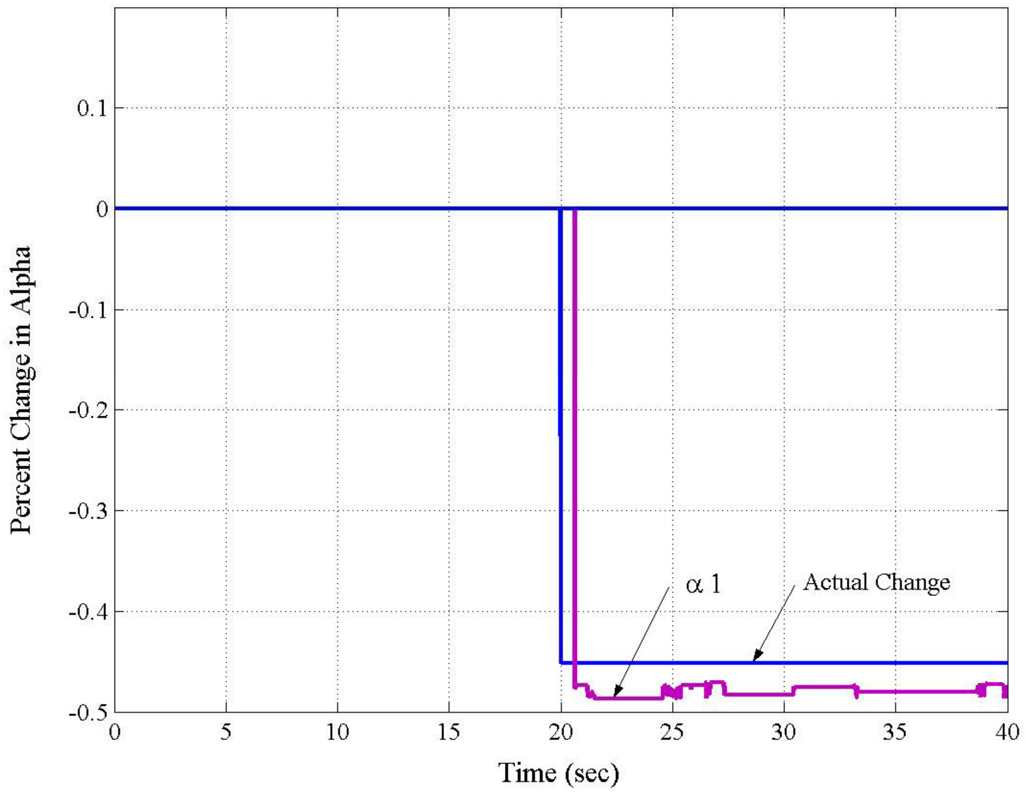
Four Degree of Freedom with Harmonic Force (No Error)



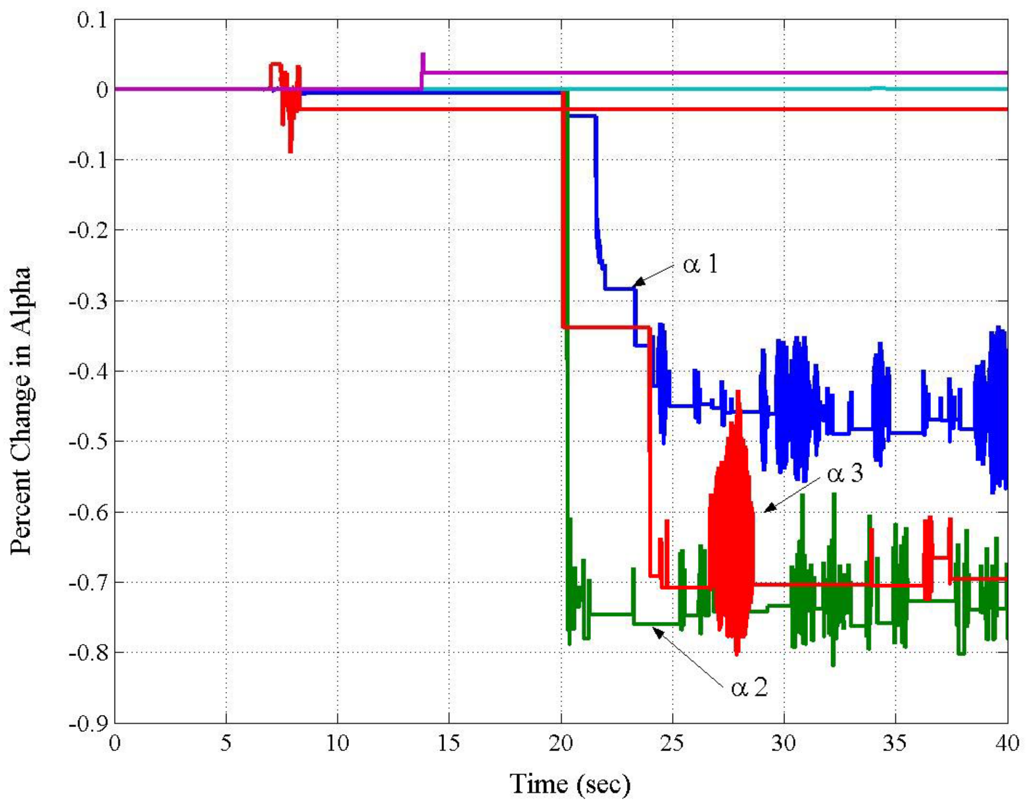
Four Degree of Freedom with Harmonic Force and Added Error



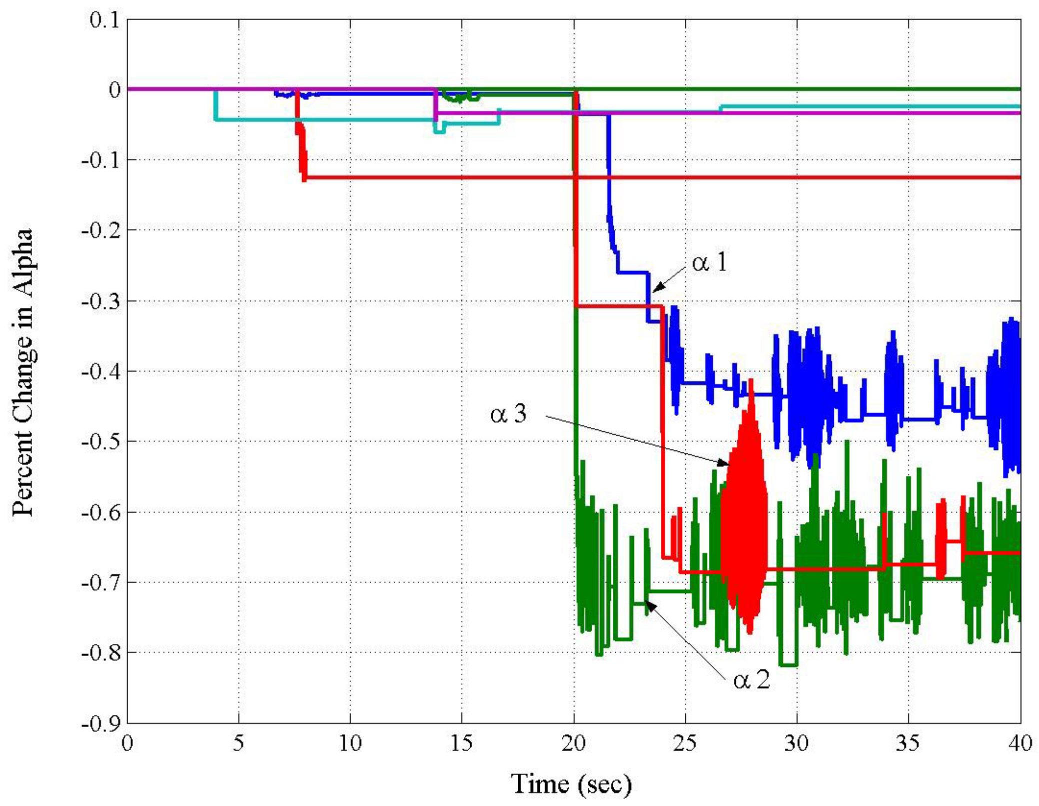
Four Degree of Freedom with Random Force (No Error)



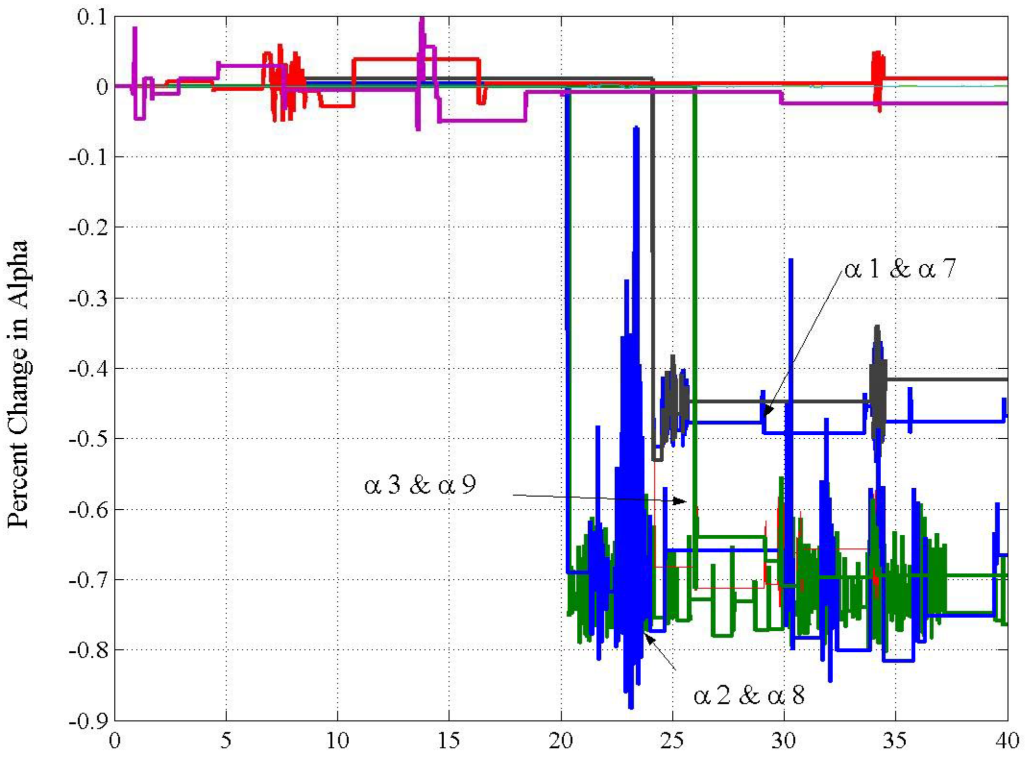
Four Degree of Freedom with Random Force and Added Error



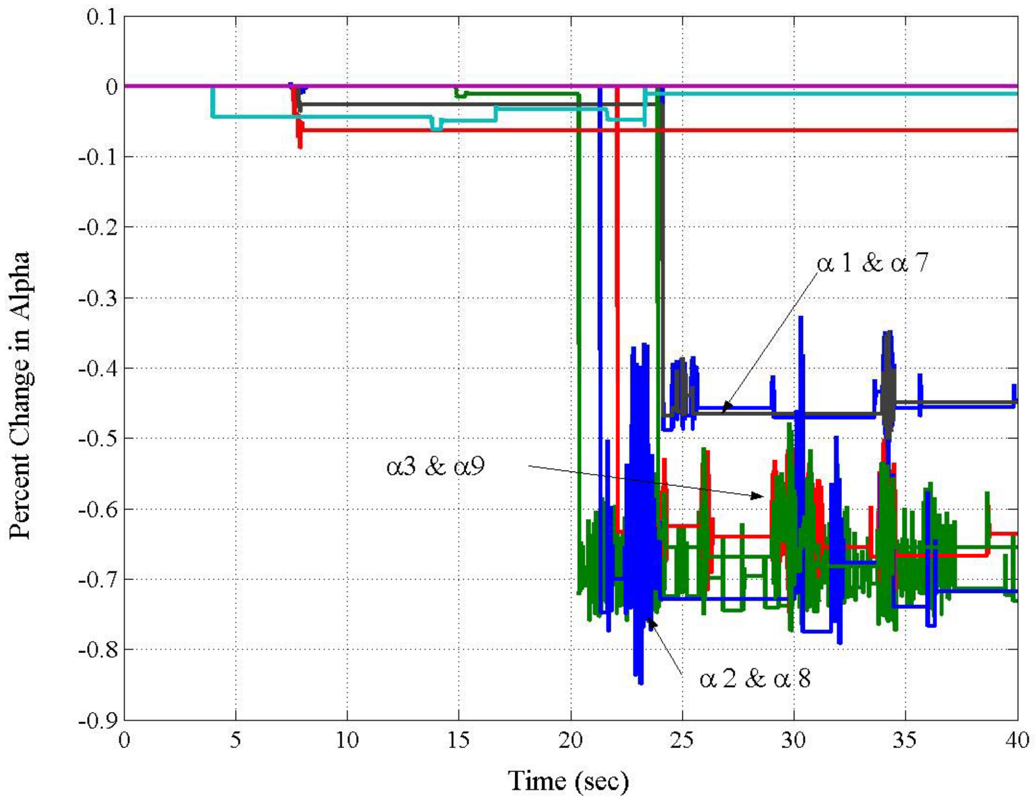
12 DOF in Case 1 and Damage Pattern 1: Own Generated Data with No Error



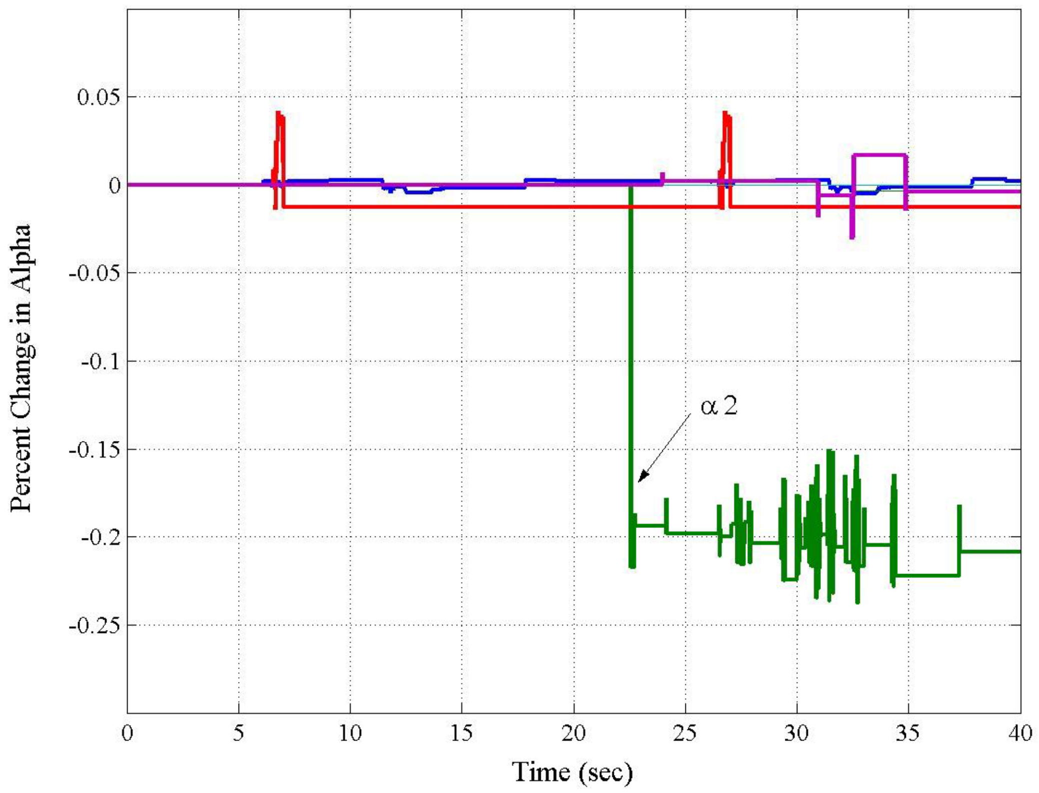
12 DOF in Case 1 and Damage Pattern 1: Own Generated Data with Added Error



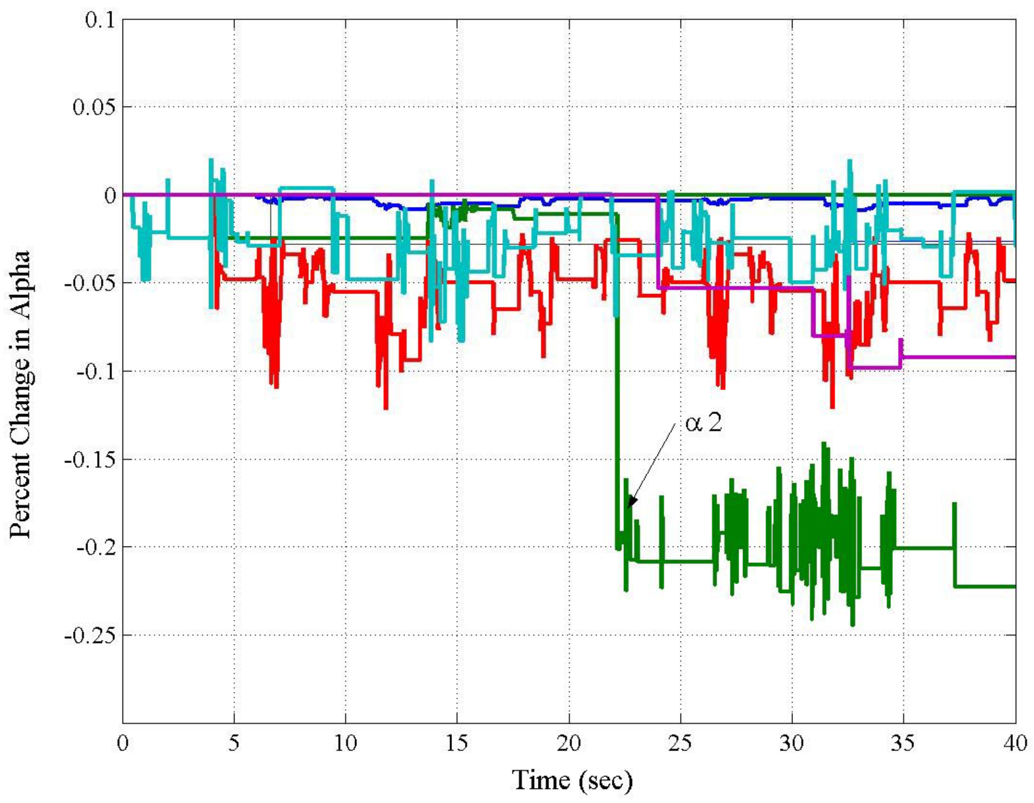
12 DOF in Case 1 and Damage Pattern 2: Own Generated Data with No Error



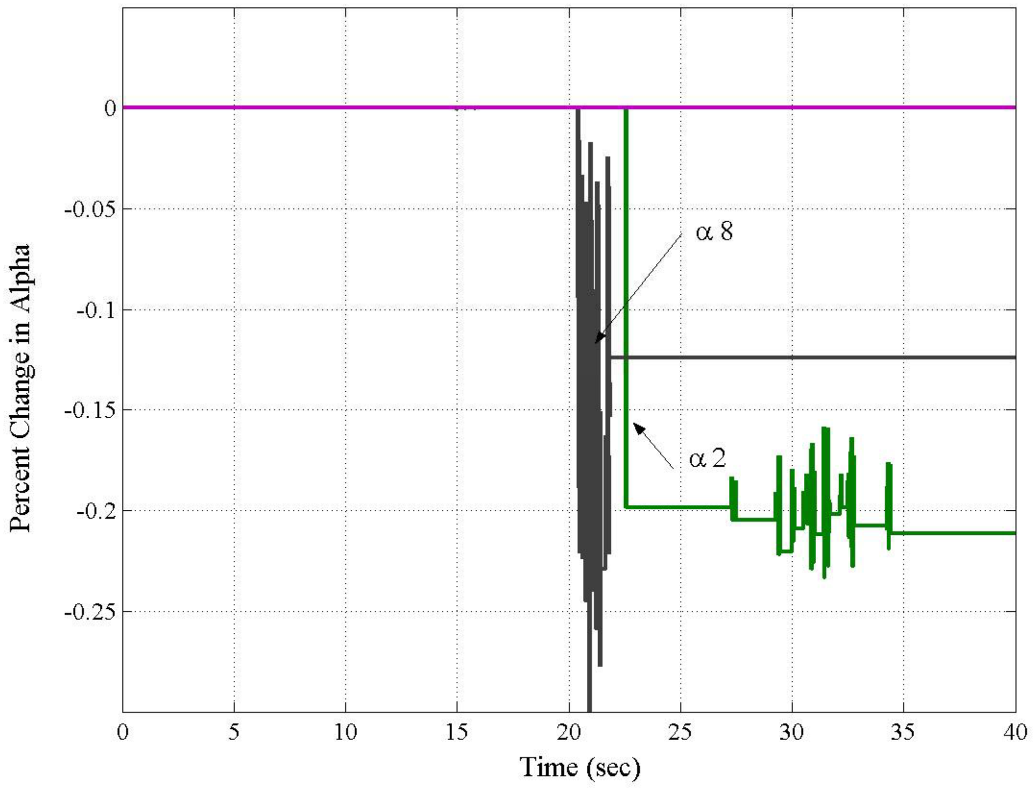
12 DOF in Case 1 and Damage Pattern 2: Own Generated Data with Added Error



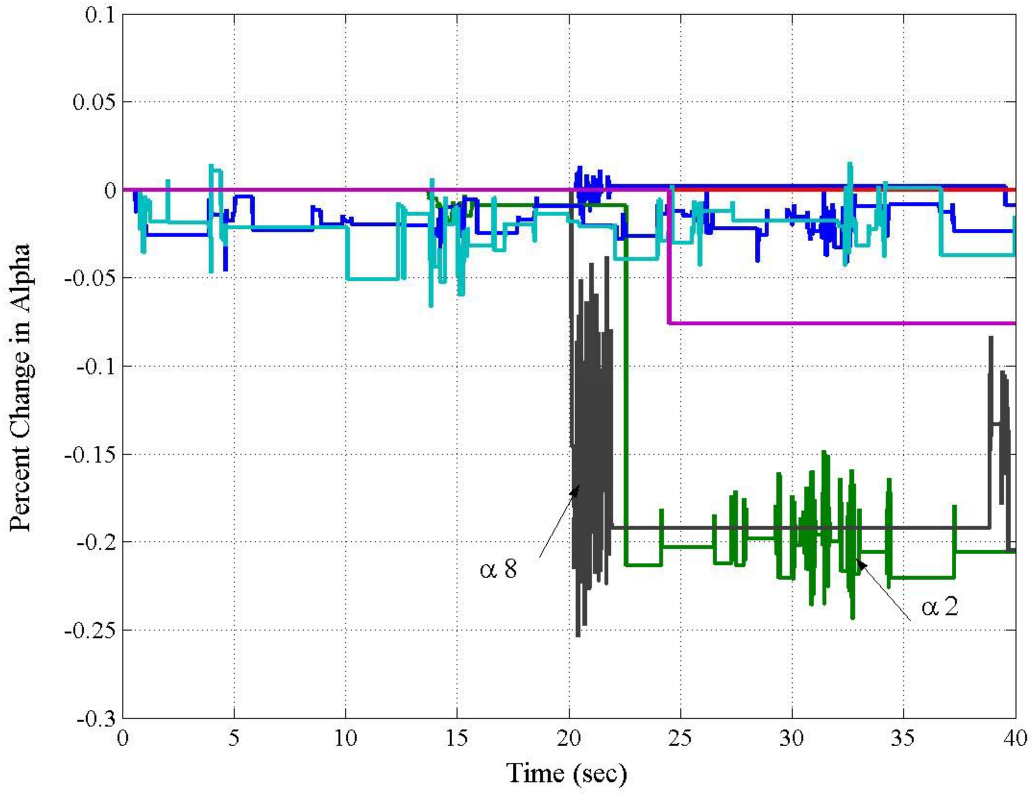
12 DOF in Case 1 and Damage Pattern 3: Own Generated Data with No Error



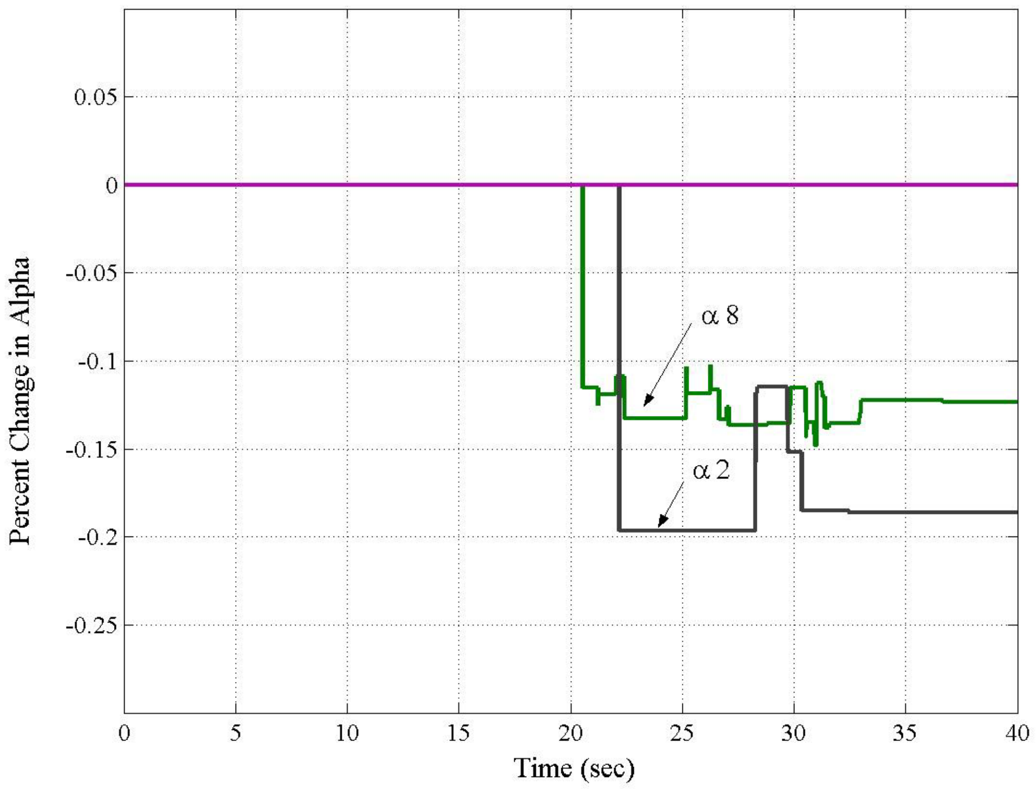
12 DOF in Case 1 and Damage Pattern 3: Own Generated Data with Added Error



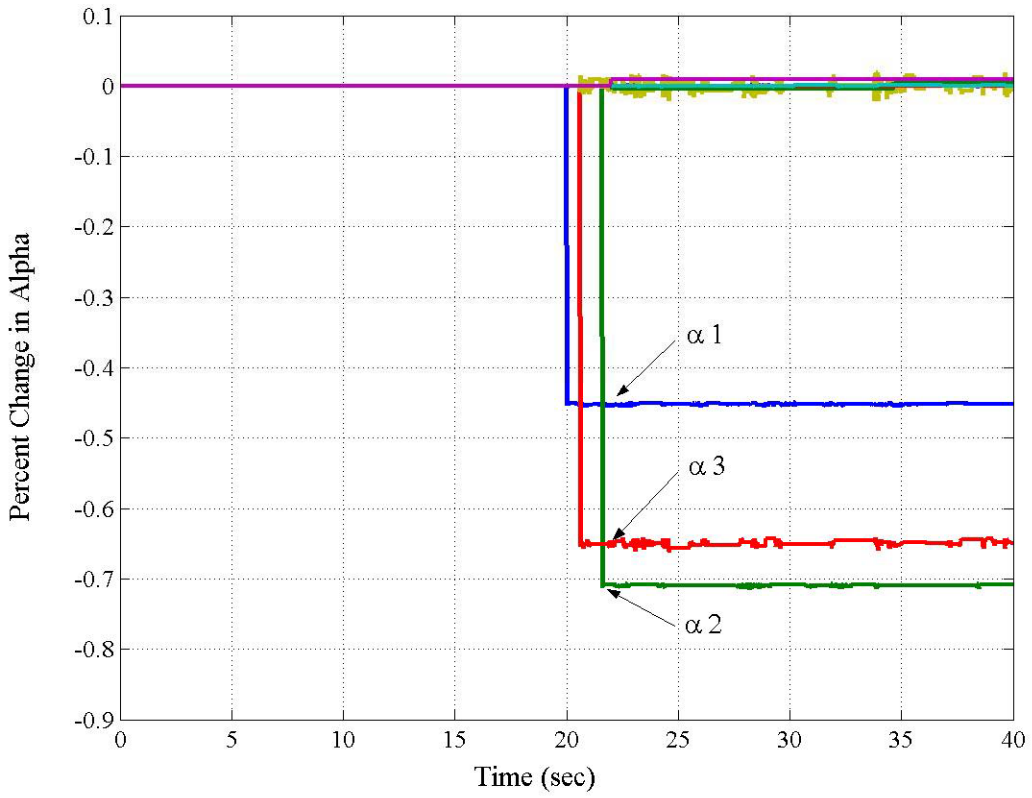
12 DOF in Case 1 and Damage Pattern 4: Own Generated Data with No Error



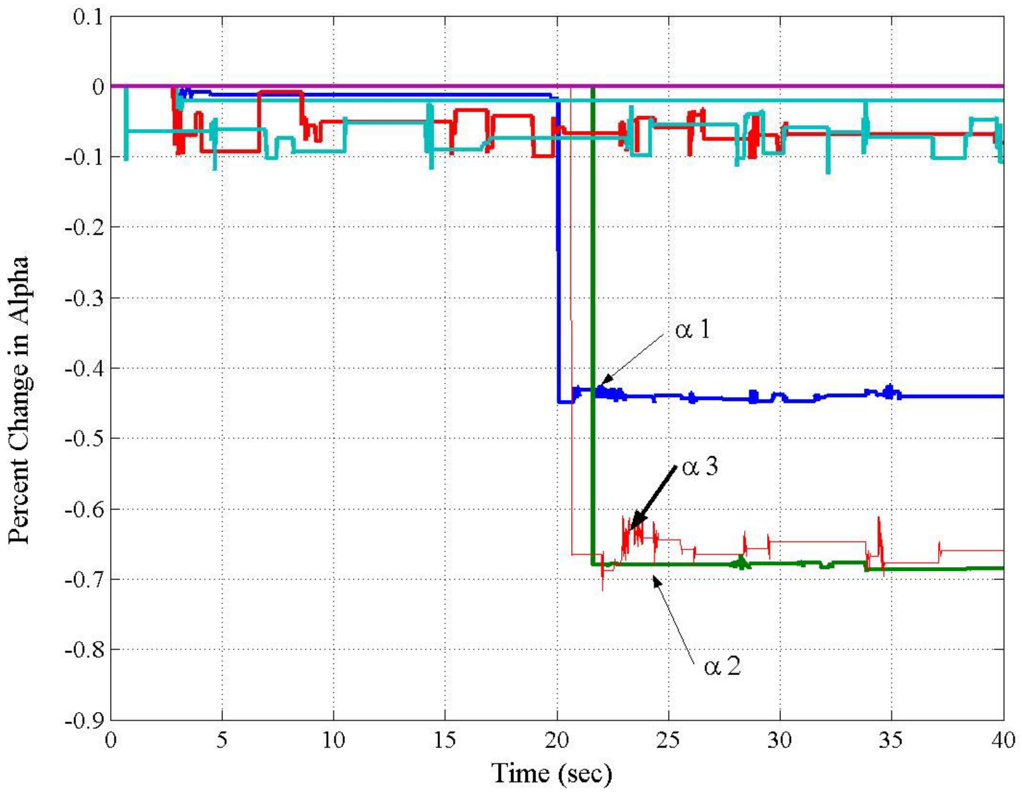
12 DOF in Case 1 and Damage Pattern 4: Own Generated Data with Added Error



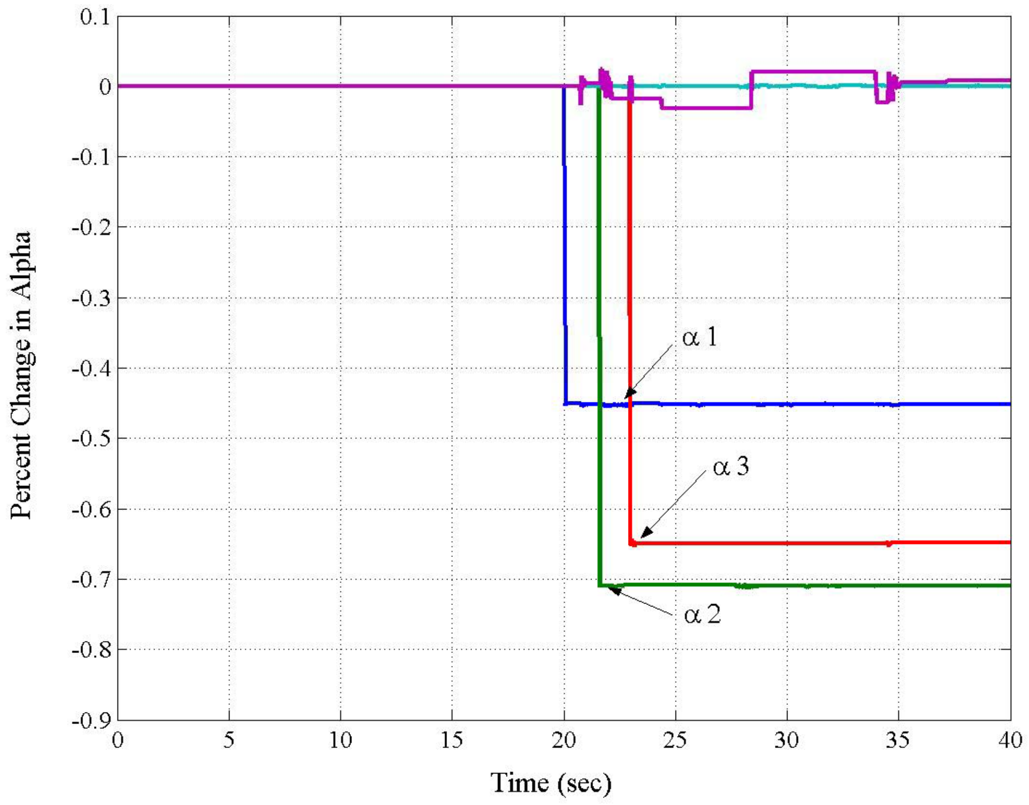
12 DOF in Case 1 and Damage Pattern 4: Data from Benchmark Problem



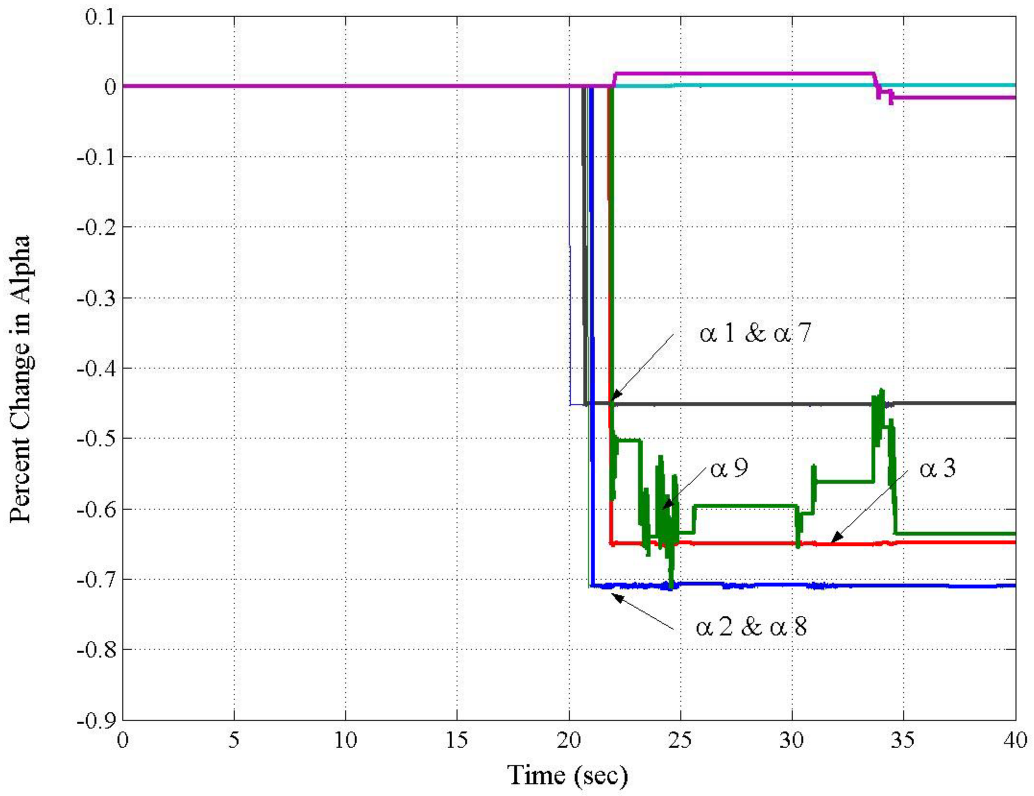
12 DOF in Case 3 and Damage Pattern 1: Own Generated Data with No Error



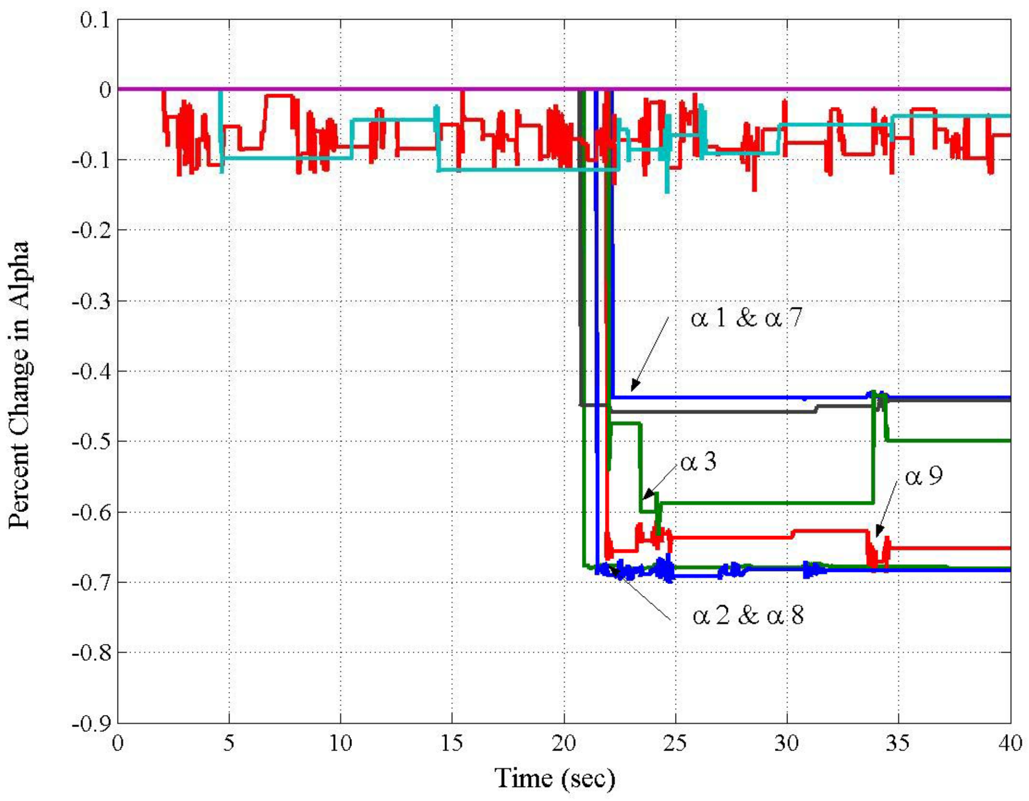
12 DOF in Case 3 and Damage Pattern 1: Own Generated Data with Added Error



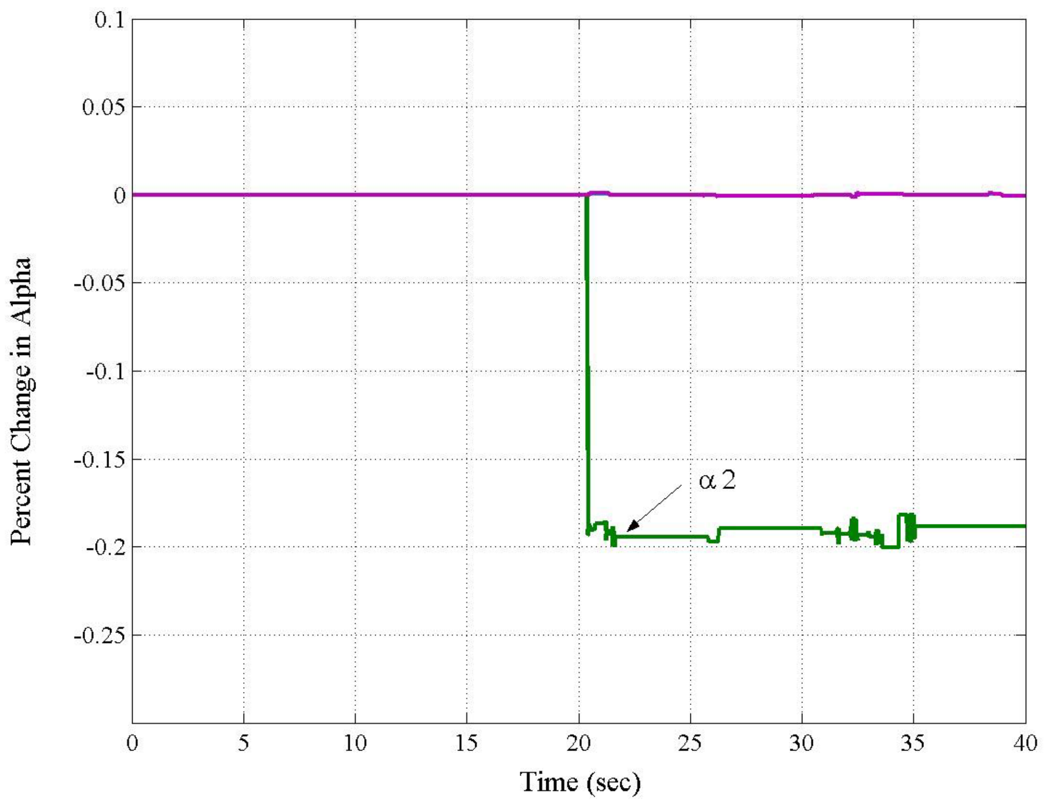
12 DOF in Case 3 and Damage Pattern 1: Data with Benchmark Problem



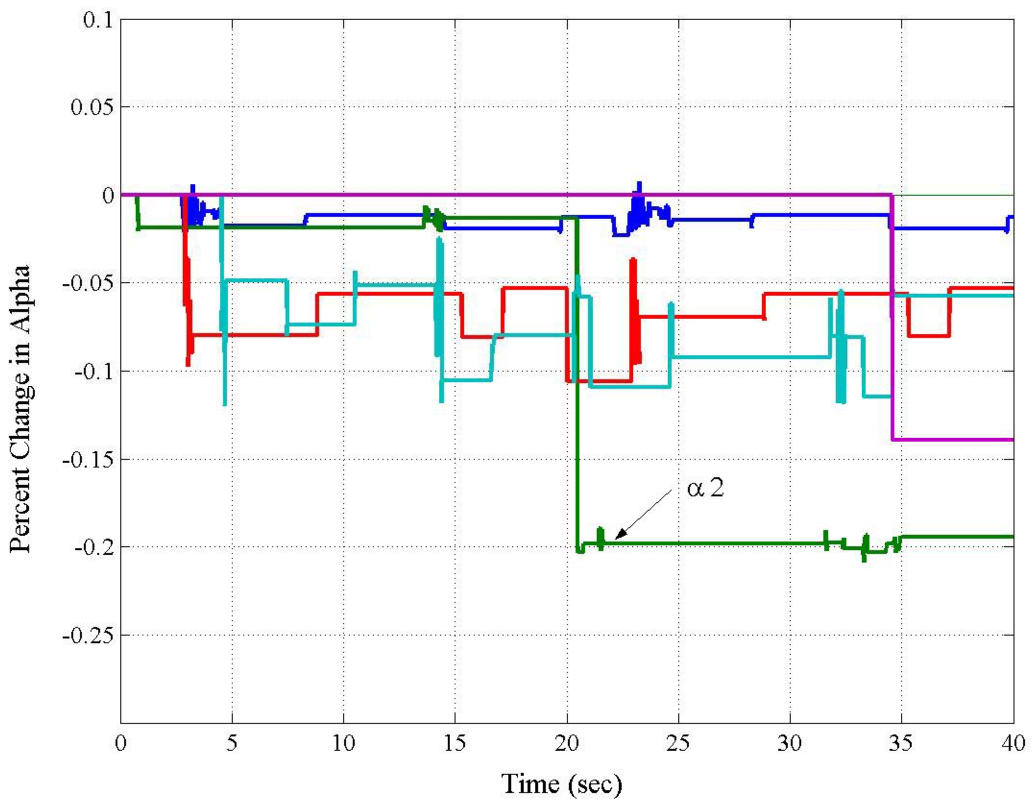
12 DOF in Case 3 and Damage Pattern 2: Own Generated Data with No Error



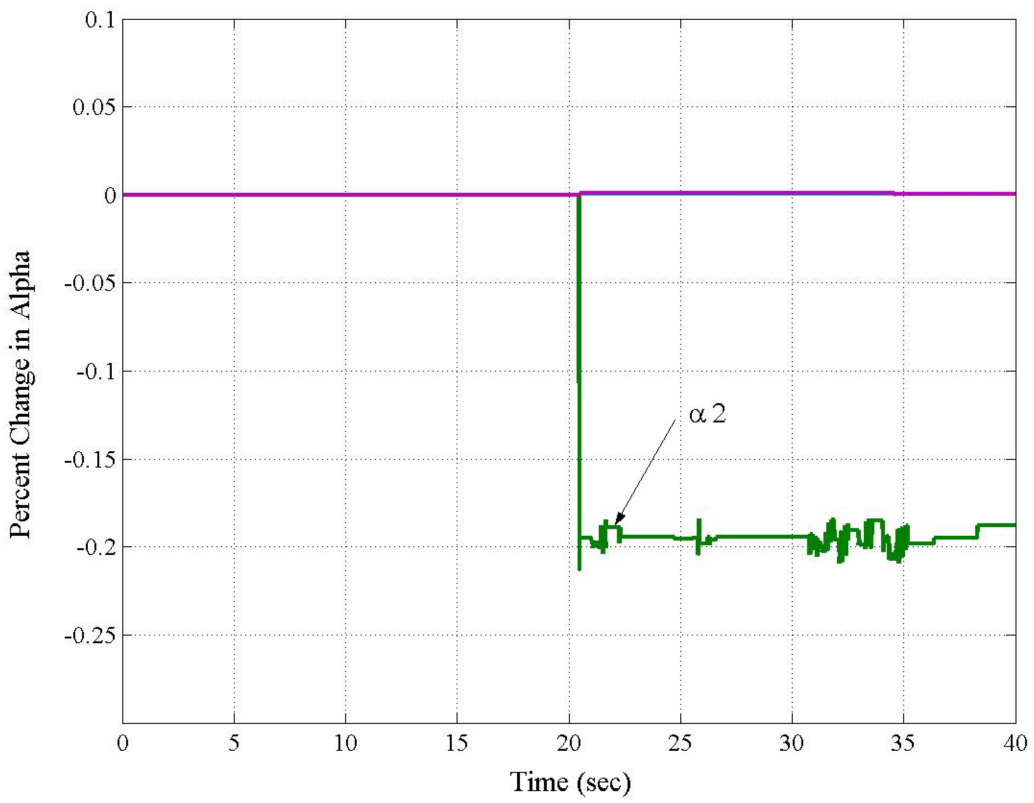
12 DOF in Case 3 and Damage Pattern 2: Own Generated Data with Added Error



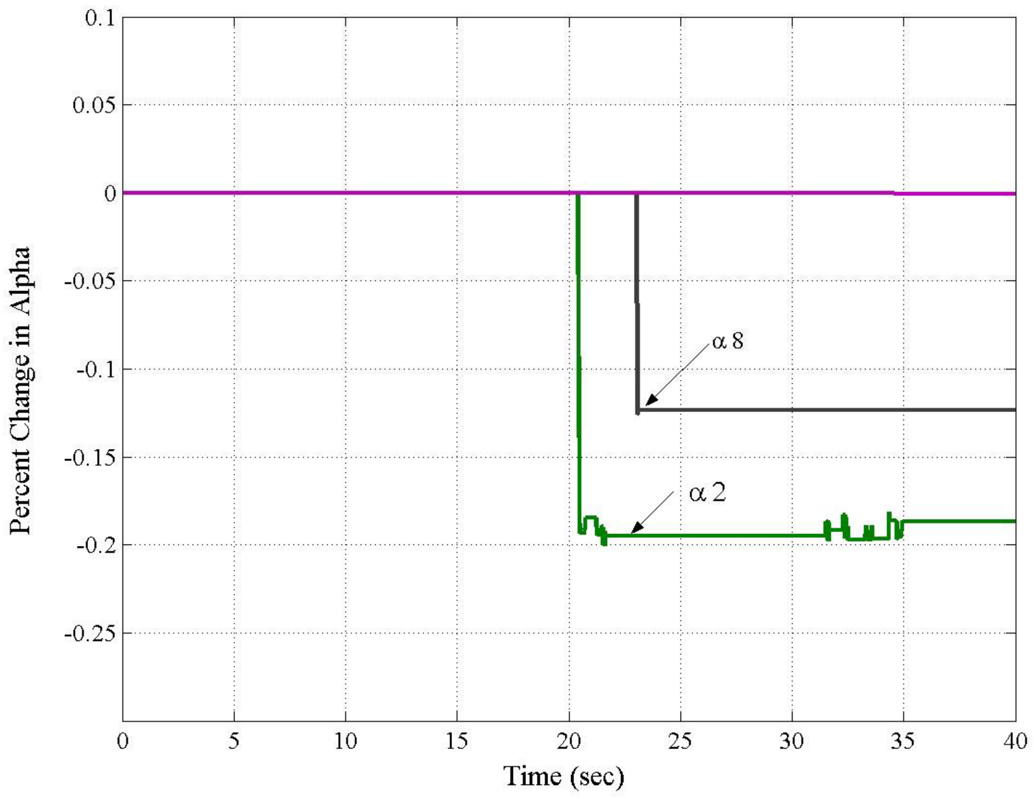
12 DOF in Case 3 and Damage Pattern 3: Own Generated Data with No Error



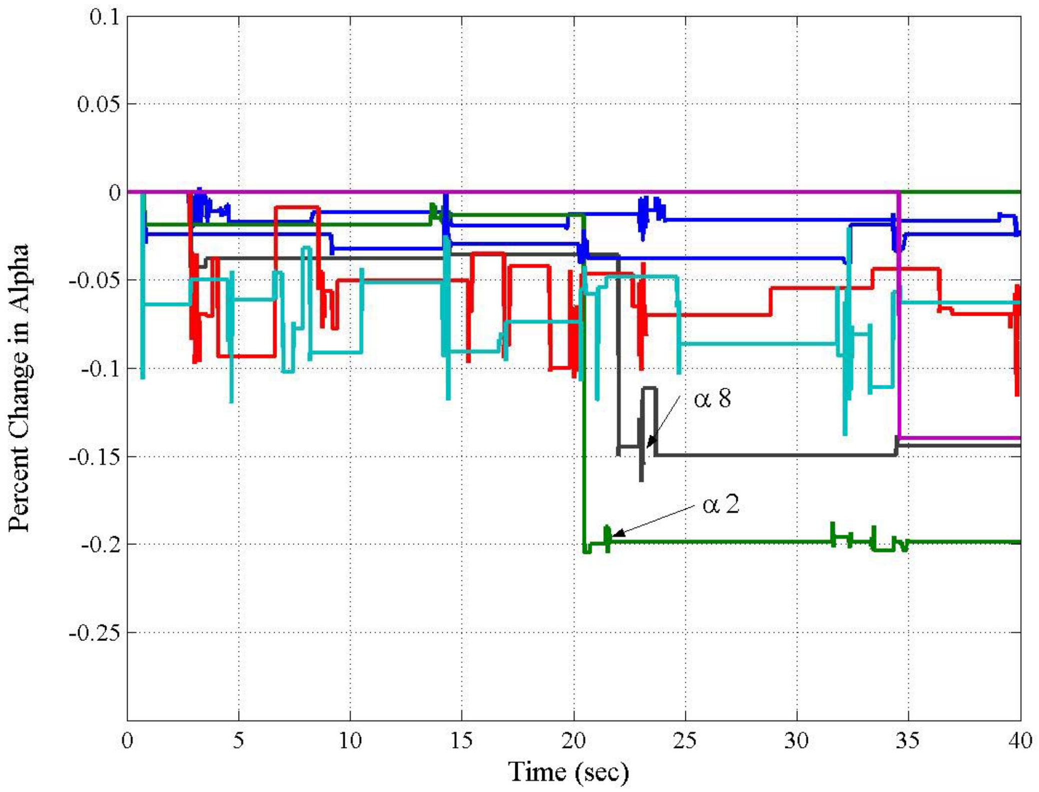
12 DOF in Case 3 and Damage Pattern 3: Own Generated Data with Added Error



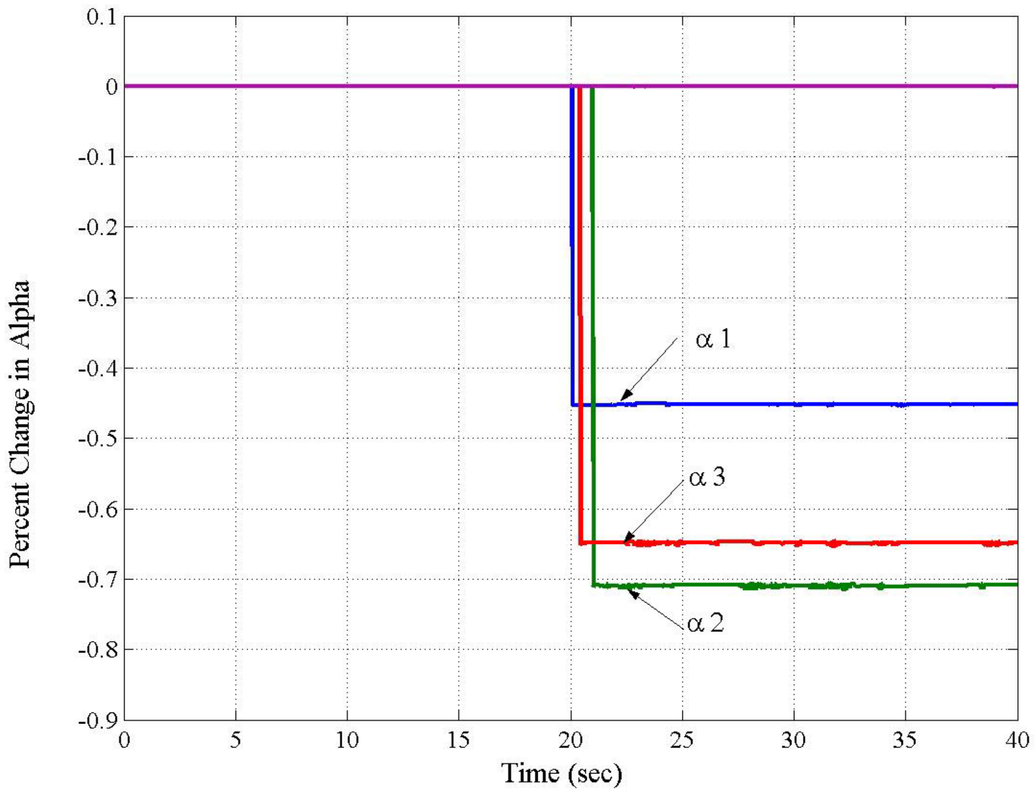
12 DOF in Case 3 and Damage Pattern 3: Data with Benchmark Problem



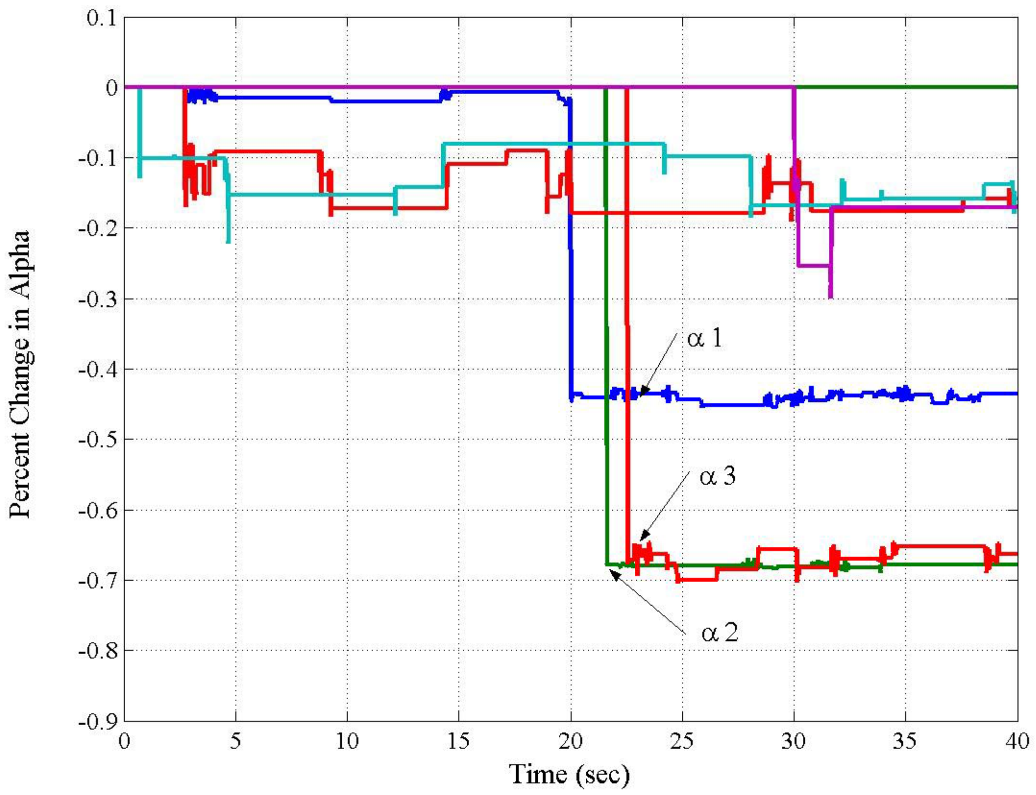
12 DOF in Case 3 and Damage Pattern 4: Own Generated Data with No Error



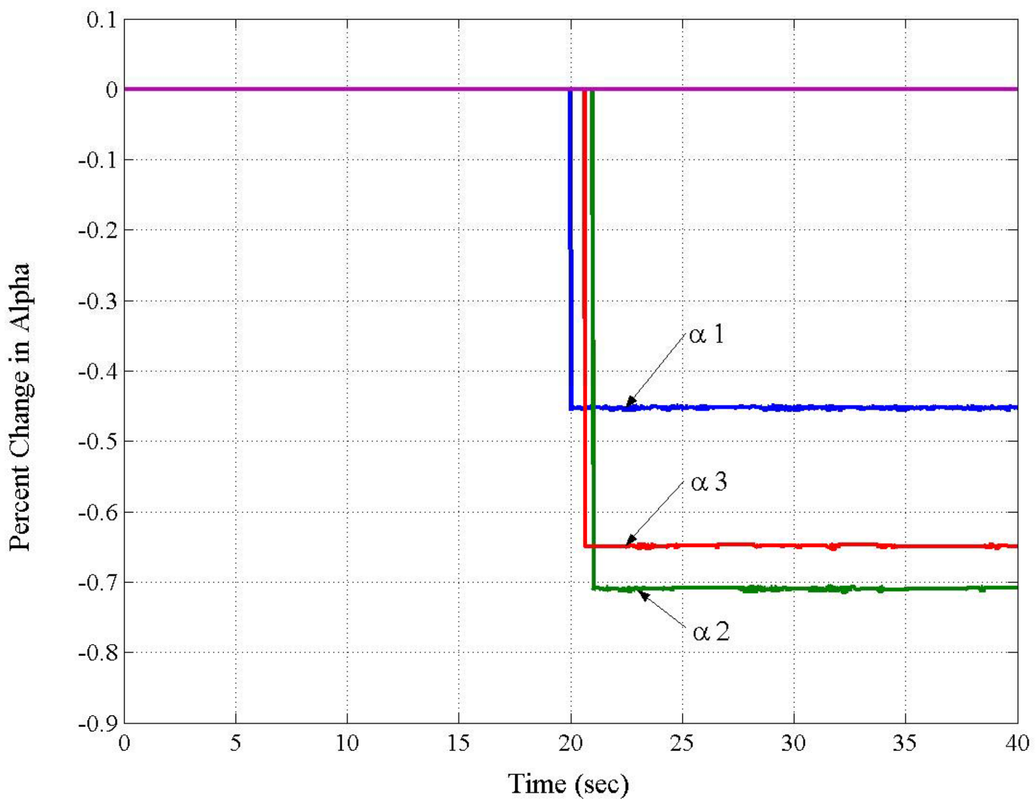
12 DOF in Case 3 and Damage Pattern 4: Own Generated Data with Added Error



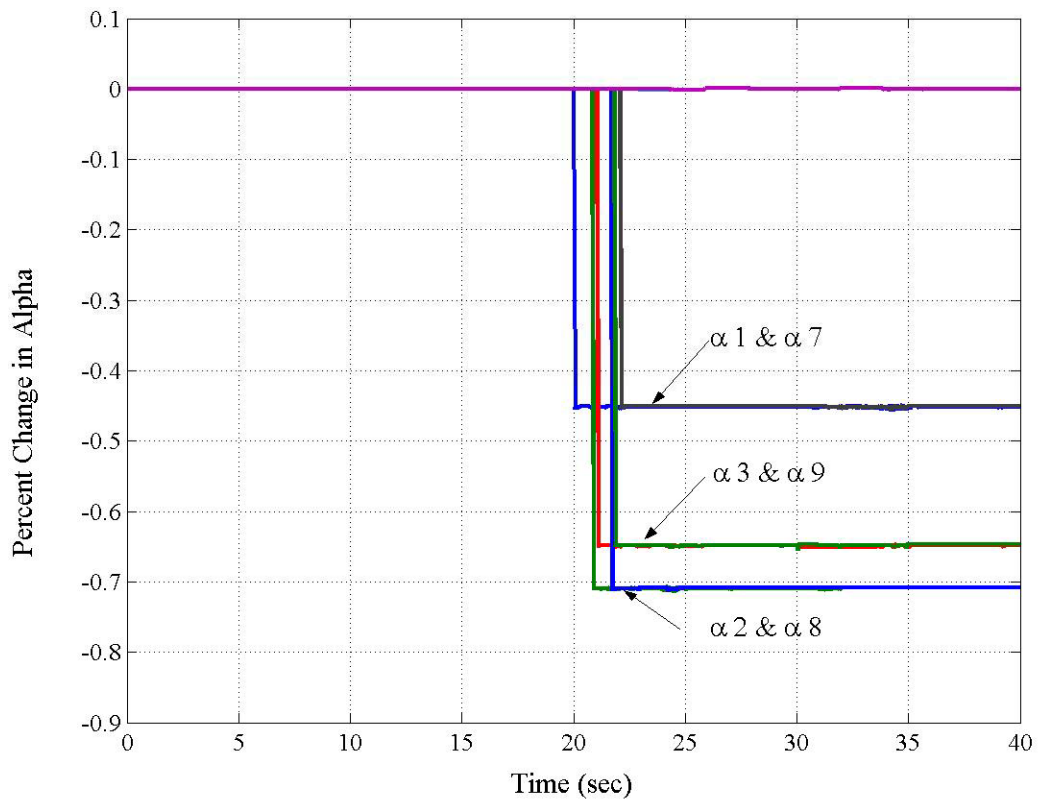
12 DOF in Case 4 and Damage Pattern 1: Own Generated Data with No Error



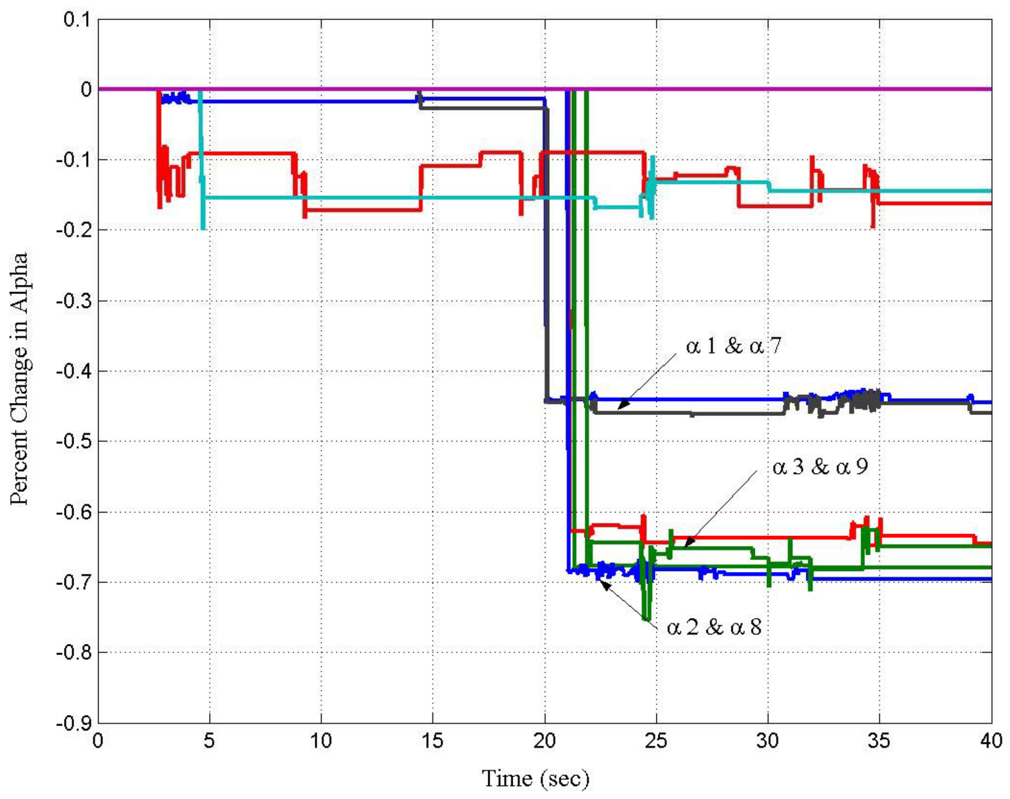
12 DOF in Case 4 and Damage Pattern 1: Own Generated Data with Added Error



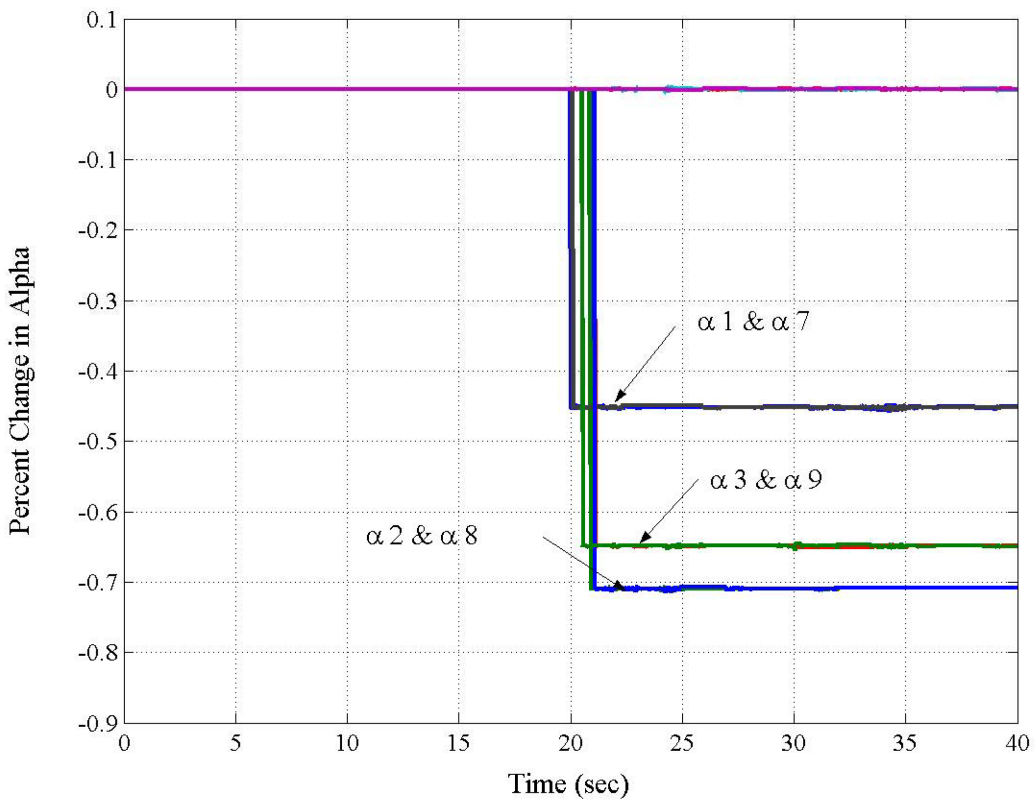
12 DOF in Case 4 and Damage Pattern 1: Data from Benchmark Problem



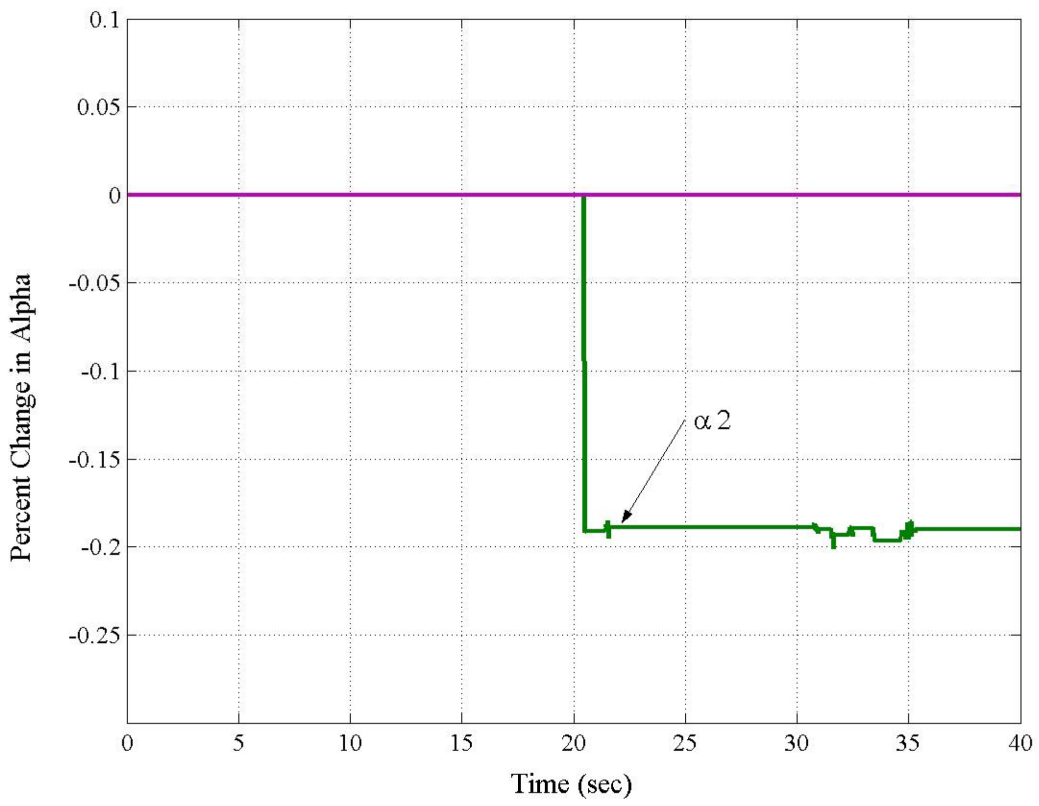
12 DOF in Case 4 and Damage Pattern 2: Own Generated Data with No Error



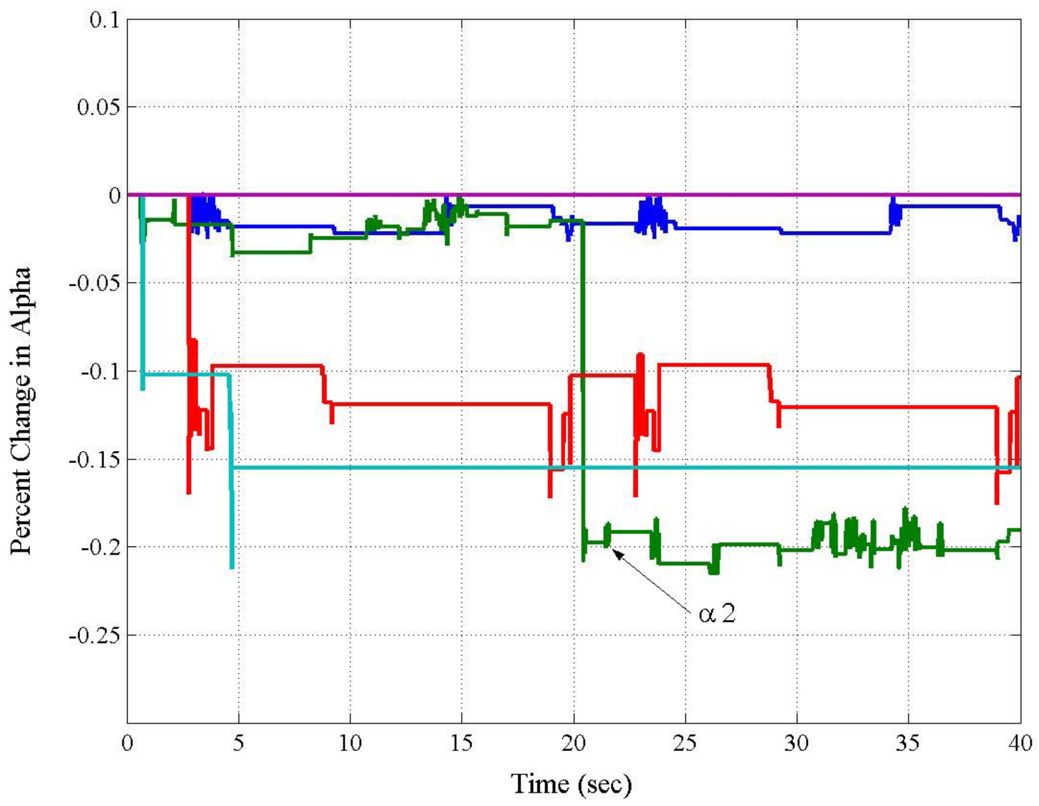
12 DOF in Case 4 and Damage Pattern 2: Own Generated Data with Added Error



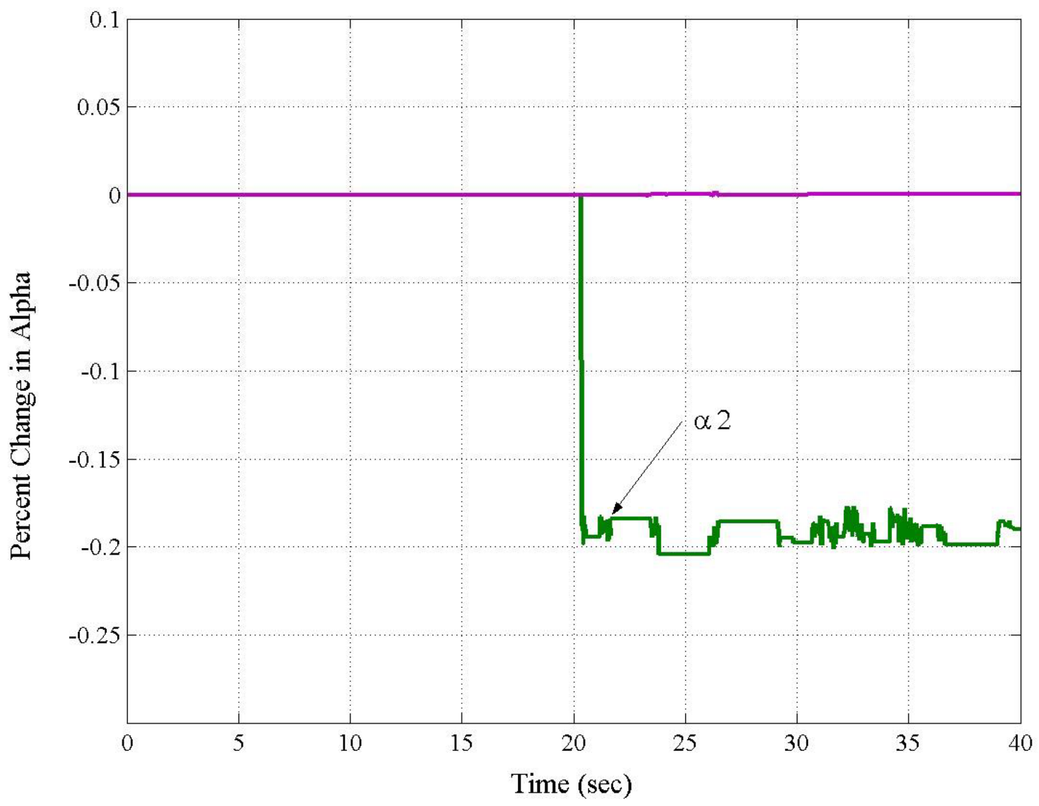
12 DOF in Case 4 and Damage Pattern 2: Data from Benchmark Problem



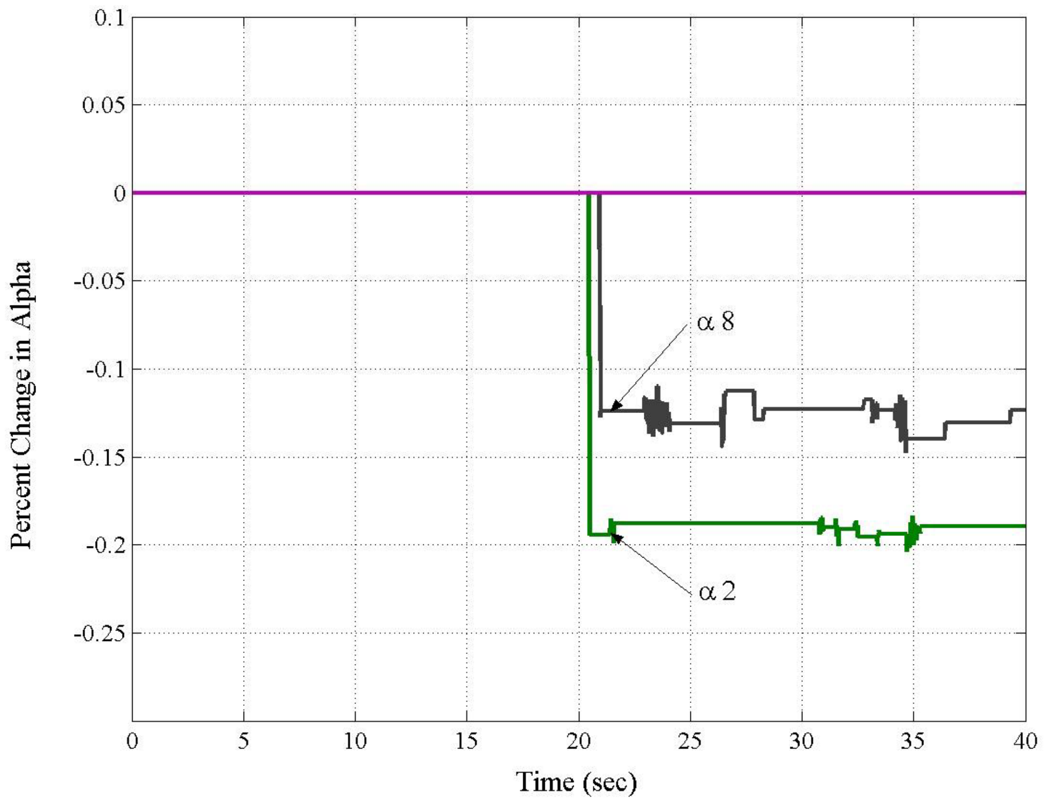
12 DOF in Case 4 and Damage Pattern 3: Own Generated Data with No Error



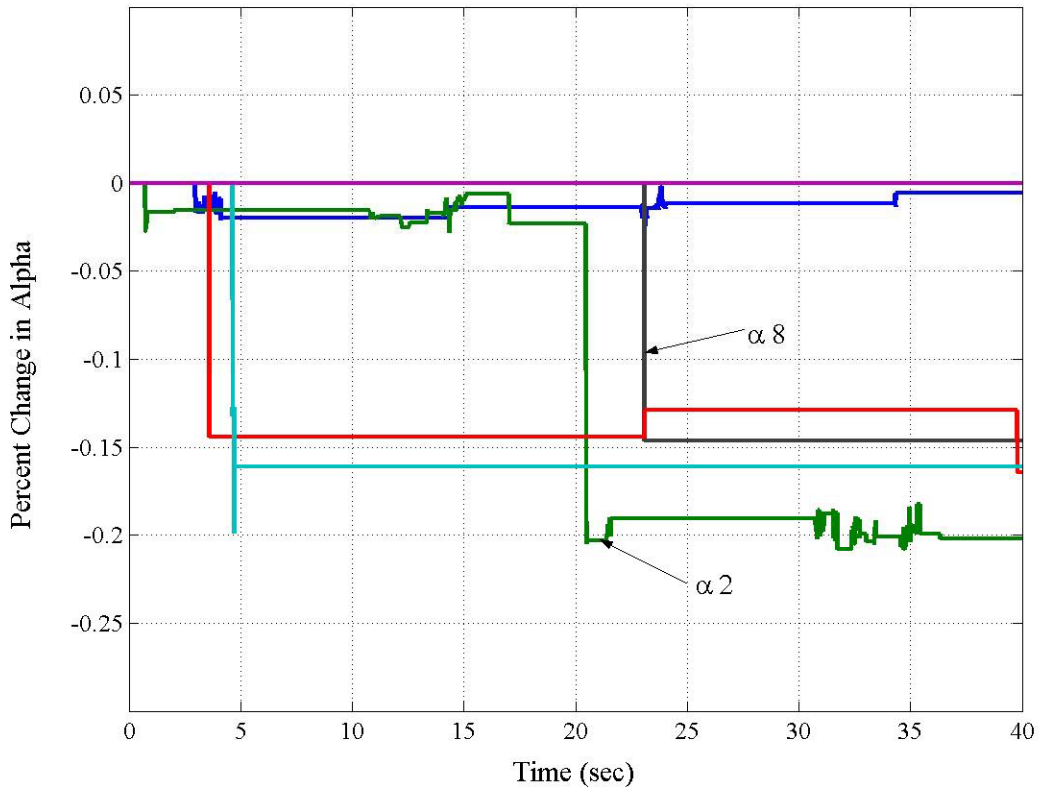
12 DOF in Case 4 and Damage Pattern 3: Own Generated Data with Added Error



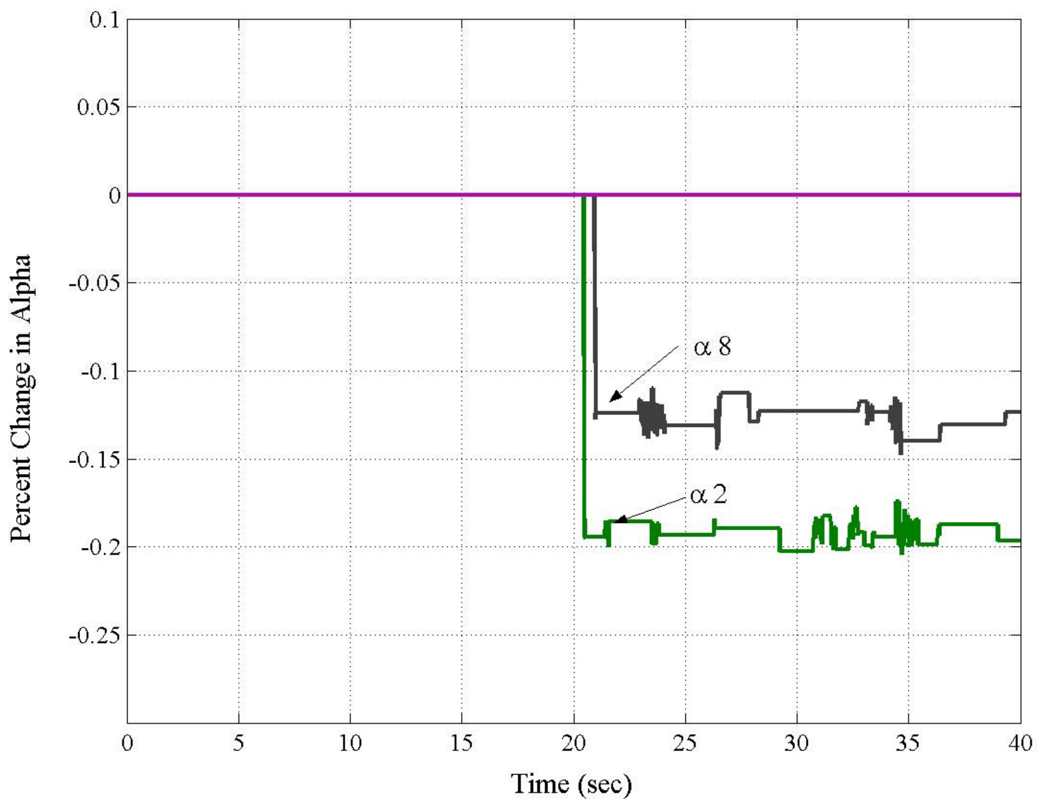
12 DOF in Case 4 and Damage Pattern 3: Data from Benchmark Problem



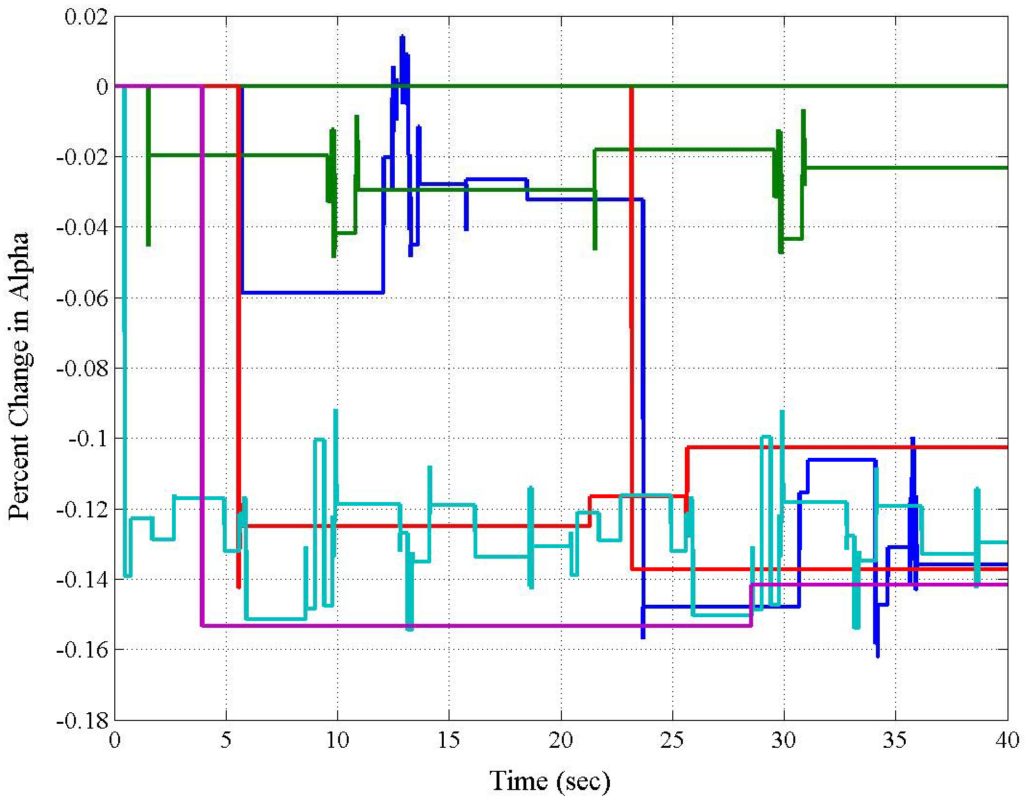
12 DOF in Case 4 and Damage Pattern 4: Own Generated Data with No Error



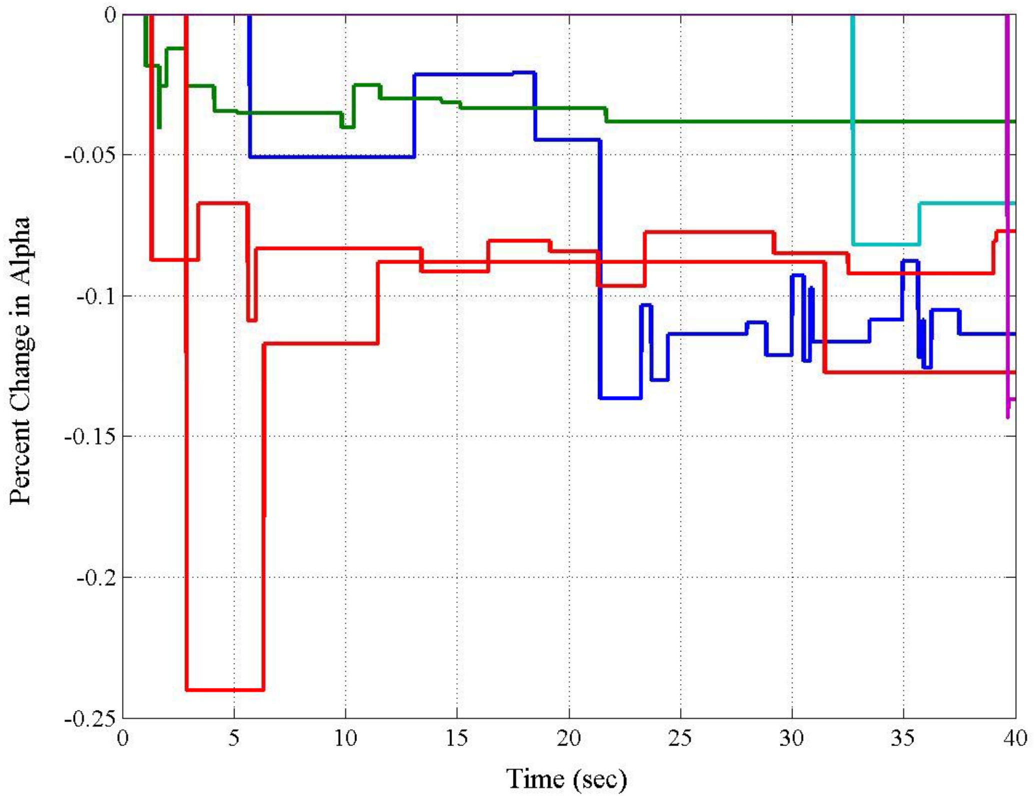
12 DOF in Case 4 and Damage Pattern 4: Own Generated Data with Added Error



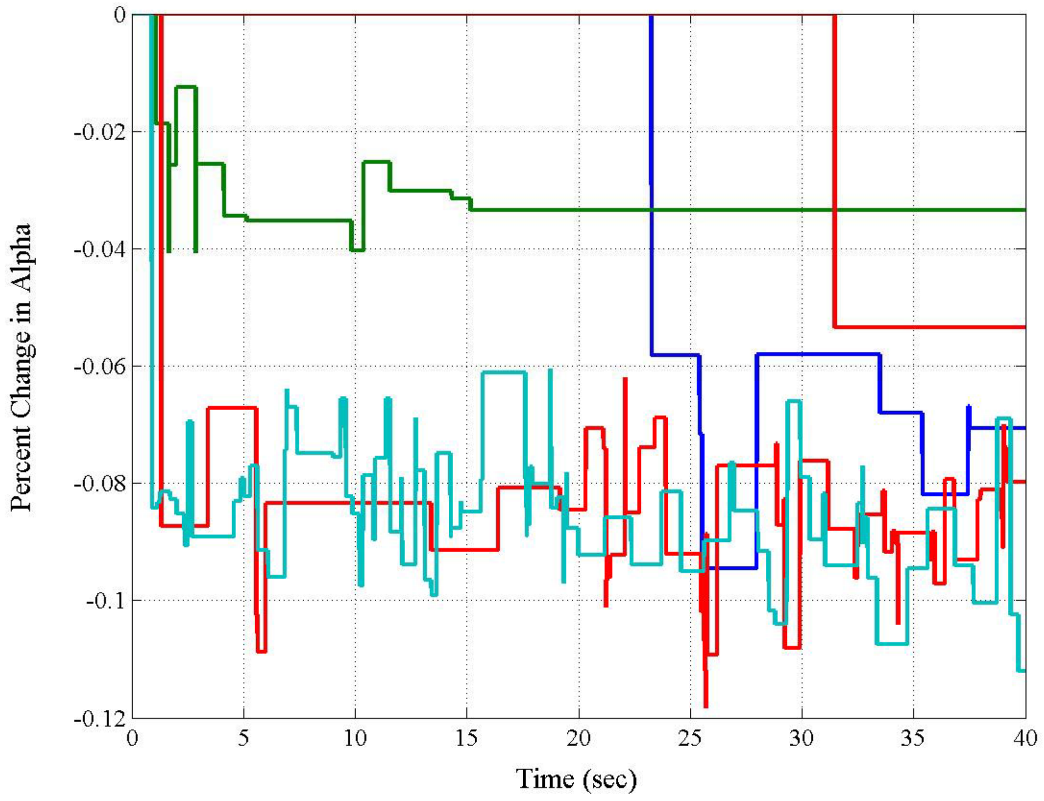
12 DOF in Case 4 and Damage Pattern 4: Data from Benchmark Problem



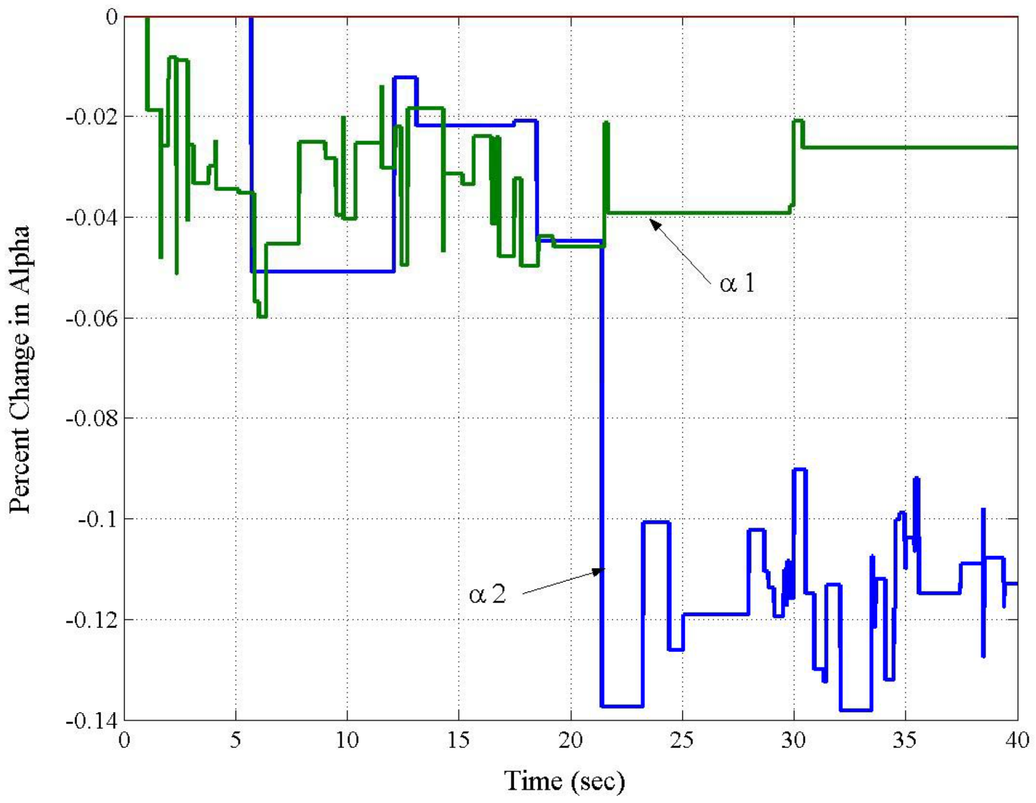
120 Degree of Freedom Phase II: Case 1



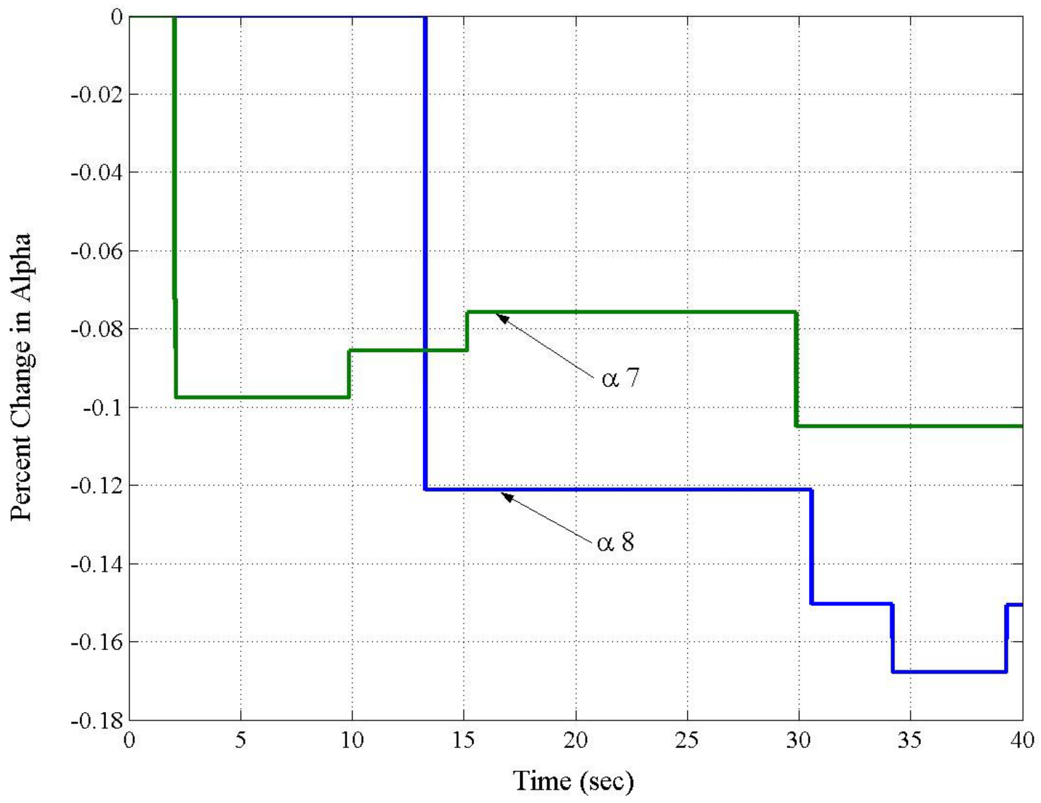
120 Degree of Freedom Phase II: Case 1 showing first 6 DOFs



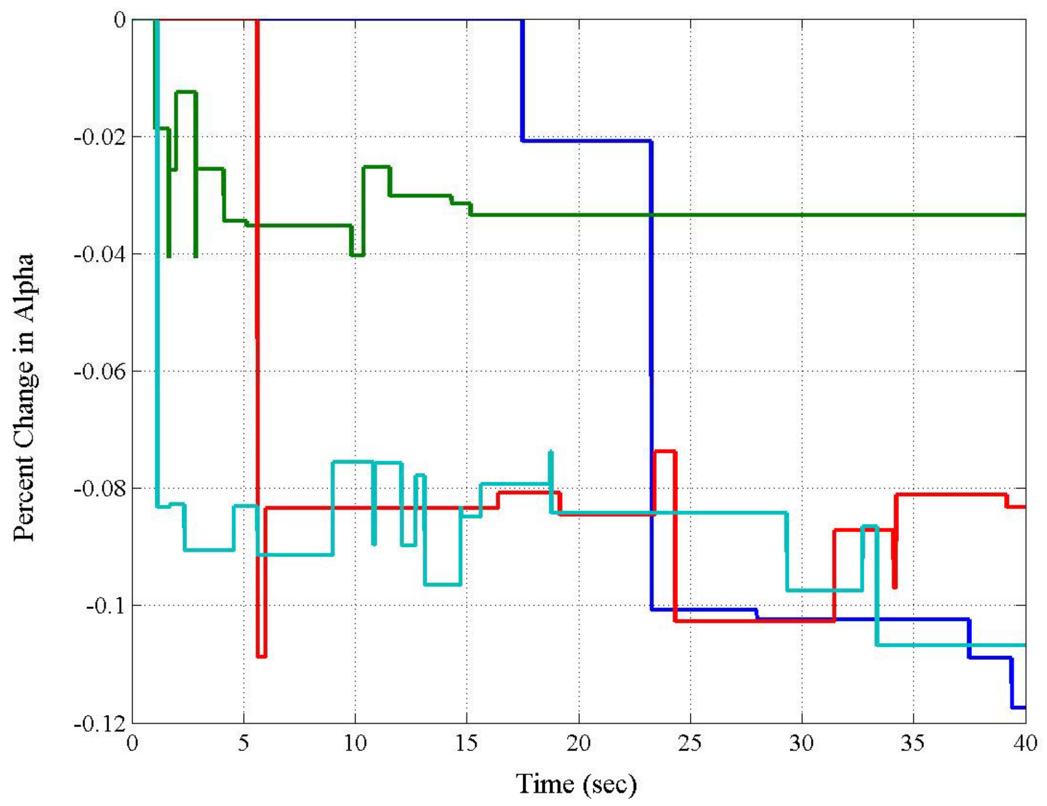
120 Degree of Freedom Phase II: Case 2



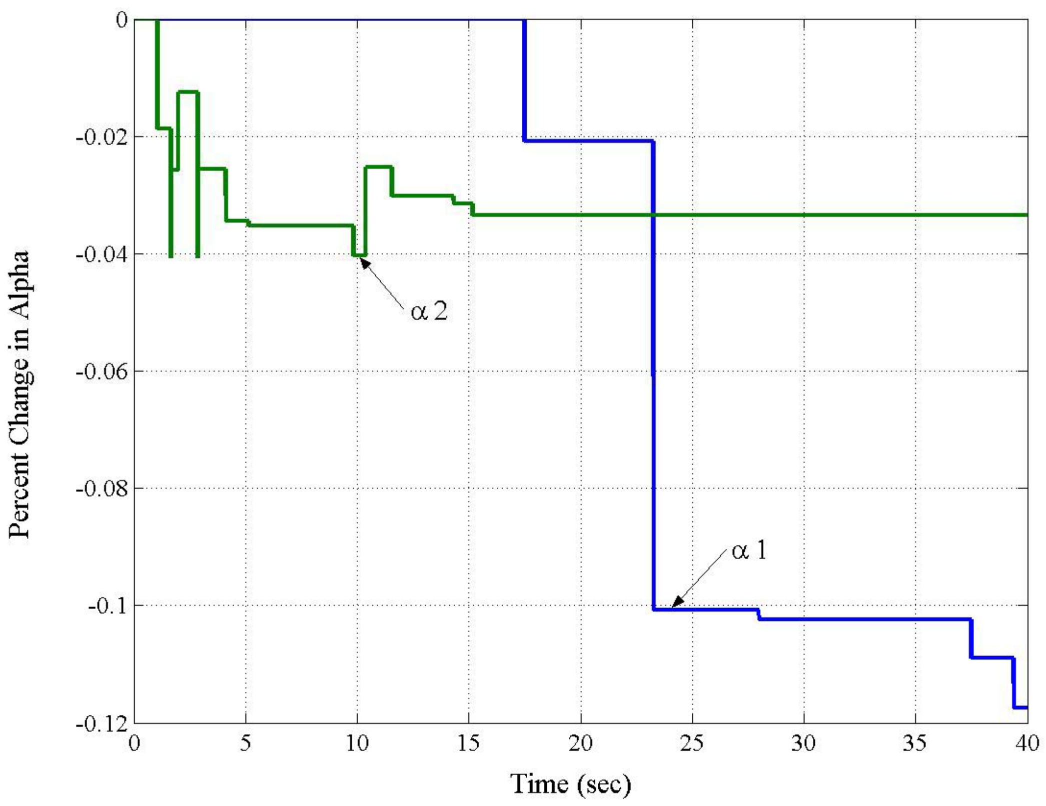
120 Degree of Freedom Phase II: Case 3 showing individual damage



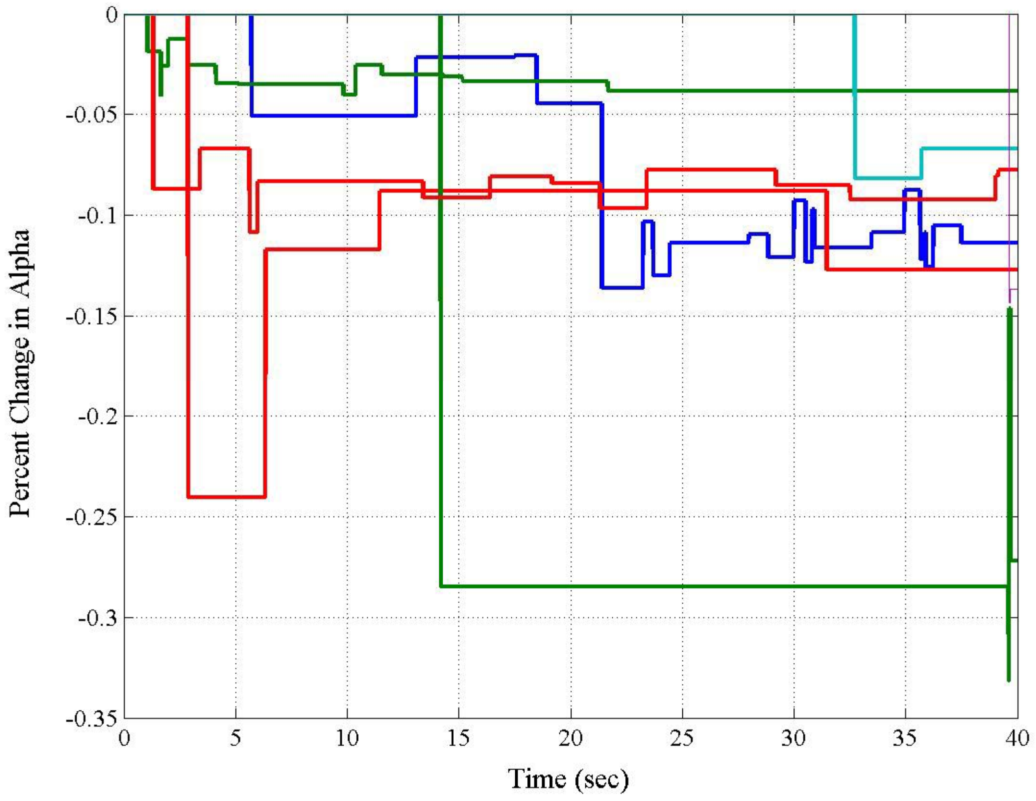
120 Degree of Freedom Phase II: Case 3 showing individual damage



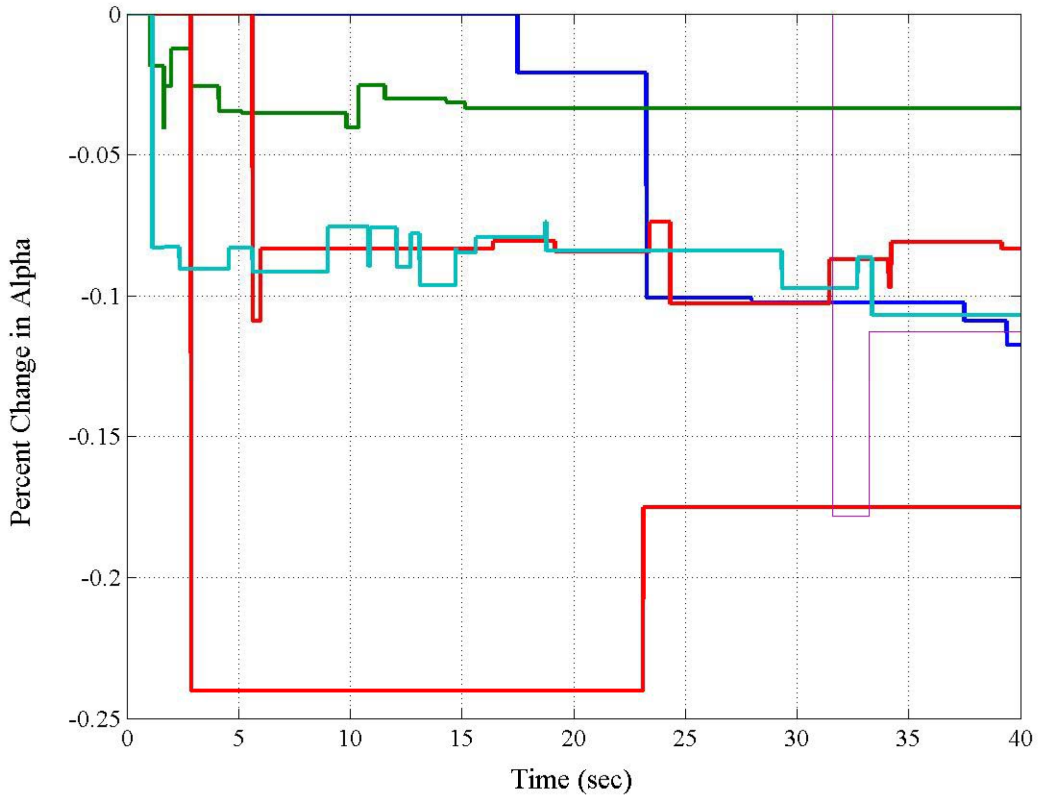
120 Degree of Freedom Phase II: Case 3 Unbraced



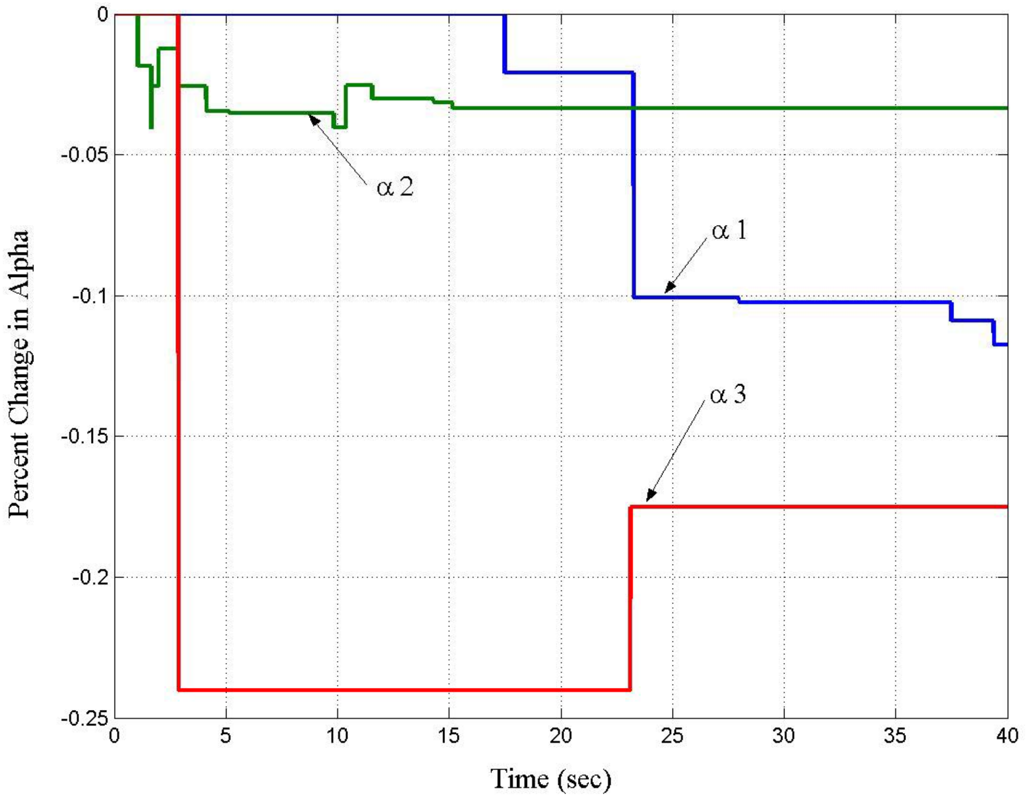
120 Degree of Freedom Phase II: Case3 Unbraced showing first 3 DOFs



120 Degree of Freedom Phase II: Blind Case 1



120 Degree of Freedom Phase II: Blind Case 2



120 Degree of Freedom Phase II: Blind Case 2 showing first 3 DOFs

APPENDIX C

TABLES AND CHARTS

Single DOF Convergence Time and Percent Change in Alpha

DOF (1)	Damage (2)	Case (3)	Error (4)	Method (5)	Time (6)	% (7)
1	1	Constant	No	Own	2.5	0.45
1	1	Constant	Yes	Own	2.9	0.59
1	1	Harmonic	No	Own	1.9	0.45
1	1	Harmonic	Yes	Own	1.8	0.33
1	1	Random	No	Own	0.6	0.47
1	1	Random	Yes	Own	0.8	0.42

Four DOF Convergence Time and Percent Change in Alpha

DOF	Damage	Case	Error	Method	1 DOF	
					Time	%
(1)	(2)	(3)	(4)	(5)	(6)	(7)
4	1	Constant	No	Own	2.6	0.45
4	1	Constant	Yes	Own	1.3	0.46
4	1	Harmonic	No	Own	1.2	0.45
4	1	Harmonic	Yes	Own	2.4	0.43
4	1	Random	No	Own	1.3	0.45
4	1	Random	Yes	Own	1.5	0.49

Three DOF Convergence Time and Percent Change in Alpha

DOF	Damage	Case	Error	Method	1 DOF		2 DOF		3 DOF	
					Time	%	Time	%	Time	%
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)
3	1	Constant	No	Own	2.2	0.45	4.6	0.71	0.6	0.65
3	1	Constant	Yes	Own	3.1	0.47	4.4	0.74	0.05	0.69
3	1	Harmonic	No	Own	3.3	0.43	4.4	0.7	0.05	0.65
3	1	Harmonic	Yes	Own	1.9	0.43	4.4	0.7	0.05	0.64
3	1	Random	No	Own	0.3	0.46	0.2	0.72	0.05	0.65
3	1	Random	Yes	Own	0.4	0.48	0.4	0.72	0.05	0.65

Twelve DOF Convergece Time and Percent Change in Alpha: Phase I, Case 1 and 3

DOF	Damage	Case	Error	Method	1 DOF		2 DOF		3 DOF		7 DOF		8 DOF		9 DOF	
					Time	%	Time	%	Time	%	Time	%	Time	%	Time	%
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)	(17)
12	1	1	No	Own	4.5	0.45	0.7	0.75	4.4	0.7						
12	1	1	Yes	Own	4.9	0.44	0.6	0.71	4.4	0.67						
12	1	1	Yes	Web	0.4	0.45	0.5	0.48	0.7	0.65						
12	2	1	No	Own	4.8	0.45	0.5	0.71	4.8	0.65	4.8	0.45	0.5	0.71	4.8	0.65
12	2	1	Yes	Own	4.1	0.45	0.9	0.72	3.8	0.66	4.1	0.45	1.1	0.73	3.7	0.66
12	2	1	Yes	Web	0.4	0.45	2.1	0.51	0.7	0.65	0.5	0.45	1.4	0.6	0.7	0.65
12	3	1	No	Own			3.1	0.2								
12	3	1	Yes	Own			2.5	0.21								
12	3	1	Yes	Web			1.3	0.13								
12	4	1	No	Own			2.5	0.2					1.7	0.12		
12	4	1	Yes	Own			2.5	0.21					1.7	0.12		
12	4	1	Yes	Web			2.2	0.21					0.5	0.12		
12	1	3	No	Own	0.05	0.45	1.6	0.71	0.5	0.65						
12	1	3	Yes	Own	0.2	0.44	1.6	0.68	0.6	0.66						
12	1	3	Yes	Web	0.05	0.45	1.6	0.71	3	0.65						
12	2	3	No	Own	0.05	0.45	0.9	0.71	1.9	0.65	0.05	0.45	1.1	0.71	3.5	0.64
12	2	3	Yes	Own	2.1	0.44	0.7	0.68	2.1	0.64	0.7	0.45	1.5	0.69	4.5	0.69
12	2	3	Yes	Web	0.8	0.45	0.9	0.71	0.2	0.65	0.7	0.45	0.9	0.71	0.5	0.65
12	3	3	No	Own			1.5	0.19								
12	3	3	Yes	Own			0.5	0.2								
12	3	3	Yes	Web			0.5	0.2								
12	4	3	No	Own			1.5	0.2					2.9	0.12		
12	4	3	Yes	Own			0.5	0.2					2.1	0.14		
12	4	3	Yes	Web			1.5	0.19					2.1	0.12		

Twelve DOF Convergece Time and Percent Change in Alpha: Phase I, Case 4 and Phase II

DOF	Damage	Case	Error	Method	1 DOF		2 DOF		3 DOF		7 DOF		8 DOF		9 DOF	
					Time	%	Time	%	Time	%	Time	%	Time	%	Time	%
12	1	4	No	Own	0.05	0.45	1	0.71	0.4	0.65						
12	1	4	Yes	Own	0.05	0.45	2.4	0.67	1.5	0.67						
12	1	4	Yes	Web	0.05	0.45	1	0.71	0.6	0.65						
12	2	4	No	Own	0.1	0.45	1.8	0.71	1.1	0.65	2.1	0.45	0.9	0.71	1.9	0.65
12	2	4	Yes	Own	0.1	0.44	1.3	0.68	1.1	0.63	2.3	0.46	1.1	0.68	2.3	0.65
12	2	4	Yes	Web	0.05	0.45	0.9	0.71	1.1	0.65	0.1	0.45	1.1	0.71	0.6	0.65
12	3	4	No	Own			0.5	0.19								
12	3	4	Yes	Own			0.6	0.2								
12	3	4	Yes	Web			0.5	0.19								
12	4	4	No	Own			1.5	0.19					0.9	0.12		
12	4	4	Yes	Own			1.5	0.19					3	0.12		
12	4	4	Yes	Web			0.5	0.2					0.9	0.12		
120	1		Yes	Web												
120	2		Yes	Web			4.9	0.07	11.1	0.05						
120	3		Yes	Web			2.1	0.11								
120	3u		Yes	Web			3.5	0.1								
120	Blind 1		Yes	Web			3.5	0.11								
120	Blind 2		Yes	Web			4.1	0.11								

VITA

Name:	Robin Huckaby Preston
Address:	c/o Dr. Luciana Barroso 3136 TAMU College Station, TX 77843-3136
Email Address:	robinpreston@hotmail.com
Education:	B.S. Civil Engineering, Texas Tech University, 2002 M.S. Civil Engineering, Texas A&M University, 2006