

2016

Shape Function Limitations within Equivalent Single-Degree-of-Freedom Analysis for Structural Elements Subject to Blast Loading

Evan Mullen
Lehigh University

Follow this and additional works at: <http://preserve.lehigh.edu/etd>



Part of the [Structural Engineering Commons](#)

Recommended Citation

Mullen, Evan, "Shape Function Limitations within Equivalent Single-Degree-of-Freedom Analysis for Structural Elements Subject to Blast Loading" (2016). *Theses and Dissertations*. 2738.

<http://preserve.lehigh.edu/etd/2738>

This Thesis is brought to you for free and open access by Lehigh Preserve. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Lehigh Preserve. For more information, please contact preserve@lehigh.edu.

Shape Function Limitations within Equivalent Single-Degree-of-Freedom Analysis for
Structural Elements Subject to Blast Loading

by

Evan M. Mullen

A Thesis

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Structural Engineering

Lehigh University
January 2016

Copyright Page

This thesis is accepted and approved in partial fulfillment of the requirements for the Master of Science.

Date

Dr. Spencer Quiel, Thesis Advisor

Dr. Panayiotis (Panos) Diplas, Chairperson of Department

Acknowledgements

I would like to first thank Sara Barker and Harry Nimoityn for their continuous support throughout my studies and research. They offered great encouragement from start to finish during this challenging, yet rewarding process.

I would also like to thank my research advisor, Dr. Spencer Quiel. His combination of practical and theoretical knowledge shaped my research and helped me complete my thesis with significant conclusions.

Table of Contents

Acknowledgements	iv
List of Tables	ix
List of Figures	x
Abstract.....	1
Chapter 1 Introduction	3
1.1 General.....	3
1.2 Background	3
1.3 Research Objectives	4
1.4 Scope of Thesis.....	5
Chapter 2 Background and Modeling.....	6
2.1 General.....	6
2.2 Numerical Solutions	7
2.3 Time Discretization	10
2.4 Loads	10
2.5 Material Properties and Plasticity.....	13
2.6 Boundary Conditions.....	15
2.6.1 Transformation Factors (For SDOF Analysis).....	16

2.7 Space Discretization	21
2.8 Element Type	23
2.9 Geometric Effects.....	24
2.10 Mass Matrix.....	26
2.11 Mode Shapes.....	27
2.12 Damping	29
Chapter 3 Single-Degree-of-Freedom Solver.....	34
3.1 General.....	34
3.2 SBEDS	34
3.2.1 Intro Tab	34
3.2.2 Input Tab	34
3.2.3 Results Tab	35
3.2.4 P-I_Diagram Tab	36
3.2.5 SDOF Output Tab.....	36
3.2.6 Database Tab	37
3.2.7 Beam_Types Tab	37
3.3 MATLAB Replica	37
3.3.1 Main File	38
3.3.2 Load File.....	38

3.3.3 Biggs File	39
3.3.4 Const_vel File	39
3.4 Solver Verification	40
3.4.1 Inconsistencies	42
Chapter 4 Multi-Degree-of-Freedom Solver	50
4.1 General	50
4.2 SAP2000	50
4.2.1 Inputs	50
4.2.2 Outputs	53
4.3 MATLAB Replica	53
4.3.1 Input File	54
4.3.2 Main File	54
4.4 Solver Verification	55
4.4.1 Inconsistencies	59
Chapter 5 Comparisons	69
5.1 General	69
5.2 Results and Analysis	70
5.2.1 First Data Set	70
5.2.2 Second Data Set	80

5.2.3 Third Data Set.....	84
5.2.4 Individual Deflected Shapes	89
Chapter 6 Conclusions and Future Research.....	99
6.1 General.....	99
6.2 Summary	99
6.3 Conclusions	100
6.4 Future Research	102
References	106
Appendix A: MATLAB Code for SBEDS Replica (SDOF Solver)	108
Appendix B: SDOF Solver Verifications (Time-History Plots).....	123
Appendix C: MATLAB Code for More Accurate Model (MDOF Solver).....	177
Appendix D: MDOF Solver Verifications (Time-History Plots)	246
Appendix E: First Data Set Plots	264
Appendix F: Second Data Set Plots	300
Appendix G: Third Data Set Plots.....	310
Vita.....	320

List of Tables

Table 2.5.1: Various Strength Increase Factors for Steel (US Army, 2008).....	13
Table 2.6.1.1a: Transformation Factors for Pinned-Pinned Beam (Biggs, 1964).....	18
Table 2.6.1.1b: Transformation Factors for Fixed-Fixed Beam (Biggs, 1964).....	18
Table 2.6.1.1c: Transformation Factors for Fixed-Pinned Beam (Biggs, 1964).....	19
Table 5.1.1: Constant Input Parameters.....	70
Table 5.1.2: Charges and Standoffs for Data Set 1.....	71
Table 5.2.3.1: Experimental Blast Data (Nassr, 2012).....	85
Table 5.2.3.2: Timoshenko Solution Calibrations to Better Match Experimental Data.....	86

List of Figures

Figure 2.4.1: Friedlander Equation Plot and Ramp Approximation with Arbitrary Values.....	11
Figure 2.4.2: Pressure-Time Plot to Approximate Friedlander Equation (ASCE/SEI 59-11, 2011).....	12
Figure 2.5.1: Stress-Strain Relationship for Two Different Materials.....	14
Figure 2.6.1.1: Transformation from Continuous System to SDOF System (US Army, 2008).....	16
Figure 2.6.1.2: Illustration of Stages of Yielding (US Army, 2008).....	20
Figure 2.6.1.3: General Resistance-Deflection Relationship for SBEDS (US Army, 2008).....	21
Figure 2.7.1: Example of Space Discretization for Two-Member Frame.....	22
Figure 3.2.5.1: Displacement-Time Plot for Arbitrary SBEDS Analysis (SBEDS, 2008)....	37
Figure 3.4.1: Acceleration Comparison of SBEDS with SDOF MATLAB Program.....	41
Figure 3.4.1.1a: Resistance-Deflection Plot for W40X655 Fixed-Pinned with Uniform Load and 0% Damping.....	43
Figure 3.4.1.1b: Resistance-Time Plot for W40X655 Fixed-Pinned with Uniform Load and 0% Damping.....	43
Figure 3.4.1.2a: Resistance-Deflection Plot for W10X12 Fixed-Fixed with Uniform Load and 0% Damping.....	44
Figure 3.4.1.2b: Resistance-Time Plot for W10X12 Fixed-Fixed with Uniform Load and	

0% Damping.....	45
Figure 3.4.1.2c: Resistance-Deflection Plot for W10X12 Fixed-Fixed with Uniform Load and 0% Damping (Magnified).....	45
Figure 3.4.1.3a: Deflection-Time Plot for W10X12 Fixed-Pinned with Point Load and 0% Damping.....	47
Figure 3.4.1.3b: Acceleration-Time Plot for W10X12 Fixed-Pinned with Point Load and 0% Damping.....	47
Figure 3.4.1.3c: Acceleration-Time Plot for W10X12 Fixed-Pinned with Point Load and 0% Damping (Magnified).....	48
Figure 3.4.1.4: Deflection-Time Plot for W10X12 Fixed-Fixed with Uniform Load and 5% Damping.....	49
Figure 4.2.1.1: Example of Material Definition Window for SAP2000.....	51
Figure 4.2.1.2: Graphical Representation of a Pinned-Fixed Beam with 0.35 k/in Load and 4 Elements.....	52
Figure 4.2.2.1: Example Plot of Deflection-Time in SAP2000.....	53
Figure 4.4.1a: Midpoint Deflection Comparison of SAP2000 with MDOF MATLAB Program.....	57
Figure 4.4.1b: Midpoint Resistance Comparison of SAP2000 with MDOF MATLAB Program.....	58
Figure 4.4.1.1 Equations for Reduced Plastic Moment Capacity in SBEDS (US Army, 2008).....	62
Figure 4.4.1.2a: Resistance-Time Plot for W10X12 Fixed-Pinned with Point Load, 2%	

Damping, and 0.2 Axial Load.....	64
Figure 4.4.1.2b: Deflection-Time Plot for W10X12 Fixed-Pinned with Point Load, 2%	
Damping, and 0.2 Axial Load.....	65
Figure 4.4.1.3a: Deflection-Time Comparison for Different Stiffness and Mass	
Models.....	67
Figure 4.4.1.3b: Resistance-Time Comparison for Different Stiffness and Mass	
Models.....	68
Figure 5.2.1.1: Deviation in K-L vs. Z for W14X48.....	72
Figure 5.2.1.2: Deviation in K-L vs. Z for W14X257.....	73
Figure 5.2.1.3: Deviation in Bernoulli-SDOF Maximum Deflection vs. Z for	
W14X257.....	75
Figure 5.2.1.4: Deviation in Timoshenko-SDOF Maximum Deflection vs. Z for	
W14X257.....	77
Figure 5.2.1.5: Deviation in Timoshenko-Bernoulli Maximum Deflection vs. Z for	
W14X257.....	79
Figure 5.2.2.1: Deviation in Bernoulli-SDOF Maximum Deflection vs. Z for Pinned-	
Pinned W14X109 (Fine Z Increment).....	81
Figure 5.2.2.2: Deviation in Timoshenko-SDOF Maximum Deflection vs. Z for Pinned-	
Pinned W14X109 (Fine Z Increment).....	83
Figure 5.2.3.1: Deviation in Bernoulli-SDOF Maximum Deflection vs. Z for Pinned-	
Pinned W14X109 (Calibrated Damping).....	87
Figure 5.2.3.2: Deviation in Timoshenko-SDOF Maximum Deflection vs. Z for Pinned-	

Pinned W14X109 (Calibrated Damping).....	89
Figure 5.2.4.1: Deflected Shape at First Yield for Pinned-Pinned W14X109 with Z=12.4 ft/lb ^{1/3}	91
Figure 5.2.4.2: Deflected Shape at First Yield for Pinned-Pinned W14X109 with Z=9.1 ft/lb ^{1/3}	91
Figure 5.2.4.3: Deflected Shape at First Yield for Pinned-Pinned W14X109 with Z=7.7 ft/lb ^{1/3}	92
Figure 5.2.4.4: Deflected Shape at First Yield for Fixed-Fixed W14X109 with Z=11.0 ft/lb ^{1/3}	94
Figure 5.2.4.5: Deflected Shape at First Yield for Fixed-Fixed W14X109 with Z=9.5 ft/lb ^{1/3}	95
Figure 5.2.4.6: Deflected Shape at First Yield for Fixed-Fixed W14X109 with Z=5.8 ft/lb ^{1/3}	95
Figure 5.2.4.7: Deflected Shape at First Yield for Fixed-Fixed W18X65 with 45 psi and 45 psi-msec.....	97
Figure 5.2.4.8: Deflected Shape at First Yield for Fixed-Fixed W18X65 with 50 psi and 50 psi-msec.....	98

Abstract

This thesis presents an investigation into the potential inaccuracies when analyzing structural elements under blast loads using equivalent single-degree-of-freedom (SDOF) systems. An equivalent SDOF system converts a continuous element into a single mass-spring-damper system, where the displacement of the mass equates to the largest deflection of the continuous element. The potential inaccuracies arise from the fact that this equivalent system uses the static deflected shape of the element. The Single-Degree-of-Freedom Blast Effects Design Spreadsheet (SBEDS) is an example of an industry accepted software package that is used for this type of simplification.

To identify inaccuracies in results from SDOF methods, the results were compared to a more accurate model. This model is a multi-degree-of-freedom (MDOF) system, which is able to better represent a structural element, due to discretizing the element into smaller segments. These smaller segments are able to respond with a more realistic deflected shape when subjected to blast loading. This is more accurate than lumping all of the information about the element to a single degree of freedom.

Using MATLAB, blast analysis of single structural elements were performed to replicate the corresponding results from two software packages: SBEDS (for SDOF analysis) and SAP2000 (for MDOF analysis). Once the MATLAB solutions were verified

and calibrated, analyses were performed using both MATLAB scripts for combinations of input parameters, such as boundary conditions, magnitude of load, and section size. After results were obtained from these analyses, they were compared, which allowed for identification of static-shape limitations of SDOF analysis.

Chapter 1 Introduction

1.1 General

This thesis presents an investigation into inaccuracies within equivalent single-degree-of-freedom (SDOF) analysis under blast loading. This chapter introduces the topic of blast design, and outlines the objectives and scope of this thesis.

1.2 Background

Dynamic loading of a structure requires dynamic analysis to obtain accurate values of deformation and internal stress. Examples of dynamic loads are live loads, earthquakes, wind, and blasts. The duration of the load distinguishes blast loads from the other dynamic loads. While these other loads can last seconds or minutes, blast loads have much shorter durations on the order of milliseconds. Whether this blast load is a result of an accidental explosion or an intended terrorist act, the load is always a fast-acting pulse.

SBEDS is used in the structural engineering industry for the “design of structural components subjected to dynamic loads using single degree of freedom (SDOF) methodology” (US Army, 2008). This program is often used for the design of these components under blast loading. By defining inputs such as the time-history of the blast load, material properties, and geometric features of the structural element in question, SBEDS performs this SDOF analysis and outputs several results that the

Structural Engineer may use to aid in the design of the component. These outputs include maximum deflection, maximum support rotation, ductility, and maximum shear force, as well as time-histories of various output variables. Due to the fact that SBEDS uses equivalent SDOF methods, the analysis requires little computing time, which is convenient for the industry.

1.3 Research Objectives

Yokoyama (2014) has shown that for large blasts at close distance, discrepancies arise between SBEDS solutions and solutions from more accurate models. Scaled distance (Z) is a common measure used to define the strength of a blast. The conventional Z cutoff for which SDOF analysis is considered to be no longer applicable is below $1.2 \text{ m/kg}^{1/3}$ ($3 \text{ ft/lb}^{1/3}$). However, Yokoyama has shown cases where applying a scaled distance above the cutoff produces significantly different results between SBEDS and more accurate models. This can be a major issue for Structural Engineers who utilize SBEDS for blast design without knowing the boundaries for which the results from SBEDS become unacceptably inaccurate.

The research presented in this thesis investigates and establishes more specific cutoffs for when SDOF analysis becomes too inaccurate. The objectives of this research are listed below:

- (1) Write MATLAB code that replicates the SBEDS solution (Single-Degree-of-Freedom (SDOF) solver)
- (2) Write MATLAB code that uses a more accurate model (Multi-Degree-of-Freedom (MDOF) solver)
- (3) Run both MATLAB programs for various input parameters, and compare solutions obtained from both solution methods.
- (4) Identify the limitations of using single-degree-of-freedom analysis when analyzing structural components under blast loading based for various input parameters.

Objective (1) will be validated against like analyses in SBEDS. Objective (2) will be validated against like analyses in SAP2000, until the limitations of SAP2000 are exceeded. Objective (3) will be completed by obtaining percent variation on the outputs for each case of inputs. For Objective (4), a maximum acceptable percent variation must be established to determine these region boundaries.

1.4 Scope of Thesis

Five additional chapters follow this chapter. Chapters 2, 3, and 4 discuss background information, modeling techniques, and the creation of both MATLAB solvers. Chapter 5 presents the comparisons of the data frames for each solver. Finally, Chapter 6 provides a summary of the research, identifies boundaries of SDOF analysis limits, and concludes the paper with future research to be explored on this topic.

Chapter 2 Background and Modeling

2.1 General

Modeling refers to the creation of a simulation that replicates an event which occurs in reality. Modeling is a very complex topic in structural engineering, as well as in other disciplines. The complexity arises from the vast amount of choices, approximations, or assumptions that the modeler may make when creating this model. Some options are better suited for certain events than others. Also, certain combinations of these choices may interact to cause very different outcomes. The user must judge each modeling choice and decide which options will best represent a realistic result.

Another problem the modeler faces is creating a model that is not only reasonably accurate, but efficient as well. Certain modeling options will be chosen that are simpler than others, in order to decrease computational runtimes. Having the most sophisticated model, which is most accurate, may be unrealistic for some applications due to the limited processing power of the modern computer.

For this paper, the events of interest are the behaviors of structural elements (beams and beam-columns) subjected to blast loading. Models of this type implement various techniques from structural dynamics, which is a combination of structural analysis and dynamics.

This chapter discusses the modeling techniques for structural dynamics, and how they specifically relate to structural elements subjected to blast loads. Sections 2.2 through 2.6 and section 2.9 pertain to all models. Section 2.6.1 pertains to SDOF models only. Sections 2.7, 2.8, and 2.10 through 2.12 pertain to MDOF models only.

2.2 Numerical Solutions

The analysis of any structural dynamics problem starts with the equation of motion, which is shown below in Equation 2.2.1:

$$m\ddot{u} + c\dot{u} + ku = f(t) \quad (2.2.1)$$

where:

m =mass

c =damping coefficient

k =stiffness coefficient

$f(t)$ =applied force

\ddot{u} =acceleration

\dot{u} =velocity

u =displacement

Displacement is the unknown variable, and velocity and acceleration are time derivatives of displacement. A closed form solution is available for this second-order

differential equation, depending on the form of the applied force. To be able to solve this equation for any form of applied force, numerical methods are often used.

Numerical methods take initial conditions, and extrapolate values incrementally, to obtain approximate solutions. Depending on the increments, the solutions may have little error compared to their closed form solutions.

Several numerical methods are available to solve a second-order differential equation.

Some examples are the *constant-velocity procedure* (Biggs, 1964), the *central difference method* (Chopra, 1995), and *Newmark's method* (Chopra, 1995). Each method approaches the problem slightly differently, but they all produce consistent results assuming an appropriate time step was chosen. An important parameter that relates to the accuracy and stability of each solution method is the natural period, T , shown below in Equation 2.2.2:

$$T = 2\pi \sqrt{\frac{m}{k}} \quad (2.2.2)$$

For the *constant-velocity procedure*, Biggs states that a time increment should not exceed one-tenth of the natural period for accurate results. He also states that the time increment should be small enough to accurately capture the load. This is especially important for blast loads, where the total duration of the load is small. For the *central difference method*, Chopra states that the solution will only be stable

(converge) if the ratio of time step to natural period of vibration is below a certain value. This value is displayed below in Equation 2.2.3a. He also states the same accuracy limit that Biggs states, which is a smaller increment than the stability limit is. *Newmark's method* is more complex, and involves two additional constants (β , γ), which control the acceleration within each time step. The stability limit for this solution method is also shown below in Equation 2.2.3b:

$$\frac{\Delta t}{T} < \frac{1}{\pi} \quad (2.2.3a)$$

$$\frac{\Delta t}{T} \leq \frac{1}{\pi\sqrt{2}} \cdot \frac{1}{\sqrt{\gamma-2\beta}} \quad (2.2.3b)$$

SBEDS uses the *constant-velocity procedure* to solve the equation of motion. The MATLAB SDOF solver will therefore implement this procedure as well. The MATLAB MDOF solver will implement *Newmark's method* for its versatility and unconditional stability for certain values of β and γ .

These methods are either classified as explicit or implicit methods. Implicit methods iterate within each time step. They calculate the internal and external forces on the structure and drive the unbalanced load (the difference of these forces) below a certain tolerance by adjusting the displacements, velocities, and accelerations, before proceeding to the next time step (Newton-Raphson iteration). Explicit methods neglect this unbalanced force and proceed through the time stepping without

analyzing unbalanced forces. The *constant-velocity procedure* is an explicit solver, while *Newmark's method* with Newton-Raphson iteration is an implicit solver.

2.3 Time Discretization

Discretization refers to the subdivision of a continuous domain into distinct quantities. Discretization is how numerical solutions can be achieved. For the equation of motion, time is the continuous domain that is discretized as time steps. As explained above, it is important to pick a time step that satisfies stability and accuracy limits for the numerical method chosen, as well as captures the load accurately.

2.4 Loads

Blast loads are very complex loads to model. How to characterize blast loads as applied to structures is not in the scope of this thesis. However, the basic concepts of determining a load function are described below. Blasts create fast-acting shock waves. These shock waves create positive pressure, followed by negative pressure on any mass in its direction of travel. Friedlander (1946) fit a curve to the pressure of blast wave over time. The Friedlander Equation is shown below:

$$P = P_s \cdot e^{\frac{-t}{t^*}} \cdot \left(1 - \frac{t}{t^*}\right) \quad (2.4.1)$$

where:

P =pressure at time t

P_s =peak pressure

t =time

t^* =time of positive duration of pressure

Figure 2.4.1 below shows a graph of this equation with arbitrarily selected values as well as a ramp approximation to the curve, which is explained after:

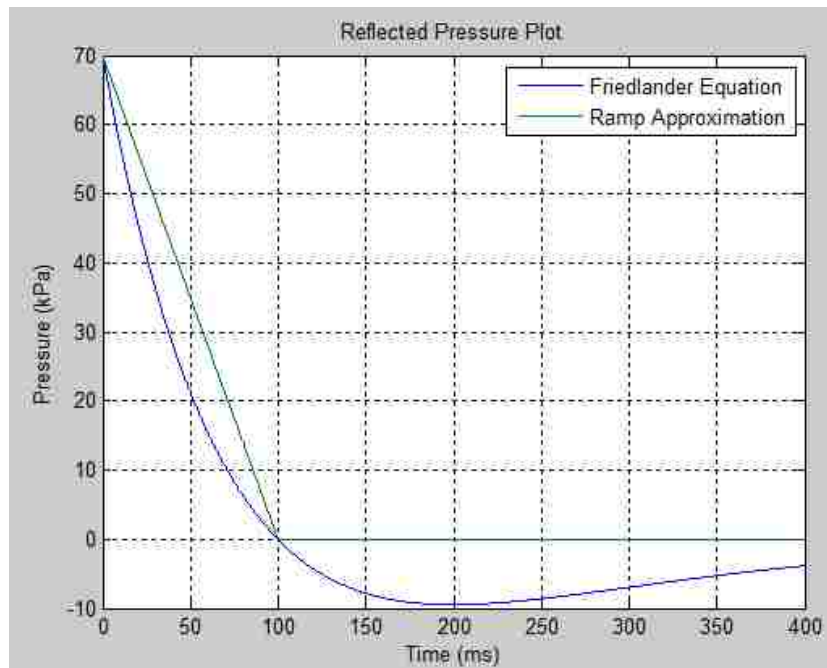


Figure 2.4.1: Friedlander Equation Plot and Ramp Approximation with Arbitrary Values

ASCE/SEI 59-11 (2011) describes a procedure for converting a weight of explosive material into a pressure-time graph that approximates the Friedlander relationship.

This graph becomes the applied load to the element by converting the pressure into a load. Figure 2.4.2 below shows the diagram depicted in the actual manual:

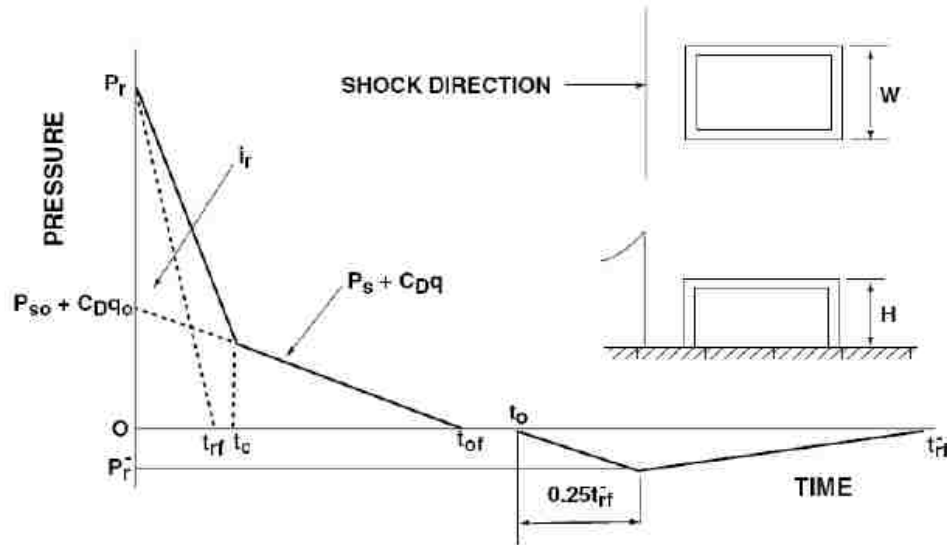


Figure 2.4.2: Pressure-Time Plot to Approximate Friedlander Equation (ASCE/SEI 59-11, 2011)

Often in practice, the negative portion of the graph is neglected, as a conservative simplification of the analysis since the negative phase will partially counteract the effects of the initial positive pressure in most cases. The scope of this thesis is not to investigate the load itself. Therefore, the loads to be applied to the structural elements will follow what is most commonly used in practice. This approach consists of obtaining a value for positive peak pressure and positive impulse from a charge (lb of TNT) and standoff (ft from structure). With the peak pressure and impulse, a simple triangular ramp-down time-history is created, with positive pressure only. This

pressure is applied to the structural element using its tributary area. Figure 2.4.1 from before shows this simplified approach.

2.5 Material Properties and Plasticity

Materials can exhibit different properties under dynamic loading, compared to static loading. Therefore, increase factors can be applied to the yield strengths of the materials in dynamic analyses. The *Methodology Manual for the Single-Degree-of-Freedom Blast Effects Design Spreadsheets (SBEDS)* summarizes these different increase factors, as shown in Table 2.5.1 below:

Material	Minimum Static Yield Strength	Average Strength Increase Factor (SIF) ^a (a)	Dynamic Strength Increase Factor (DIF) (c)
Cold Formed Panels, Beams	200 – 400 MPa (30,000 - 60,000 psi)	1.21	1.1
Steel	200 – 240 MPa (30,000 - 36,000 psi)	1.1	1.29
Steel	290 – 400 MPa (42,000 - 60,000 psi)	1.05	1.19
Steel	515 – 690 MPa (75,000 - 100,000 psi)	1.0	1.09

*Also referred to as an average strength factor (ASF)

Table 2.5.1: Various Strength Increase Factors for Steel (US Army, 2008)

Modeling plastic behavior can be complicated and is material dependent. For example, hot-rolled steel exhibits a linear stress-strain relationship until its yield point, at which it gains no increase in stress with additional strain. At a certain strain value, it then exhibits strain hardening, and has an increase in stress with additional strain.

This strain-hardening portion of the curve is strain-rate dependent, which causes much complexity to modeling this behavior. Reinforced concrete has a less distinct linear stress-strain relationship, and has no distinct point of yielding. The stress-strain curve continues to decrease in slope until a maximum stress is observed. Because of the complexity and variation in stress-strain behavior, assumptions are usually made about the stress-strain curve that do not introduce significant error into the solution, and at the same time, make computations significantly simpler. A crude representation of the stress-strain behaviors of the two materials explained is shown below in Figure 2.5.1:

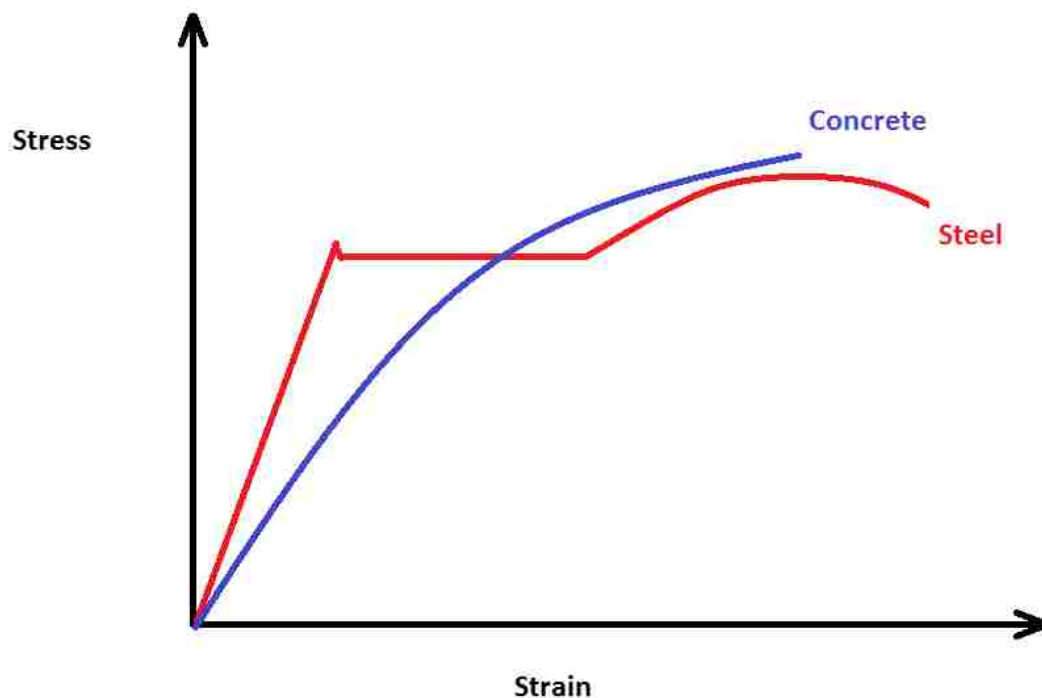


Figure 2.5.1: Stress-Strain Relationship for Two Different Materials

Blast loads often have enough energy in such a short duration to cause yielding in the structural elements they are applied to. Due to these conditions, plasticity must be considered in blast analysis.

For this thesis, only hot-rolled steel is considered. Hardening of the material is neglected to simplify the analysis without risking much inaccuracy. Therefore, the stress-strain relationship used in both MATLAB solvers follows an elastic-perfectly plastic relationship. For the analysis, once the internal moment reaches the plastic moment, that location is considered fully plastic until rebounding (change in sign of velocity). After rebounding, the section becomes elastic again, but with residual displacement. Yielding within the cross-section is neglected. Additionally, all types and modes of buckling are not considered. All elements considered are assumed to be non-slender and not susceptible to lateral torsional buckling. Only compact sections are considered to avoid local buckling considerations as well. Buckling limit states are outside the scope of this thesis.

2.6 Boundary Conditions

In reality, boundary conditions are never fully rigid nor fully rotatable. Connections will have partial restraint of rotation. However, for simplicity, connections are often modeled as either fully moment-connected, or fully pinned. For this thesis, the boundary conditions considered are fully fixed, fully pinned, and free (no constraint).

2.6.1 Transformation Factors (For SDOF Analysis)

SDOF analysis is computationally convenient because the quantities in the equation of motion are scalars. For MDOF analysis, these quantities become matrices, which requires matrix computations. For this reason, SBEDS makes use of certain transformation factors that convert beams (MDOF systems with potentially infinite DOFs) to SDOF systems. Figure 2.6.1.1 below shows this transformation pictorially:

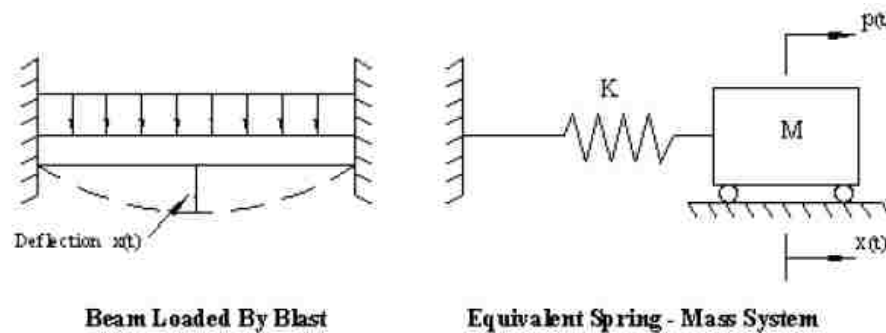


Figure 2.6.1.1: Transformation from Continuous System to SDOF System (US Army, 2008)

As explained in *Methodology Manual for the Single-Degree-of-Freedom Blast Effects Design Spreadsheets (SBEDS)*, “[a]n equivalent SDOF system is defined as a system that has the same energy in terms of work energy, strain energy, and kinetic energy, as the blast-loaded component responding in a given, assumed mode shape.” The mathematical derivations of these definitions are listed in the following equations:

$$K_M = \frac{M_e}{M_t} = \frac{\int_0^L m(x)[\phi(x)]^2 dx}{\int_0^L m(x) dx} \quad (2.6.1.1)$$

where:

K_M =mass factor

M_e =equivalent SDOF mass

M_t =total mass of MDOF element

$m(x)$ =distributed mass along MDOF element

$\emptyset(x)$ =static deflected shape (unit normalized)

L =length of element

$$K_L = \frac{F_e(t)}{F_t(t)} = \frac{\int_0^L p(x,t)\emptyset(x)dx}{\int_0^L p(x,t)dx} \quad (2.6.1.2)$$

where:

K_L =load factor

$F_e(t)$ =equivalent SDOF load

$F_t(t)$ =total load on MDOF element

$p(x, t)$ =distributed load along MDOF element

Using the above equations, these factors are calculated. The mass of the system gets multiplied by the mass factor, and the load on the system and the resistance of the system get multiplied by the load factor. These factors vary for each static deflected shape. Therefore, they change based on the boundary condition, the shape of loading, and the state of plasticity. By dividing the mass factor by the load factor, a single

“load-mass” factor can be used as well. Biggs originally derived these factors for various conditions, which are shown below in Tables 2.6.1.1a, 2.6.1.1b, and 2.6.1.1c:

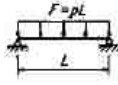
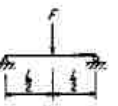
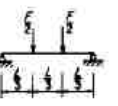
Loading diagram	Strain range	Load factor K_L	Mass factor K_M		Load-mass factor K_{LM}		Maximum resistance R_m	Spring constant k	Dynamic reaction V
			Concentrated mass*	Uniform mass	Concentrated mass*	Uniform mass			
	Elastic	0.64	0.50	0.78	$\frac{89R_p}{L}$	$\frac{384EI}{5L^3}$	$0.39R + 0.11F$
	Plastic	0.50	0.33	0.66	$\frac{89R_p}{L}$	0	$0.38R_m + 0.12F$
	Elastic	1.0	1.0	0.49	1.0	0.49	$\frac{49R_p}{L}$	$\frac{48EI}{L^3}$	$0.78R - 0.28F$
	Plastic	1.0	1.0	0.33	1.0	0.33	$\frac{49R_p}{L}$	0	$0.75R_m - 0.25F$
	Elastic	0.87	0.76	0.52	0.87	0.60	$\frac{69R_p}{L}$	$\frac{58.4EI}{L^3}$	$0.525R - 0.025F$
	Plastic	1.0	1.0	0.56	1.0	0.56	$\frac{69R_p}{L}$	0	$0.52R_m - 0.02F$

Table 2.6.1.1a: Transformation Factors for Pinned-Pinned Beam (Biggs, 1964)

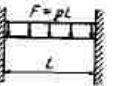
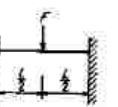
Loading diagram	Strain range	Load factor K_L	Mass factor K_M		Load-mass factor K_{LM}		Maximum resistance R_m	Spring constant k	Effective spring constant k_{eff}	Dynamic reaction V
			Concentrated mass*	Uniform mass	Concentrated mass*	Uniform mass				
	Elastic	0.53	...	0.41	...	0.77	$\frac{129R_p}{L}$	$\frac{384EI}{L^3}$	$0.36R + 0.14F$
	Elastic-plastic	0.64	...	0.50	...	0.78	$\frac{8}{L}(9R_p + 9R_{pm})$	$\frac{384EI}{5L^3}$	$\frac{307EI}{L^3}$	$0.39R + 0.11F$
	Plastic	0.50	...	0.33	...	0.66	$\frac{8}{L}(9R_p + 9R_{pm})$	0	$0.38R_m + 0.12F$
	Elastic	1.0	1.0	0.37	1.0	0.37	$\frac{4}{L}(9R_p + 9R_{pm})$	$\frac{192EI}{L^3}$	$0.71R - 0.21F$
	Plastic	1.0	1.0	0.33	1.0	0.33	$\frac{4}{L}(9R_p + 9R_{pm})$	0	$0.75R_m - 0.25F$

Table 2.6.1.1b: Transformation Factors for Fixed-Fixed Beam (Biggs, 1964)




Loading diagram	Strain range	Load factor K_L	Mass factor K_M		Load-mass factor K_{LM}		Maximum resistance R_m	Spring constant k	Ejection spring constant k_{st}	Dynamic reaction V
			Concentrated mass*	Uniform mass	Concentrated mass*	Uniform mass				
	Elastic	0.58	0.45	0.78	$\frac{88M_p}{L}$	$\frac{185EI}{L^3}$	$\frac{160EI}{L^3}$	$V_1 = 0.26R + 0.12F$ $V_2 = 0.43R + 0.19F$
	Elastic-plastic	0.64	0.50	0.78	$\frac{4}{L}(3R_m + 2M_{pm})$	$\frac{384EI}{5L^3}$		$V = 0.39R + 0.11F \pm 3M_{pm}/L$
	Plastic	0.50	0.33	0.66	$\frac{4}{L}(3R_m + 2M_{pm})$	0		$V = 0.38R_m + 0.12F \pm 3M_{pm}/L$
	Elastic	1.0	1.0	0.43	1.0	0.43	$\frac{16M_p}{3L}$	$\frac{107EI}{L^3}$	$\frac{108EI}{L^3}$	$V_1 = 0.26R + 0.07F$ $V_2 = 0.54R + 0.14F$
	Elastic-plastic	1.0	1.0	0.49	1.0	0.49	$\frac{2}{L}(3R_m + 2M_{pm})$	$\frac{48EI}{L^3}$		$V = 0.78R - 0.28F \pm 3M_{pm}/L$
	Plastic	1.0	1.0	0.33	1.0	0.33	$\frac{2}{L}(3R_m + 2M_{pm})$	0		$V = 0.76R_m - 0.25F \pm 3M_{pm}/L$
	Elastic	0.81	0.67	0.45	0.83	0.55	$\frac{6M_p}{L}$	$\frac{133EI}{L^3}$	$\frac{122EI}{L^3}$	$V_1 = 0.17R + 0.17F$ $V_2 = 0.32R + 0.33F$
	Elastic-plastic	0.87	0.78	0.52	0.87	0.60	$\frac{2}{L}(3R_m + 3M_{pm})$	$\frac{56EI}{L^3}$		$V = 0.525R - 0.025F \pm 3M_{pm}/L$
	Plastic	1.0	1.0	0.56	1.0	0.56	$\frac{2}{L}(3R_m + 3M_{pm})$		$V = 0.52R_m - 0.02F \pm 3M_{pm}/L$

Table 2.6.1.1c: Transformation Factors for Fixed-Pinned Beam (Biggs, 1964)

Displacement is calculated at each time step of the solution. From the original beam, a value of displacement can be solved for, corresponding to first yielding of the section. If the displacement of the equivalent system reaches this value of displacement, the transformation factors change from the elastic factors to the elastic-plastic factors or the plastic factors, depending on which boundary condition is used. After rebound, the factors go back to the elastic factors and the tracking restarts in the opposite direction.

Certain boundary conditions have two sets of transformation factors, while other boundary conditions have three sets. This is because depending on the boundary condition, the beam can develop several locations of sequential yielding before

becoming a collapse mechanism (not able to resist any further increase in loading). For example, pinned-pinned beams yield in the center and then become a collapse mechanism. Fixed-fixed beams first yield at the supports simultaneously, and the beam essentially becomes a pinned-pinned beam, until it yields again at the center, forming the collapse mechanism. An illustration of this process is shown below in Figure 2.6.1.2:

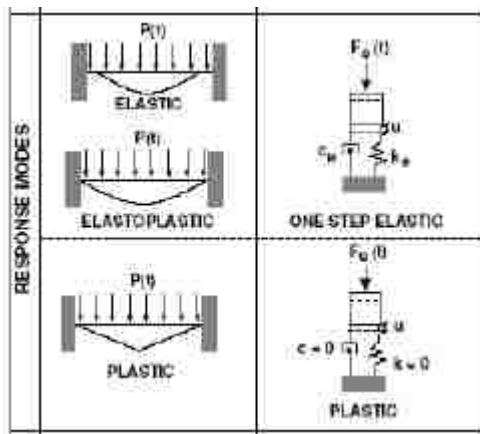


Figure 2.6.1.2: Illustration of Stages of Yielding (US Army, 2008)

SBEDS makes use of these factors in its analysis. By tracking the displacements at which yielding occurs, SBEDS creates a load-displacement graph that tracks how much resistance the system can have at any given displacement. Figure 2.6.1.3 on the following page shows the general resistance-deflection relationship that SBEDS uses. This resistance-deflection relationship is equivalent to the stress-strain relationship mentioned earlier, and takes into account the varying transformation factors at different points of yielding.

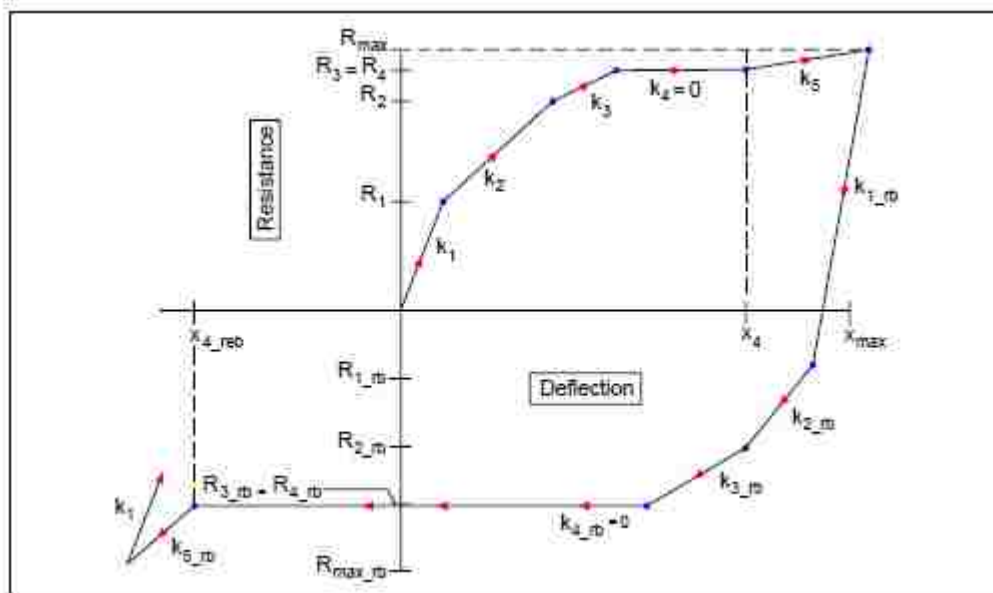


Figure 2.6.1.3: General Resistance-Deflection Relationship for SBEDS (US Army, 2008)

2.7 Space Discretization

For SDOF analysis, the mass of the system is located at a point. There is no need to discretize space, because the mass is all located at the same point in space. However, for MDOF analysis, the mass of the system for a structural element is distributed along the element itself. The solution to the displacement of each point along this element can be solved for using continuum analysis. However, this is complex and not computationally efficient. It is therefore necessary to discretize space along the element, so that mass can be lumped at specific points (nodes) along the element. This allows matrices to be assembled that represent the mass, stiffness, damping, and applied loads for the element. These matrices can then be used in a matrix

formulation of any numerical method described above. An example of space discretization for a two-member frame is shown below in Figure 2.7.1.

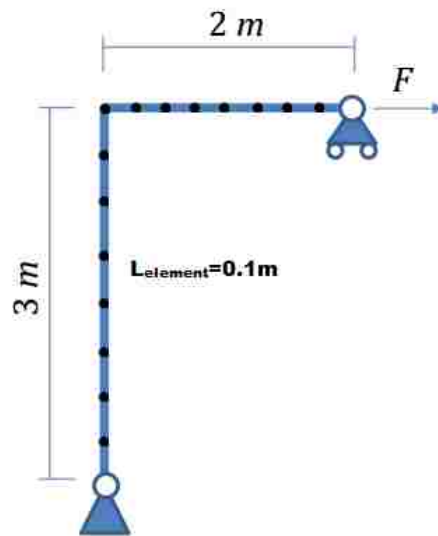


Figure 2.7.1: Example of Space Discretization for Two-Member Frame

The two continuous elements of lengths 3 and 2 meters are discretized into multiple elements of length 0.1 meters. The mass (and any applied loads) of the continuous elements becomes lumped at each node using tributary widths. Each discretized element has its own stiffness and damping coefficients as well, which relate to the overall stiffness and damping of the structure.

2.8 Element Type

In MDOF analysis, an element type must be selected. Different types of elements are capable of exhibiting different types of response behavior. For example, the Euler-Bernoulli Beam only considers deformation due to bending, while the Timoshenko Beam considers deformation due to both bending and shear. It is important to know which type of element is appropriate to use. If the element in question is slender, Euler-Bernoulli Beams give good approximations of behavior. However, for short elements, shear deformation is significant, and Timoshenko Beams must be used to obtain more accurate results. In “A First Course in the Finite Element Method” (Logan, 2012), stiffness matrices for both types of elements are derived. Below displays the elemental elastic stiffness matrix of a Timoshenko Beam:

$$\mathbf{k}^e = \frac{EI}{L^3(1+r)} \begin{pmatrix} 12 & 6L & -12 & 6L \\ 6L & (4+r)L^2 & -6L & (2-r)L^2 \\ -12L & -6L & 12 & -6L \\ 6L & (2-r)L^2 & -6L & (4+r)L^2 \end{pmatrix} \quad (2.8.1)$$

where:

\mathbf{k}^e =elemental elastic stiffness matrix

E =elastic modulus of element

I =moment of inertia of element

L =length of element

$$r = \frac{12EI}{A_sGL^2}$$

A_s =shear area of element

G =shear modulus of element

By substituting zero for r , this stiffness matrix reduces to the Euler-Bernoulli elastic stiffness matrix.

After a system is discretized in space, elemental stiffness matrices are calculated for each discretized element. These elemental stiffness matrices are then assembled into an overall global stiffness matrix, by utilizing the connectivity of the structure.

2.9 Geometric Effects

When axial load is applied to a beam, it becomes classified as a beam-column, because it resists axial load like a column as well as transverse load like a beam. Axial load has an effect on the stiffness of the element. In general, tension in a member will increase its stiffness, while compression will decrease its stiffness. Different methods may be implemented to account for this decrease in stiffness in SDOF analysis. One method, which is implemented by SBEDS, applies additional transverse load to the element for the following time step, based on the static axial load on the member as well as the deflection at the current time step. Here, the stiffness is not directly manipulated, but the end result is the same: a larger deflection.

For MDOF analysis, a geometric stiffness matrix can be used to account for this change in stiffness. This geometric stiffness matrix is added to the elastic stiffness matrix, either before assembly as element matrices, or after assembly as a global matrix. In “Structural Element Stiffness, Mass, and Damping Matrices” Gavin (2014) derives geometric stiffness matrices for both Euler-Bernoulli and Timoshenko elements. By substituting zero into the shear factor, the Timoshenko geometric stiffness matrix reduces to the Euler-Bernoulli matrix. Below is the elemental geometric stiffness matrix for Timoshenko Beam:

$$kg := \frac{P}{L \cdot (1+r)^3} \begin{bmatrix} \frac{6}{5} + 2r + r^2 & \frac{L}{10} & \frac{-6}{5} - 2r - r^2 & \frac{L}{10} \\ \frac{L}{10} & \frac{2 \cdot L^2}{15} + \frac{L^2 \cdot r}{6} + \frac{L^2 \cdot r^2}{12} & \frac{-L}{10} & \frac{-L^2}{30} + \frac{L^2 \cdot r}{6} - \frac{L^2 \cdot r^2}{12} \\ \frac{-6}{5} - 2r - r^2 & \frac{-L}{10} & \frac{6}{5} + 2r + r^2 & \frac{-L}{10} \\ \frac{L}{10} & \frac{-L^2}{30} + \frac{L^2 \cdot r}{6} - \frac{L^2 \cdot r^2}{12} & \frac{-L}{10} & \frac{2 \cdot L^2}{15} + \frac{L^2 \cdot r}{6} + \frac{L^2 \cdot r^2}{12} \end{bmatrix}$$

(2.9.1)

where:

kg =elemental geometric stiffness matrix

P =applied axial load (tension positive)

L =length of element

r defined earlier in Equation 2.8.1

2.10 Mass Matrix

In MDOF analysis, mass matrices can be modeled differently as well. A simple way to model mass matrices is to lump half of the elements mass at each node (assuming a two-node element), thus creating a diagonal mass matrix. A more comprehensive approach involves the use of shape functions, which come from the Finite Element Method (not covered in this thesis). “A First Course in the Finite Element Method” (Logan, 2012) derives mass matrices using these shape functions. This “consistent-mass matrix” is shown below:

$$\mathbf{m}^e = \frac{\rho AL}{420} \begin{pmatrix} 156 & 22L & 54 & -13L \\ 22L & 4L^2 & 13L & -3L^2 \\ 54 & 13L & 156 & -22L \\ -13L & -3L^2 & -22L & 4L^2 \end{pmatrix} \quad (2.10.1)$$

where:

\mathbf{m}^e =elemental mass matrix

ρ =density of element

A =area of element

L =length of element

Przemieniecki (1968) derived the consistent-mass matrix using Timoshenko theory.

Substituting zero for the shear factor in the Timoshenko mass matrix reduces the matrix to the Euler-Bernoulli mass matrix. The Timoshenko mass matrix requires

significantly more space to display than the Euler-Bernoulli consistent-mass matrix, and is therefore not shown.

After a system is discretized in space, elemental mass matrices are calculated for each discretized element. These elemental mass matrices are then assembled into an overall global mass matrix, by utilizing the connectivity of the structure. This process is identical to the process for assembling the global stiffness matrix.

2.11 Mode Shapes

In MDOF analysis, once the global matrices are assembled, there are two options for solving the displacements using numerical methods. The first option is to directly input the matrices into Equation 2.2.1 from earlier, and then use the matrix formulation of whichever numerical method is chosen. This involves matrix computations.

The second option is to use transformations to create another set of matrices, called modal matrices. These matrices are diagonal, which allows Equation 2.2.1 to be decoupled. This means that instead of having one N by x matrix equation (N is number of DOFs), there are instead N scalar equations. Each of these scalar equations can be solved independently using any chosen numerical method. The solution to these equations are modal displacements, which then get transformed back into physical displacements.

To obtain the modal stiffness and mass matrices, a matrix eigenvalue problem must be solved, which is shown below in Equation 2.11.1:

$$\mathbf{k}\phi_n = \omega_n^2 \mathbf{m}\phi_n \quad (2.11.1)$$

where:

\mathbf{k} =global stiffness matrix

\mathbf{m} =global mass matrix

ϕ_n =mode shape for mode n of N

ω_n =natural frequency of vibration for mode n of N

Once the mode shapes are determined, they can be concatenated into a matrix, Φ .

The global stiffness and mass matrices can then be pre-multiplied and post-multiplied by Φ to create the modal stiffness and modal mass matrices, which are diagonal as stated previously. These new matrices can be substituted into Equation 2.2.1, with the unknown quantities now being modal displacements (q_n), modal velocities (\dot{q}_n), and modal accelerations (\ddot{q}_n). These form decoupled equations, since the modal matrices are diagonal. Therefore, the modal displacements can be solved for, for each mode individually. Once these are determined, a transformation brings the modal displacements into the physical displacements. The same equation applies to velocities and accelerations. Equation 2.11.2 shows this transformation:

$$\mathbf{u}(t) = \sum_{r=1}^N \phi_r q_r(t) \quad (2.11.2)$$

where:

$\mathbf{u}(t)$ =vector of physical displacements

This second method has two benefits. The first benefit is the fact that the differential equations become decoupled and are simpler to solve independently. The second benefit is that higher modes can be neglected in the transformation back to physical coordinates. These higher modes have large corresponding natural frequencies of vibration. This means the periods of vibration for these higher modes are very short, which can lead to inaccuracy and instability in the numerical solution, as described in Chapter 2.2. By neglecting these higher modes, the accuracy and stability limits become more feasible to satisfy when choosing a time step.

However, Chopra (1995) states that, “[c]lassical modal analysis is [...] not applicable to the analysis of nonlinear systems.” Therefore, a different formulation will be explained in the following section.

2.12 Damping

Damping refers to the loss of energy due to friction, which causes a decrease in the response over time. Damping can be strain-rate dependent, which can make it very

complex to model. In SDOF analysis, damping is modeled using the damping coefficient, c , in Equation 2.2.1 from earlier. SBEDs allows users to input a damping ratio, which is used to calculate c for the analysis. In *Methodology Manual for the Single-Degree-of-Freedom Blast Effects Design Spreadsheets (SBEDS)*, reference is made to TM 5-1300, which recommends using damping ratios of 5% for steel, and 1% for reinforced concrete. Below is another excerpt from the methodology manual:

“UFC 3-340-01 recommends that damping values commonly used for earthquake analysis and design [...] can be used for elastic and elastoplastic component response only. Therefore, the damping term is set to zero for all plastic response. This is recommended because the damping forces for structure response to earthquake loading were developed primarily for elastic component response, which involves deformation over the whole volume of the component. After ultimate resistance, deformation consists primarily of localized hinging at maximum moment regions, which occurs over a much smaller volume, so that no significant additional damping occurs” (Us Army, 2008).

In MDOF analysis, there are many modeling options for damping, which vary in complexity. Chopra (1995) presents three main options: *Caughey Damping*, superposition of modal damping matrices, and nonclassical damping. The first two options fall under the category of classical damping.

The formulation of the natural modes explained in the previous section are constructed in a way so that the mass and stiffness matrices are orthogonal to each mode. This means that pre-multiplying either matrix by the transpose of any mode

shape, and post-multiplying the result by the same mode shape will result in 0. For a damping matrix to be classically damped, it must also satisfy this mode orthogonality.

In the first method, *Caughey Damping*, damping ratios for the first J modes can be specified. The rest of the damping ratios that are not specified are automatically calculated. Equations 2.12.1a and 2.12.1b below are used to calculate this type of damping matrix:

$$\mathbf{c} = \mathbf{m} \sum_{l=0}^{J-1} a_l [\mathbf{m}^{-1} \mathbf{k}]^l \quad (2.12.1a)$$

$$\zeta_n = \frac{1}{2} \sum_{l=0}^{J-1} a_l \omega_n^{2l-l} \quad (2.12.1b)$$

where:

\mathbf{c} =global damping matrix

ζ_n =damping ratio for mode n

J=number of modes to specify damping ratios for

First, damping ratios are specified for J modes. Then, a_l constants are solved for using Equation 2.12.1b. Once these constants are determined, the global damping matrix can be determined. A special case of this is called *Rayleigh Damping*, where only two modes have determined damping ratios ($J=2$).

The second method, superposition of modal damping matrices, is very similar to *Caughey Damping*. Damping ratios are specified for any number of modes. However, instead of the modes not specified being automatically calculated, there is no damping in these modes. Equation 2.12.2 below is used to calculate the classical damping matrix using superposition of modal damping matrices:

$$\mathbf{c} = \mathbf{m} \left(\sum_{n=1}^J \frac{2\zeta_n \omega_n}{M_n} \phi_n \phi_n^T \right) \mathbf{m} \quad (2.12.2)$$

where:

M_n =modal mass for mode n (n th diagonal term on modal mass matrix)

The third method, non-classical damping, involves modifying the eigenvalue formulation for determining mode shapes. In the previous section, the eigenvalue problem was formulated by neglecting damping. By including damping, the matrix eigenvalue problem becomes the quadratic eigenvalue problem. Equation 2.12.3 below represents this formulation:

$$(\lambda_n^2 \mathbf{m} + \lambda_n \mathbf{c} + \mathbf{k}) \psi_n = \mathbf{0} \quad (2.12.3)$$

where:

λ_n =eigenvalue of problem

ψ_n =eigenvector of problem

λ_n can be thought of as equivalent to ω_n^2 in the matrix eigenvalue problem and ψ_n can be thought of as equivalent to ϕ_n in the matrix eigenvalue problem. The solution to this equation yields complex numbers.

Under nonlinear dynamic analysis, damping becomes even more complex. Once an element yields, with the sudden reduced stiffness, the velocity is likely to increase suddenly. Since viscous damping is proportional to the velocity, there would be increased damping at this point of yielding, which is not realistic. To mitigate this issue, many solvers implement a reduction in damping during yielding. An example of this is to reduce the elemental damping matrices that correspond to any element that yields by a factor of 10^{-3} . This reduction would make the damping force associated with the increased velocity insignificant enough not to overdamp the system during yielding.

Chapter 3 Single-Degree-of-Freedom Solver

3.1 General

This chapter discusses all aspects of the SDOF analysis. This includes the modeling techniques that SBEDS implements, how SBEDS was replicated using MATLAB, and some minor inconsistencies between theory and output.

3.2 SBEDS

SBEDS is a Microsoft Excel-based spreadsheet used to design structural elements under dynamic loading. It is commonly used for blast design. There are several sheets to the file, which are explained in the following sections.

3.2.1 Intro Tab

The Intro tab allows the user to select a type of component used for the analysis, such as a steel beam, or a reinforced concrete beam. In this tab, the user can also choose between English and Metric units. For this thesis, English units will be used, as this is the standard in practice. The Intro tab also lists instructions and comments about the program.

3.2.2 Input Tab

The Input tab allows users to input several parameters for the analysis. Specifically, yellow cells specify user input cells. Element size, braced length, boundary conditions,

supported weight, axial load, lateral load functions, time step, damping ratio, and incident angle are some input parameters in this tab. A recommended time step is displayed here. This recommended time step is the smallest of several values, which are listed below (US Army, 2008):

- 10% of the natural period
- 10% of the smallest time increment in a manually input, point-wise linear blast load
- 3% of the equivalent triangular positive phase duration or 1.5% of the equivalent triangular negative phase duration of an input charge weight-standoff blast load
- 3% of the smallest calculated time between local maxima and minima points of an input blast load file
- eight natural periods divided by 2900 time steps

This tab also displays the resistance-deflection relationship for the chosen parameters, as well as yield displacements and other important parameters. Once the input parameters are entered, a green button labelled "RUN SDOF" runs the analysis. Results of this analysis include maximum values of displacement, rotation, and ductility.

3.2.3 Results Tab

This tab displays several plots once the analysis is run. These plots include displacement-time, load-time, resistance-time, resistance-displacement, and dynamic shear-time.

3.2.4 P-I_Diagram Tab

This tab allows the user to create a pressure-impulse diagram for the selected element.

3.2.5 SDOF Output Tab

This tab shows the tabulated data for the analysis. These values are the plots in the Results tab. This output tab displays columns of output, such as deflection, velocity, acceleration, resistance, and the load-mass factor for each time step. SBEDS uses the *constant-velocity procedure* to calculate the output, as explained in section 2.2 earlier. This method solves the displacement, velocity, and acceleration at each time step. When the value of the displacement reaches a yield value (calculated in the Input tab), the load-mass factor changes to the appropriate factor, and the time stepping continues. On rebound, the load-mass factor changes back to the elastic value and the process starts over on rebound. The program calculates these values for a total of 2900 time steps, no matter what size time step is chosen. Figure 3.2.5.1 on the following page shows the displacement-time plot for an arbitrary analysis:

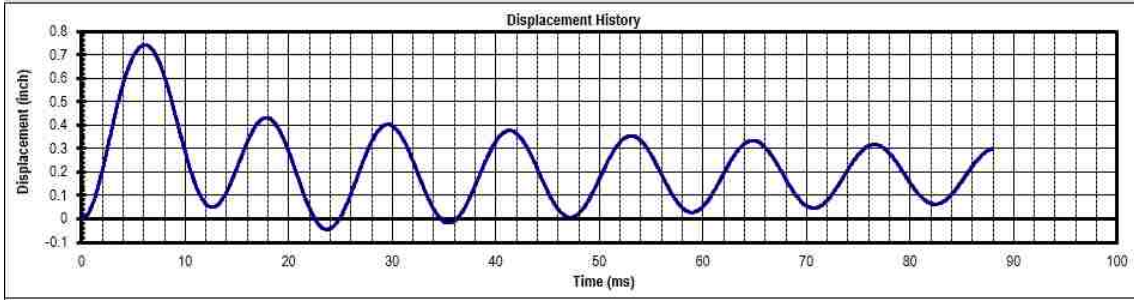


Figure 3.2.5.1: Displacement-Time Plot for Arbitrary SBEDS Analysis (SBEDS, 2008)

3.2.6 Database Tab

The Database tab has various constants that the analysis uses, depending on what the user chooses as inputs. These constants include the Biggs transformation factors for each condition, yield stresses and elastic moduli for certain materials, and increase factors for yield values.

3.2.7 Beam_Types Tab

This tab is similar to the Database tab. In this tab, properties of every cross-section type are listed for the analysis to reference. For every section type (Wide Flange, Tube Section, etc.), properties are listed such as weight, plastic section modulus, and moment of inertia. These are all needed for the analysis.

3.3 MATLAB Replica

The first step in quantifying the inaccuracies in SBEDS is to create a program that takes the same inputs as SBEDS, performs an equivalent SDOF analysis, and returns the

same outputs as SBEDS. The reason this has to be done with programming is because this program will be used several times for ranges of different input parameters, in order to create data frames. These data frames will be compared to data frames created from the MDOF solver. MATLAB was used to write this program.

To keep the program organized like SBEDS is, several function files were created that have different features. These functions are equivalent to the different tabs that SBEDS has. For the full set of program files, see Appendix A: MATLAB Code for SBEDS Replica (SDOF Solver).

3.3.1 Main File

The first program file, called “main.m,” is the main part of the program. From here, variables are passed to and from all other functions for analysis. This main file starts with declarations of all inputs. This is equivalent to the Intro and Input tabs from SBEDS. All of the input parameters are then converted to a consistent set of units so that all further calculations are in consistent units. The rest of the main file consists of functions that transfer variables and perform calculations.

3.3.2 Load File

The first function, called “Load.m,” creates the loading function. Four inputs are taken from the main file (initial applied pressure [psi], length of pressure [ms], time step

[ms], and number of steps). This load function creates a triangular load function and passes the value of the load at each time step back to the main file.

3.3.3 Biggs File

The next function, called “Biggs.m,” specifies which transformation factors will be used for the equivalent SDOF analysis. This function takes two inputs from the main file (load type [point load or uniform load], and boundary conditions [pinned-pinned, fixed-fixed, fixed-pinned, or cantilevered]). From these two inputs, all the transformation factors are defined, and passed back to the main file.

3.3.4 Const_vel File

The next function, called “const_vel,” performs the bulk of the calculations. Here, the *constant-velocity* procedure is implemented. Initial conditions on the displacement and velocity, the Biggs transformation factors, mass, damping ratio, stiffness, the loading function, time step, and number of steps are all passed to this function. For each time step, the value of the resistance is compared to the values of resistance that would cause yielding, and therefore would cause the transformation factors to change. Once the value of resistance reaches 95% of maximum resistance, the damping ratio changes to zero until rebound. SBEDS implements this modeling option for damping, as explained earlier.

For each time step, every value of time, loading, displacement, velocity, acceleration, stiffness, resistance, load-mass factor, and internal shear are stored into a matrix called "alldata." This is the same format as the SDOF Output tab in SBEDS, so that it is easy to compare the output of SBEDS to the output of this MATLAB program. All of these calculated variables are then sent back to the main file, where final post-processing is performed, at which point plots are created of the outputs.

3.4 Solver Verification

To ensure that this MATLAB program functions identically to SBEDS, several sets of inputs were analyzed with both programs and compared. The element chosen for comparison was a W10X12 Grade 50 steel beam, with length of 20 ft and tributary width of 0.5 ft. The duration of the pressure load was 17.8 ms, which is a common duration length in practice. Every boundary condition and load type were tested, with damping ratios of 0%, 5%, and 20%. The initial pressure load was also varied for each combination to get the maximum displacement of the system to be in each region of plasticity.

Once all of these cases were verified, other parameters were changed as well. This included changing the duration of the pressure load to 8 ms, adding additional weight (30 psf, which converted to additional mass) to the system, and changing the beam shape to a W40X655.

An example of how the two analyses were compared is shown below in Figure 3.4.1:

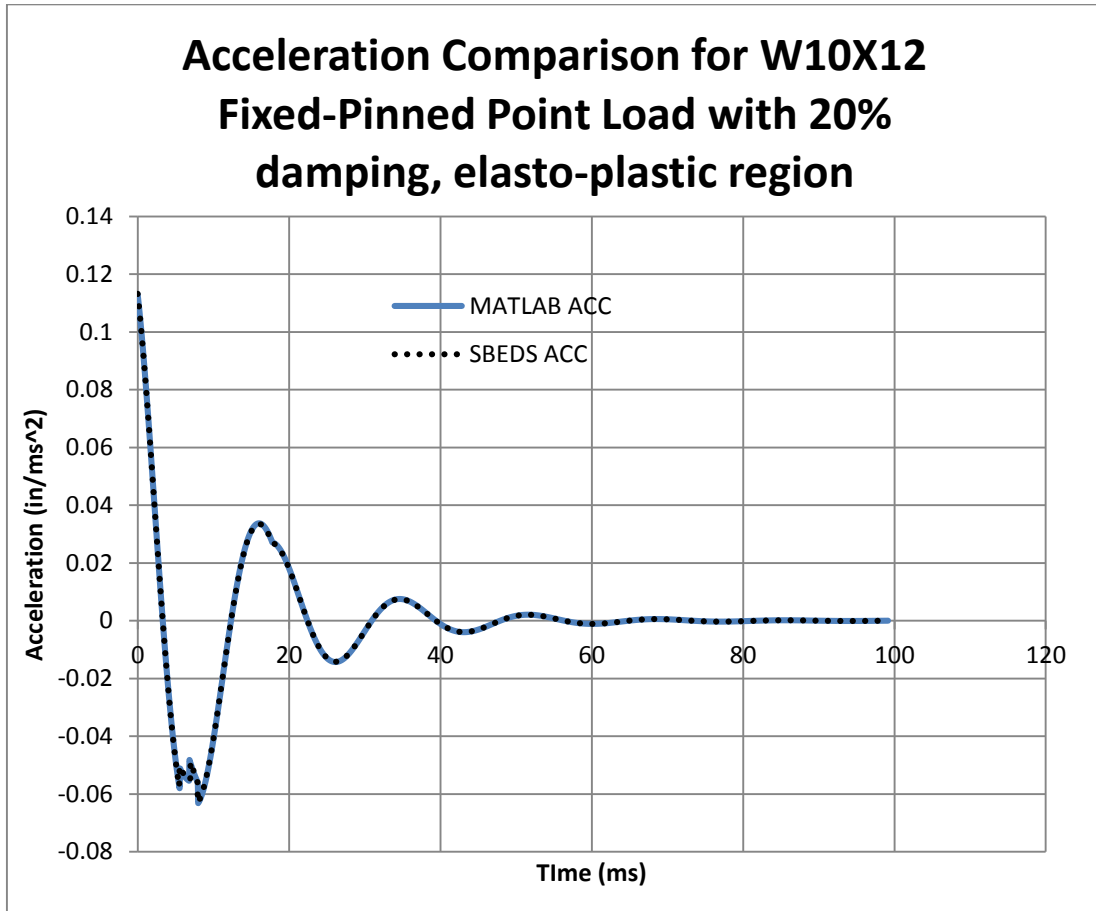


Figure 3.4.1: Acceleration Comparison of SBEDS with SDOF MATLAB Program

The data from the SDOF Output tab from SBEDS was pasted into a new Excel document. The data from the “alldata” matrix was also pasted into this Excel document. Plots were created for the displacement, velocity, acceleration, resistance, and internal shear force for each set parameter variations. Since the plots of both sets of outputs overlapped, the MATALB replica was verified as being identical to SBEDS.

For the full set of verifications, see Appendix B: SDOF Solver Verifications (Time History Plots).

3.4.1 Inconsistencies

While comparing each set of parameters to SBEDS, an inconsistency was found between the theory stated in the SBEDS Manual and the SBEDS output. Additionally, some minor inconsistencies between SBEDS and the MATLAB replica were displayed in the plots, which are the results of certain modeling choices.

The inconsistency between SBEDS theory and output was in the resistance-deflection relationship. Figure 2.6.1.3 from earlier shows this relationship that SBEDS apparently always follows. The results from the W40X655 element, however, showed a different resistance-deflection relationship. Figures 3.4.1.1a and 3.4.1.1b below shows the resistance-deflection output for this W40X655 element, which underwent a load large enough to create a fully plastic resistance:

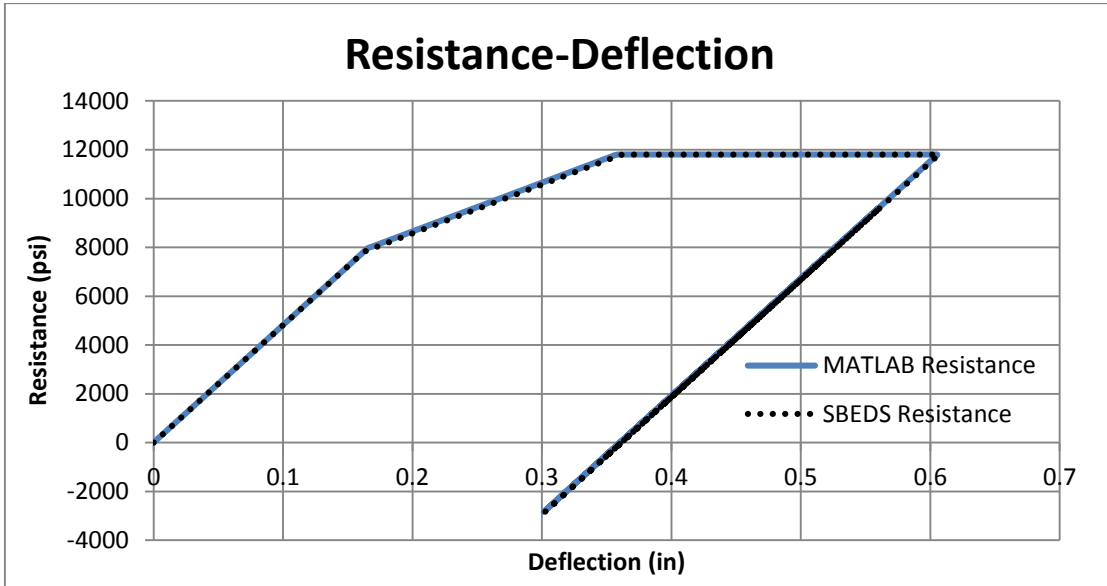


Figure 3.4.1.1a: Resistance-Deflection Plot for W40X655 Fixed-Pinned with Uniform Load and 0% Damping

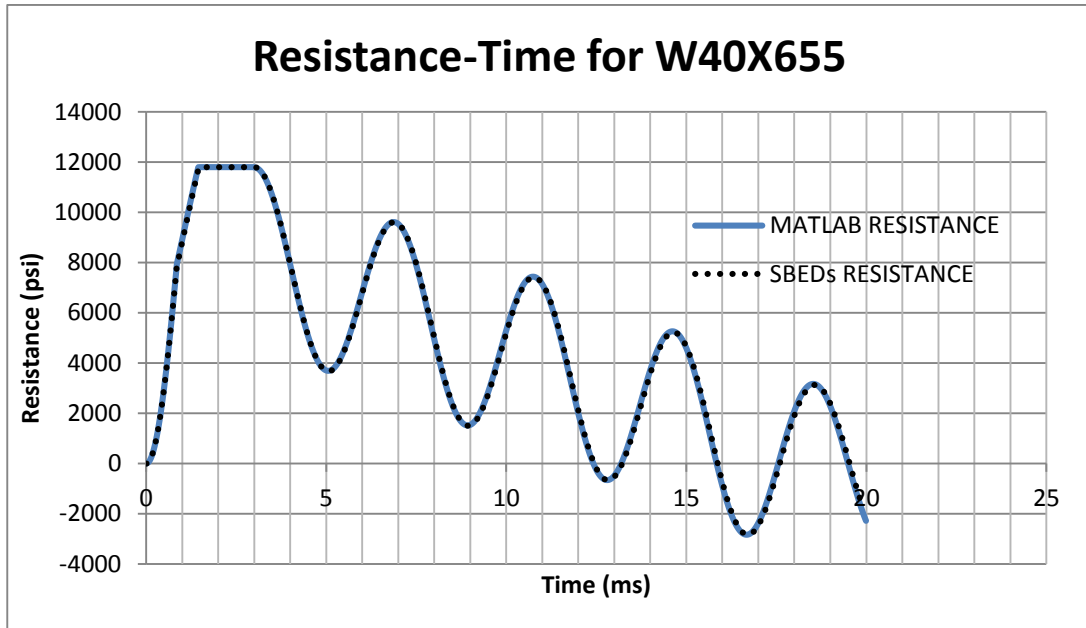


Figure 3.4.1.1b: Resistance-Time Plot for W40X655 Fixed-Pinned with Uniform Load and 0% Damping

From Figure 3.4.1.1a, it can be seen that the first yield transition occurs at a resistance of approximately 8000 psi. Once the mass goes through rebound, and then inbound again, the resistance reaches this value again. This can be seen in Figure 3.4.1.1b, since there is no overlap. However, the resistance function does not change slopes here (the stiffness stays the same). In other cases, the resistance-deflection relationship is consistent with the SBEDS theory. Figures 3.4.1.2a, 3.4.1.2b, and 3.4.1.2c below show this with the W10X12 element, which also underwent a load large enough to create a fully plastic resistance:

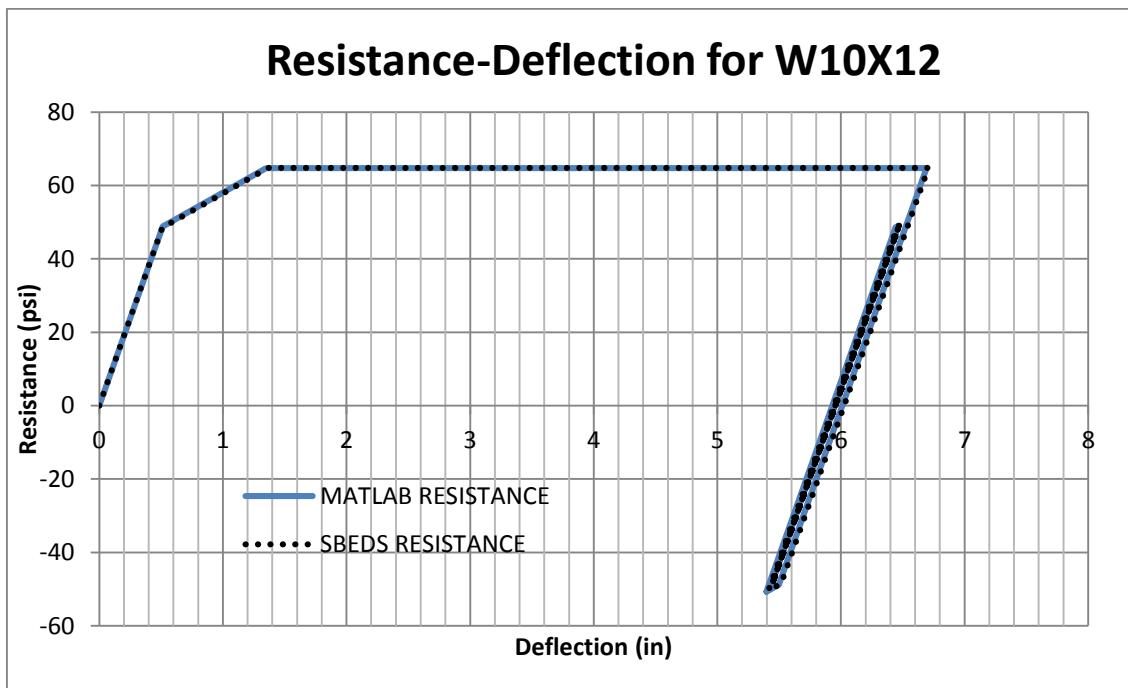


Figure 3.4.1.2a: Resistance-Deflection Plot for W10X12 Fixed-Fixed with Uniform Load and 0% Damping

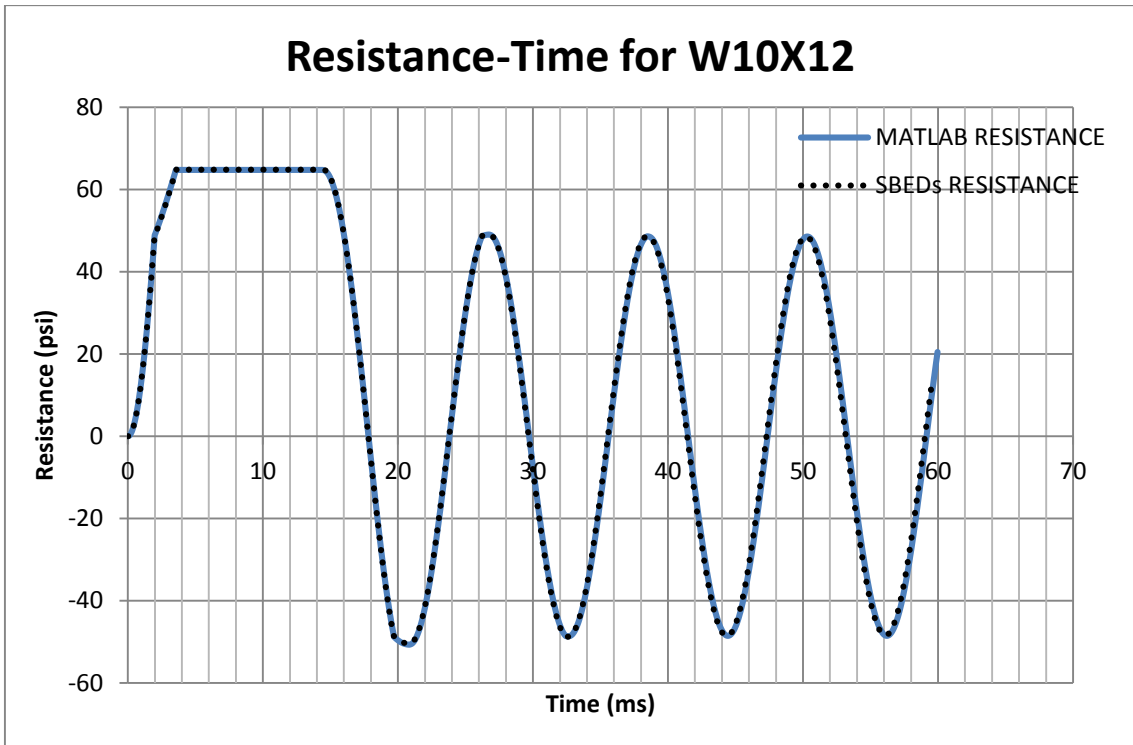


Figure 3.4.1.2b: Resistance-Time Plot for W10X12 Fixed-Fixed with Uniform Load and 0% Damping

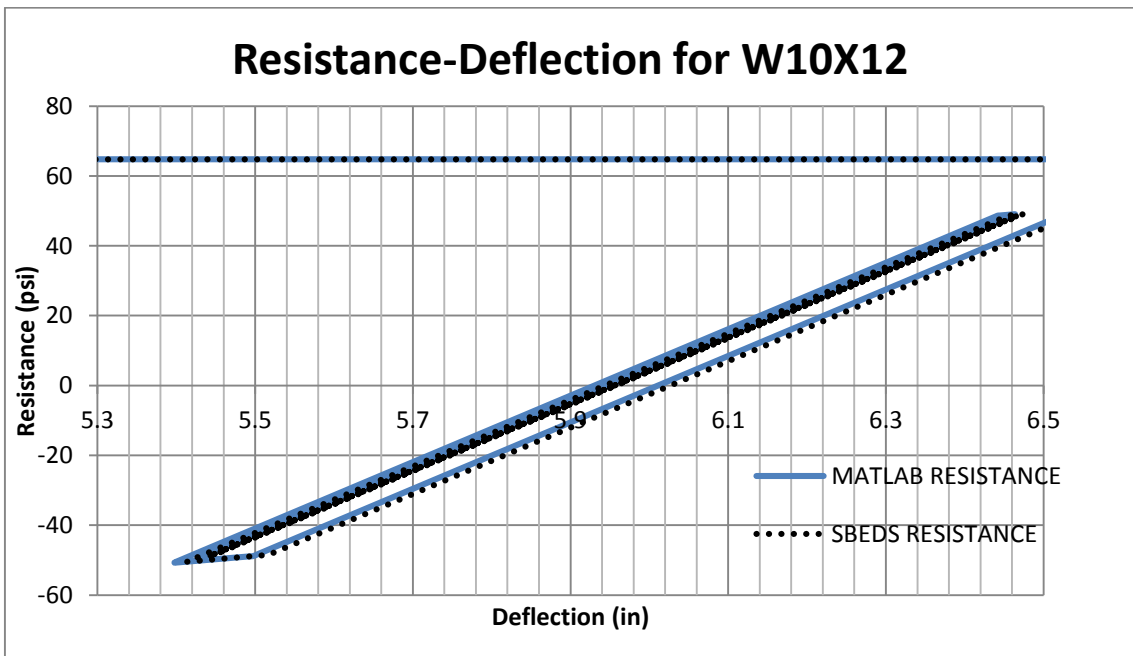


Figure 3.4.1.2c: Resistance-Deflection Plot for W10X12 Fixed-Fixed with Uniform Load and 0% Damping (Magnified)

From Figure 3.4.1.2a, it can be seen that the first yield transition occurs at a resistance of approximately 50 psi. Once the mass goes through rebound, and then inbound again, the resistance reaches this value again. This can be seen in Figure 3.4.1.2b, since there is no overlap. For this analysis, the resistance function does change in slope. Figure 3.4.1.2c shows this (the stiffness changes). In other cases, the resistance-deflection relationship is consistent with the SBEDS theory.

To account for this inconsistency in the MATLAB replica, a datum resistance value is stored during the time stepping. This datum resistance is subtracted from the resistance that is checked against the resistance values to cause the transformation factors to change (yield regions). This allows the replica to match SBEDS for both situations. Although this inconsistency does exist in SBEDS, it does not cause any major issues while using the program. This is because design is based on maximum response, and most maximum responses occur before rebound, which is before this issue ever arises.

The modeling choices made while creating the MATLAB replica led to some minor inconsistencies in the plots when validating each case. Figures 3.4.1.3a, 3.4.1.3b, and 3.4.1.3c on the following pages show one of these discrepancies:

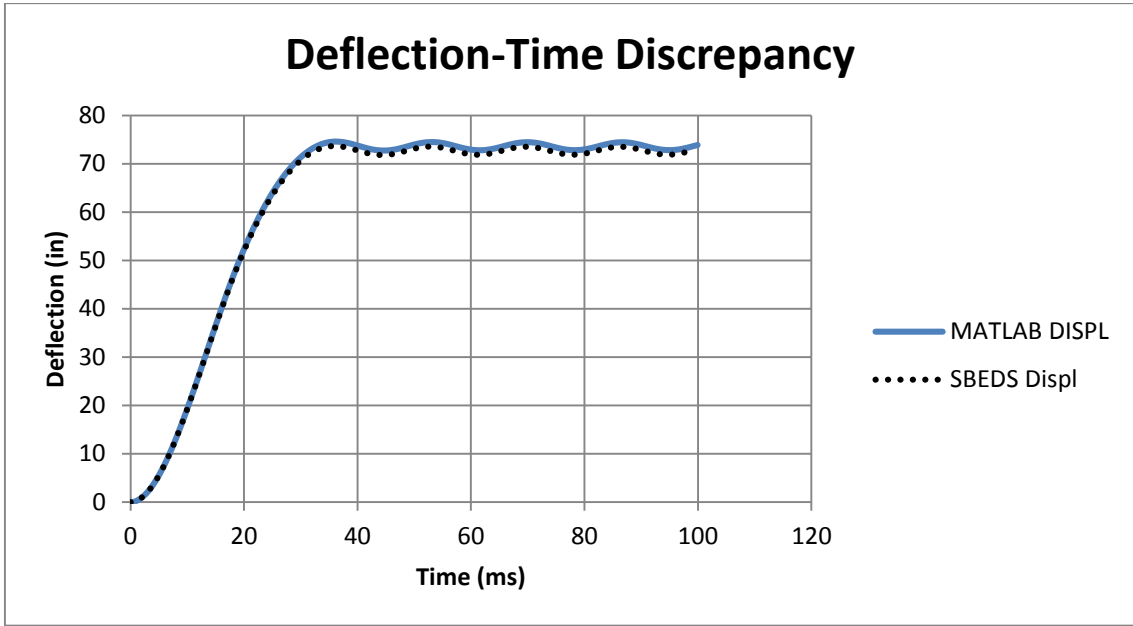


Figure 3.4.1.3a: Deflection-Time Plot for W10X12 Fixed-Pinned with Point Load and 0% Damping

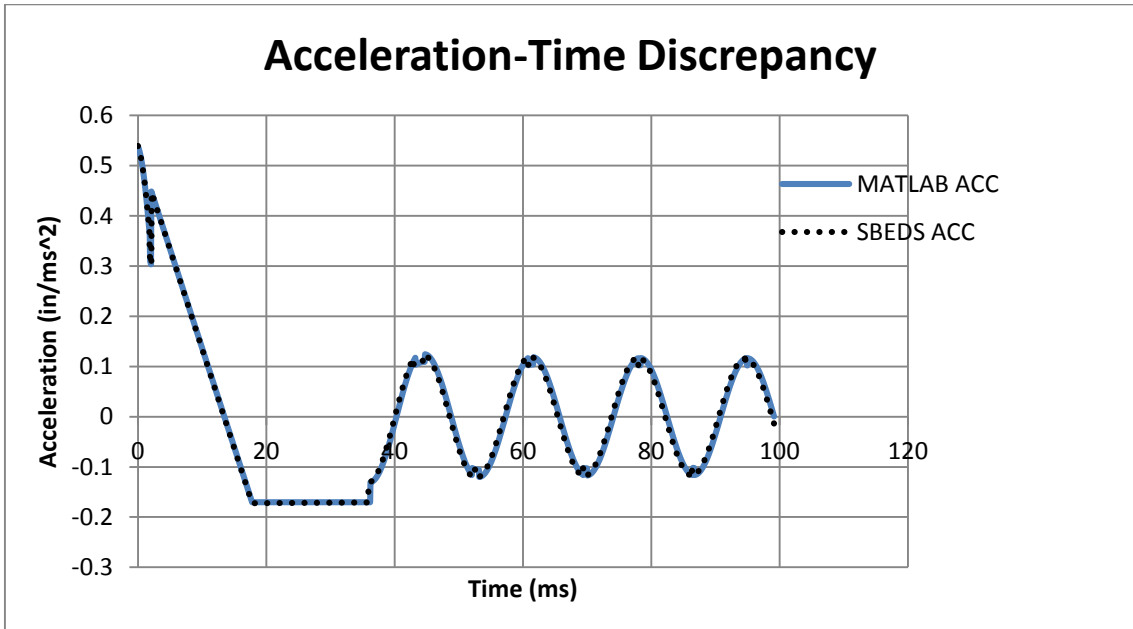


Figure 3.4.1.3b: Acceleration-Time Plot for W10X12 Fixed-Pinned with Point Load and 0% Damping

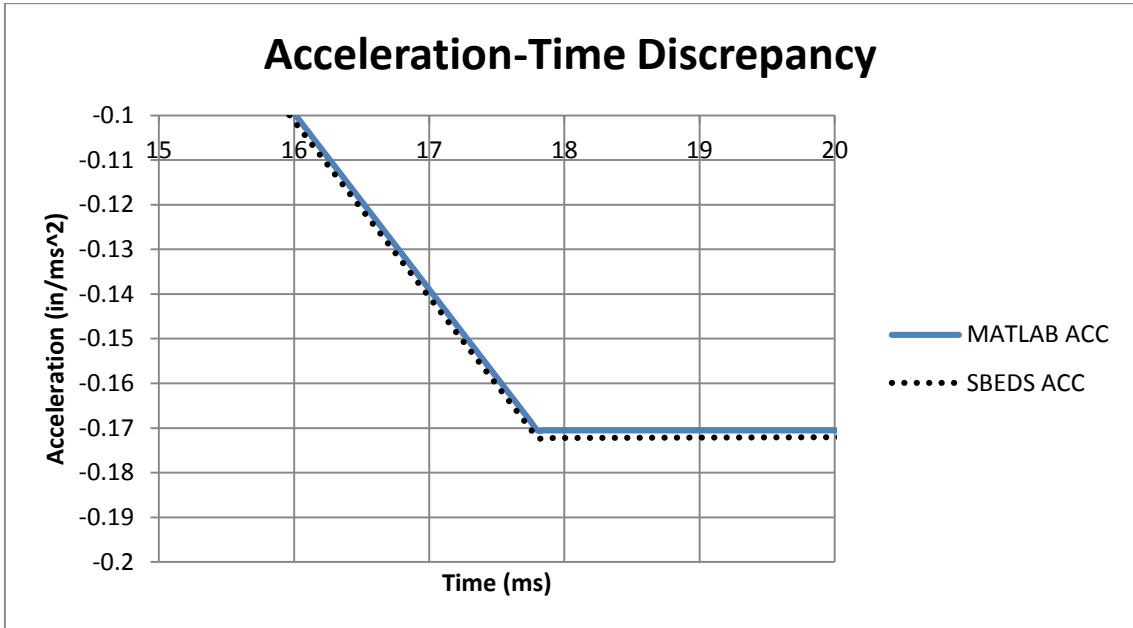


Figure 3.4.1.3c: Acceleration-Time Plot for W10X12 Fixed-Pinned with Point Load and 0% Damping (Magnified)

As seen in the above plots, the acceleration in SBEDS is slightly more negative than the MATLAB replica, which causes the SBEDS displacement to be decreased. This is due to the time step at which the transformation factors change, as a result of the resistance reaching the yield value. For this specific case, the SBEDS solution spent a few more time steps in the elasto-plastic region than the MATLAB solution did. This additional stiffness for the elasto-plastic time steps is what caused the deflection to be smaller for the SBEDS solution. The reason for these additional elasto-plastic time steps may be due to the amount of significant digits that Excel stores compared to the amount that MATLAB stores, and if SBEDS rounds the deflection values at which the stiffness changes. However, this is a minor discrepancy (1.3% error), and when damping is introduced, the plots overlap one another.

Another slight discrepancy is shown in Figure 3.4.1.4 below:

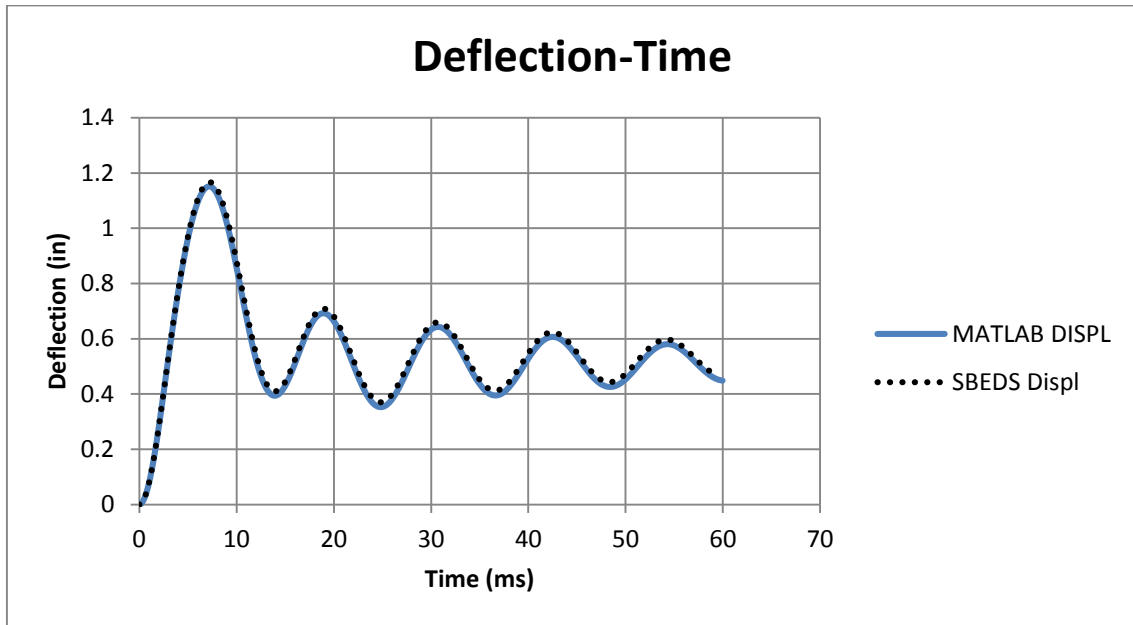


Figure 3.4.1.4: Deflection-Time Plot for W10X12 Fixed-Fixed with Uniform Load and 5% Damping

This discrepancy is due to the fact that the MATLAB program does not reset the resistance as the exact yield value when transitioning. Instead, it uses the calculated resistance from the previous time step, and therefore slightly overshoots the value of resistance. Since the resistance is larger for one time step, the deflection is slightly lower for the rest of the solution. However, this decrease is small enough (1.3% error) to still be considered accurate with respect to SBEDS's output.

Chapter 4 Multi-Degree-of-Freedom Solver

4.1 General

This chapter discusses all aspects of the MDOF analysis. This includes the modeling techniques that SAP2000 implements, how the single-beam analysis in SAP2000 was replicated using MATLAB, and some minor inconsistencies between the two solvers as well as SAP2000 methodology and SBEDS methodology.

4.2 SAP2000

SAP2000 is a software package used to perform structural analysis, structural dynamic analysis, and structural design. It is commonly used in the structural engineering industry. The structural dynamic analysis capabilities of SAP2000 are the primary focus of this verification study. This is a more precise version of the analysis that SBEDS performs. The increase in accuracy comes from the fact that SAP2000 performs MDOF analysis, whereas SBEDS performs SDOF analysis. Some of the key features of this tool are explained in the following sections.

4.2.1 Inputs

SAP2000 has a graphical user interface as opposed to being a spreadsheet like SBEDS. First, a grid is created for the nodes, or degrees of freedom to be placed. Next, definitions of materials and section properties are specified, including material

nonlinear hinges which be applied to the nodes. Once these are specified, the user can create the structural element or system with nodes and lines. An example of the material definition window is shown below in Figure 4.2.1.1:



Figure 4.2.1.1: Example of Material Definition Window for SAP2000

Once the geometry is defined with associated materials and section properties, boundary conditions and loads can be applied to the structure, as well as mass

definitions, which will be used in dynamic analysis. These loads can be linked to time-history plots, and damping and a numerical method can be specified for the time-history analysis. SAP2000 is limited by having lumped mass matrices only; it does not use consistent-mass matrices. However, shear area can be included or neglected, thereby specifying Timoshenko stiffness matrices or Euler-Bernoulli stiffness matrices, respectively. Geometric effects due to axial load can be included or neglected as well. Figure 4.2.1.2 below shows the graphical representation of a defined structure in SAP2000:

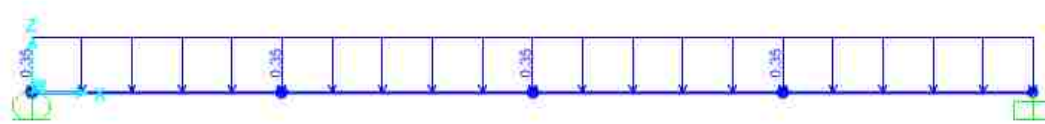


Figure 4.2.1.2: Graphical Representation of a Pinned-Fixed Beam with 0.35 k/in Load and 4 Elements

Damping in SAP2000 is specified with *Rayleigh Damping* factors. These factors, α and β are multiplied by the elemental mass and elastic stiffness matrices respectively. An important note is that SAP2000 uses the initial assembled damping matrix throughout the whole time-history analysis. If a degree of freedom corresponding to an element yields, its corresponding elemental stiffness-proportional damping matrix is multiplied by a factor between 10^{-2} and 10^{-3} , and then the global damping matrix is reassembled. This is used until the degree of freedom that yielded reverses direction, at which the original damping matrix is used again.

4.2.2 Outputs

Once the input parameters are chosen, a time-history can be performed, and all the output data is accessible. This includes time-history data in spreadsheet format for displacements, velocities, accelerations, internal forces, and reactions. SAP2000 has a feature for plotting the data as well. Shown below in Figure 4.2.2.1 is an example plot from SAP2000:

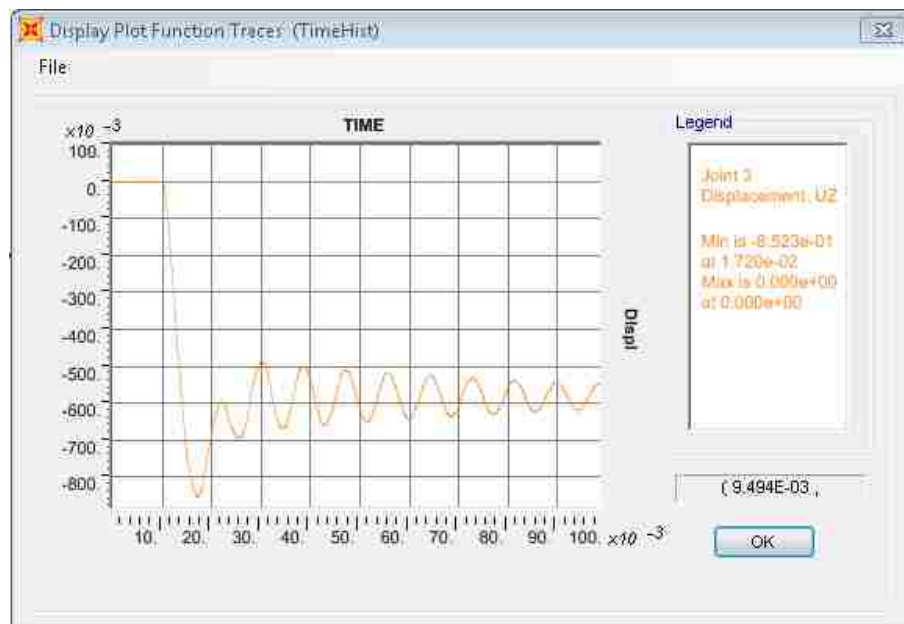


Figure 4.2.2.1: Example Plot of Deflection-Time in SAP2000

4.3 MATLAB Replica

The next step in quantifying the inaccuracies in SBEDS is to create a program that takes the same inputs as SAP2000 (and SBEDS), performs a MDOF analysis like SAP2000, and

returns the outputs of interest. MATLAB was used to write this program as well, as explained earlier. This code is provided in Appendix C: MATLAB Code for More Accurate Model (MDOF Solver).

4.3.1 Input File

The first function file, called “input.m,” contains all the input data for the analysis, specified by the user. This file allows the user to specify the geometry of the structure, type of mass matrices to be used (consistent or lumped), type of elements to be used (Timoshenko or Euler-Bernoulli), β and γ factors for *Newmark’s method*, tolerance of unbalanced load for Newton-Raphson iteration, shape of the loading time-history curve, boundary conditions, force values and locations scaled by the time-history curve, axial load, *Rayleigh Damping* factors, length of the system, number of elements (mesh), material parameters (elastic modulus, yield stress, density), section properties (cross-sectional area, moment of inertia, shear area), and non-structural masses.

4.3.2 Main File

The program file, called “mainNR.m,” is the bulk of the code. It first reads the data from the input function and performs preprocessing on this data to setup all the matrices and vectors for time-history analysis. The code follows the steps that a standard direct stiffness method or finite element method approach would follow, by assigning all the properties to each element, and then assembling global matrices based on the boundary conditions. Several small functions are used throughout this

process, such as a function that compute equivalent nodal loads for distributed loads (called “g_element.m”) and a function that calculates the stiffness and mass matrices for each element passed to it (called “k_element.m”).

Once all of this setup is complete, a “while” loop runs through the time stepping of *Newmark’s method* with Newton-Raphson iteration. If an element yields, its stiffness and mass matrices are passed to a function, called “condense.m,” that performs static condensation to release the degree of freedom that hinged. After releasing this degree of freedom, the resistance of this degree of freedom is capped at the plastic moment capacity, since the solution is incremental. When this element yields, the rows and columns in the stiffness-proportional damping matrix associated with the degree of freedom that yielded are scaled by 10^{-3} . SAP2000’s solution scales the entire stiffness-proportional damping matrix; however, by only scaling the rows and columns associated with the yielded degree of freedom, the MATLAB solution matches SAP2000 more closely.

Once the time stepping is complete, some post-processing is performed on the data to allow the data to be stored in a more efficient manner and plotted.

4.4 Solver Verification

To ensure that this MATLAB program functions as close to SAP2000 as possible, several sets of inputs were analyzed with both programs and compared. The element

chosen for comparison was a W10X12 Grade 50 steel beam, with length of 10 ft. Four elements were used for the mesh. Tributary width was not used because for these solvers, force is directly specified, as opposed to specifying pressure and tributary width. Therefore a ramp-down load was used with a specific value of force or force per length. The duration of the forcing load was 17.8 ms. Every boundary condition and load type were tested, with 2% damping in the first two modes, as well as no damping. Peak loads were also varied for each combination to allow the output to vary significantly. Axial loading was also incorporated into the verifications, as proportions of the compressive capacity of the section. These values were 0, 0.2 and 0.4. A case was also performed with additional mass, equal to ten times the self-weight.

An example of how the two analyses were compared is shown on the following pages in Figures 4.4.1a and 4.4.1b. A 40k point load was used here as the initial value of the ramp-down load function.

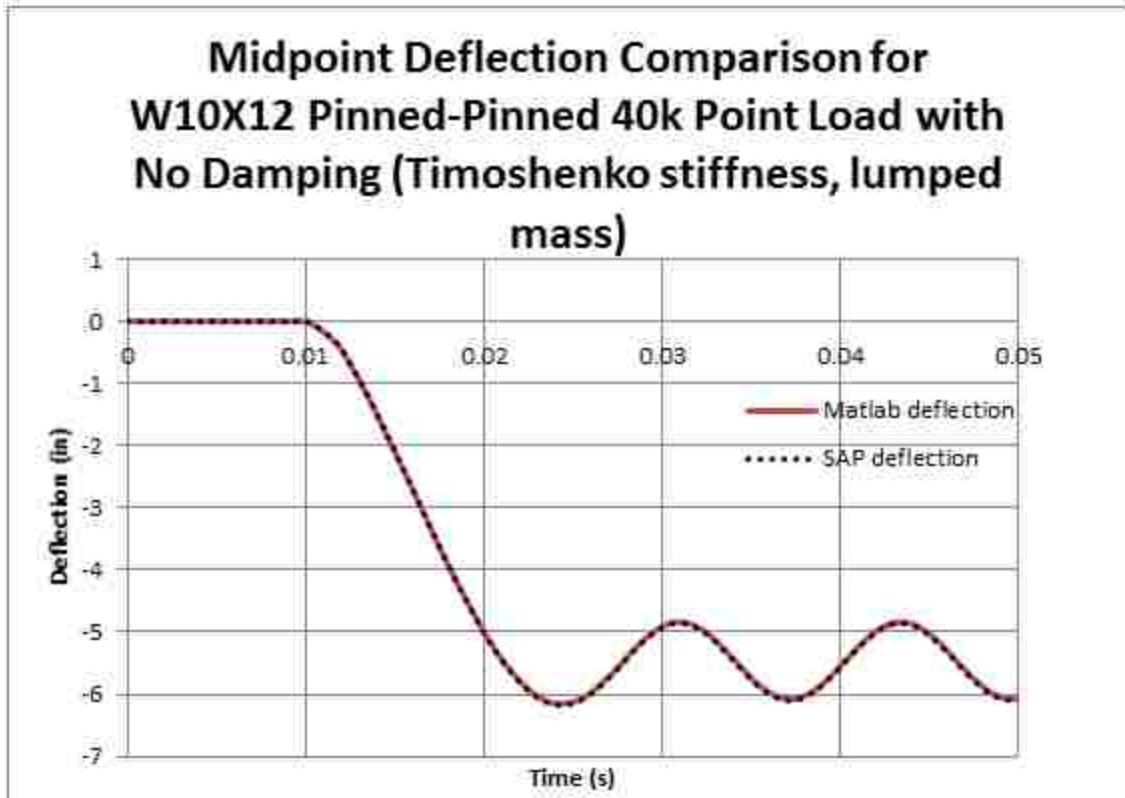


Figure 4.4.1a: Midpoint Deflection Comparison of SAP2000 with MDOF MATLAB Program

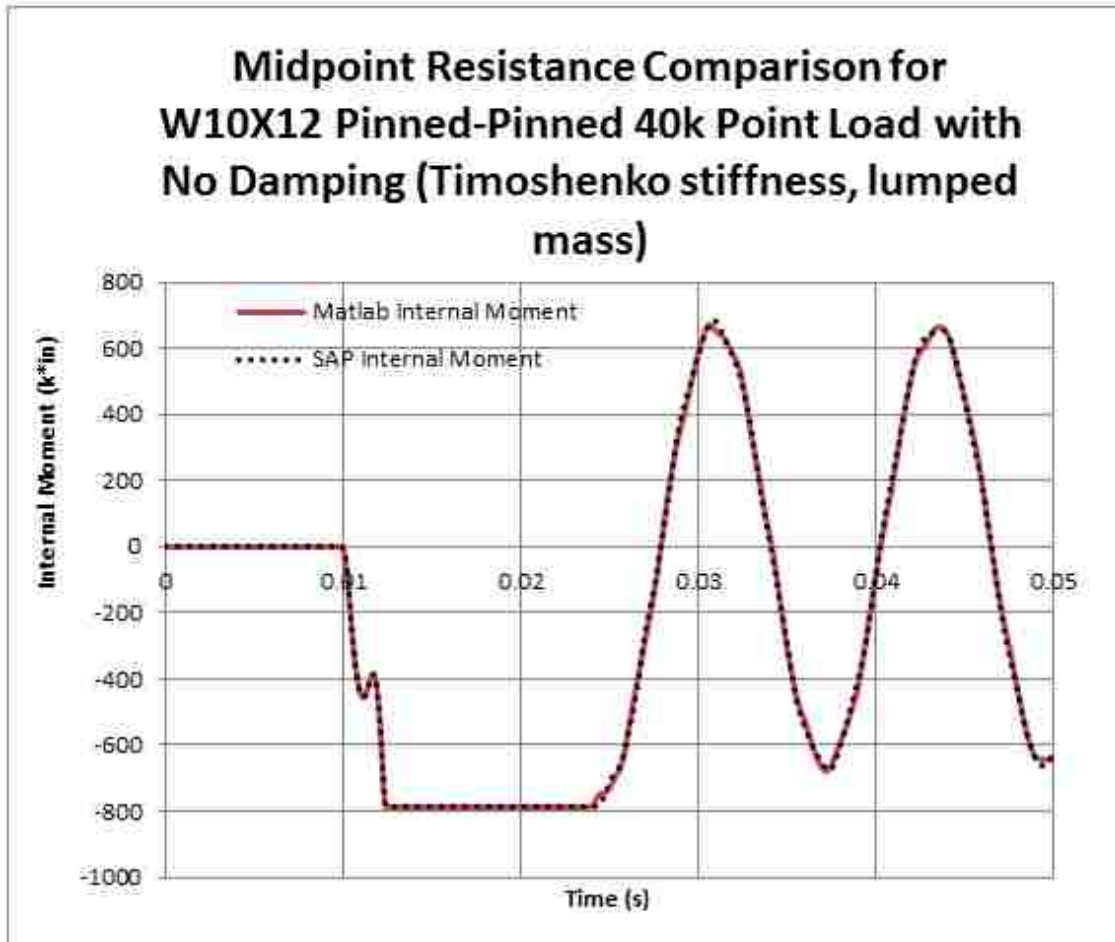


Figure 4.4.1b: Midpoint Resistance Comparison of SAP2000 with MDOF MATLAB Program

The output from SAP2000 and the MATLAB program was pasted into a new Excel document for comparison. Plots were created for deflection and resistance (internal moment) at midpoint for each set of parameters. Even though the maximum deflection does not occur at the midpoint for fixed-pinned boundary conditions, the midpoint was still used as the comparison for convenience. For cantilevered boundary conditions, the deflection was measured at the free end, and the internal resistance was measured at the support end. Since the plots of both sets of outputs overlapped

well, the MATLAB replica was verified as being a good representation of what SAP2000 would output. For the full set of verifications, see Appendix D: MDOF Solver Verifications (Time-History Plots).

4.4.1 Inconsistencies

While comparing each set of parameters to SAP2000, some discrepancies were noticed between the two solutions. This could be the result of modeling choices; however, these discrepancies are minor and do not cause the MATLAB solution to deviate enough from the SAP2000 solution to cause significant differences in outputs.

With some of the test comparisons, the maximum deflections at midspan varied, and the resistance curves were not identical. However, the deflections from the MATLAB solution never deviated from the SAP2000 deflections by more than 3%. Therefore, the MATLAB solution was accurate enough to be substituted for SAP2000 for the analyses. This discrepancy in deflection and resistance could be attributed to several minor differences in modeling choices between SAP2000 and the MATLAB replica:

1. When an element in SAP2000 undergoes yielding in the time-history analysis, the time step of yielding is broken up into smaller increments. This allows the solution find more accurately when the yielding occurs, since the yielding occurs within a time step. The MATLAB replica does not increase the time step of yielding; instead, if the resistance of an element

surpasses the plastic capacity, its corresponding resistance vector (all internal forces in the beam), are scaled back until the highest value of resistance is the plastic capacity. This is a more inaccurate method, however since all analyses are performed with very small time increments to begin with (from the nature of a blast load), the difference in the two modeling choices is negligible.

2. Another difference in the two solutions is how the damping matrix is reduced during yielding. As explained previously, SAP2000 reduces any stiffness-proportional damping matrix that corresponds to a degree of freedom undergoing yielding. Once a rotational degree of freedom yields, the stiffness components of the damping matrices corresponding to elements that share this yielded global degree of freedom are reduced by a factor between 10^{-2} and 10^{-3} . By implementing this strategy into the MATLAB replica, deflections were significantly larger than the deflections in SAP2000. To better represent SAP2000's output, more damping was required during yielding to reduce the deflections. By only reducing the rows and columns of the stiffness-proportional damping matrices associated with yielded global degrees of freedom (by a factor of 10^{-3}), the MATLAB replica matched the SAP2000 output within 3% error (compared to as much as 300% error before). All verification cases with damping use

this method of row/column-only reduction, to get as close to SAP2000 as possible.

3. The final differences between the models are related to adding axial load to the beam, creating a beam-column. The first of these differences is related to the reduction in the plastic moment capacity of a section due to axial load. There are several resources that can be used for calculating this reduction in plastic moment capacity. Some of these include Equation 5-4 from *Prestandard and Commentary for the Seismic Rehabilitation of Buildings* (FEMA 356) and Equations H-1a and H1-1b from *Steel Construction Manual Fourteenth Edition* (AISC). Since the goal is to compare SBEDS to other models, it makes the most sense for the model being compared to SBEDS to have the same reduced plastic capacity calculation to stay consistent. In the SBEDS Methodology Manual, the reduced plastic capacity calculation is outlined. Figure 4.4.1.1 on the following page is taken from the SBEDS manual:

Equation 6-9

$$M_{du1} = M_m \left(1 - \frac{P}{P_{cr}} \right) \left[\frac{1}{C_m} \left(1 - \frac{P}{P_e} \right) \right] \quad \text{where } \frac{1}{C_m} \left(1 - \frac{P}{P_e} \right) = 1 \text{ in SBEDS}$$

$$M_{du2} = 1.18 M_{du} \left(1 - \frac{P}{P_y} \right) \leq M_{du}$$

$$M_{du} = \text{MIN}(M_{du1}, M_{du2})$$

Note: $M_{duA} = M_{duA1}$ for metal studs

Equation 6-10

$$P_{cr} = 0.658^{\lambda_c^2} f_{dy} A \quad \text{if } \lambda_c \leq 1.5 \quad \text{where } \lambda_c = \frac{KL}{r\pi} \sqrt{\frac{f_{dy}}{E}}$$

$$P_{cr} = \frac{0.877}{\lambda_c^2} f_{dy} A \quad \text{if } \lambda_c > 1.5$$

where:

M_{duA} = ultimate dynamic moment capacity accounting for axial compressive load

P = applied axial compression load including input static load and peak dynamic axial load

P_e = Euler buckling load

P_{cr} = axial load capacity accounting for column slenderness (see Equation 6-10)

P_y = axial load capacity for short column based on beam-column cross section area = $f_{dy}A$

A = cross sectional area

M_m = moment capacity accounting for bracing to compression flange in weak axis direction (see Equation 6-6)

M_{du} = moment capacity accounting for fully braced component = $f_{dy}Z$ (see

Equation 6-4)

C_m = coefficient applied to bending term in moment-axial load interaction formula and dependent upon column curvature caused by applied moments (see AISC Design Specification)

E = Young's modulus

KL/r = larger slenderness ratio considering strong and weak bending axes where L = unbraced length, K = effective length factor based on boundary conditions, and r = radius of gyration about each axis. K assumed = 1.0 in weak bending axis. Critical KL/r is calculated for inbound response and also used for rebound response in SBEDS.

Figure 4.4.1.1 Equations for Reduced Plastic Moment Capacity in SBEDS (US Army, 2008)

By simply entering a yield stress for the material in SAP2000, and then performing the time-history analysis with applied axial load, the yielding occurred at an internal moment value different than expected. This value varied significantly from the value that SBEDS uses for the same set of inputs. Hand calculations were performed to identify the plastic moment from Equation 5-4 from *Prestandard and Commentary for the Seismic Rehabilitation of Buildings* (FEMA 356) and Equations H-1a and H1-1b from *Steel Construction Manual Fourteenth Edition* (AISC), but neither value matched the value from SAP2000. Unable to determine how SAP2000 calculates reduced plastic capacity, a different approach was taken. A different yield stress was manually inputted into the SAP2000 material until the plastic moment capacity matched that of SBEDS. This is how all of the verification tests for the MATLAB replica were performed.

After the plastic moment capacities matched, there was still a discrepancy between SAP2000 and the MATLAB replica. This difference arose from including geometric effects, which would implement the geometric stiffness matrices in the solution. In the SAP2000 solution, the internal axial force in the beam-column fluctuated. Since the axial force has an effect on the geometric stiffness matrix, this was changing the geometric stiffness matrix at every time step. In the MATLAB replica, the axial load is applied statically before the time-history analysis, and then this axial load is held constant through the time stepping, and therefore so does the geometric stiffness matrix. In SAP2000, a static analysis with the axial load was also applied to the system before the time-history analysis. However, during the time-history analysis, the axial

degrees of freedom continued to move, causing this change in axial internal force. SBEDS holds the axial load constant through the time-history analysis, so this was the modeling choice chosen for the MATLAB solution as well, even though SAP2000 was unable to keep the axial load constant through the time-history analysis.

This difference in modeling choice is most likely causing the discrepancy between SAP2000's and the MATLAB replica's internal moment resistance plots. Once the plastic capacity is reached, the MATLAB solution stays at this value until rebound. However, the SAP2000 solution oscillates around the plastic capacity. This is shown below in Figure 4.4.1.2a:

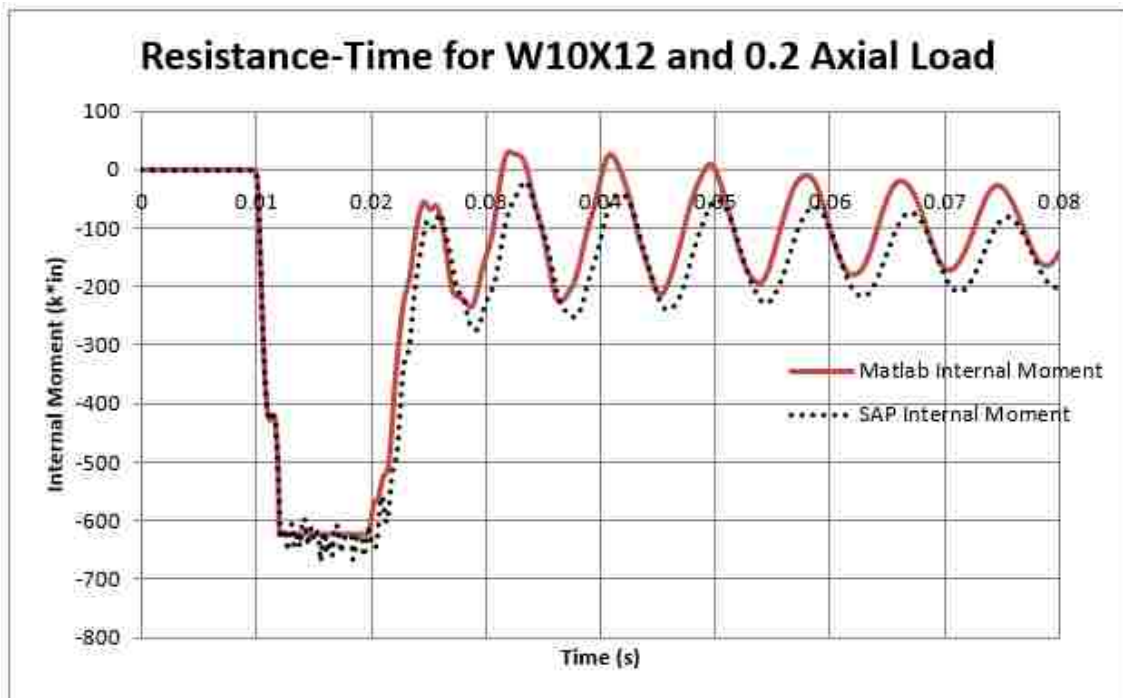


Figure 4.4.1.2a: Resistance-Time Plot for W10X12 Fixed-Pinned with Point Load, 2% Damping, and 0.2 Axial Load

The fact that the resistance does not flat-line during yielding changes the rest of the solution. However, this discrepancy does not result in significant deflection deviation, as shown below in Figure 4.4.1.2b:

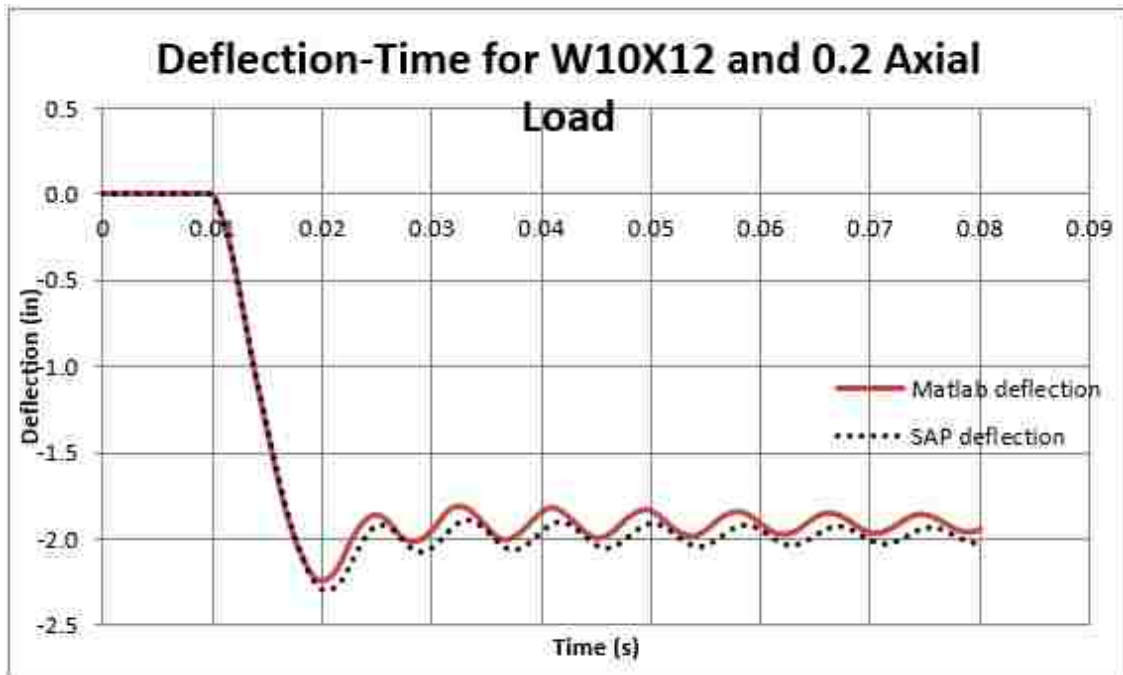


Figure 4.4.1.2b: Deflection-Time Plot for W10X12 Fixed-Pinned with Point Load, 2% Damping, and 0.2 Axial Load

The discrepancy in the resistance plot has little effect on the deflection. For this particular case, there is 2.5% deviation in the deflection from the SAP2000 solution, which is acceptable. Also, by varying the yield stress in SAP2000, the deviation can be further reduced.

All of the verifications thus far were performed with lumped mass matrices. SAP2000 is unable to implement a consistent-mass matrix. However, after all these verifications were performed, a final comparison was performed. This analysis held everything constant (W10X12 pinned-pinned with 40k point load), and only varied the mass matrix between a lumped mass matrix and a consistent-mass matrix. The comparisons to SBEDS are performed with consistent-mass matrices, so a comparison of the lumped mass solution to the consistent-mass matrix solution provides a verification that the consistent-mass matrix is properly defined, even though the consistent-mass matrix exceeds the capabilities of SAP2000. Figures 4.4.1.3a and 4.4.1.3b on the following pages show these final comparisons:

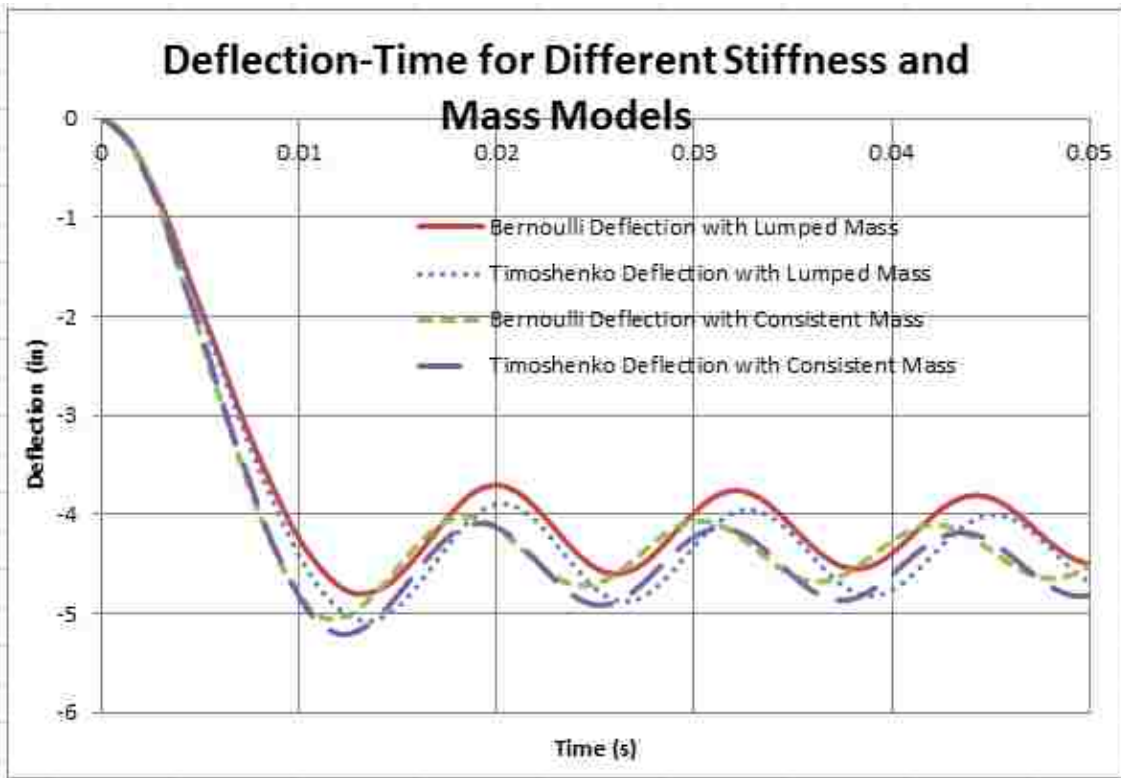


Figure 4.4.1.3a: Deflection-Time Comparison for Different Stiffness and Mass Models

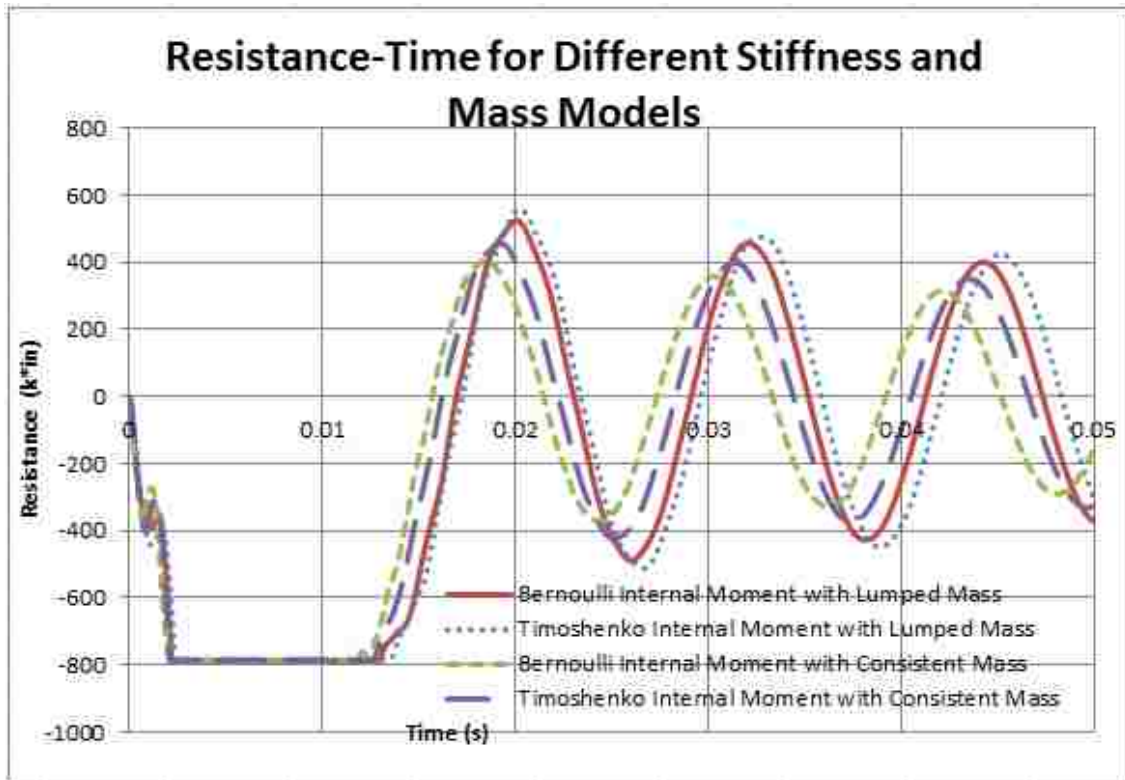


Figure 4.4.1.3b: Resistance-Time Comparison for Different Stiffness and Mass Models

The figures above show that by using a consistent-mass matrix, the solution changes slightly; however the implementation of the consistent-mass matrix is shown as not significantly affecting the solution because the overall resistance-time and deflection-time plots remain very similar. It is expected that the period should change slightly when changing from a lumped mass matrix to a consistent-mass matrix, since the period is dependent upon the stiffness and mass distribution.

Chapter 5 Comparisons

5.1 General

This chapter discusses the different input parameters that were varied for all of the analyses, and the corresponding output of each case. This includes incorporating the two created MATLAB codes into one overall code and which output parameters were saved into the data frames for each analysis.

In section 8.1.2 of Yokoyama's thesis, he introduces a stiffness factor that compares the overall deflected shape of SDOF solution to the MDOF solution, for a specified time step. This "K_L factor" takes the effective stiffness of the MDOF solution and divides by the SDOF stiffness. When the deflected shape of the MDOF solution matches that of the SDOF solution, this factor will be near the load transformation factor explained in section 2.6.1 earlier. Deviation of this "K_L factor" from the load transformation factor means that the two solutions develop different overall deflected shapes (Yokoyama, 2011).

The parameters varied for the analyses were charge of TNT (lb), standoff distance from charge to structure (ft), axial load as proportion of axial capacity of the element, boundary conditions, damping ratio, section shape, and added weight (psf). Other input parameters were held constant for all three data sets. Table 5.1.1 on the following page summarizes the constant parameters:

Parameter	Length of Element	Loading	Tributary Width	Elastic Modulus	Yield Stress	Mesh of Element	Type of Element and Mass
Value	15 ft	Uniform	15 ft	29,000 ksi	50 ksi	16 elements	consistent-mass
Comment	Standard for Column	More applicable to blast	Conservative for spacing	Steel	Steel	Less than 1 ft per element	Most accurate

Table 5.1.1: Constant Input Parameters

Additionally, a static and dynamic increase factor is applied to the yield stress, as SBEDS uses. These values vary from 1.00 to 1.05 (static) and 1.19 to 1.24 (dynamic), as explained in the following sections. Whenever a Bernoulli element is specified, the corresponding consistent-mass matrix for a Bernoulli element is used. Whenever a Timoshenko element is specified, the corresponding consistent-mass matrix for a Timoshenko element is used.

5.2 Results and Analysis

The following sections contain selections from the full sets of analyses performed, with comments on the selected plots.

5.2.1 First Data Set

A first set of analyses were performed, with large increments in each varied parameter, to get a sense of the regions of discrepancy between SDOF and MDOF analysis. Charges from 10 lb to 2000 lb were paired with standoffs from 3 ft to 250 ft.

However, only combinations which resulted in a scaled distance between 0.25 and 10 $m/kg^{1/3}$ were included. A summary table of the combinations of charges and standoffs is shown below:

Standoff (ft)	3	5	10	25	50	100	250
Charges (lb)	10	10	10	10	10		
	25	25	25	25	25		
	50	50	50	50	52		
		200	200	200	200	200	
		500	500	500	500	500	
			2000	2000	2000	2000	2000

Table 5.1.2: Charges and Standoffs for Data Set 1

Axial loads were 0, 0.2, and 0.4 proportion of compressive axial capacity. Boundary conditions were pinned-pinned, fixed-pinned, and fixed-fixed. Damping was set to 2%, and in the first two modes for the MDOF analysis. Section shapes included were W14X48, W14X53, W14X61, W14X82, W14X109, W14X132, W14X145, and W14X257. The added weights used were 0 psf, 15 psf, 40 psf, and 81.25 psf. These values represent no weight, glazing, brick, and concrete wall. The static and dynamic increase factors were set to 1.05 and 1.19, respectively.

For this first set, 4 sets of plots were created for each beam section. All graphs are plotted against scaled distance, Z. The first set of plots displays the deviation in the K_L factor using Bernoulli elements from the appropriate load factor for the SDOF solution. The time step used for comparison was the time step in which first yield occurred. No

axial load is incorporated here because the SDOF solution stiffness does not take into account axial load, so the plots would not be comparable. Additionally, Timoshenko elements are not included in these plots because the overall shape for a Timoshenko beam will always be different than the overall shape for a SDOF model. Two examples of this set of plots are shown below:

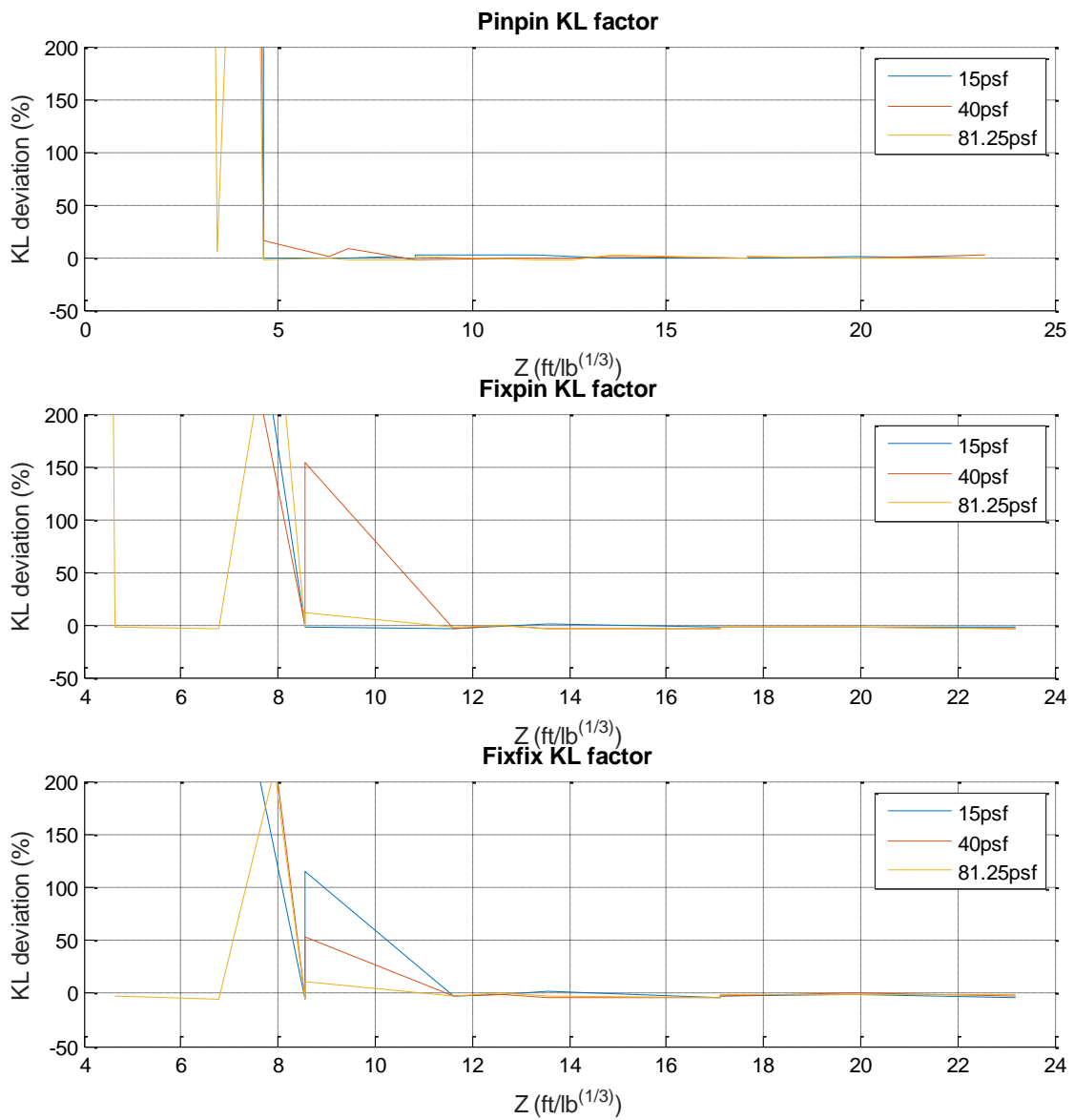


Figure 5.2.1.1: Deviation in K_L vs. Z for W14X48

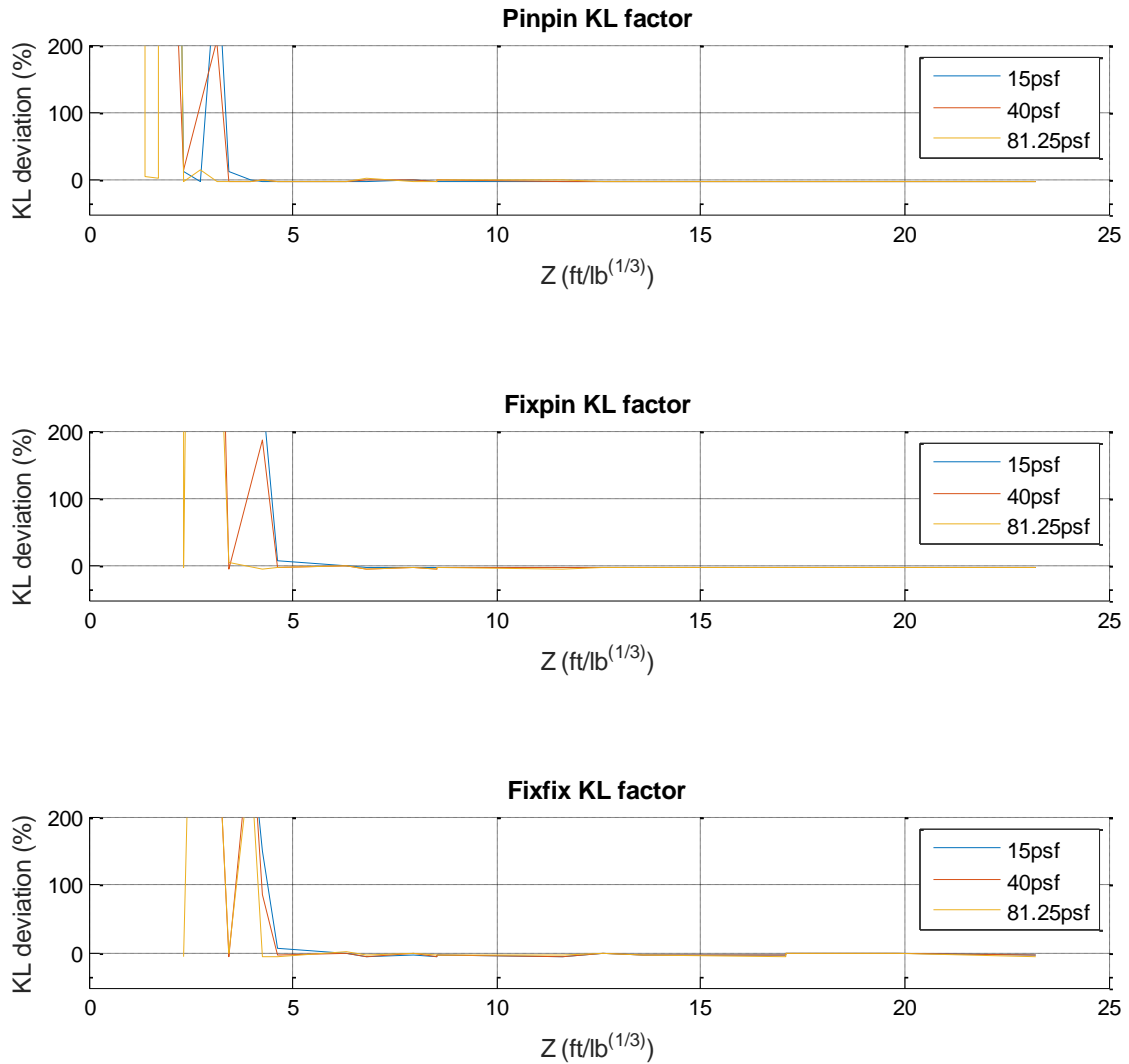


Figure 5.2.1.2: Deviation in K_L vs. Z for W14X257

Depending on the boundary condition, the Z at which the MDOF solution begins to deviate from the SDOF solution will change. Additionally, the beam section size changes this deviation point as well, since adding stiffness will reduce the potential for yielding under the same load. Since the Z values had such large increments, the deviation appears to grow instantaneously. Additionally, the solution may break down

at very low Z values near $2 \text{ ft/lb}^{1/3}$, but a limit can clearly be seen. The deviation curves would be more gradual if finer increments were to be used. From this data, it can be seen that the supposed $3 \text{ ft/lb}^{1/3}$ limit does not necessarily apply universally for all analyses, and this limit is highly dependent upon the element's stiffness and boundary condition. For a pinned-pinned W14X257, this limit may be acceptable, but by decreasing the section size to a W14X48 and changing the boundary conditions to fixed-fixed, the limit is at roughly $12 \text{ ft/lb}^{1/3}$. From these plots, the added weight does not seem to have significant effect on changing the limit for a specific set of conditions.

The second set of plots displays the deviation in maximum deflection using Bernoulli elements from the SDOF maximum deflection. Axial load was implemented for these plots. An example of this set of plots is shown on the next page:

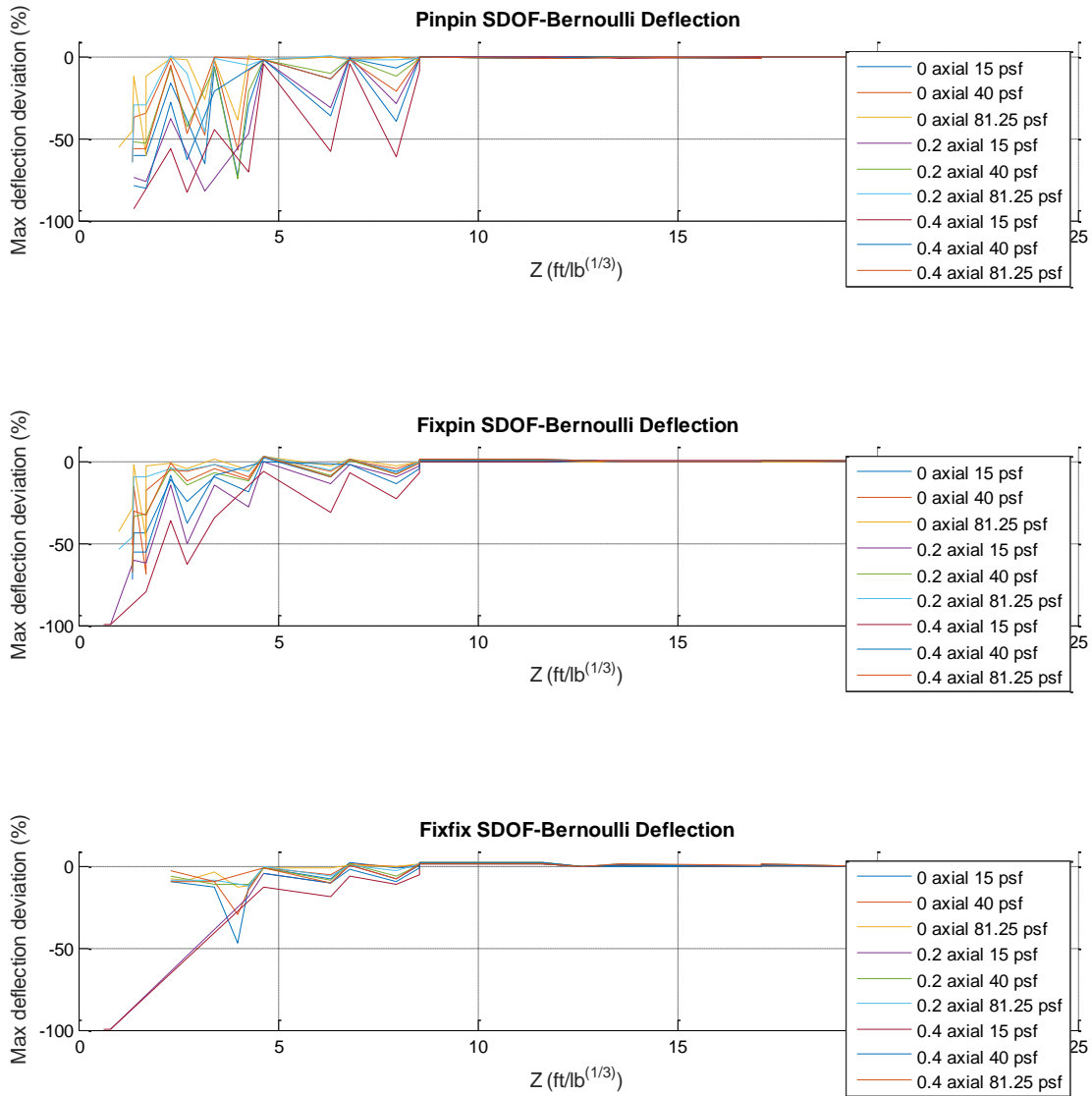


Figure 5.2.1.3: Deviation in Bernoulli-SDOF Maximum Deflection vs. Z for W14X257

In these plots, the significant deviations start to occur under $8.5 \text{ ft}/\text{lb}^{1/3}$. This is interesting because Figure 5.2.1.2 indicates consistent overall shape until $12 \text{ ft}/\text{lb}^{1/3}$. This shows that even though deflected shapes at first yield may be the same, their magnitudes can be different. This is still a deviation in solution. Adding axial load creates more deviation in solutions. This may be due to the fact that the SDOF

solution takes axial load and applies additional equivalent transverse load, while the MDOF solution decreases the stiffness of the element (using the geometric stiffness matrix). Since the deviation values become negative, the Bernoulli maximum deflection is smaller than the SDOF solution.

The third set of plots displays the deviation in maximum deflection using Timoshenko elements from the SDOF maximum deflection. An example of this set of plots is shown on the following page:

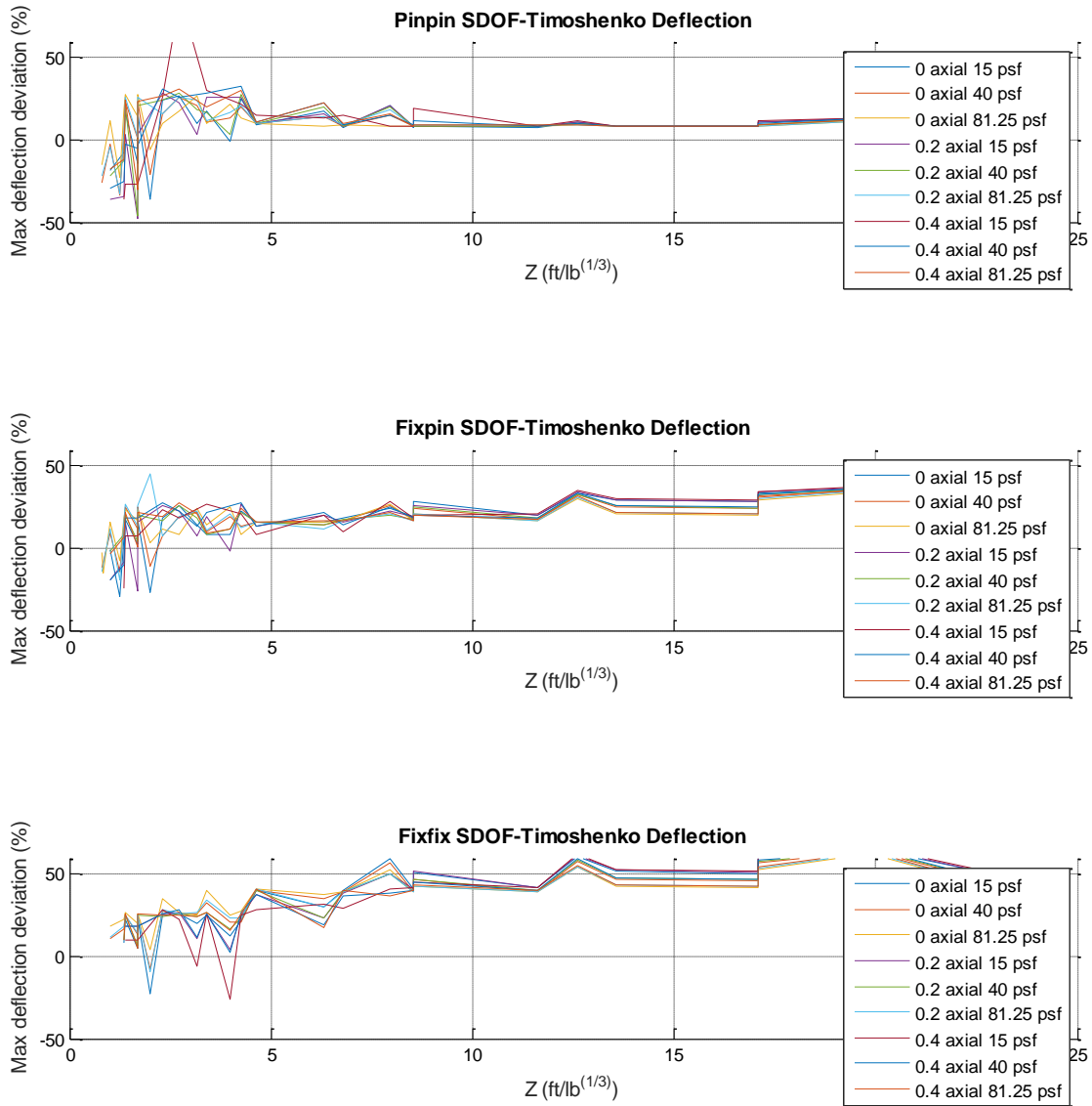


Figure 5.2.1.4: Deviation in Timoshenko-SDOF Maximum Deflection vs. Z for W14X257

The above plots show how different maximum deflections may be when shear deformation is included. For very high Z values, deviation is at 25% to 50%. Since the Bernoulli solution starts to undershoot the SDOF solution below a certain Z, and since Timoshenko elements allow more overall deformation of the system, the two facts almost negate each other. The absolute values of the deviations in Figure 5.2.1.4 are

smaller than those in Figure 5.2.1.3. However, the deviations in Figure 5.2.1.4 are still significant. A definitive limit on Z cannot be seen because the Timoshenko solution is always above or below the SDOF solution.

The final set of plots displays the deviation in maximum deflection using Timoshenko elements from maximum deflection using Bernoulli elements. An example of this set of plots is shown on the following page:

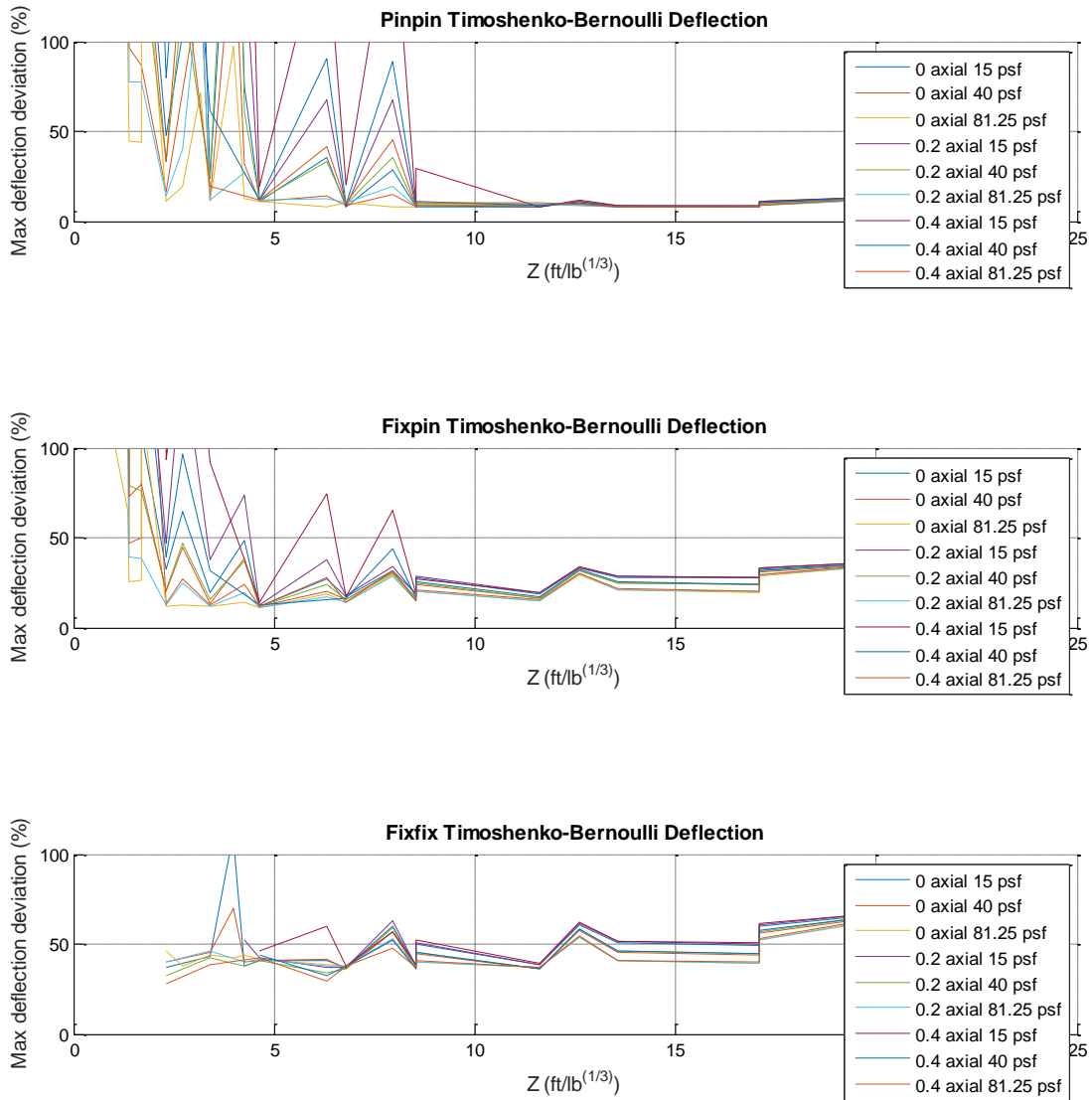


Figure 5.2.1.5: Deviation in Timoshenko-Bernoulli Maximum Deflection vs. Z for W14X257

This set of plots shows that the Timoshenko solution always produces a larger maximum deflection than the Bernoulli solution, since the deviation is always positive. Also, the deviations grow as Z decreases, especially for pinned-pinned and fixed-fixed boundary conditions. This makes sense because these boundary conditions are less stiff, so the inclusion of shear deformation has a larger effect for these conditions.

For this first data set, each curve held boundary conditions, added weight, and axial load constant. However, different combinations of charge and standoff were lumped together on this curve. The fact that these curves are not smooth at high Z values shows that using Z to identify a limit for SDOF analysis is not necessarily a viable option. Different combinations of charge and standoff that create a similar Z value have different deviations. This can be seen in Figure 5.2.1.5 for the fixed-pinned and fixed-fixed conditions, where there is an immediate jump in deviation at a Z of $17ft/lb^{1/3}$. For two different combinations of charge and standoff that create a Z of $17ft/lb^{1/3}$, the deviation in maximum deflection changes. This is why the plots in the first data set seem to oscillate (in addition to the large increment in Z for each data point).

The full set of plots for the first data set is located in Appendix E: First Data Set Plots.

5.2.2 Second Data Set

The second set focused on the fact that two different sets of charges and standoffs that create the same Z value can have different results. This data set used a W14X109 (mid-stiffness from the first data set) with 2% damping, and used the same modes for damping as the first data set for MDOF analysis. With a 25 lb charge and a 200 lb charge, standoffs were back-calculated to establish a range of Z values from 2.5 to 15

$ft/lb^{1/3}$ in smaller increments than the first data set, so that a finer solution set could be obtained. By separating the curves by charge and by creating a finer Z increment, the oscillations and jagged nature of the curves from the first data set are removed. The same combinations of boundary conditions, added weight, and axial loads were used as the first data set. The same deviations from the first data set were plotted against Z. An example of one of these plots is shown below:

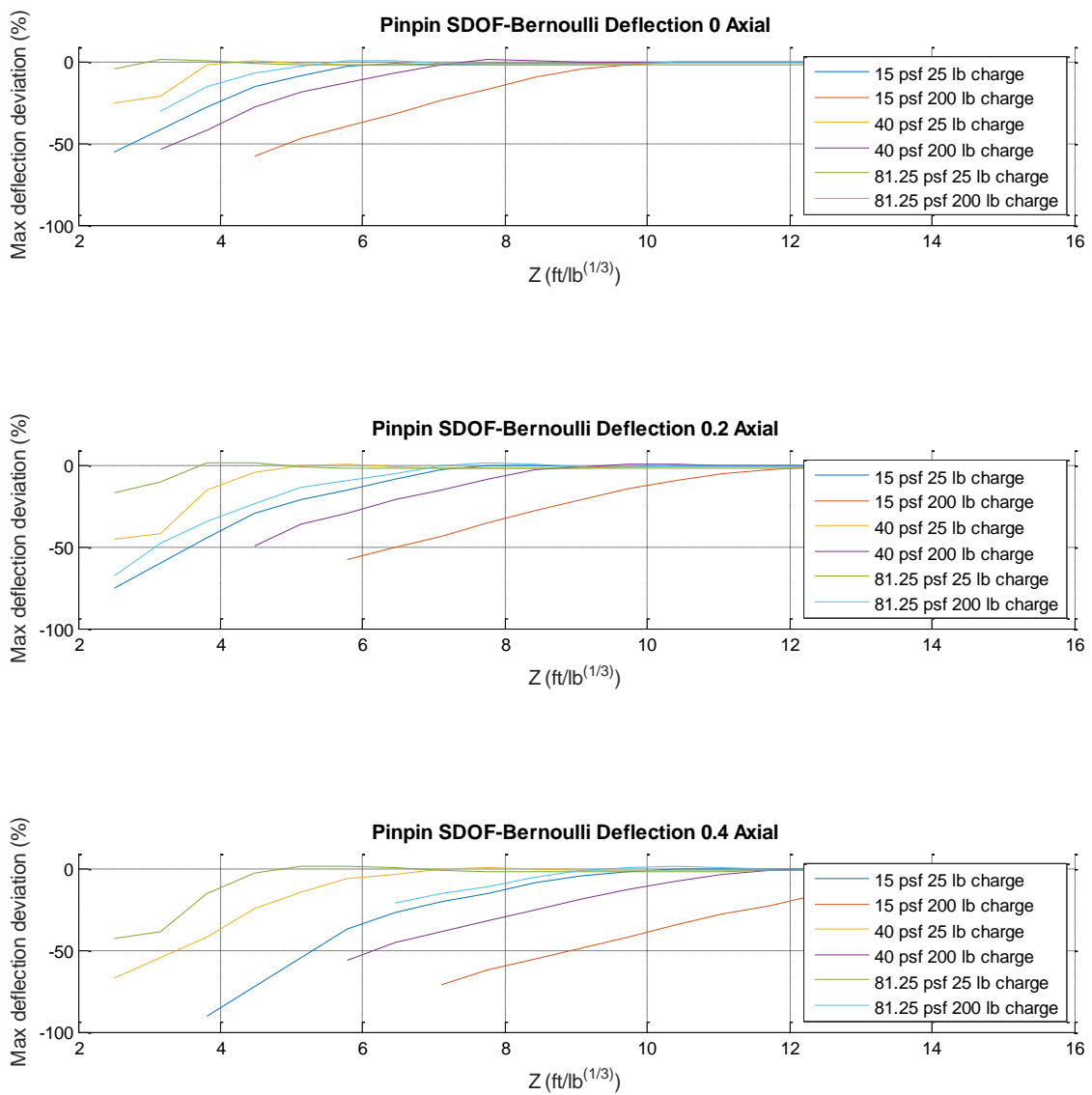


Figure 5.2.2.1: Deviation in Bernoulli-SDOF Maximum Deflection vs. Z for Pinned-Pinned W14X109 (Fine Z Increment)

Since the different charges are not in the same curve anymore, the plots are smoother and follow a more distinguishable path. The above plots show that for a pinned-pinned, W14X109 member, increasing the charge while also increasing Z, decreasing supported weight, and adding axial load all cause the Bernoulli MDOF solution to deviate from the SDOF solution at a higher Z value. The curve of various standoffs with no axial load, 81.25 psf added weight, and a 25 lb charge never significantly deviates from the SDOF solution, even at Z values lower than the accepted limit of $3ft/lb^{1/3}$. Meanwhile, the curve with 0.4 axial load, 15 psf added weight, and a 200 lb charge shows 20% deviation from the SDOF solution at a Z of $12ft/lb^{1/3}$. These two curves represent the extremes for the pinned-pinned condition, but they illustrate the fact that significant deviations start to develop at very different Z values, depending on the conditions.

The figure on the following page shows the same plots as Figure 5.2.2.1, but using Timoshenko elements instead of Bernoulli elements, to see the effect of shear deformation.

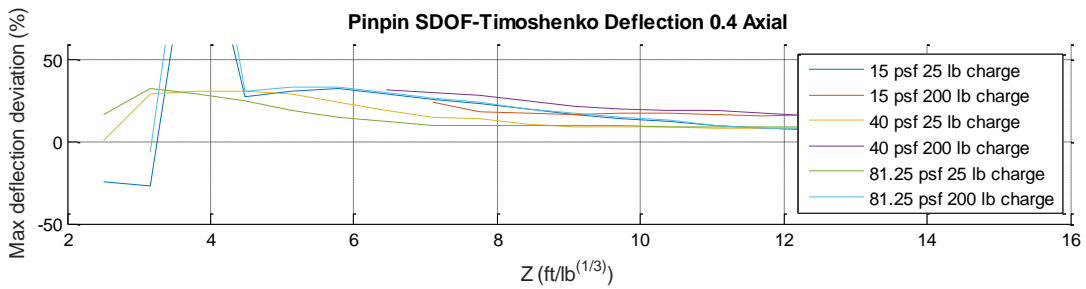
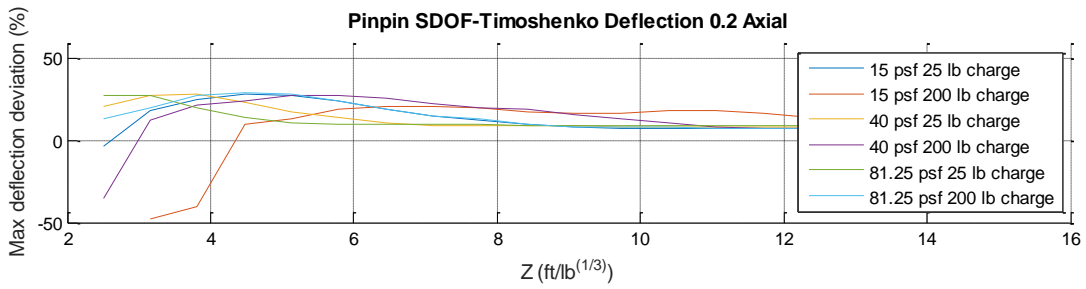
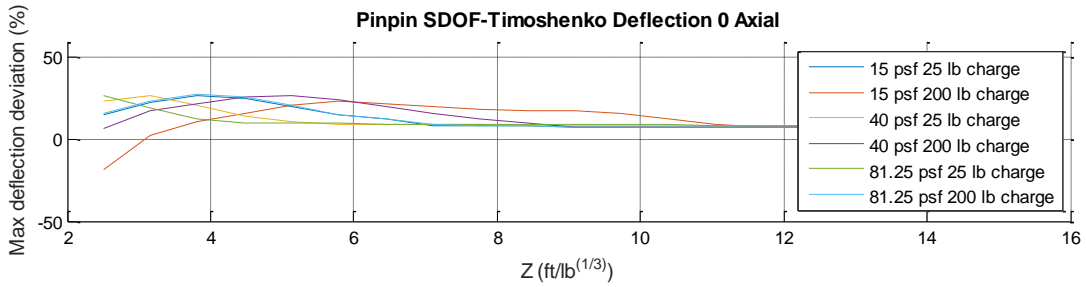


Figure 5.2.2.2: Deviation in Timoshenko-SDOF Maximum Deflection vs. Z for Pinned-Pinned W14X109 (Fine Z Increment)

As stated previously in the first data set, the fact that the Bernoulli MDOF solution undershoots the SDOF solution, and the fact that the Timoshenko MDOF solution allows more deformation than the Bernoulli MDOF solution, tends to negate one another. For the plot with 0 axial load, the Timoshenko solution overestimates the maximum deflection of the SDOF solution by roughly 8%, until a Z of approximately

$11.5ft/lb^{1/3}$. At this point, only the curve for 15 psf added weight and 200 lb charge starts to show increased deviation. By a Z of $9ft/lb^{1/3}$, the curve for 40 psf added weight and 200 lb charge starts to deviate more as well. However, all these deviations stay under 30%, and by very low Z values, the curves tend to stop increasing and start to decrease – this trend was also shown in the Bernoulli plots.

Due to all these factors, especially since different charges exhibit larger deviations than other charges for the same scaled distance, it is difficult to attach a single value on the limit of Z for accuracy of SDOF analysis to blast loads.

The full set of plots for the second data set is located in Appendix F: Second Data Set Plots.

5.2.3 Third Data Set

Experimental blast testing was performed on various beams and beam-columns, which was published in “Experimental Performance of Steel Beams under Blast Loading” by Amr A. Nassr and others. The results from 5 blast shots are shown on the following page in Table 5.2.3.1. In the same paper, models were created to replicate the blast and results. In the paper, no static increase factor was used, and a dynamic increase factor of 1.24 seemed to match the experimental data the best (for holding the increase factor constant throughout the analysis). For this reason, the dynamic

increase factor of 1.24 was used for the third data set, even though this is larger than the conventional value.

Shot	Section designation	Charge mass (kg)	Stand-off distance (m)	Scaled distance (m/kg ^{1/3})	Axis of bending	Test beams
1	W150 × 24	50	10.30	2.80	x-x	1B1, 1B2, 1B3
2	W150 × 24	100	10.30	2.22	y-y	2B1, 2B2, 2B3
3	W150 × 24	150	9.00	1.69	x-x	3B1, 3B2, 3B3
4	W150 × 24	250	7.00	1.11	x-x	4B1, 4B2, 4B3
5	W200 × 71	250	9.50	1.51	x-x	5B1

Table 5.2.3.1: Experimental Blast Data (Nassr, 2012)

The third data set was the same as the second set, but with two changes. This third set re-calibrated the SDOF and MDOF models to better match the experimental data. For this reason, the static increase factor was set to 1 and the dynamic increase factor was set to 1.24 for data set three. The first mode was damped at 2%, and the second chosen mode to be damped by 2% for MDOF analysis was adjusted until the Timoshenko solution came closest to the experimental data. The Timoshenko solution is assumed to be the most accurate solution, so once the variables were calibrated using the Timoshenko solution, these calibrated variables were applied to the Bernoulli and SDOF solutions, where applicable. Only simply supported data was available. For the 16 element model in use, damping in the first and 48th modes resulted in a maximum deflection closest to the experimental data. This represented the last mode of the system, and therefore fixed-pinned and fixed-fixed conditions had damping applied to their first and last modes as well. A summary of the adjustments and results of the adjustments are displayed on the next page in Table 5.2.3.2:

	Measured Maximum Deflection (mm)	Timoshenko Maximum Deflection (mm)	Second Mode Set to 2% Damping	SIF,DIF
Shot 1	6.9	7.1	48 th (Last)	1.00, 1.24
Shot 3	33.2	32.7	46 th	1.00, 1.24
Shot 5	62.8	68.2	30 th	1.00, 1.24

Table 5.2.3.2: Timoshenko Solution Calibrations to Better Match Experimental Data

To best match experimental deflection from each shot, the second mode to be damped at 2% changed for each shot. Therefore, the last mode was the chosen mode for data set three. It is difficult to say which chosen mode is most accurate because there may be other factors that are causing the discrepancy in deflection, such as the fact that the pressure gages along the length of the beam reported different pressure values for the same time step (the experimental load was non-uniform). Additionally, one of Nassr’s conclusions is that, “[u]sing constant dynamic increase factor to estimate the material strength because of strain rate might not provide a realistic assessment of the actual effect of the strain rate on the dynamic response of beams.” The models for this thesis implement a constant dynamic increase factor, so this is why calibrations were made to other variables in order to best match the experimental data.

Once the model was re-calibrated, this third data set was performed, and the same types of plots were created with this new data. The figure on the following page shows one of these plots:

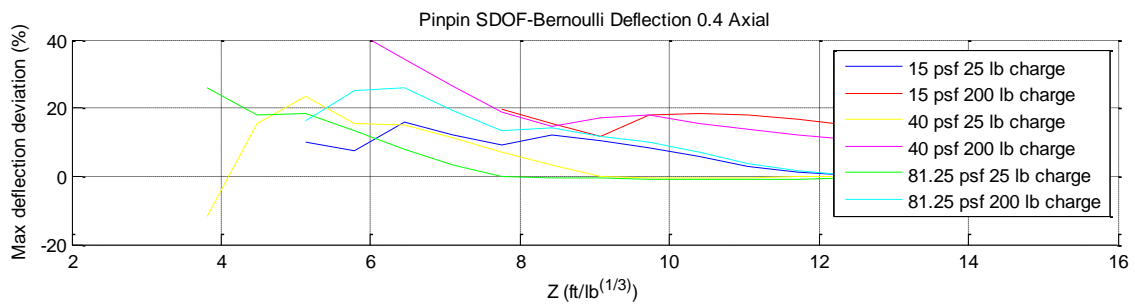
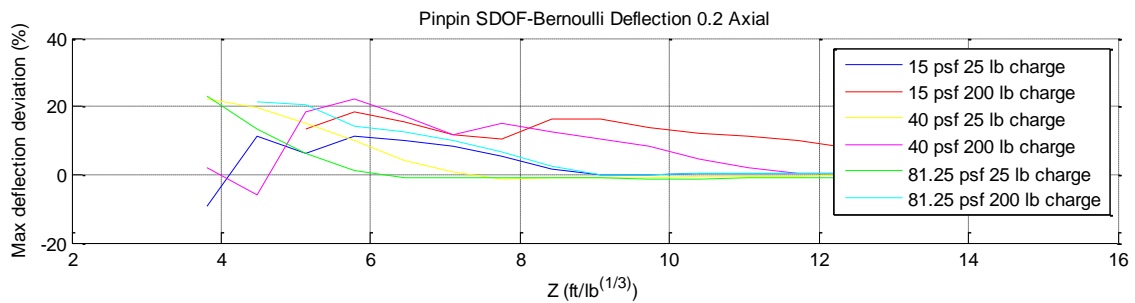
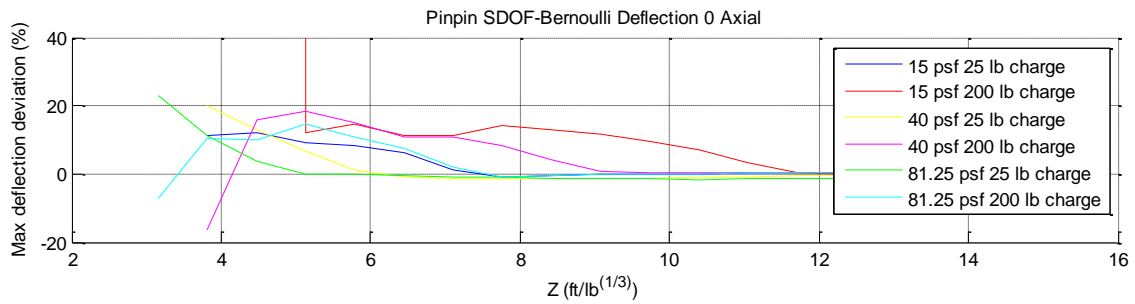


Figure 5.2.3.1: Deviation in Bernoulli-SDOF Maximum Deflection vs. Z for Pinned-Pinned W14X109 (Calibrated Damping)

From the second data set, these conditions led the Bernoulli solution to undershoot the SDOF solution. Altering the static and dynamic increase factors has little effect on the results, since these values are used in both the SDOF and MDOF solutions to define

when yielding in an element will occur. Additionally, using a static increase factor and dynamic increase factor of 1.05 and 1.24 respectively creates a net increase factor of 1.25. This third set of data uses a net increase factor of 1.24, which is very close to 1.25. The damping is the main difference between data set two and data set three. Now that damping for the MDOF model has been decreased, the Bernoulli solution overshoots the SDOF solution. This third set of data was calibrated using the Timoshenko solution, so the following plots are even more relevant:

In these plots, the MDOF solution overshoots the SDOF solution even more. The plots follow the same trends as every other set of plots mentioned, with larger deviations depending on the set of conditions used. One again, this supports the fact that it is not accurate to use a single value of Z to distinguish when SDOF analysis is applicable.

The full set of plots for the third data set is located in Appendix G: Third Data Set Plots.

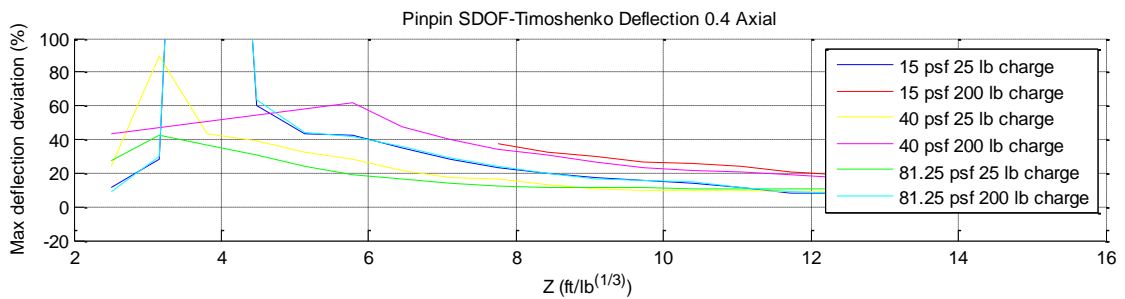
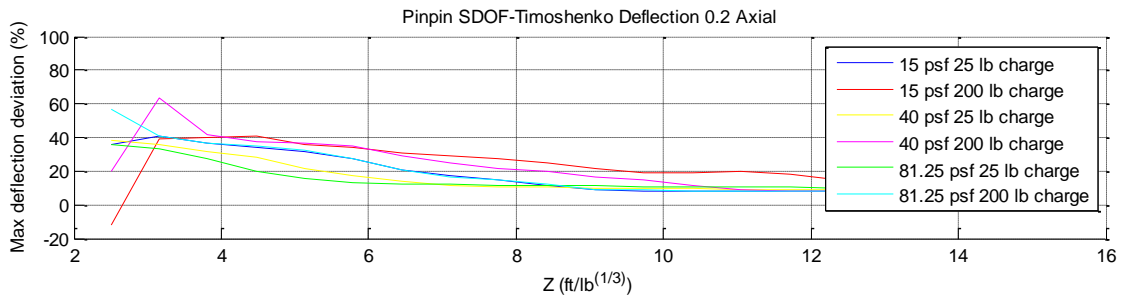
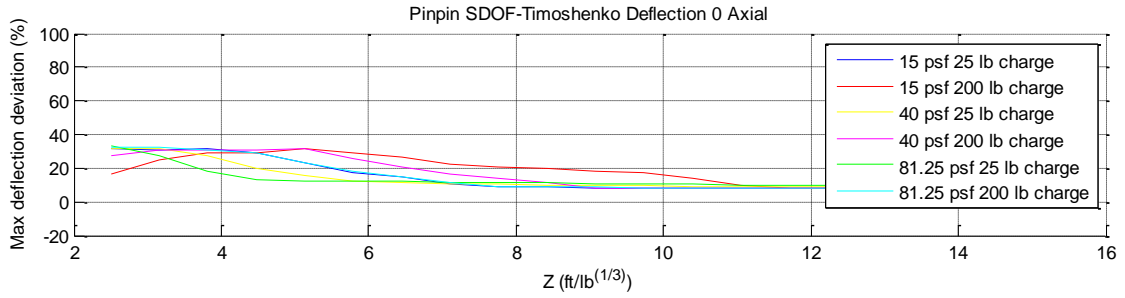


Figure 5.2.3.2: Deviation in Timoshenko-SDOF Maximum Deflection vs. Z for Pinned-Pinned W14X109 (Calibrated Damping)

5.2.4 Individual Deflected Shapes

This fourth section consists of individual chosen analyses where the MDOF solution deviated significantly from the SDOF solution. Included is the same set of inputs which Yokoyama (2011) identified as causing significant deviation in the MDOF deflected

shape from the SDOF deflected shape. Plots here display the SDOF, Bernoulli MDOF, and Timoshenko MDOF deflected shapes of the element at first yield. Input parameters stayed consistent with parameters from the third data set, such as dynamic increase factors and damping options.

The first individual case investigates a pinned-pinned W14X109 beam with a 200 lb charge, 15 psf added weight, and no axial load. Cases were taken from the first plots in Figures 5.2.3.1 and 5.2.3.2, using Z values of approximately 12.4, 9.1, and $7.7 \text{ ft}/\text{lb}^{1/3}$. In Figure 5.2.3.1, these cases encompass little deviation to large deviation in maximum deflection for the Bernoulli elements.

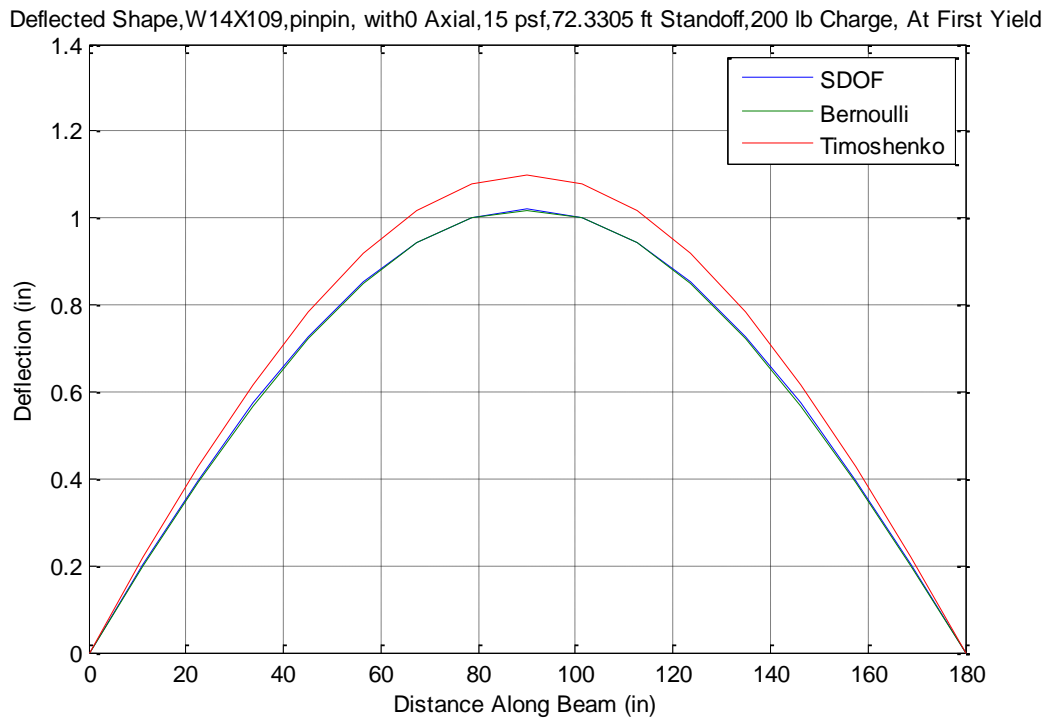


Figure 5.2.4.1: Deflected Shape at First Yield for Pinned-Pinned W14X109 with $Z=12.4 \text{ ft}/\text{lb}^{1/3}$

Deflected Shape, W14X109, pinpin, with 0 Axial, 15 psf, 53.0937 ft Standoff, 200 lb Charge, At First Yield

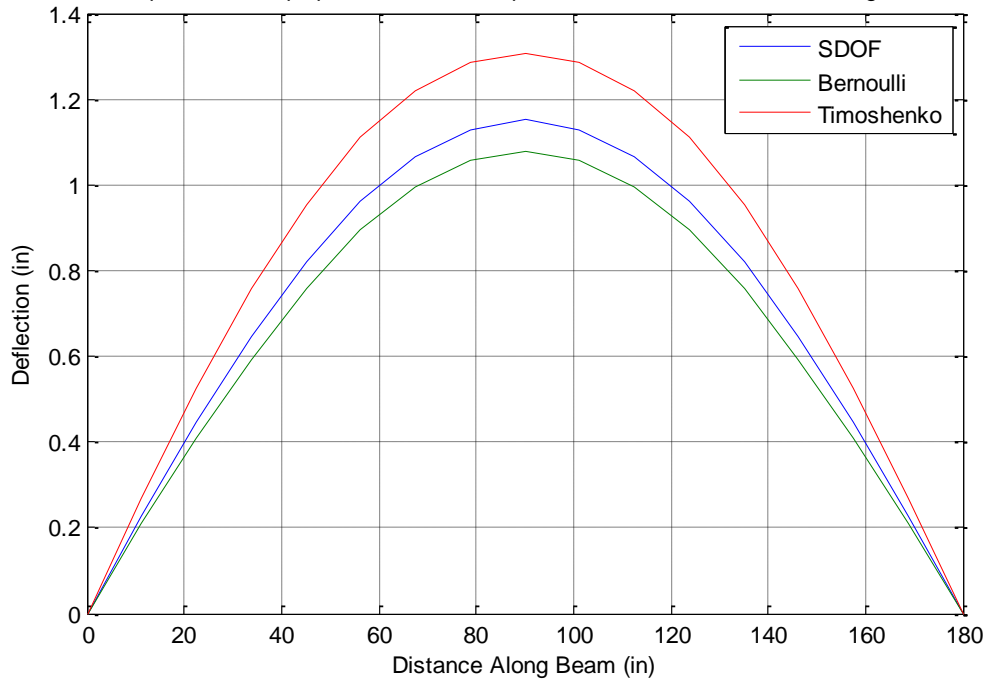


Figure 5.2.4.2: Deflected Shape at First Yield for Pinned-Pinned W14X109 with $Z=9.1 \text{ ft}/\text{lb}^{1/3}$

Deflected Shape, W14X109, pinpin, with 0 Axial, 15 psf, 45 ft Standoff, 200 lb Charge, At First Yield

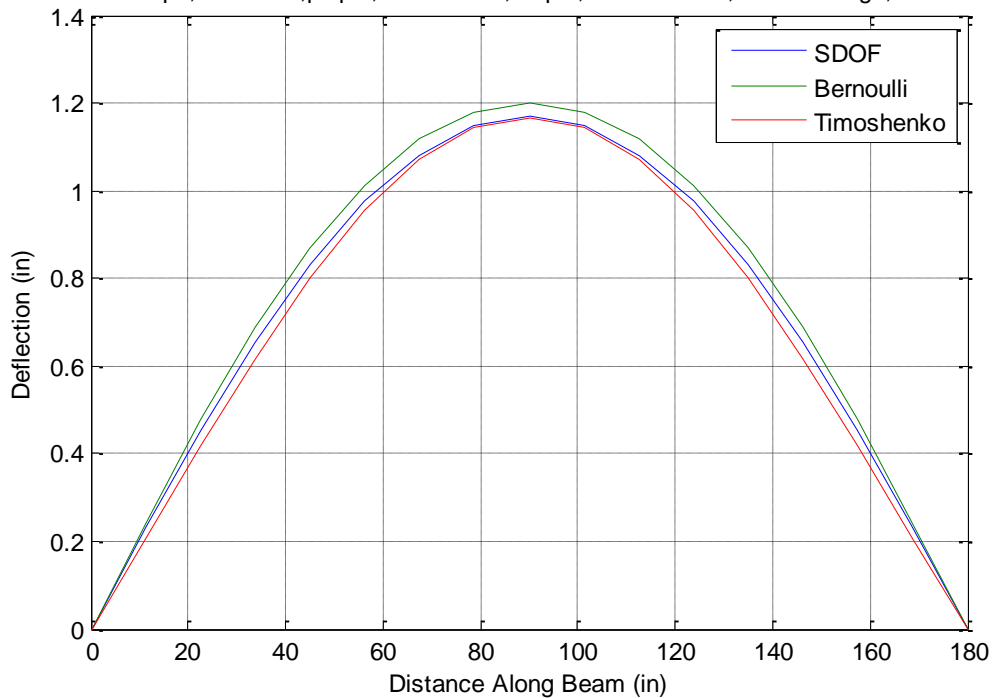


Figure 5.2.4.3: Deflected Shape at First Yield for Pinned-Pinned W14X109 with $Z=7.7 \text{ ft}/\text{lb}^{1/3}$

Figure 5.2.4.1 shows little deviation in maximum deflection for the Bernoulli elements. This is consistent with how the deflected shape looks at first yield; the Bernoulli solution fits almost exactly on the SDOF solution. The standoff and charge combination here must not be strong enough to cause the MDOF solution to deviate the maximum deflection significantly. Both the Bernoulli and SDOF solutions yield at a midspan deflection of 1.02 inches.

By decreasing the standoff, the solutions deviate from one another, as shown in Figure 5.2.4.2. Now, the Bernoulli solution yields at 1.08 inches, while the SDOF solution yields at 1.15 inches. Because the SDOF solution assumes a certain deflected shape through the whole analysis, the only reason for an increase in deflection at first yield would be the duration and magnitude of the load versus the time step. By decreasing the duration and/or increasing the magnitude of the load, the change in resistance will be larger for every time step, and the time step under first yield will overshoot the theoretical deflection at first yield, depending on the size of the time step. However, for a MDOF analysis, since each node can deflect independently, a change in the deflection at first yield can be caused by the relative deflection between nodes, which may vary different combinations of duration and magnitude of load. In addition, the MDOF deflection at first yield is dependent upon the time step for the same reason as the SDOF solution. From Figure 5.2.3.1, this case with a Z of $9.1 \text{ ft/lb}^{1/3}$ shows a

deviation of 11.5%. However, the deviation at first yield is -6.1%. This means that during the time steps past yield, the Bernoulli solution deflects enough to change its relative deviation from -6.1% to 11.5% compared to the SDOF solution. This shows that most of the deviation comes from the time steps post-yield.

Decreasing the standoff even further, the deflected shapes change relative to each other again, as shown in Figure 5.2.4.3. Here, the Bernoulli deflection at first yield is 1.20 inches, while the SDOF deflection at first yield is 1.17 inches. This is a 2.6% deviation. From Figure 5.2.3.1, this case with a Z of $7.7 \text{ ft}/\text{lb}^{1/3}$ shows a deviation of 14%. Once again, this shows that the deviation in maximum deflection for the Bernoulli solution is primarily due to the time steps post-yield.

The second individual case investigates a fixed-fixed W14X109 beam with a 200 lb charge, 15 psf added weight, and no axial load. Cases were taken from equivalent first plots in Figures 5.2.3.1 and 5.2.3.2 for a fixed-fixed condition, using Z values of approximately 11.0, 9.5, and $5.8 \text{ ft}/\text{lb}^{1/3}$ (see Appendix G: Third Data Set Plots for these plots for a fixed-fixed condition). These cases encompass little deviation to large deviation in maximum deflection for the Bernoulli elements.

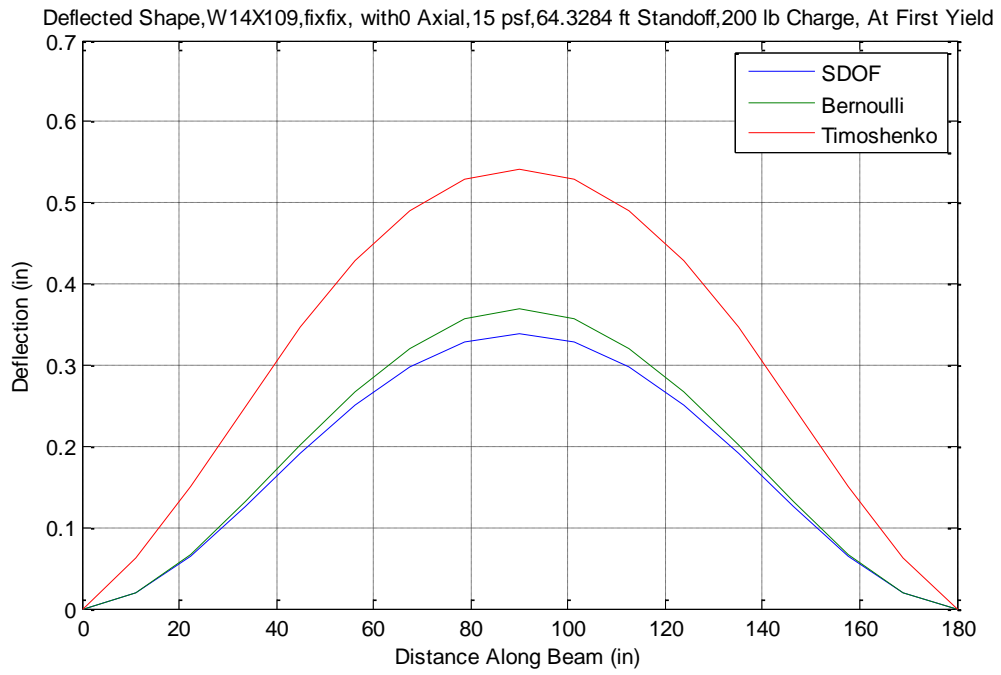


Figure 5.2.4.4: Deflected Shape at First Yield for Fixed-Fixed W14X109 with $Z=11.0 \text{ ft} / \text{lb}^{1/3}$

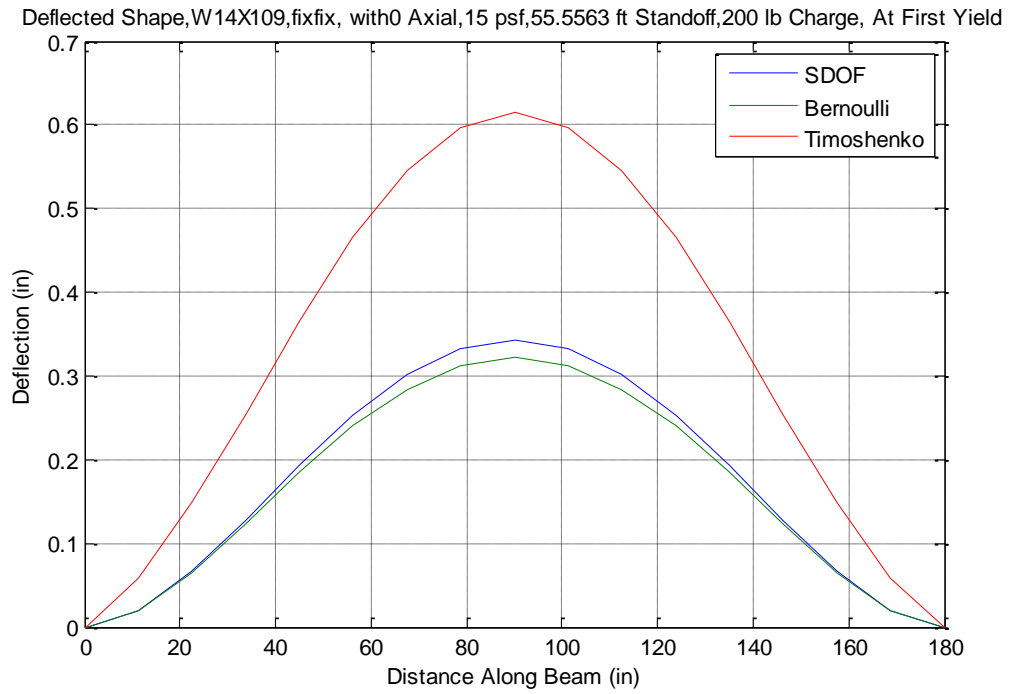


Figure 5.2.4.5: Deflected Shape at First Yield for Fixed-Fixed W14X109 with $Z=9.5 \text{ ft} / \text{lb}^{1/3}$

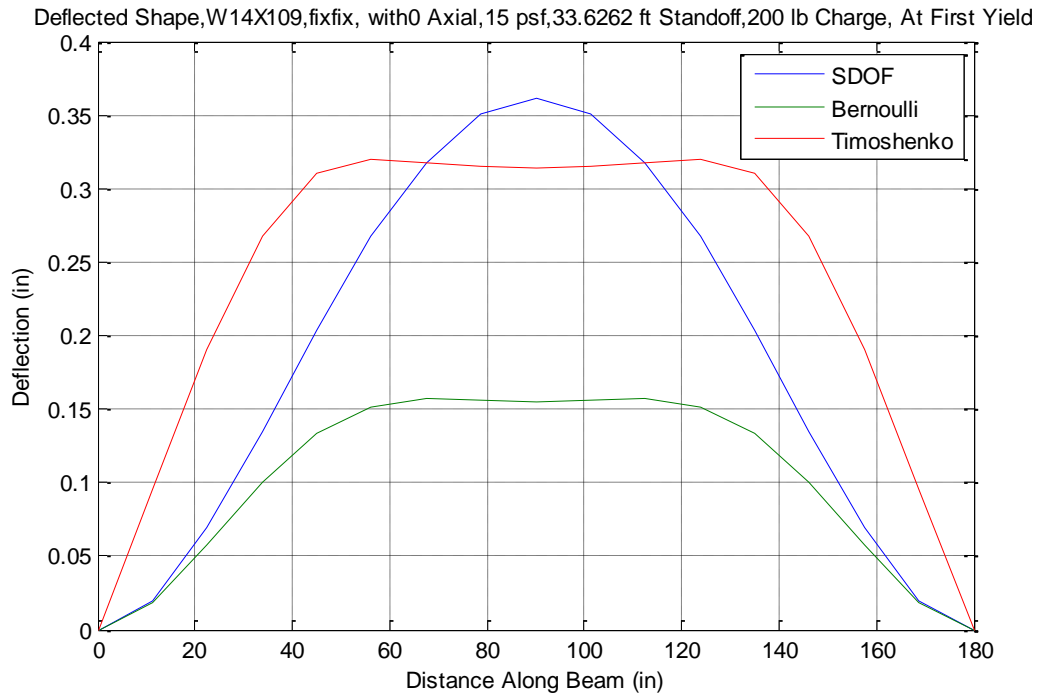


Figure 5.2.4.6: Deflected Shape at First Yield for Fixed-Fixed W14X109 with $Z=5.8 \text{ ft}/\text{lb}^{1/3}$

The same remarks from Figures 5.2.4.1 through 5.2.4.3 can be applied to Figures 5.2.4.4 and 5.2.4.5. However, as Z is decreased for fixed-fixed boundary conditions, the deflected shapes at first yield for the MDOF solutions become drastically different than for the SDOF solution. Since the SDOF solution assumes a deflected shape, only the magnitude of the overall shape may change at first yield. However, the MDOF solutions may have any shape, especially as the number of elements used increases. From Figure 5.2.4.6, the MDOF solutions yield in locations different than at midspan. The beam will yield where the relative displacement between consecutive nodes is large enough to cause a resistance equal to the yield moment. This can occur at any

node. For this set of conditions, this happens to occur at approximately quarter-span (and three-quarter-span). The combination of boundary conditions, load, and duration of load causes the beam to deflect in a way where the quarter-span (and three-quarter-span) experiences the largest relative deflection. If the shapes at first yield look this dissimilar and the majority of the deviation comes from the time steps post-yield, then these solutions will be vastly different, since they start the post-yield time step at vastly different shapes already.

The last individual case is a replica of Yokoyama's case (2011). The inputs for this case consist of a fixed-fixed W18X65 beam with no added weight and no axial load. The length and tributary width are 30 ft and 10 ft, respectively. Yokoyama directly inputted a peak pressure and impulse, instead of obtaining these from a certain charge and standoff. Yokoyama used a peak pressure of 100 psi and an impulse of 100 psi-msec. The case here has peak pressures of 45 and 50 psi, and impulses of 45 and 50 psi-msec. This is so the duration of the load stays the same for each pressure-impulse combination, and is consistent with the duration from Yokoyama's case. The following figures show the deflected shapes at first yield:

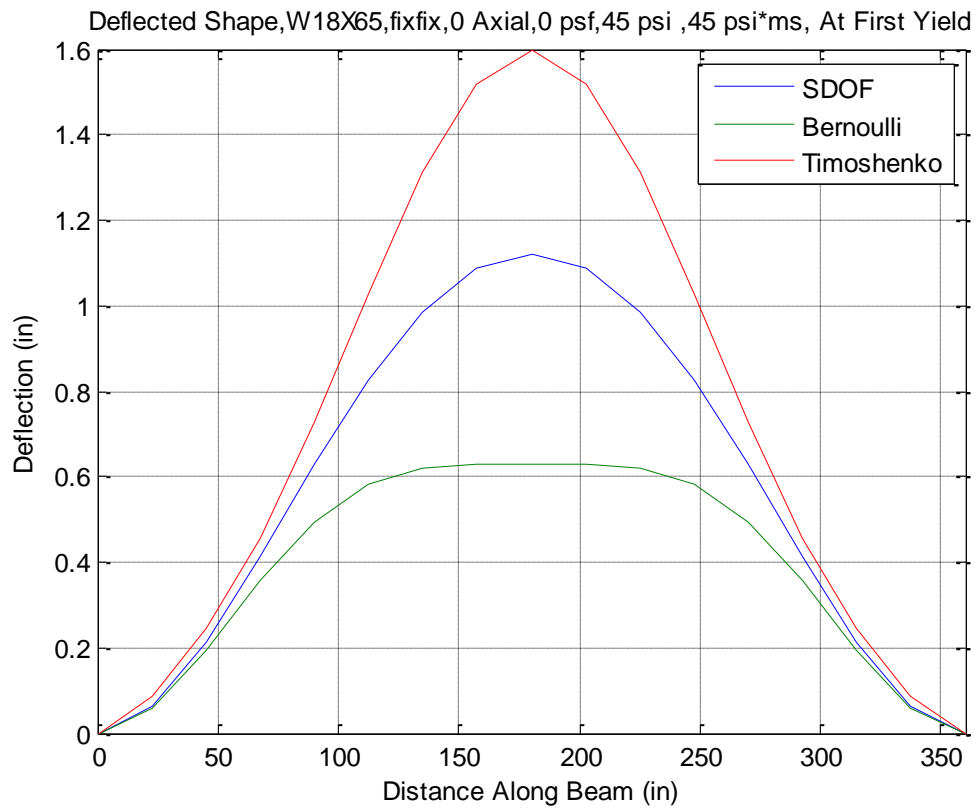


Figure 5.2.4.7: Deflected Shape at First Yield for Fixed-Fixed W18X65 with 45 psi and 45 psi-msec

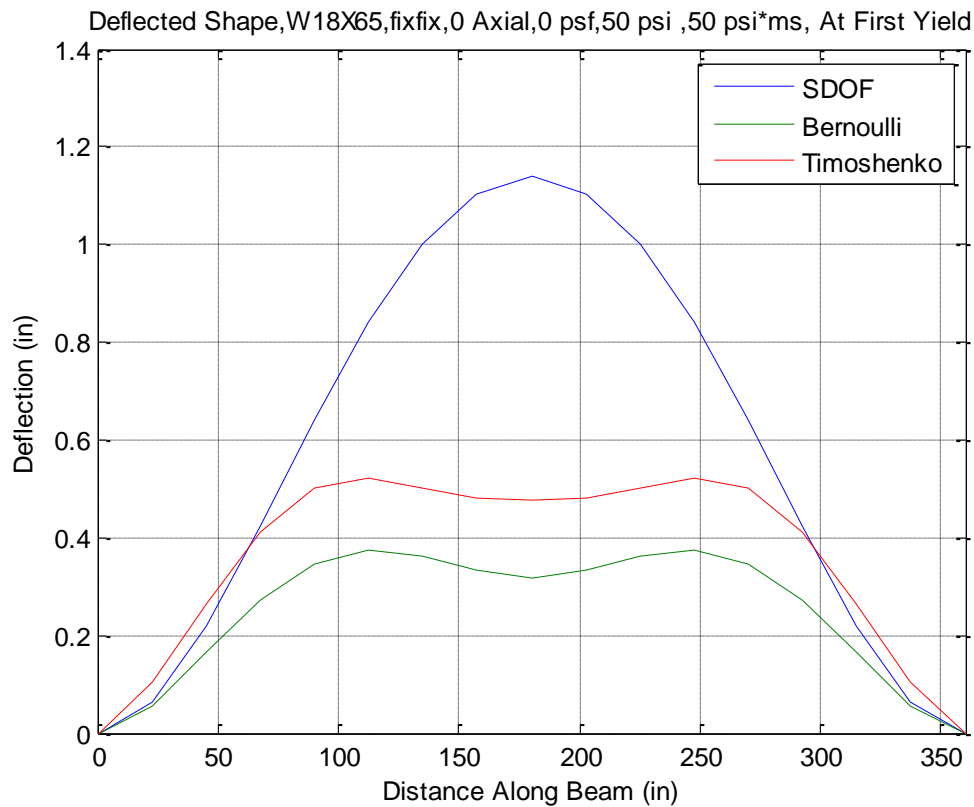


Figure 5.2.4.8: Deflected Shape at First Yield for Fixed-Fixed W18X65 with 50 psi and 50 psi-msec

In Yokoyama’s article, “SDOF Limits on Static Deflected Shape Assumptions,” he plots a deflected shape for his SDOF analysis and MDOF analysis, using Bernoulli elements with 100 psi and 100 psi-msec. The shapes from Figure 5.2.4.8 above are consistent with the results from Yokoyama.

The yielding of the element at locations different than midspan supports the fact that SDOF methodology may not be the best choice for analysis, under certain circumstances.

Chapter 6 Conclusions and Future Research

6.1 General

This final chapter discusses and summarizes the results from the research. Also included in this chapter are areas for future study.

6.2 Summary

Single-Degree-of-Freedom and Multi-Degree-of-Freedom analyses are two different methods for obtaining the time-history results (displacement, internal force, etc.) for a structural element under a dynamic load. They attempt to model reality by discretizing the continuous domains, time and space. MDOF analysis is generally accepted as a more precise representation of structural behavior than SDOF analysis. However, SDOF analysis is more computationally efficient, and is therefore more commonly used in practice.

Under blast loading, the structural element experiences load which is large in magnitude and short in duration. These fast-acting loads may cause deflected shapes that cannot be represented by SDOF analysis. SDOF analysis assumes a deflected shape for the element, which is based on its static deflected shape. These blast loads may cause the element to respond with a deflected shape very different than the static shape. This is where the discrepancy between SDOF and MDOF analysis arises.

Scaled distance (Z) is a measure of the intensity of a blast load. The conventional Z cutoff for which SDOF analysis is assumed to be no longer applicable is below $1.2 m/kg^{1/3}$ ($3ft/lb^{1/3}$). However, results from this research show from several sets of initial conditions that this cutoff is too low, and that using scaled distance may not even be an appropriate factor for determining a limit on SDOF analysis.

6.3 Conclusions

For this thesis, an absolute deviation larger than 15% is considered too large to still be considered accurate. Several figures presented in Chapter 5 that display deviation in maximum deflection reach deviation values larger than this 15% limit before Z decreases to $3ft/lb^{1/3}$.

The results from the analyses displayed in this research point towards the fact that establishing a simple boundary for when SDOF analysis becomes inaccurate is not as simple as stating a Z value. Results from Figures 5.2.3.1 and 5.2.3.2 for 15 psf added weight and a 200 lb charge show that the MDOF solution can deviate 15% to 20% (for Bernoulli and Timoshenko elements, respectively) from the SDOF solution at a Z of $9ft/lb^{1/3}$. This is three times the accepted limit. Using other sets of inputs, the deviation can be much smaller for a Z value down to $5ft/lb^{1/3}$ for 81.25 psf added weight and a 25 lb charge. Several factors affect how MDOF analysis may deviate from SDOF analysis. Different combinations of charges and standoffs may produce the

same Z value, but will have different values for deviation in maximum deflection. This fact alone is enough to prove that using Z as a limit is not appropriate. A better way may be to create charge-standoff surfaces. This will eliminate the issue of different combinations of charge and standoff that have the same Z leading to different amounts of deviation. However even this surface method may be too simple, since this neglects other factors that are important to consider. This includes strain rate effects, which will change the dynamic increase factor, and damping, which will change the results.

The effect of shear deformation and distributed mass are two factors that work against each other, but can produce very different results from SDOF analysis as well. Conventional SDOF analysis does not take into account the effects of shear deformation, but it may in fact be a significant proportion of the deformation of the element, which would increase the overall deflection. With the implementation of an MDOF system, the mass becomes distributed along the element, instead of lumped into one point, as is done in SDOF analysis. This distribution will tend to decrease the overall deflection of the element, due to the inertial forces being distributed to each degree of freedom. These forces balance with each stiffness force at each degree of freedom, which will decrease each degree of freedom's movement compared to an equivalent SDOF system. Although shear deformation and distributed mass work against each other, it is hard to tell by how much, and the result could be an overall

increase or decrease in maximum deflection compared to that of the SDOF system. For this reason, Z is too simple of a parameter to use as a limit for SDOF accuracy.

As load intensifies, SDOF and MDOF solutions for pinned-pinned conditions vary in magnitude. However, the shape of the deflected element remains fairly similar.

However, for fixed-fixed conditions, there is a point where the MDOF solution will yield at points along its length not predicted by static analysis. This causes the rest of the solution (post-yield) for MDOF analysis to be very different than for SDOF analysis.

This change in overall shape at first yield can occur at $5.8ft/lb^{1/3}$, which approximately twice the currently accepted limit.

6.4 Future Research

Several areas of the research included in this paper have room to be expanded upon and further developed. It is very imperative that the model being used be as accurate as possible. Therefore, more blast load testing needs to be recorded for further verification of the MDOF model. Damping and dynamic increase factors play a large role in the sensitivity of the model. Therefore, it is very important that these factors be set to values that most closely represent the testing results.

The MDOF model used for this research has limitations as well. A more accurate model can be used to achieve more accurate results. This model could account for

material nonlinearity, since the model for this research used an elastic-perfectly plastic material. Steel is to a certain extent elastic-perfectly plastic, but only if the strain in the steel does not get high enough to cause strain hardening. Much of the analyses in this research have large enough strains to cause strain hardening, yet this was neglected. However, since strain hardening is also neglected in SDOF analysis, the comparison is valid. Additionally, yielding was lumped at the nodes for MDOF analysis. In reality, there is a region of plasticity that spreads outward along the element from the point of initial yielding. Fiber elements may be used to better represent this plastic flow. Additionally, fiber elements would be able to distinguish between initial yielding of the outermost fiber and full yielding of the section, which would change the results of an analysis.

Only steel material was used in this research, for simplicity in analysis. The result of reinforced concrete elements under blast load is a topic that can be explored, as the resistance curves for these elements are different than those for steel. Considering failure modes due to bending about the strong axis only, reinforced concrete sections can fail in concrete crushing or steel reinforcement rupture. The stress-strain curve for concrete softens for most of the curve, whereas the stress-strain curve for steel is linear until a definitive point of yielding (see Figure 2.5.1). Due to the different material properties of concrete, reinforced concrete sections will have different responses than steel.

For MDOF analysis, geometric effects are accounted for in this research with the use of the geometric stiffness matrix. However, the co-rotational formulation could be implemented in the MDOF solution to obtain a more accurate representation of geometric effects, since this formulation takes into account the deformations from the chord of each element, instead of only taking into account deformations from a global coordinate system. The results of the MDOF solution with a co-rotational formulation may be more precise at predicting response due to blast loading.

Under initial axial loading, the moment capacity of the section will be reduced, due to initial stress within the element. There are different equations that model this decrease in capacity, or P-M interaction. Further investigation on the best P-M interaction to use would increase the accuracy of this research. However, SDOF and MDOF analysis used in this research use the same P-M interaction equation, so there is no inconsistency in solutions with regards to P-M interaction.

Lastly, the loading under blast analysis can be further investigated. The load used in this research is converted from a charge and standoff into a peak pressure and impulse with a ramp-down shape. This load is then applied to the element as a uniform load. However, depending on the standoff of the charge, the load may not be uniform along the length of the element at different time steps. However, results from this research are consistent between SDOF and MDOF analysis, since they both represent the load

as uniform. Additionally, negative-phase can be introduced to the forcing function, to better represent a real blast.

References

AISC (2011), *Steel Construction Manual Fourteenth Edition*. American Institute of Steel Construction.

ASCE/SEI 59-11 (2011), *Blast Protection of Buildings*. American Society of Civil Engineers.

Biggs, J. M. (1964), *Introduction to Structural Dynamics*. McGraw-Hill, Inc.

Chopra, A.K. (1995), *Dynamics of Structures*. Pearson Education, Inc.

FEMA 356 (2000), *Prestandard and Commentary for the Seismic Rehabilitation of Buildings*. American Society of Civil Engineers.

Friedlander, F. G. (1946), "The diffraction of sound pulses. I. Diffraction by a semi-infinite plate." *Proc. Roy. Soc. Lond. A*, pp. 186, 322-344.

Gavin, Henri P (2014), *Structural Element Stiffness, Mass, and Damping Matrices*. Department of Civil and Environmental Engineering, Duke University.

Logan, Daryl L (2012), "A First Course in the Finite Element Method." Cengage Learning.

Nassr, A.A. (2012), "Experimental Performance of Steel Beams under Blast Loading." *Journal of Performance of Constructed Facilities*. American Society of Civil Engineers. Vol. 26, Issue 5, pp. 600-619.

Przemieniecki, J. S. (1968), *Theory of Matrix Structural Analysis*. McGraw-Hill, Inc.

US Army Corps of Engineers (2008), *Methodology Manual for the Single-Degree-of-Freedom Blast Effects Design Spreadsheets (SBEDS)*. U.S. Army Corps of Engineers Protective Design Center Technical Report.

Yokoyama, T. (2014, November 4), "SDOF Limits on Static Deflected Shape Assumptions." Hinman Consulting Engineers Internal Technical Brief. Retrieved by request from tyokoyama@hce.com.

Yokoyama, T. (2014), "Limits to Deflected Shape Assumptions of the SDOF Methodology for Analyzing Structural Components Subject to Blast Loading." *Journal of Performance of Constructed Facilities*. American Society of Civil Engineers. Vol. 29, Issue 5.

Yokoyama, T. (2011), *Verification and Expansion of Single-Degree-of-Freedom Transformation Factors for Beams Using a Multi-Degree-of-Freedom Non-Linear Numerical Analysis Method*. Master of Science Thesis, Department of Architectural Engineering, California Polytech State University, San Luis Obispo, California.

Appendix A: MATLAB Code for SBEDS Replica (SDOF Solver)

Main Program (Executable)

```
clc
clear

%% For SBEDs Check (Enter the result of an SBEDS analysis to compute
error

y_yield_SBED=0;
y_yield2_SBED=0;
rotation_SBED=0;
ymax_SBED=0;
%duct_SBED=0;

%% inputs

del_t =.03; % time increment for Newmark (ms)

p_giv=14.6161; % input pressure of pulse (psi)

t_giv=15.2042; %length of pulse (ms)

bc='pinpin'; %set as pinpin, fixpin, fixfix, or cantil

load_type = 'uniform'; % set as uniform for uniform, pointd for point

zeta=0.02; %damping ratio of element (enter as decimal)

axial=0; %axial load in pounds

L=10; %length of beam (ft)

B=5; %tributary width that gets transferred to element (ft) if
distributed load

cross_area=3.54; %cross-sectional are (in^2)

A=B*L; %tributary area that gets transferred to element (ft^2) if
point load

E=29000; %modulus of elasticity (ksi)

fy=50; %yield strength (ksi)

stat=1.05; %static increase factor for yield strength

dyn=1.19; %dynamic increase factor for yield strength
```

```

I=53.8; %moment of inertia (inches^4)

Z = 12.6; %Plastic section modulus (in^3)

weight=12; %weight of element in (lb/ft)

weight_supported=0; %supported weight in psf

steps = 2000; % number of steps for Newmark

u_init=0; % initial displ
v_init=0; % initial vel (ft/s)

gam=1/2; %factor for Newmark
beta=1/4; %factor for Newmark (between 1/4 and 1/6)

%% unit conversions to s, lb, in

L=L*12;

B=B*12;

A=A*144;

weight=weight/12;

weight_supported=weight_supported/144;

%%

[t p load] = Load(t_giv,p_giv,del_t,steps,load_type,B,A); % creates
linearly decreasing pressure

g=0.0003864;

weight=weight*L/A+weight_supported;

m=weight/g; %convert weight/in to mass in psi with g in in/s^2

fy=fy*dyn*stat;

rg=sqrt(I/cross_area);

if bc=='cantil'
    klr_fact=2;

```

```

elseif bc=='pinpin'
    klr_fact=1;
elseif bc=='fixpin'
    klr_fact=.7;
elseif bc=='fixfix'
    klr_fact=.5;
end

Klr=klr_fact*L/rg;

Cc=Klr/pi*sqrt(fy/E);

if Cc<=1.5
    Pcr=(.658^(Cc^2))*fy*1000*cross_area;
else
    Pcr=.877/Cc^2*fy*1000*cross_area;
end

Mpset(1)=fy*1000*Z;

Mpset(2)=(1-axial/Pcr)*fy*1000*Z;

Mpset(3)=(1-axial/(fy*1000*cross_area))*1.18*Z*fy*1000;

Mp=min(Mpset);

Mp=Mp/1000;

axial=axial/B;

if load_type=='pointd' %for P-delta effects

    if bc=='cantil'
        k_fact=1/L^2;
        axial=axial*k_fact;
    else
        k_fact=4/L^2;
        axial=axial*k_fact;
    end

elseif load_type=='unifrm'

    if bc=='cantil'
        k_fact=2/L^2;
        axial=axial*k_fact;
    else
        k_fact=8/L^2;
        axial=axial*k_fact;
    end
end

```



```
end
```

```
[k_m k_F k R F m k_V_RF] = Biggs(L, bc, load_type, E, I, Mp, load, m,  
A); % creates Biggs Factors
```

```
%%
```

```
wn=sqrt(k/m); %natural freq
```

```
wd=wn*sqrt(1-zeta^2); %damped freq
```

```
c=zeta*2*m*wn; %damping coeff
```

```
[u, v, a, alldata, V] = const_vel(bc, load_type, u_init, v_init,  
m, c, k, k_F, k_m, R, t, F, zeta, wn, wd, steps, del_t, p, A, k_V_RF,  
axial);
```

```
%figure(1)  
%plot(t*1000,p*1000)  
%title('Load')  
%xlabel('time (ms)')  
%ylabel('Pressure (psi)')
```

```
%figure(2)  
%hold on  
%plot(t*1000,u,'r');  
%plot(t*1000,u_actual);  
%grid on  
%legend('Newmark', 'Actual');  
%title('Displacements')  
%xlabel('Time (ms)')  
%ylabel('Deflection (in)')  
%hold off
```

```
%figure(3)  
%plot(t*1000,u_residual);  
%title('Residual')  
%xlabel('Time (ms)')  
%ylabel('Deflection (in)')
```

```
ymax=max(u)  
Vmax=max(V)
```

```

%ymax=max(u_actual);

half=L/2;

if bc=='cantil'
    rotation=atan(ymax/L);

else
    rotation=atan(ymax/half);
end

rotation=rotation*180/pi;

R=R/A*1000; %converts R values to psi.

y_yield=R(1)/(k(1));

y_yield2=y_yield+(R(2)-R(1))/(k(2));

%if duct==1
%    ductility=ymax/y_yield;

%elseif duct==2
%    ductility=ymax/y_yield2;
%end

er(1)=Error(y_yield, y_yield_SBED); %first hinge displacement

sdf=1;

if bc=='pinpin'

    sdf=0;
end

if bc=='cantil'

    sdf=0;
end

if bc=='fixfix'
    if load_type=='pointd'

        sdf=0;
    end
end

```

```

if sdf==0
    er(2)=0;
else
    er(2)=Error(y_yield2, y_yield2_SBED); %2nd hinge displacement
end

er(4)=Error(rotation, rotation_SBED); % total rotation
er(3)=Error(y_max, y_max_SBED); %max displacement
%er(5)=Error(ductility, duct_SBED) %ductility

```

Load Function

```

function [t p load] = Load(t_giv,p_giv,del_t,steps,load_type,B,A)

t(1)=0;
p(1)=p_giv;

for i=2:1:steps

    if t(i-1)<t_giv-del_t

        t(i)=t(i-1)+del_t;
        p(i)=p(i-1)-p_giv/t_giv*del_t;

    else

        t(i)=t(i-1)+del_t;
        p(i)=0;

    end

end

if load_type == 'uniform'
    load=p.*B;
end

if load_type == 'pointd'
    load=p.*A;
end

end

```

Biggs Function

```
function [k_m k_F k R F m k_V_RF] = Biggs(L, bc, load_type, E, I, Mp,
load, m, A)
```

```
if bc == 'pinpin'
```

```
    if load_type == 'uniform'
```

```
        k_F=[.64 .64 .5];
        k_m=[.5 .5 .33];
        R=[0 8*Mp/L];
        k=[384*E*I/5/L^3 384*E*I/5/L^3 2*10^-6/1000*A]/A*1000;
        k_V_RF=[.39 .11; .39 .11; .38 .12];
```

```
    elseif load_type == 'pointd'
```

```
        k_F=[1 1 1];
        k_m=[.49 .49 .33];
        R=[0 4*Mp/L];
        k=[48*E*I/L^3 48*E*I/L^3 2*10^-6/1000*A]/A*1000;
        k_V_RF=[.78 -.28; .78 -.28; .75 -.25];
```

```
    end
```

```
elseif bc == 'fixpin'
```

```
    if load_type == 'uniform'
```

```
        k_F=[.58 .64 .5];
        k_m=[.45 .5 .33];
        R=[8*Mp/L 12*Mp/L];
        k=[185*E*I/L^3 384*E*I/5/L^3 2*10^-6/1000*A]/A*1000;
        k_V_RF=[.43 .19; .43 .19; .46 .12];
```

```
    elseif load_type == 'pointd'
```

```
        k_F=[1 1 1];
        k_m=[.43 .49 .33];
        R=[16*Mp/3/L 6*Mp/L];
        k=[107*E*I/L^3 48*E*I/L^3 2*10^-6/1000*A]/A*1000;
        k_V_RF=[.97 -.28; .97 -.28; .92 -.25];
```

```

end

elseif bc == 'fixfix'

    if load_type == 'unifrm'

        k_F=[.53 .64 .5];
        k_m=[.41 .5 .33];
        R=[12*Mp/L 16*Mp/L];
        k=[384*E*I/L^3 384*E*I/5/L^3 2*10^-6/1000*A]/A*1000;
        k_V_RF=[.36 .14; .39 .11; .38 .12];

    elseif load_type == 'pointd'

        k_F=[1 1 1];
        k_m=[.37 .37 .33];
        R=[0 8*Mp/L];
        k=[192*E*I/L^3 192*E*I/L^3 2*10^-6/1000*A]/A*1000;
        k_V_RF=[.71 -.21; .71 -.21; .75 -.25];

    end

elseif bc == 'cantil'

    if load_type == 'unifrm'

        k_F=[.4 .4 .5];
        k_m=[.26 .26 .33];
        R=[0 2*Mp/L];
        k=[8*E*I/L^3 8*E*I/L^3 2*10^-6/1000*A]/A*1000;
        k_V_RF=[.69 .31; .69 .31; .75 .25];

    elseif load_type == 'pointd'

        k_F=[1 1 1];
        k_m=[.24 .24 .33];
        R=[0 Mp/L];
        k=[3*E*I/L^3 3*E*I/L^3 2*10^-6/1000*A]/A*1000;
        k_V_RF=[1.74 -.74; 1.74 -.74; 1.5 -.5];

    end

end

```

```

end

k_F=transpose(k_F);

if load_type == 'uniform'

    F=load*L;

elseif load_type == 'pointd'

    F=load;

end

```

Constant Velocity Function

```

function [u, v, a, alldata, V] = const_vel(bc, load_type, u_init,
v_init, m, c, k, k_F, k_m, R, t, F, zeta, wn, wd, steps, del_t, p, A,
k_V_RF, axial)

R=R/A*1000; %converts R values to psi.

y_yield=R(1)/(k(1));

y_yield2=y_yield+(R(2)-R(1))/(k(2));

skip2=0;

Rmax=.95*R(2);

KLM=k_m./transpose(k_F);

c_cr=2*sqrt(k(1)*m*KLM(1));

c_use=zeta*c_cr;

c_min=.05/100*c_cr;

R_datum=0;

%% initial constant velocity calcs

i=1;

```

```

t(i)=0;

u(i)=u_init;

v(i)=v_init;

equiv_pdelt(i)=axial*u(i);

if u(i)<=y_yield

    c=c_use;
    Res(i)=u(i)*k(1);
    a(i)=(p(i)+equiv_pdelt(i)-c*v(i)-Res(i))/(m*KLM(1));
    stiffness(i)=k(1);
    factor(i)=KLM(1);
    V(i)=k_V_RF(1,1)*Res(i)+k_V_RF(1,2)*p(i);

elseif u(i)>y_yield && u(i)<y_yield2

    Res(i)=R(1)+(u(i)-y_yield)*k(2);

    if Res(i)<Rmax
        c=c_use;
    elseif Res(i)>=Rmax
        c=c_min;
    end

    a(i)=(p(i)+equiv_pdelt(i)-c*v(i)-Res(i))/(m*KLM(2));
    stiffness(i)=k(2);
    factor(i)=KLM(2);
    V(i)=k_V_RF(2,1)*Res(i)+k_V_RF(2,2)*p(i);

elseif u(i)>=y_yield2

    Res(i)=R(2)+(u(i)-y_yield2)*k(3);
    c=c_min;
    a(i)=(p(i)+equiv_pdelt(i)-c*v(i)-Res(i))/(m*KLM(3));
    stiffness(i)=k(3);
    factor(i)=KLM(3);
    V(i)=k_V_RF(3,1)*Res(i)+k_V_RF(3,2)*p(i);

end

d_u(i)=.5*a(i)*del_t^2+v(i)*del_t;

alldata(i,1)=t(i);
alldata(i,2)=p(i);
alldata(i,3)=equiv_pdelt(i);
alldata(i,4)=u(i);

```

```

alldata(i,5)=v(i);
alldata(i,6)=a(i);
alldata(i,7)=stiffness(i);
alldata(i,8)=Res(i);
alldata(i,9)=d_u(i);
alldata(i,10)=factor(i);
alldata(i,11)=V(i);

i=2;

%% Main Constant Velocity loop

while i<=steps %loop for Constant Velocity iterations

    t(i)=t(i-1)+del_t;

    u(i)=u(i-1)+d_u(i-1);

    v(i)=(u(i)-u(i-1))/del_t+.5*a(i-1)*del_t;

    equiv_pdelt(i)=axial*u(i);

    R_check=abs(Res(i-1)-R_datum); %amount of change in resistance
    from inflection point ("0 strain")

    %%%elastic region%%

    if R_check<=R(1) || skip2==1

        Res(i)=Res(i-1)+(u(i)-u(i-1))*k(1);

        if Res(i)>R(2) %prevents going past maximum resistance

            Res(i)=R(2);

        end

        if Res(i)<-R(2) %prevents going past maximum resistance
        (negative)

            Res(i)=-R(2);

        end

```



```

        if sign(Res(i))==sign(v(i)) %decreases damping when close to
yielding

            if Res(i)<Rmax
                c=c_use;
            elseif Res(i)>=Rmax
                c=c_min;
            end

        else

            c=c_use;
        end

        a(i)=(p(i)+equiv_pdelt(i)-c*v(i)-Res(i))/(m*KLM(1));
        stiffness(i)=k(1);
        factor(i)=KLM(1);
        V(i)=k_V_RF(1,1)*Res(i)+k_V_RF(1,2)*p(i);

        if sign(v(i))~=sign(v(i-1)) %once an inflection point occurs
(sign change of velocity), R_datum is set as the current resistance

            R_datum=Res(i);

        end

        %with skip2, it forces stage 1 on turnaround while the net
%Resistance is still larger than yield Resistance. Resets the
%datum
        if sign(Res(i))~=sign(Res(i-1))
            skip2=0;
            R_datum=0;

        end

        %%%%elasto-plastic region%%%%%%%%

        elseif R_check>R(1) && R_check<R(2)

            Res(i)=Res(i-1)+(u(i)-u(i-1))*k(2);

            if Res(i)>R(2) %prevents going past maximum resistance

                Res(i)=R(2);

            end

```

```

        if Res(i)<-R(2)%prevents going past maximum resistance
(negative)

            Res(i)=-R(2);

        end

        if Res(i)<Rmax %decreases damping when close to yielding
            c=c_use;
        elseif Res(i)>=Rmax
            c=c_min;
        end

        a(i)=(p(i)+equiv_pdelt(i)-c*v(i)-Res(i))/(m*KLM(2));
        stiffness(i)=k(2);
        factor(i)=KLM(2);
        V(i)=k_V_RF(2,1)*Res(i)+k_V_RF(2,2)*p(i);

        if sign(v(i))~=sign(v(i-1)) %once an inflection point occurs
(sign change of velocity), R_datum is set as the current resistance

            R_datum=Res(i);
            skip2=1;

        end

        % %%%plastic region%%

        elseif R_check>=R(2)

            Res(i)=Res(i-1)+(u(i)-u(i-1))*k(3);

            c=c_min;
            a(i)=(p(i)+equiv_pdelt(i)-c*v(i)-Res(i))/(m*KLM(3));
            stiffness(i)=k(3);
            factor(i)=KLM(3);
            V(i)=k_V_RF(3,1)*Res(i)+k_V_RF(3,2)*p(i);

            if sign(v(i))~=sign(v(i-1)) %once an inflection point occurs
(sign change of velocity), R_datum is set as the current resistance

                R_datum=Res(i);
                skip2=1;

            end

        end

```

```

end

d_u(i)=u(i)-u(i-1)+a(i)*del_t^2;

alldata(i,1)=t(i);
alldata(i,2)=p(i);
alldata(i,3)=equiv_pdelt(i);
alldata(i,4)=u(i);
alldata(i,5)=v(i);
alldata(i,6)=a(i);
alldata(i,7)=stiffness(i);
alldata(i,8)=Res(i);
alldata(i,9)=d_u(i);
alldata(i,10)=factor(i);
alldata(i,11)=V(i);

i=i+1;

end

figure(1)
subplot(2,2,1)
plot(t,a)
title('acc')
xlabel('time')
ylabel('Acceleration')
grid on

subplot(2,2,2)
plot(t,v)
title('vel')
xlabel('time')
ylabel('Velocity')
grid on

subplot(2,2,3)
plot(t,u)
title('displ')
legend('Displacement');
xlabel('time')
ylabel('displacement')
grid on

```

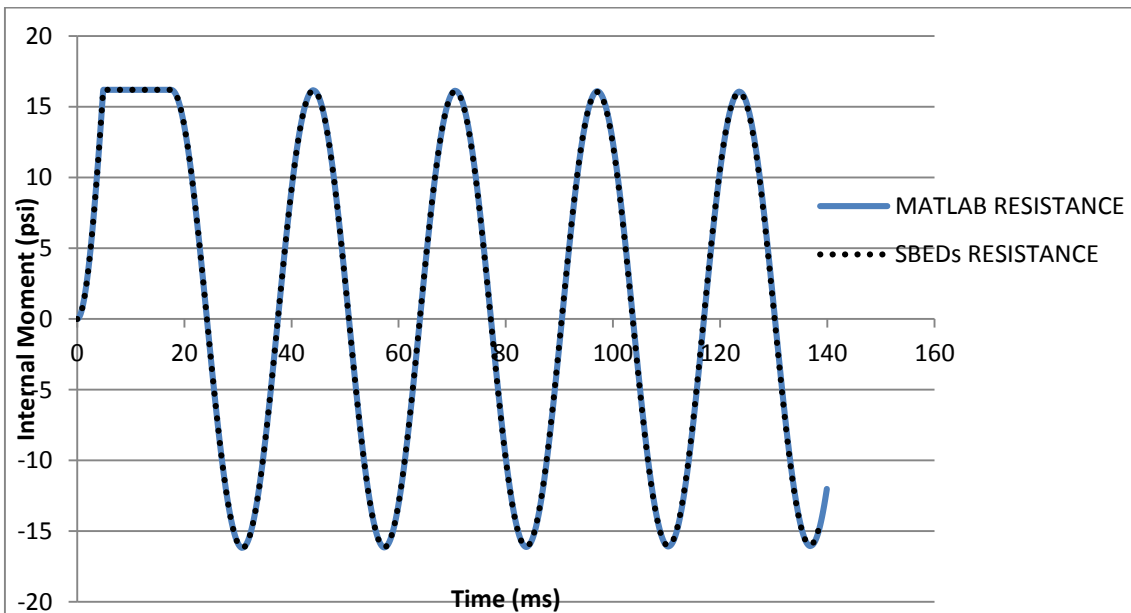
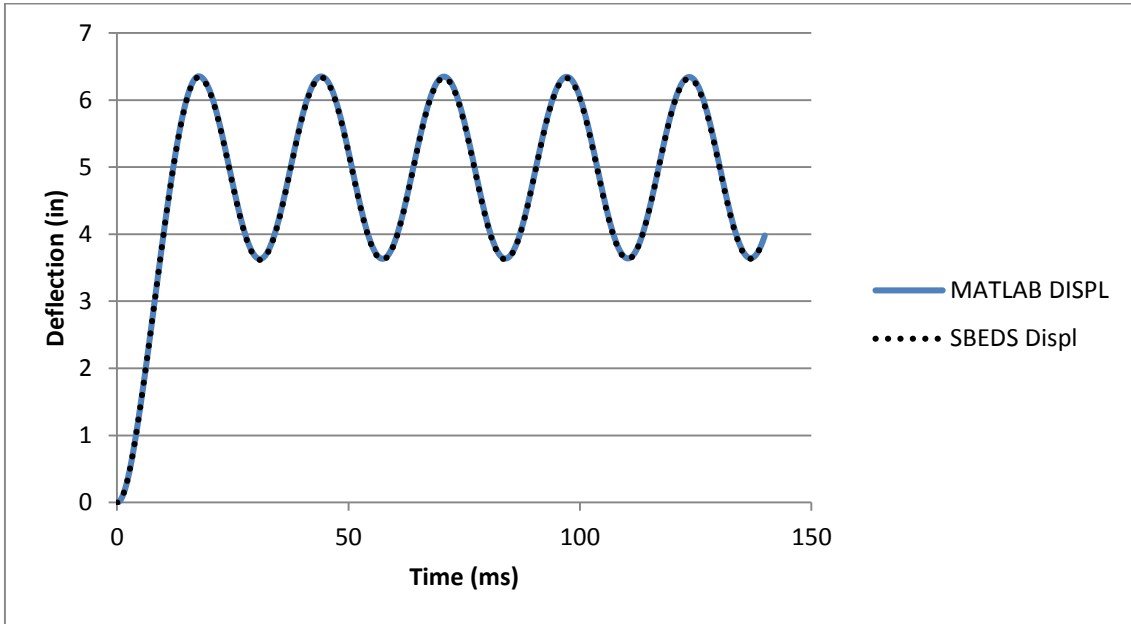
Error Function

```
function [error] = Error(i, j)
error=(j-i)/j*100;
```

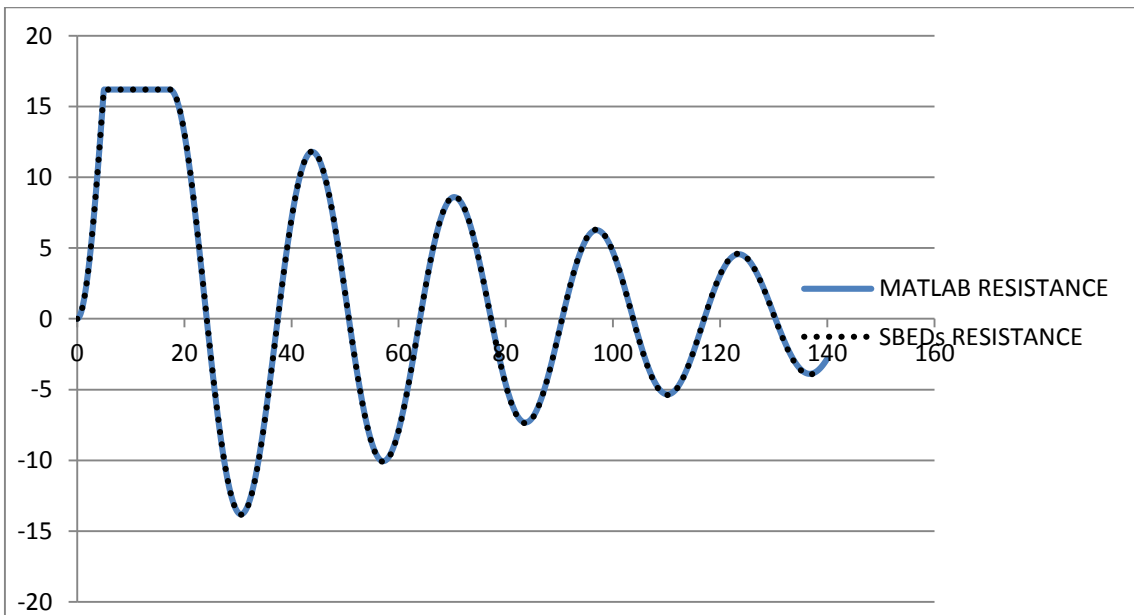
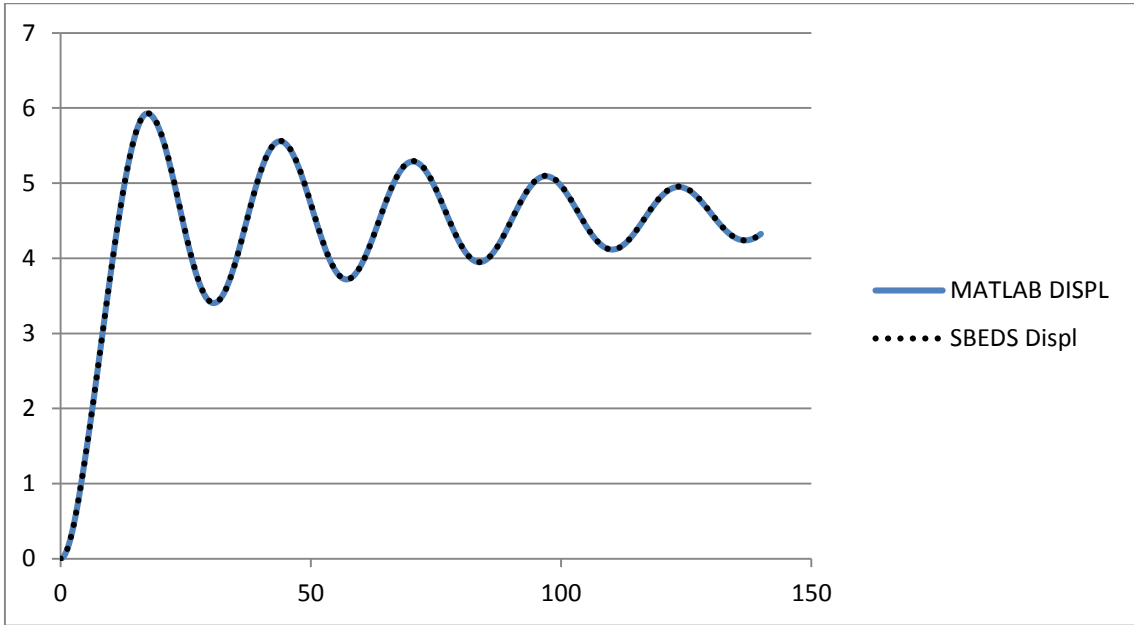
Appendix B: SDOF Solver Verifications (Time-History Plots)

All verifications use 15 ft Length and 0.5 ft Tributary Width

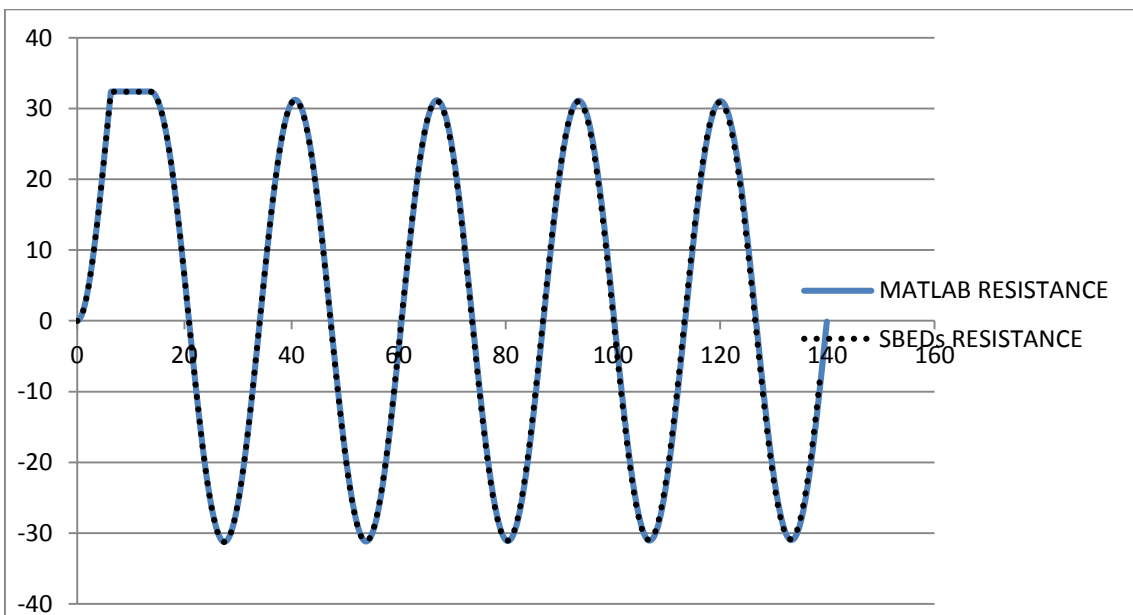
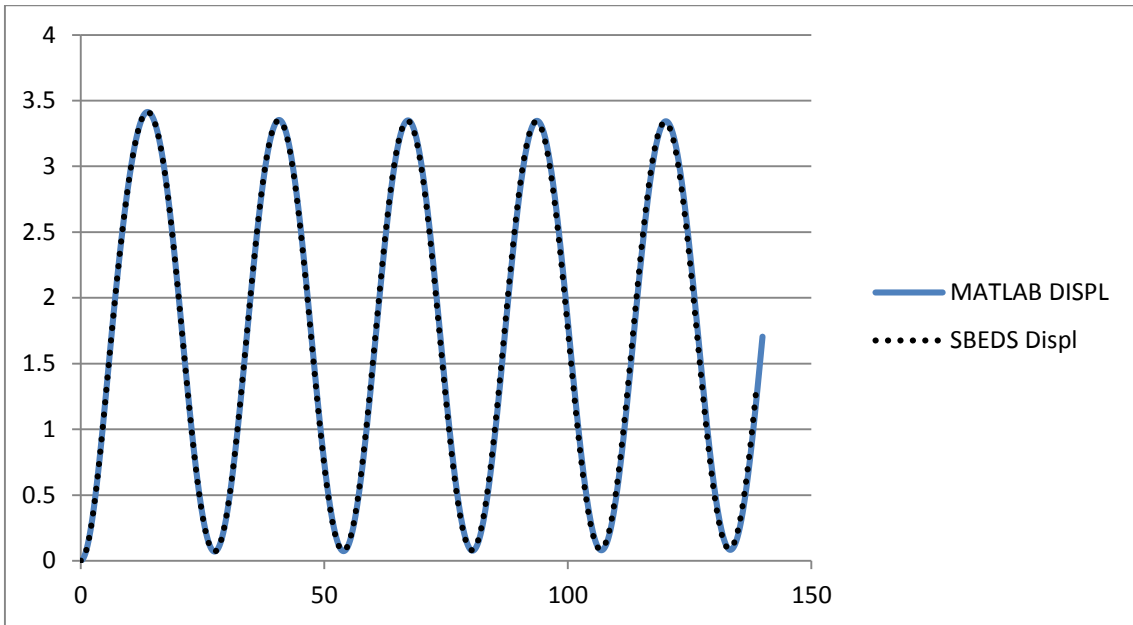
W10X12 Pinned-Pinned Point Load (30 psi in 17.8 ms) 0% Damping



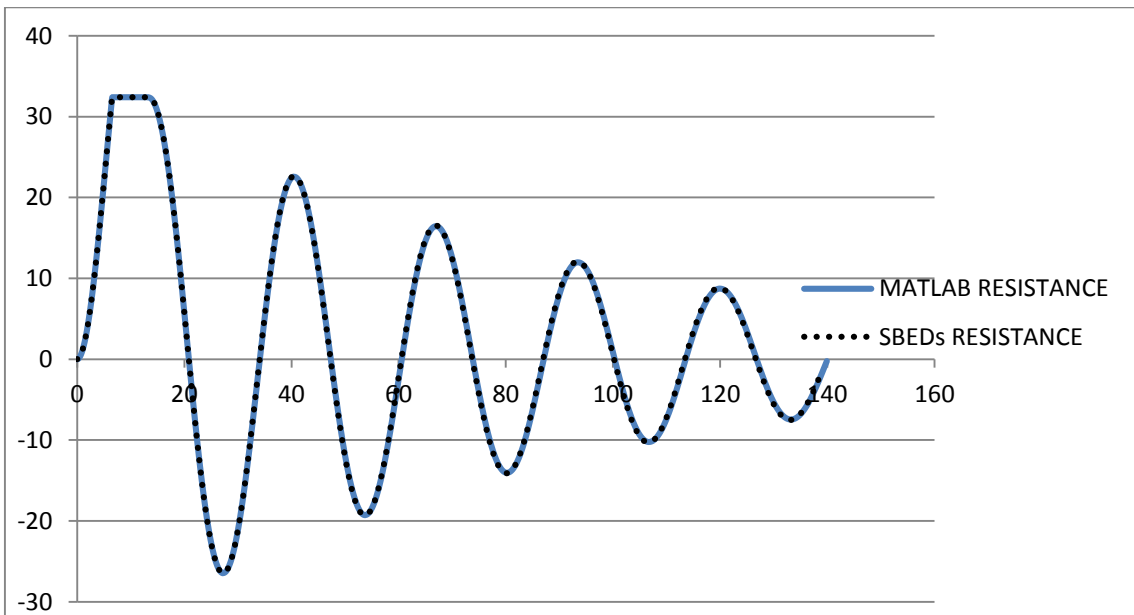
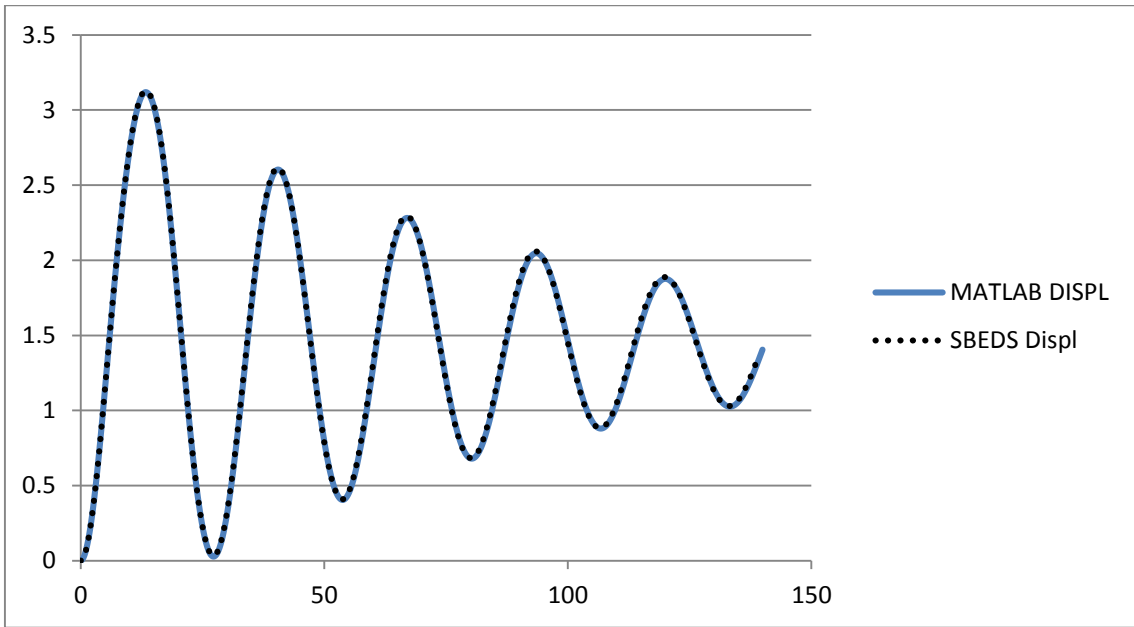
W10X12 Pinned-Pinned Point Load (30 psi in 17.8 ms) 5% Damping



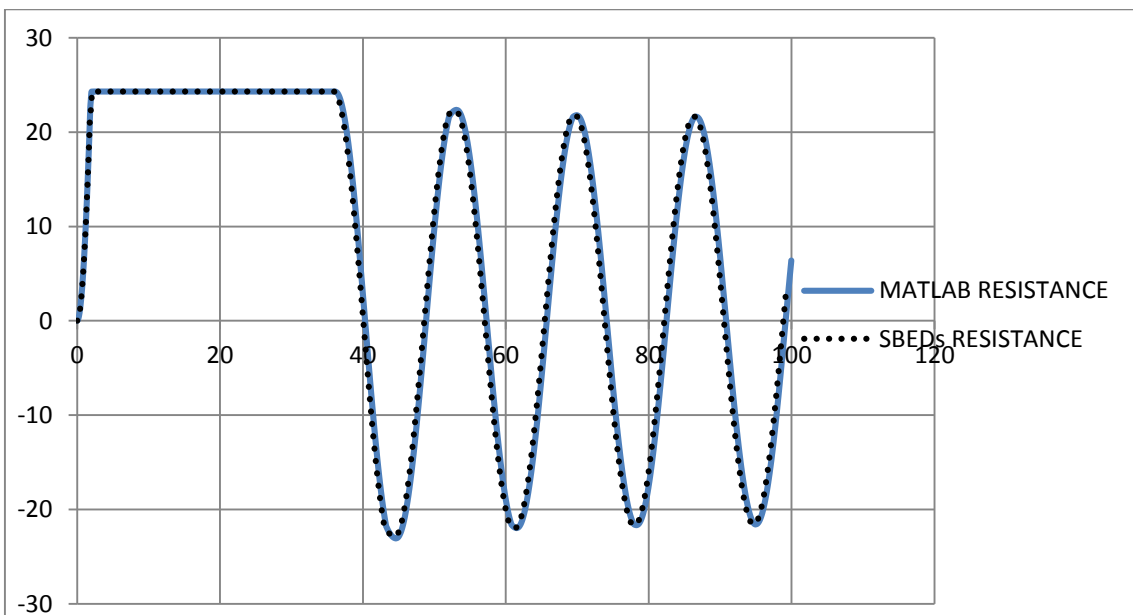
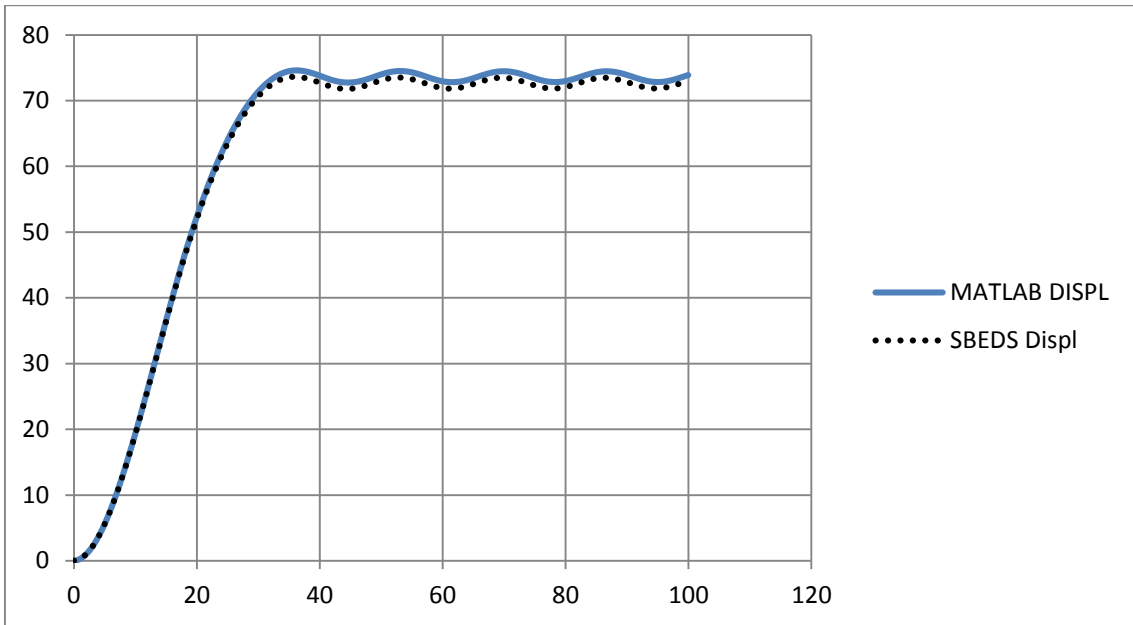
W10X12 Pinned-Pinned Uniform Load (40 psi in 17.8 ms) 0% Damping



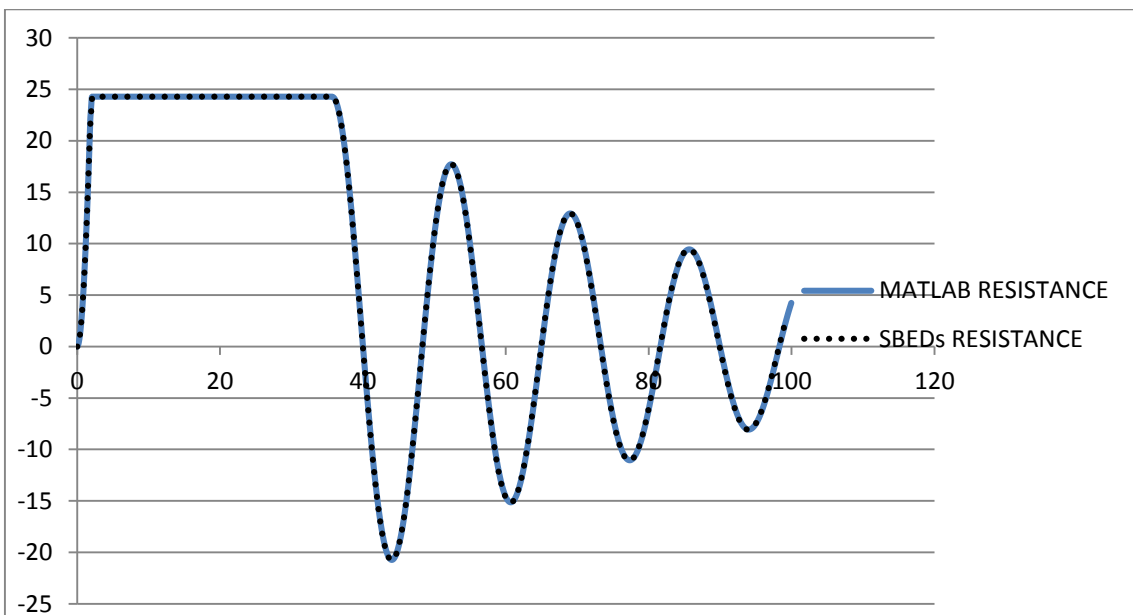
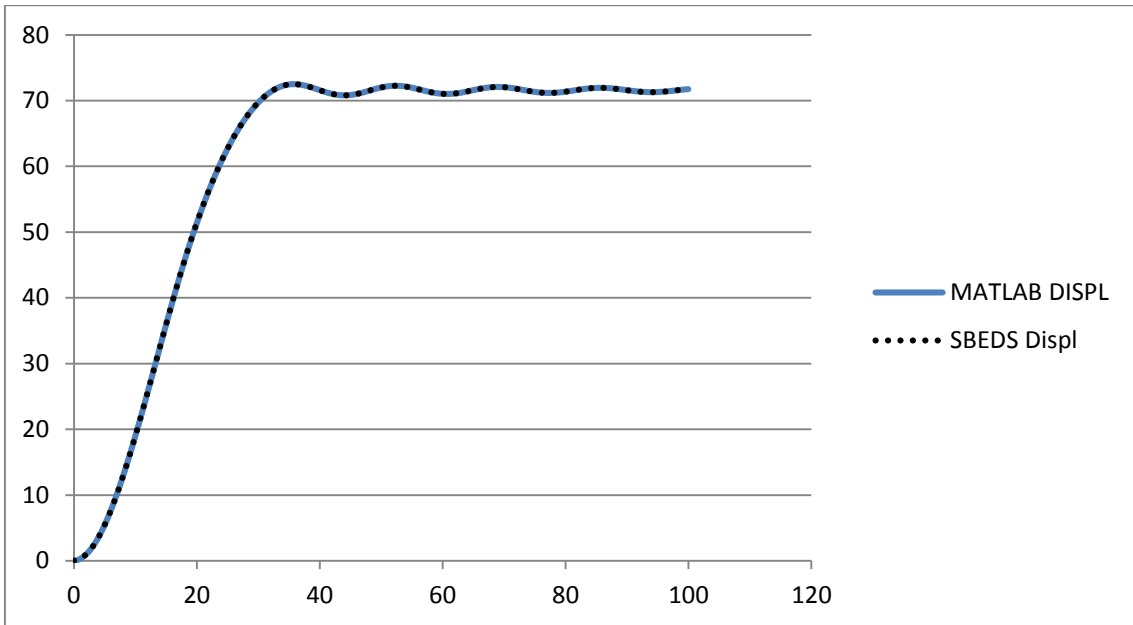
W10X12 Pinned-Pinned Uniform Load (40 psi in 17.8 ms) 5% Damping



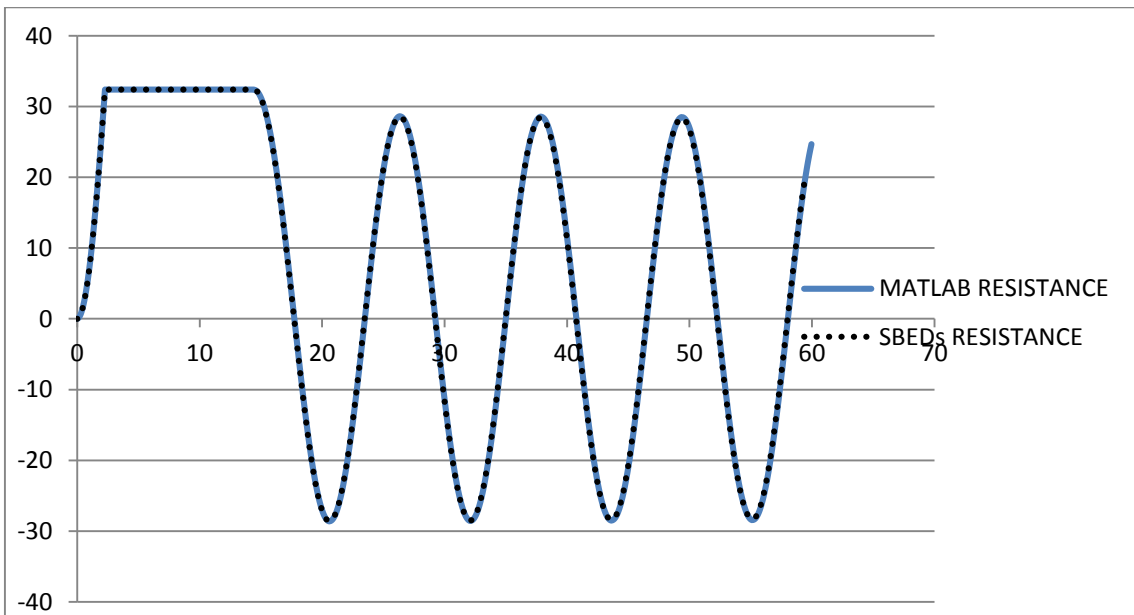
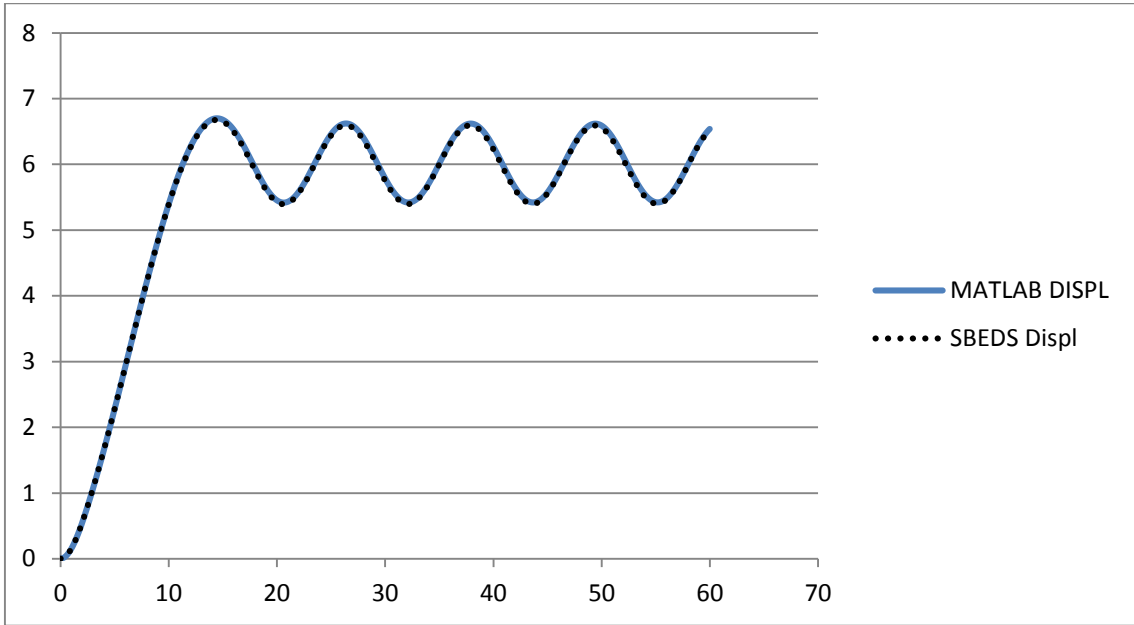
W10X12 Fixed-Pinned Point Load (100 psi in 17.8 ms) 0% Damping



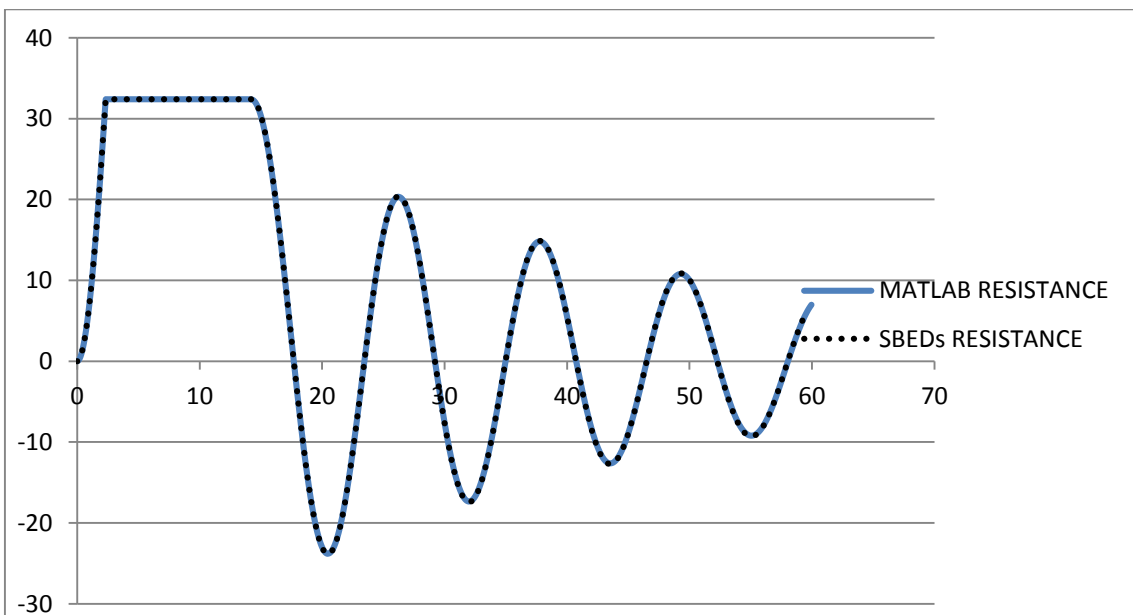
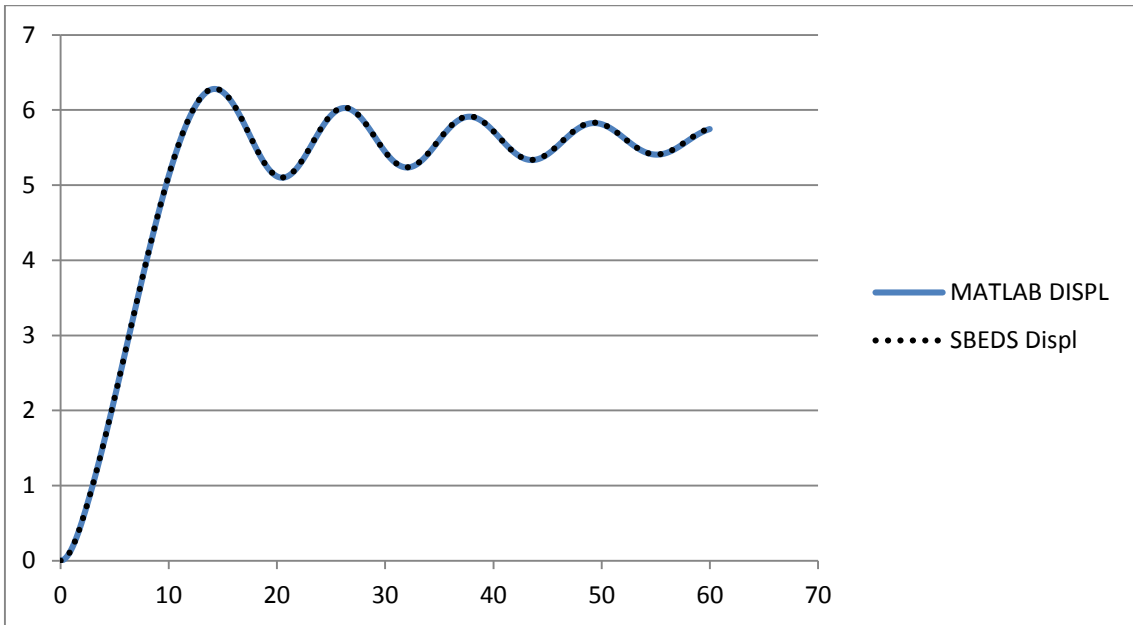
W10X12 Fixed-Pinned Point Load (100 psi in 17.8 ms) 5% Damping



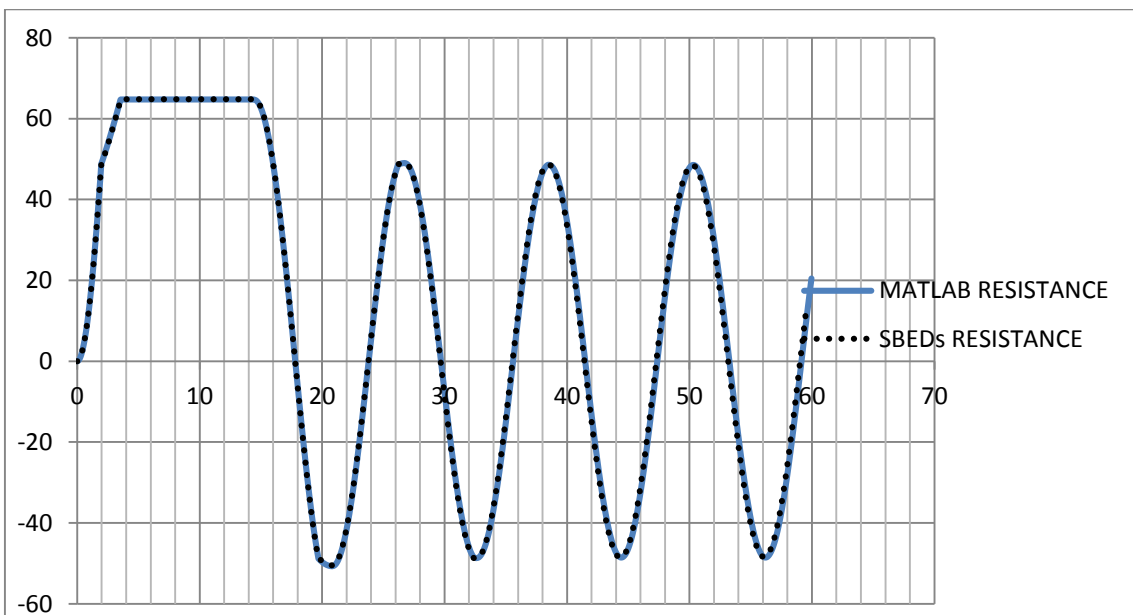
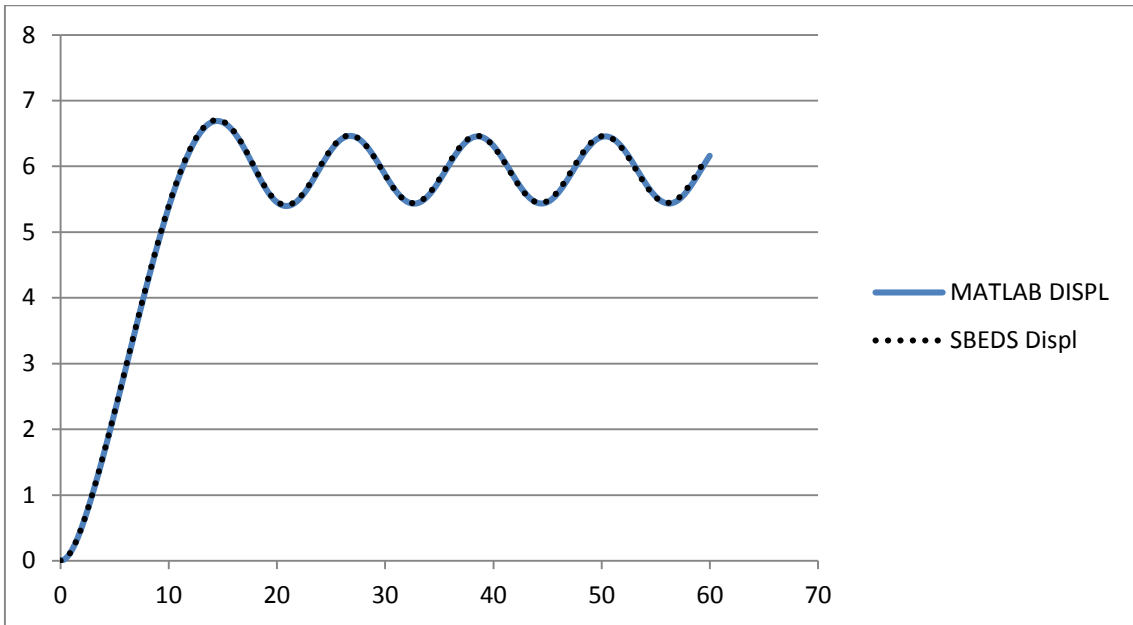
W10X12 Fixed-Fixed Point Load (50 psi in 17.8 ms) 0% Damping



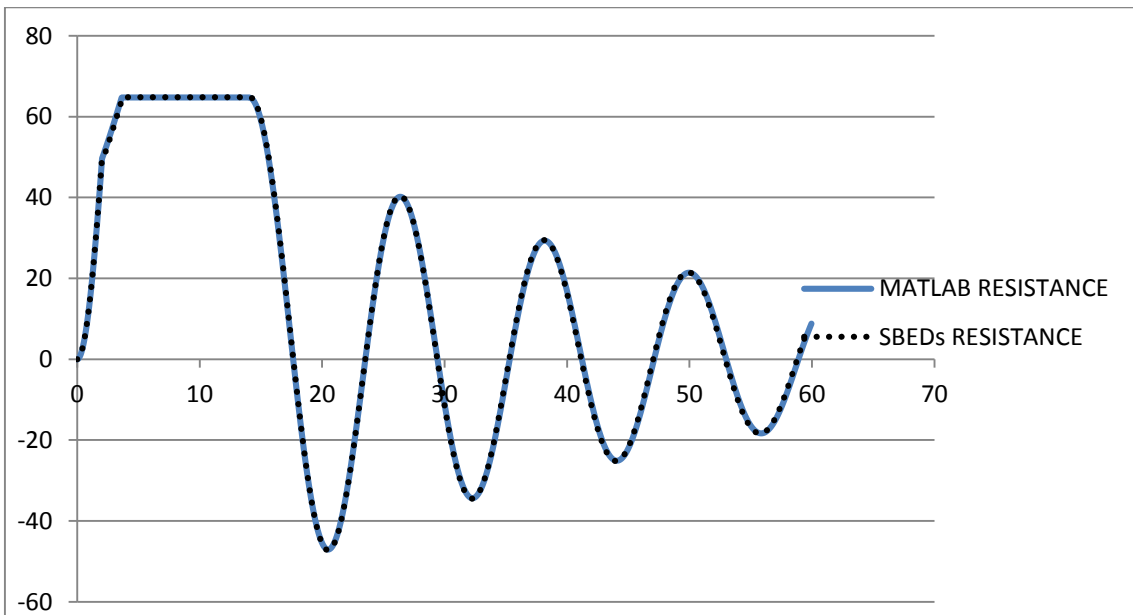
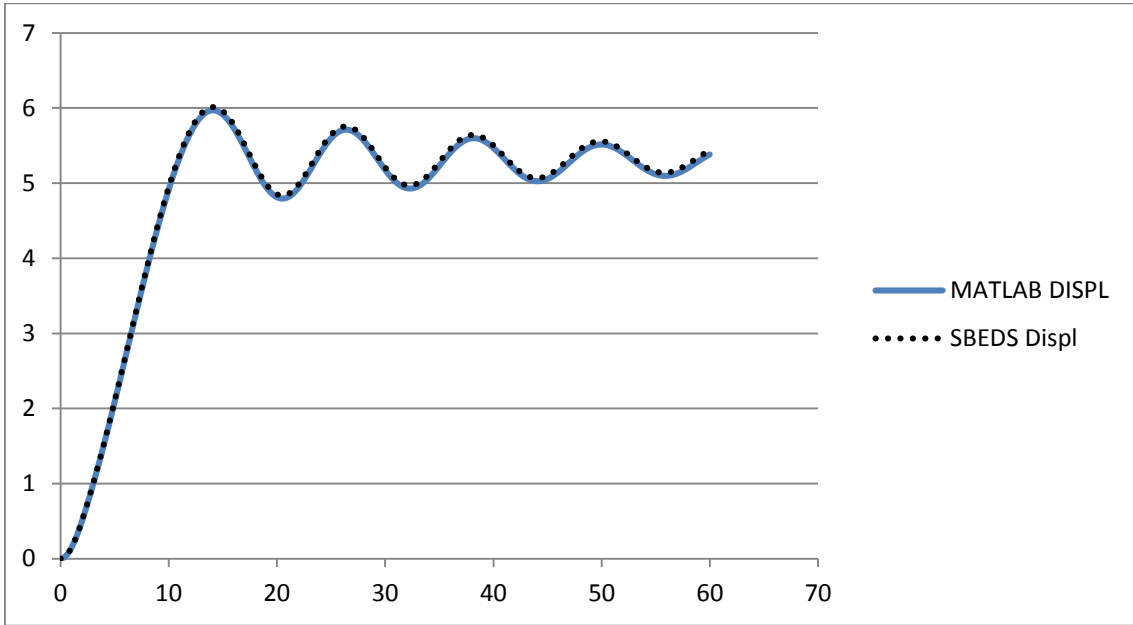
W10X12 Fixed-Fixed Point Load (50 psi in 17.8 ms) 5% Damping



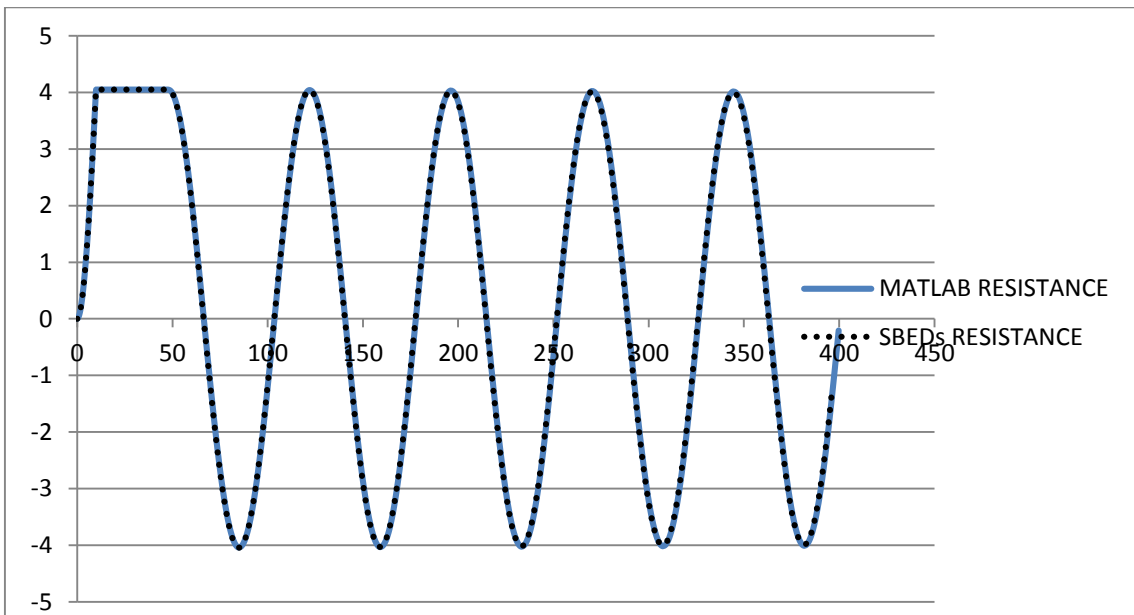
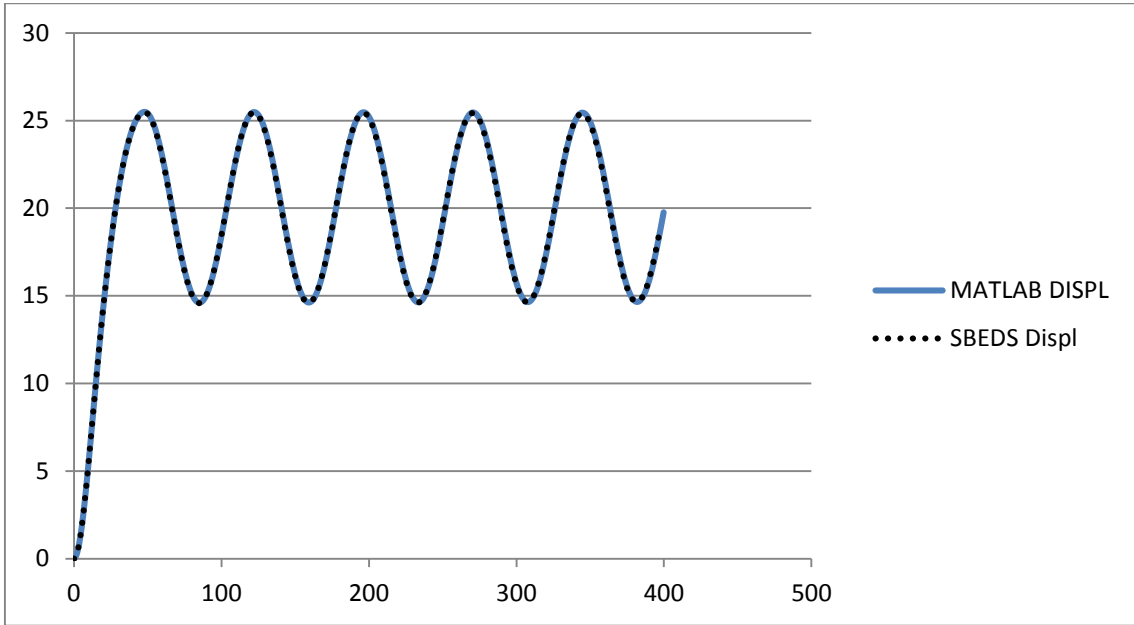
W10X12 Fixed-Fixed Uniform Load (100 psi in 17.8 ms) 0% Damping



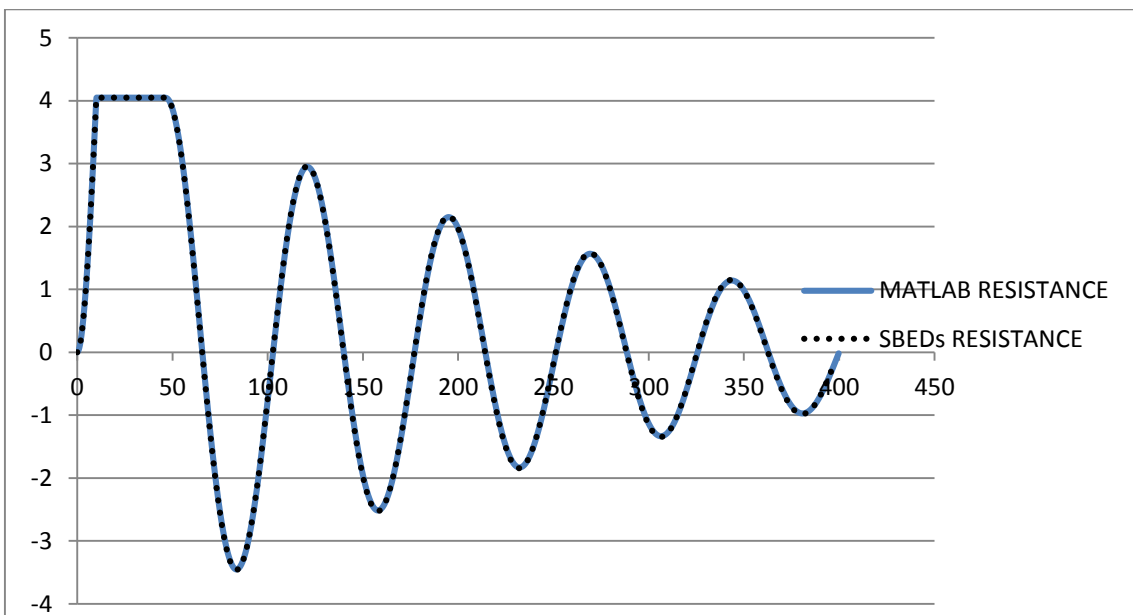
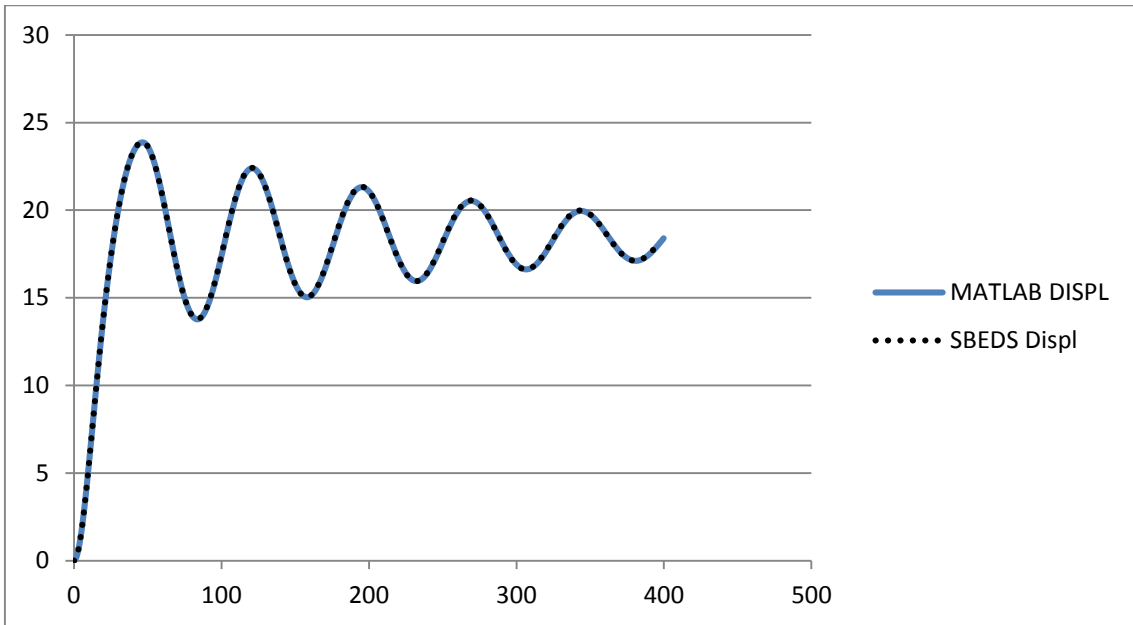
W10X12 Fixed-Fixed Uniform Load (100 psi in 17.8 ms) 5% Damping



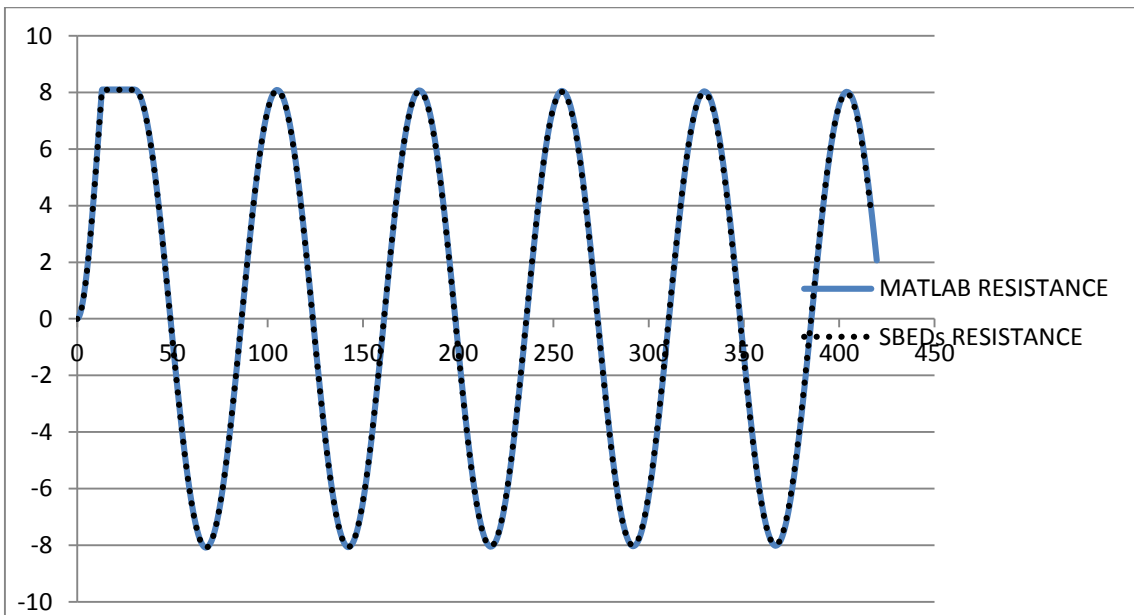
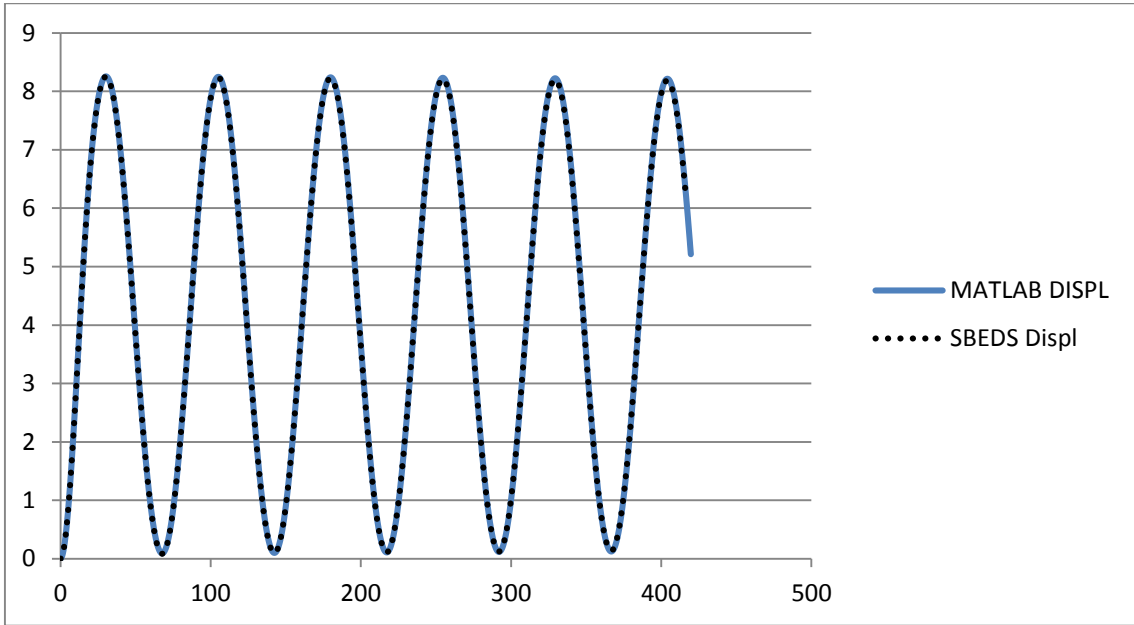
W10X12 Cantilever Point Load (15 psi in 17.8 ms) 0% Damping



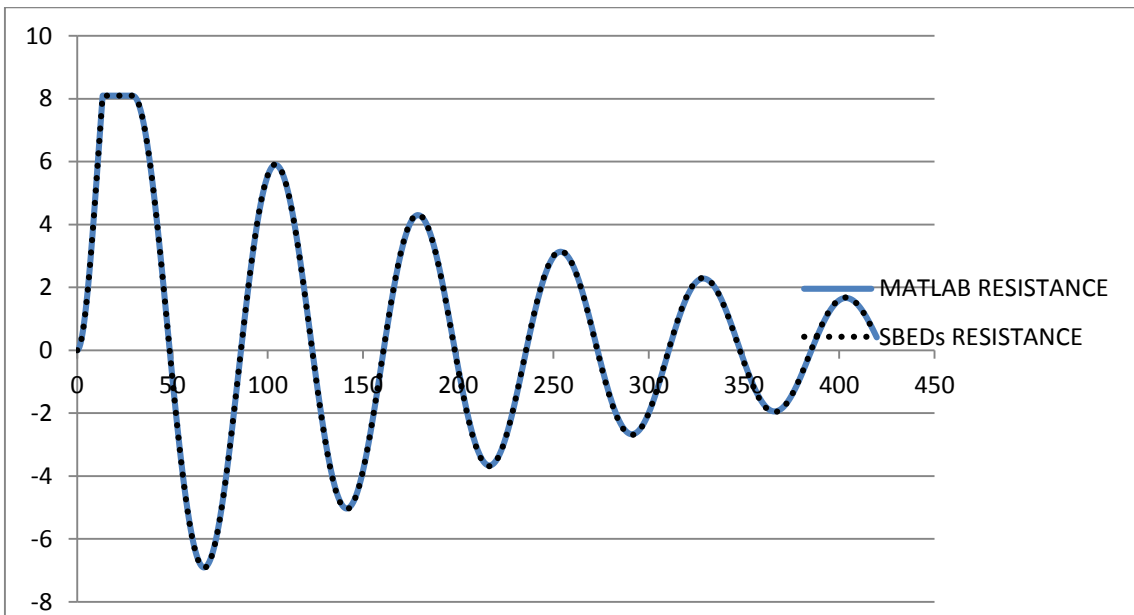
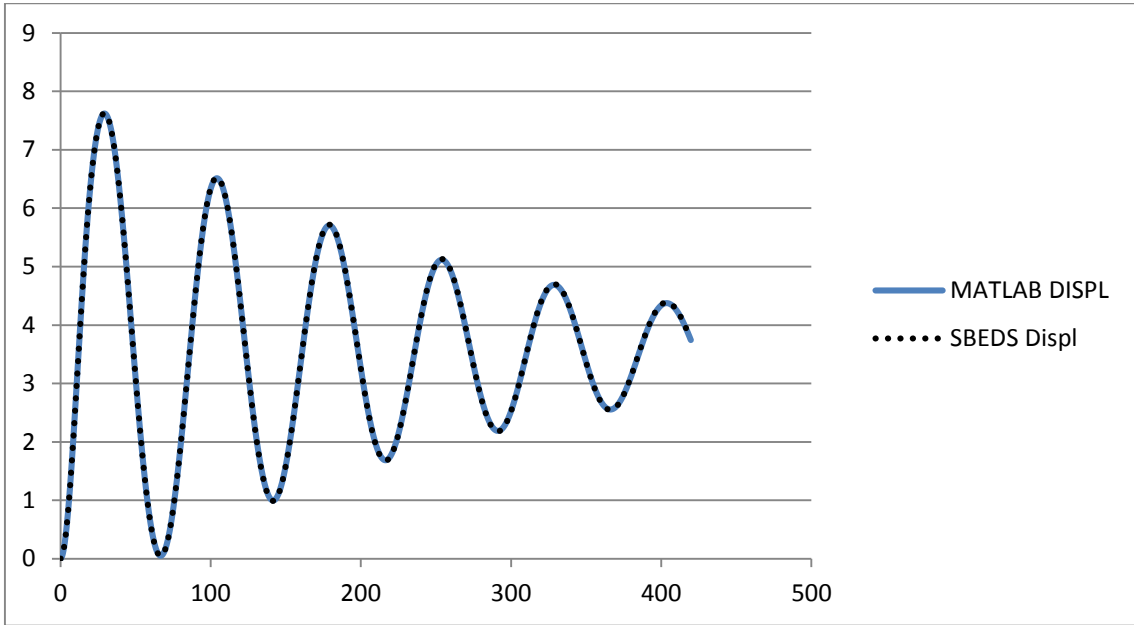
W10X12 Cantilever Point Load (15 psi in 17.8 ms) 5% Damping



W10X12 Cantilever Uniform Load (20 psi in 17.8 ms) 0% Damping

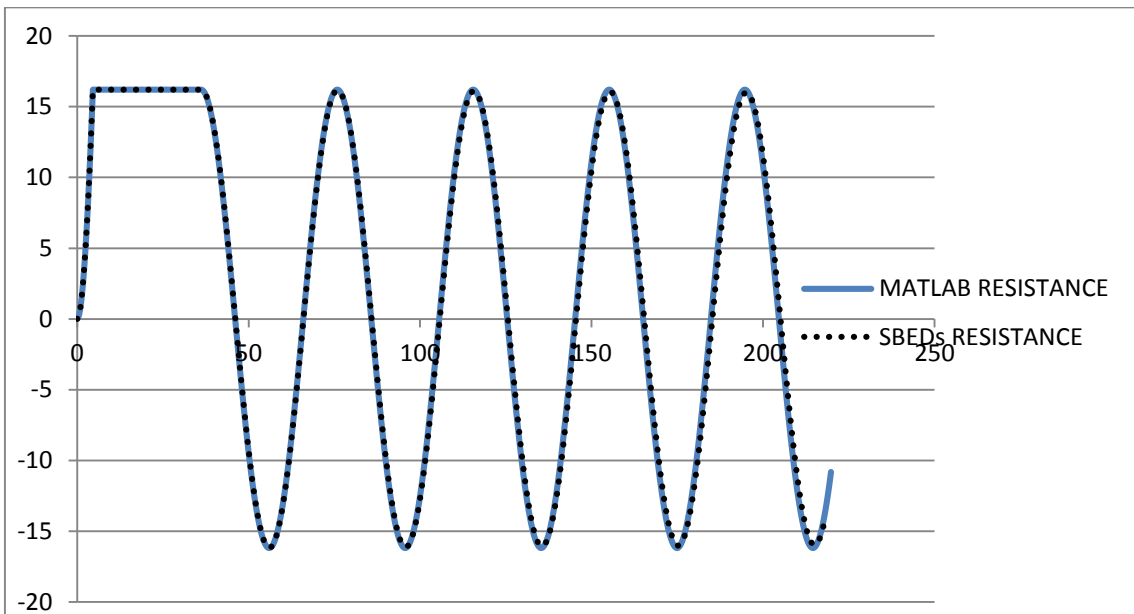
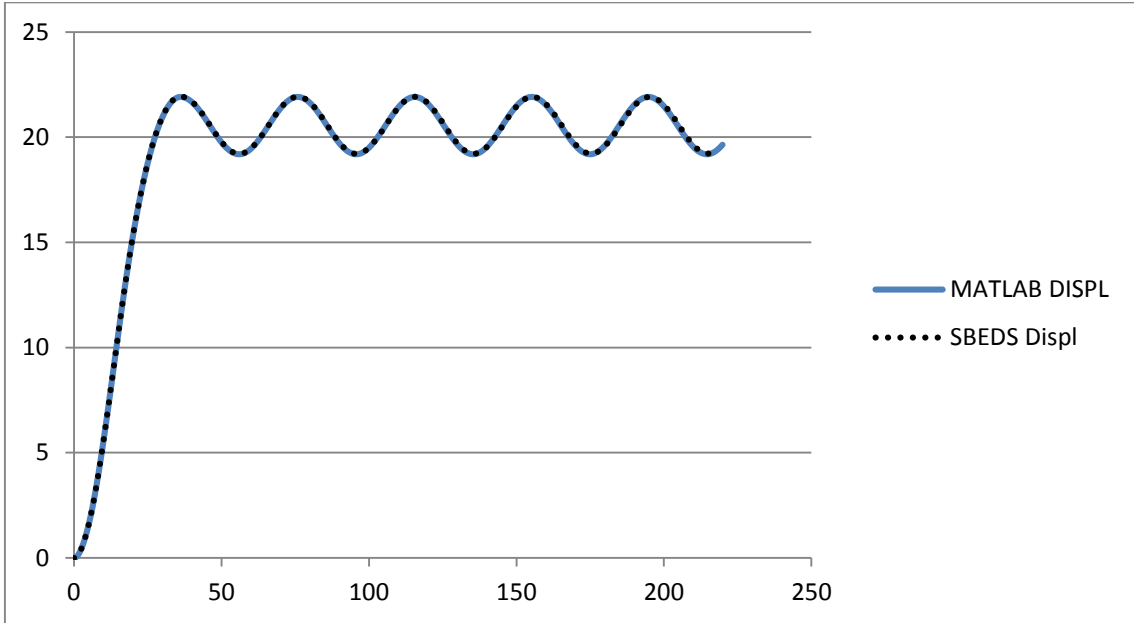


W10X12 Cantilever Uniform Load (20 psi in 17.8 ms) 5% Damping



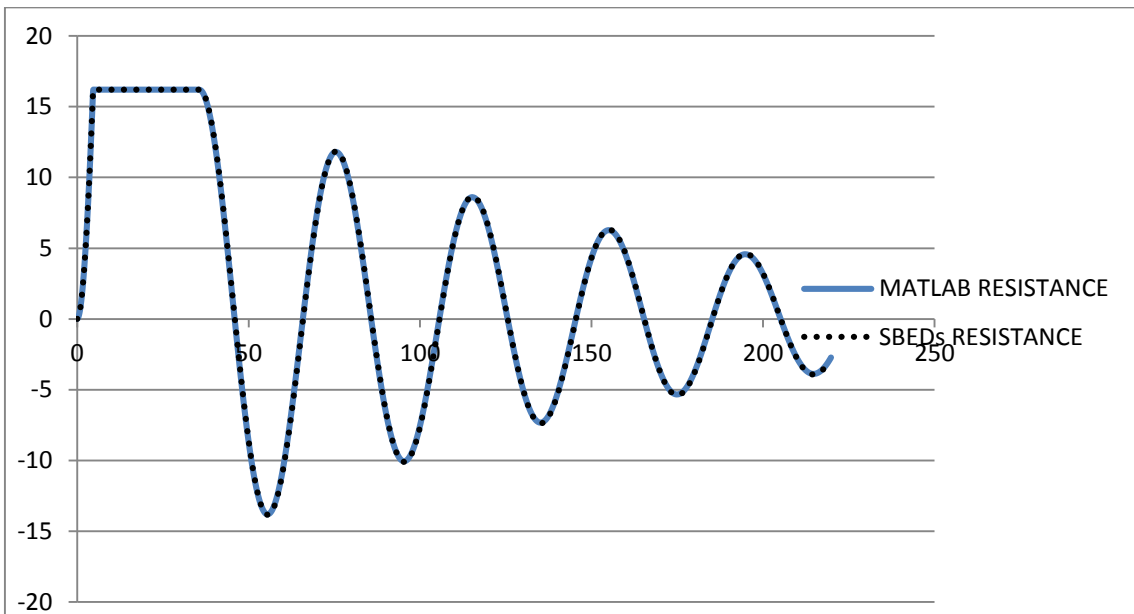
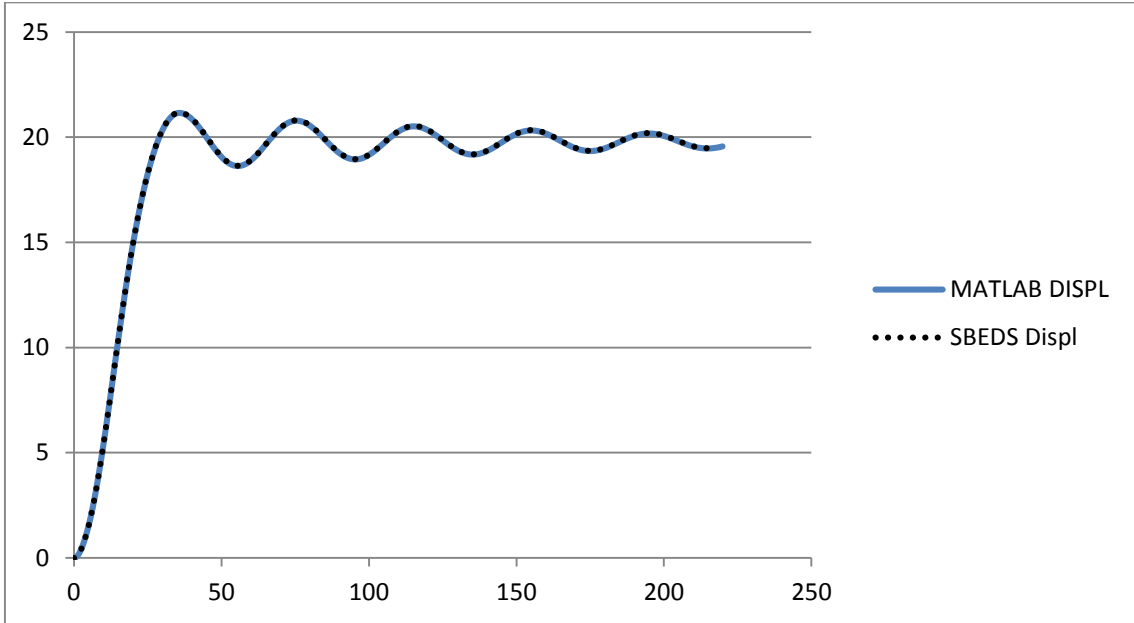
W10X12 Pinned-Pinned Point Load (70 psi in 17.8 ms) 0% Damping 30 psf Added

Weight



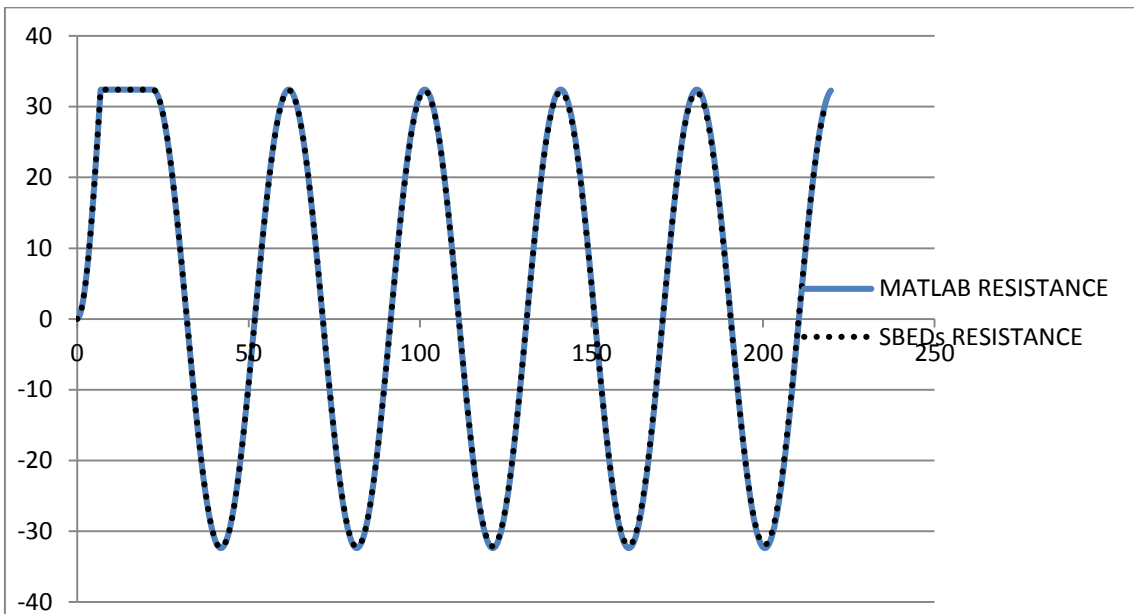
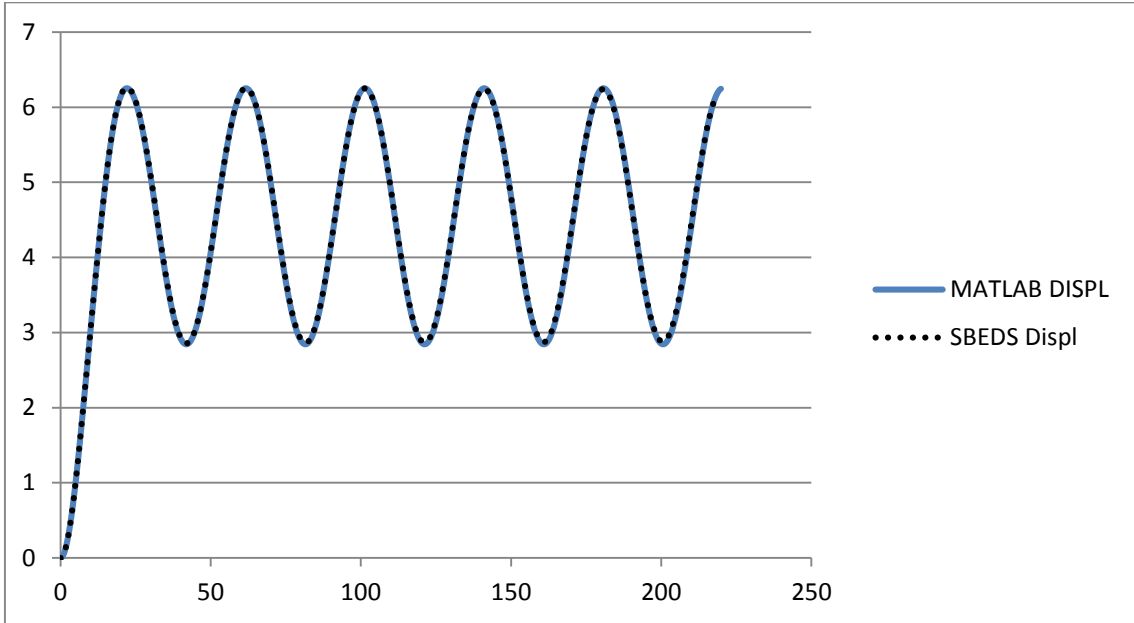
W10X12 Pinned-Pinned Point Load (70 psi in 17.8 ms) 5% Damping 30 psf Added

Weight



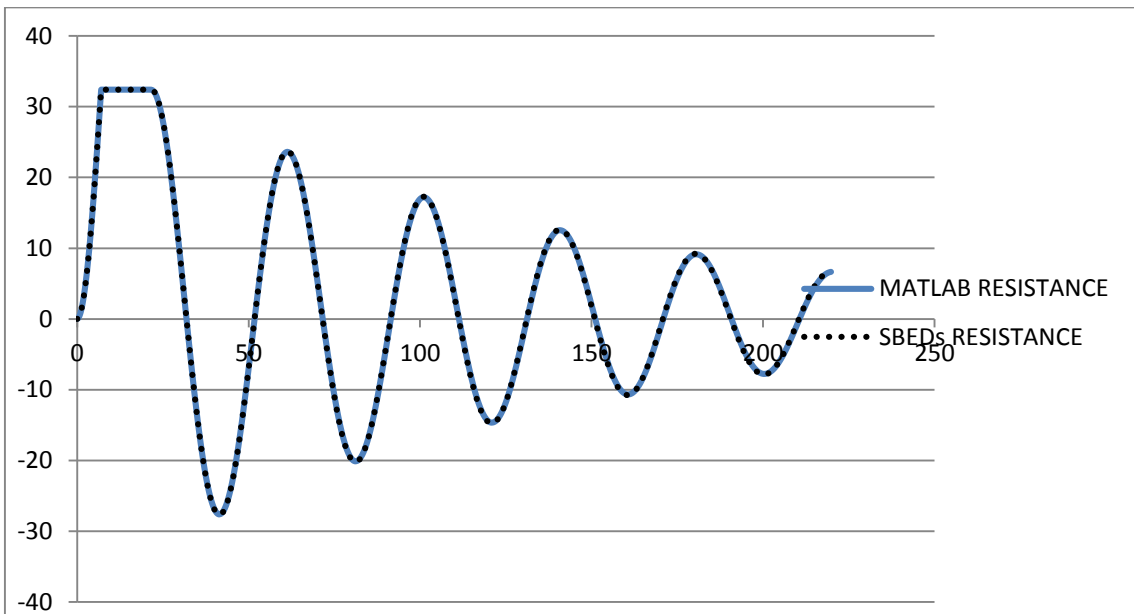
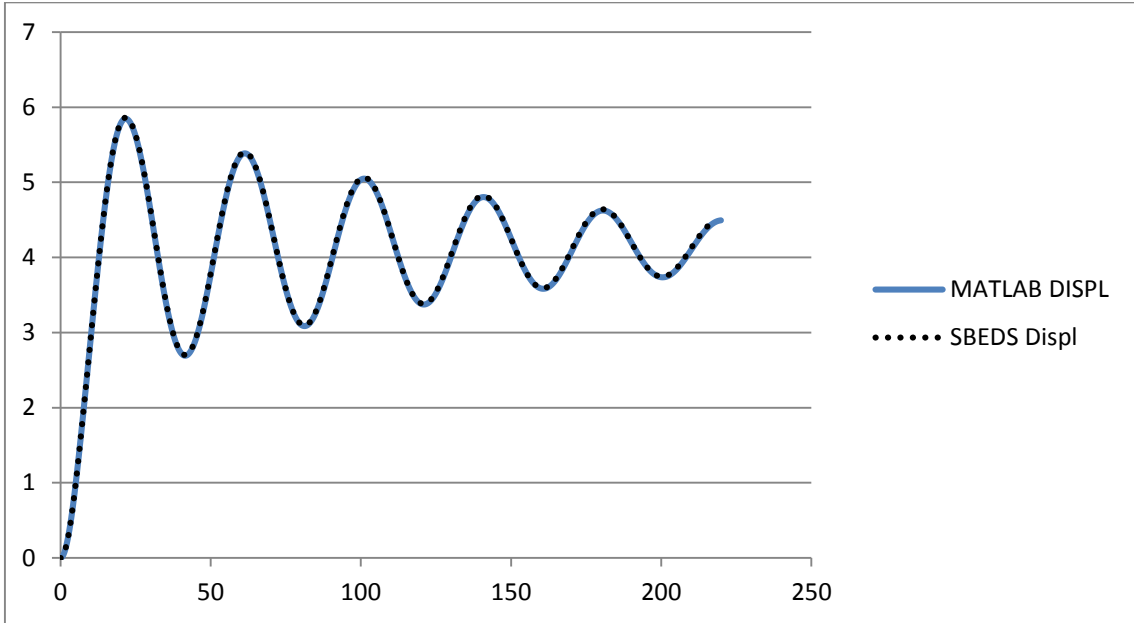
W10X12 Pinned-Pinned Uniform Load (70 psi in 17.8 ms) 0% Damping 30 psf Added

Weight

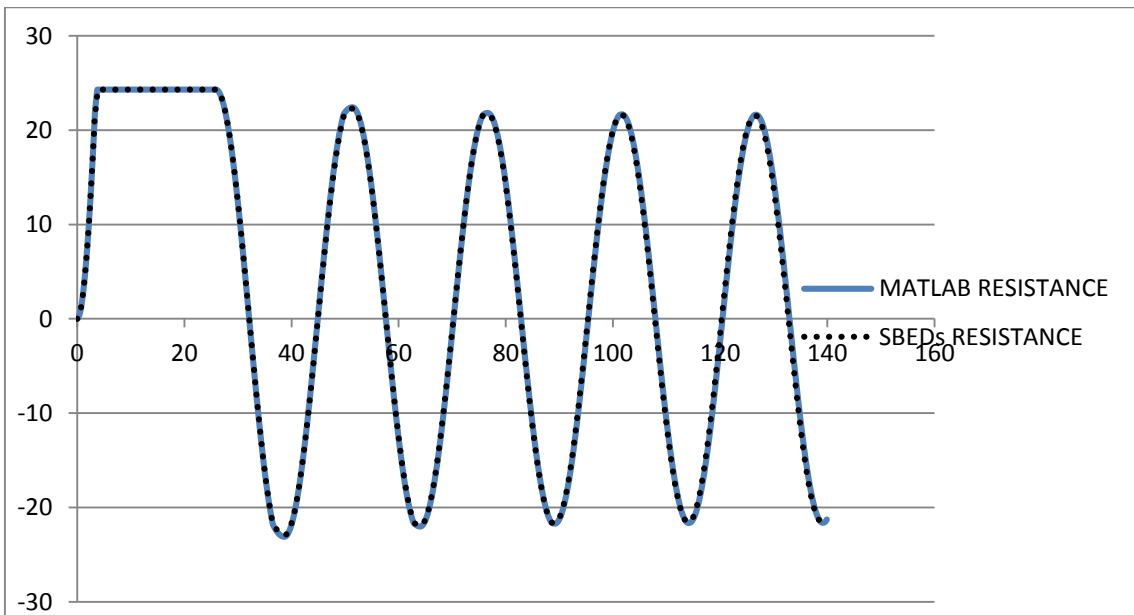
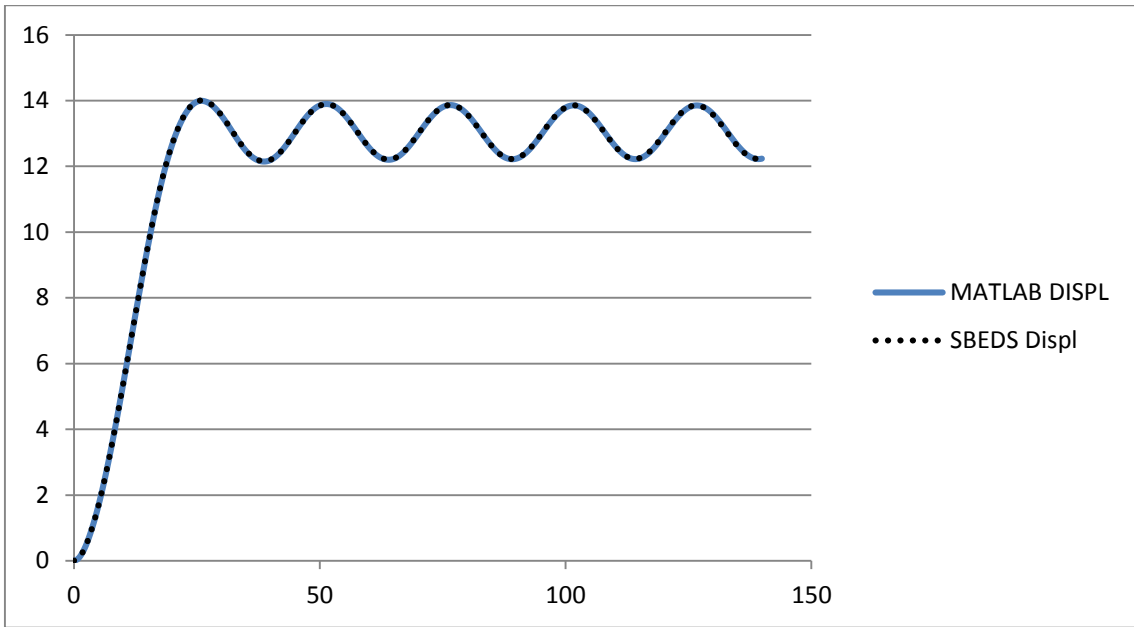


W10X12 Pinned-Pinned Uniform Load (70 psi in 17.8 ms) 5% Damping 30 psf Added

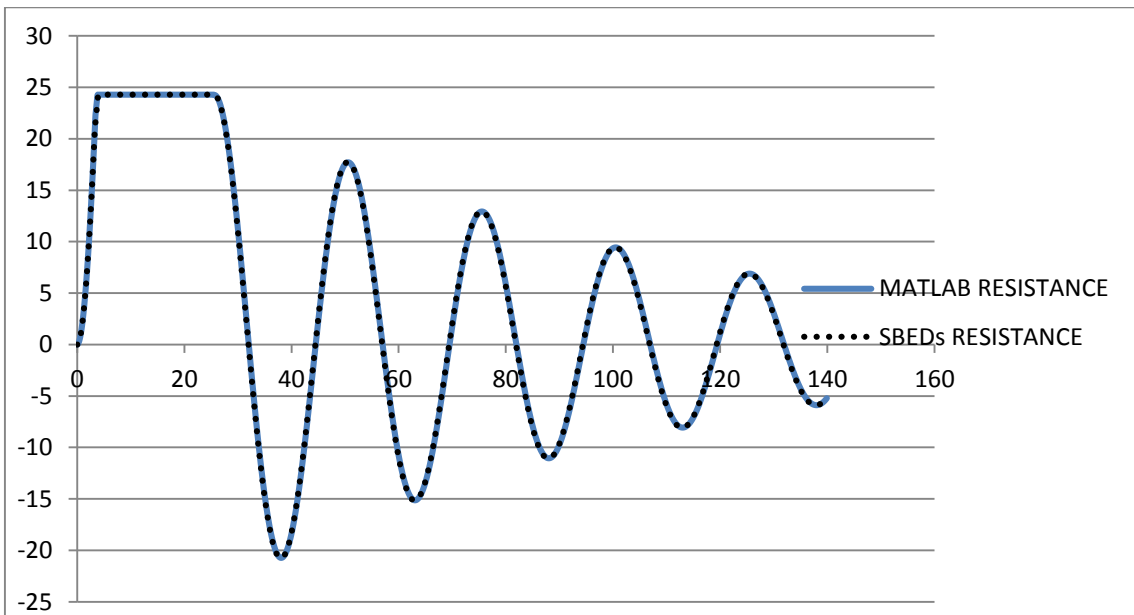
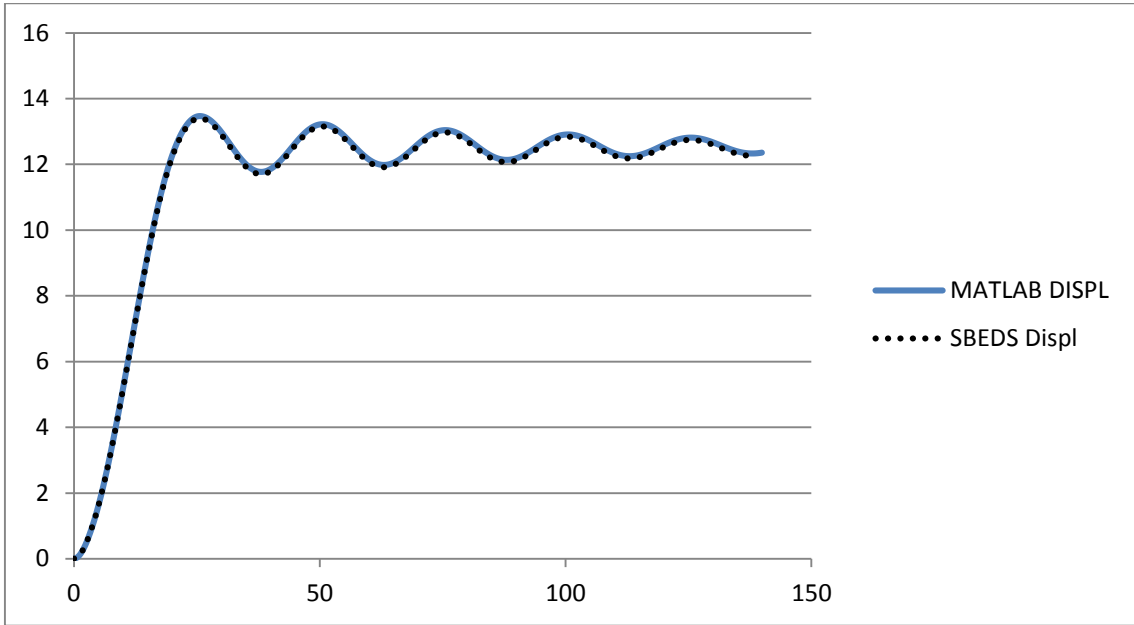
Weight



W10X12 Fixed-Pinned Point Load (70 psi in 17.8 ms) 0% Damping 30 psf Added Weight

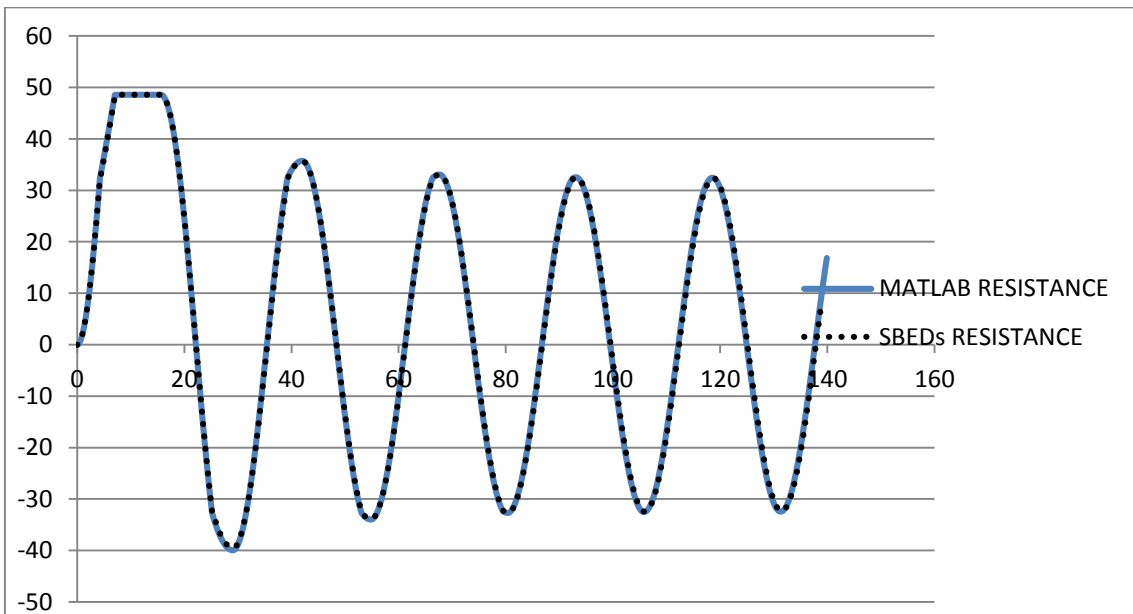
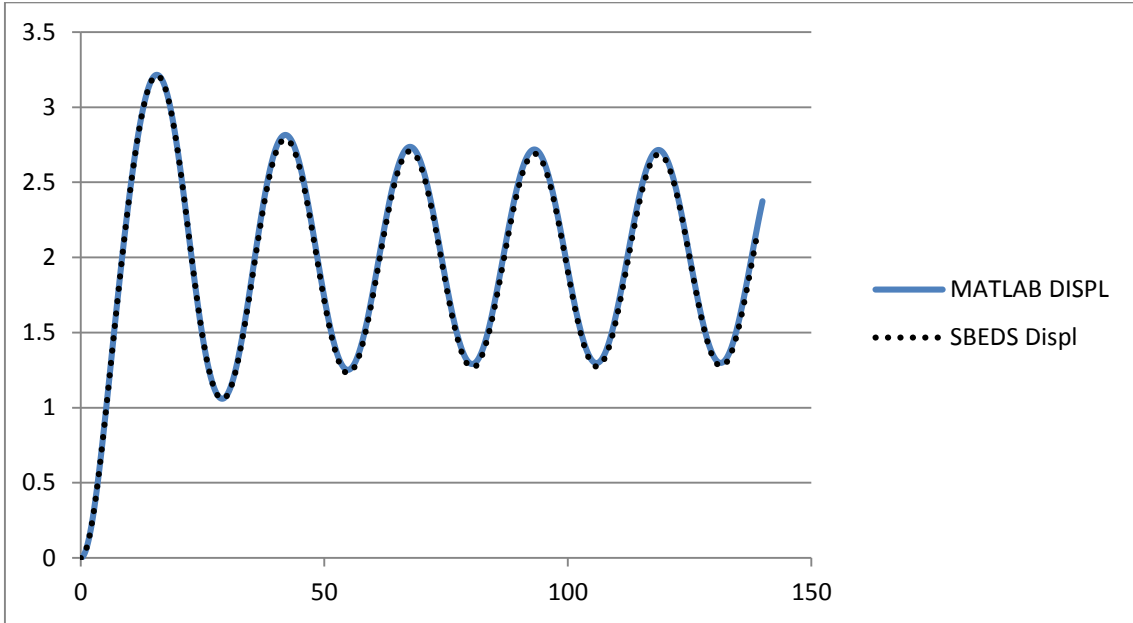


W10X12 Fixed-Pinned Point Load (70 psi in 17.8 ms) 5% Damping 30 psf Added Weight



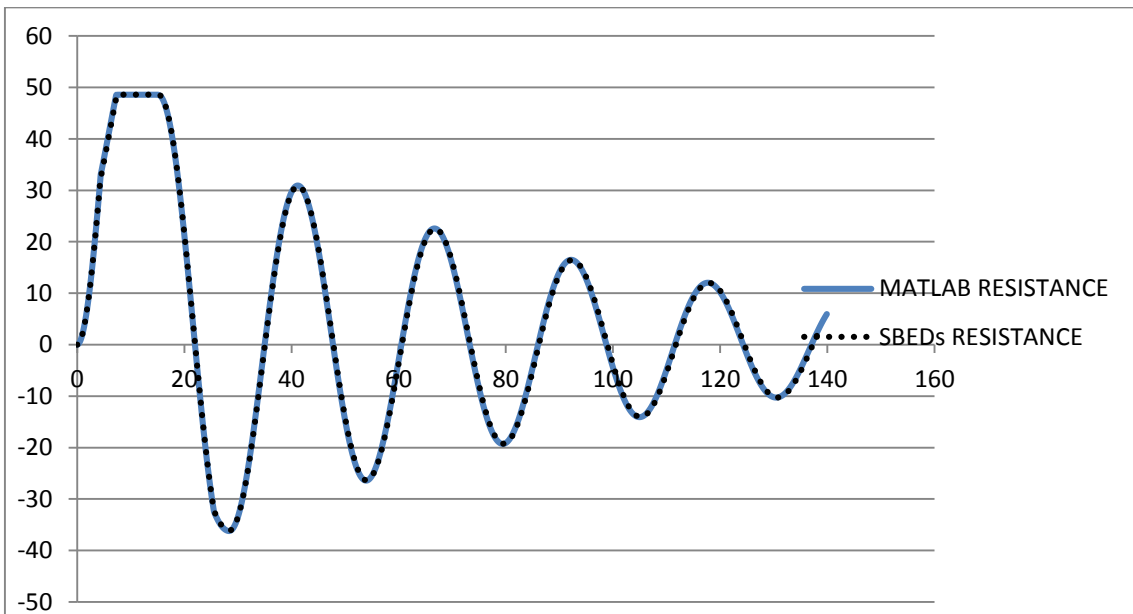
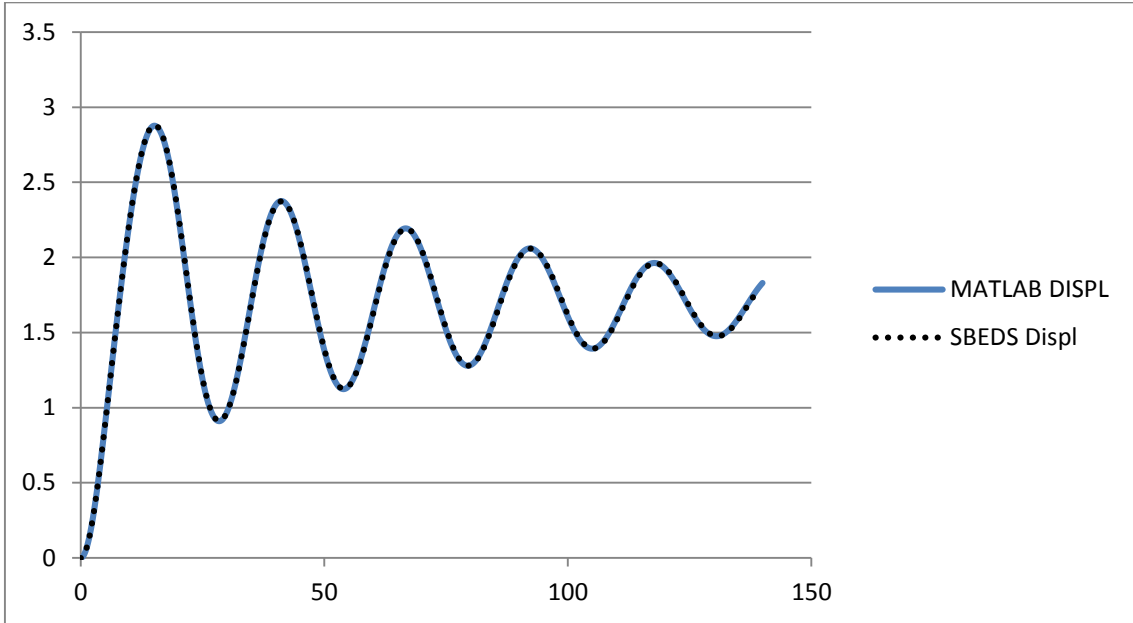
W10X12 Fixed-Pinned Uniform Load (70 psi in 17.8 ms) 0% Damping 30 psf Added

Weight

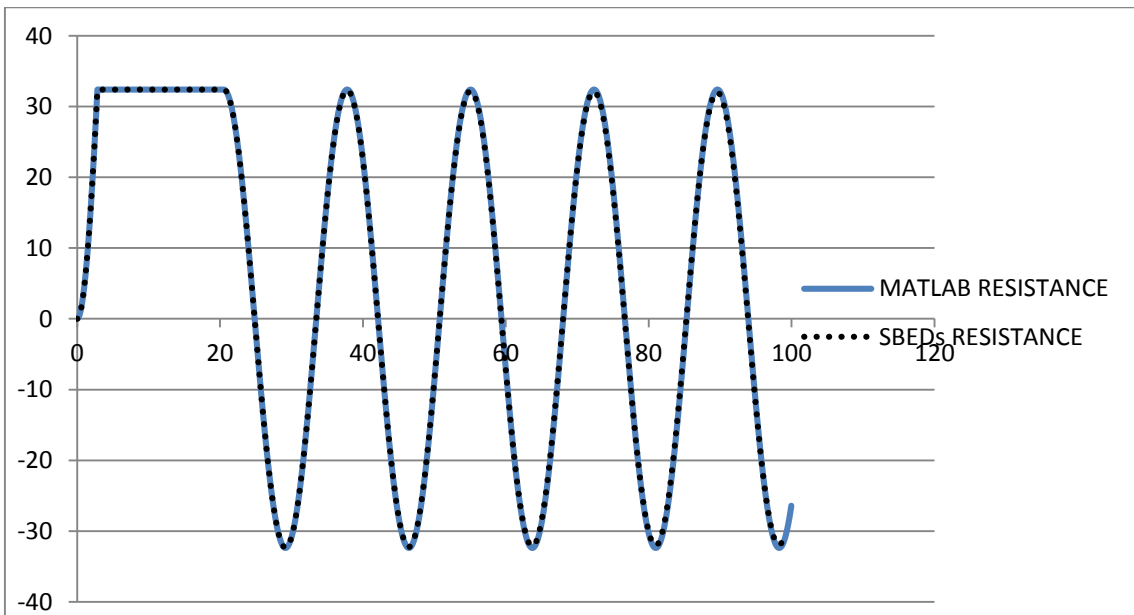
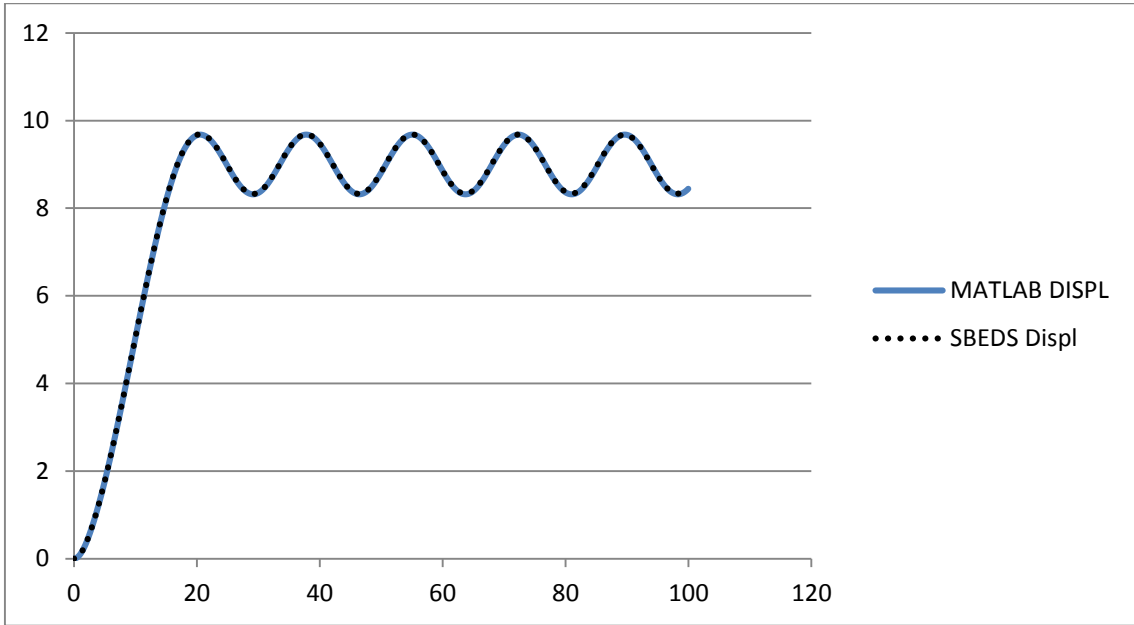


W10X12 Fixed-Pinned Uniform Load (70 psi in 17.8 ms) 5% Damping 30 psf Added

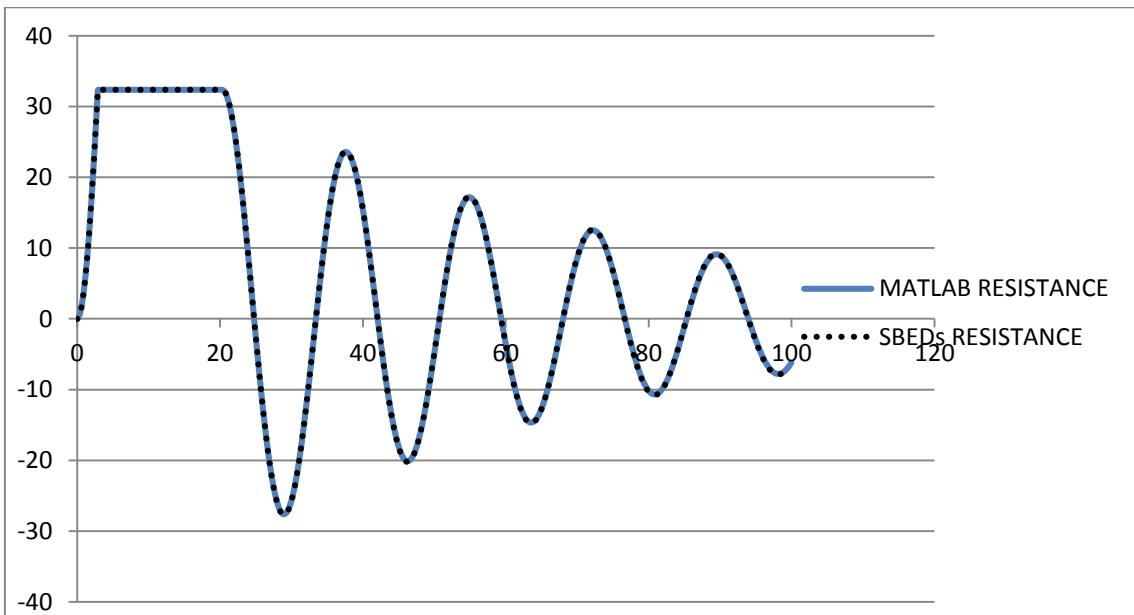
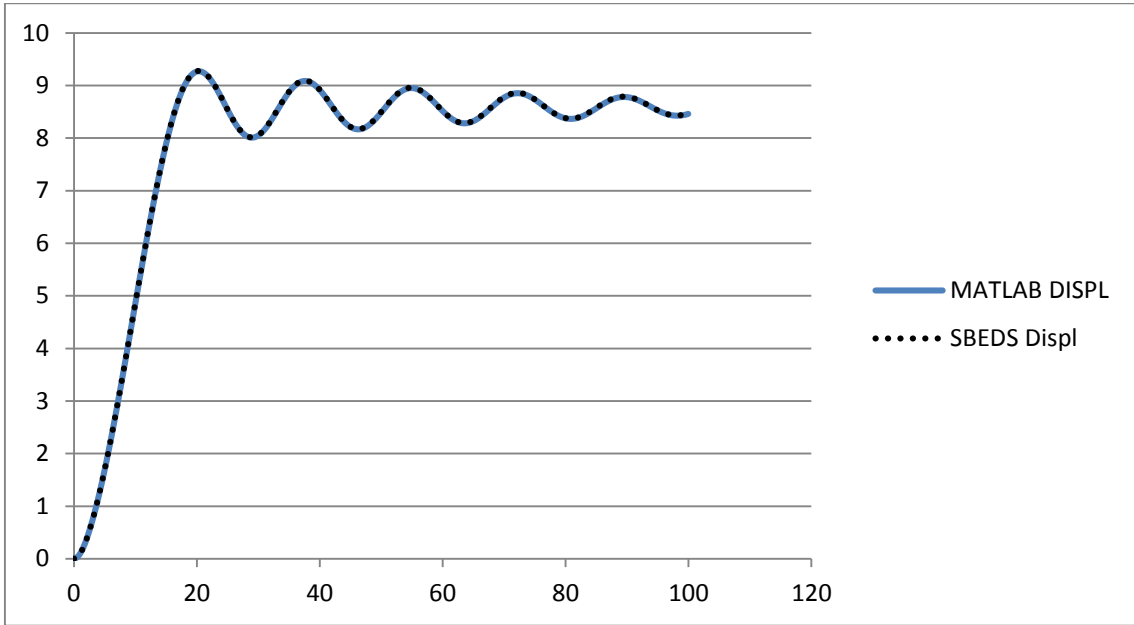
Weight



W10X12 Fixed-Fixed Point Load (70 psi in 17.8 ms) 0% Damping 30 psf Added Weight

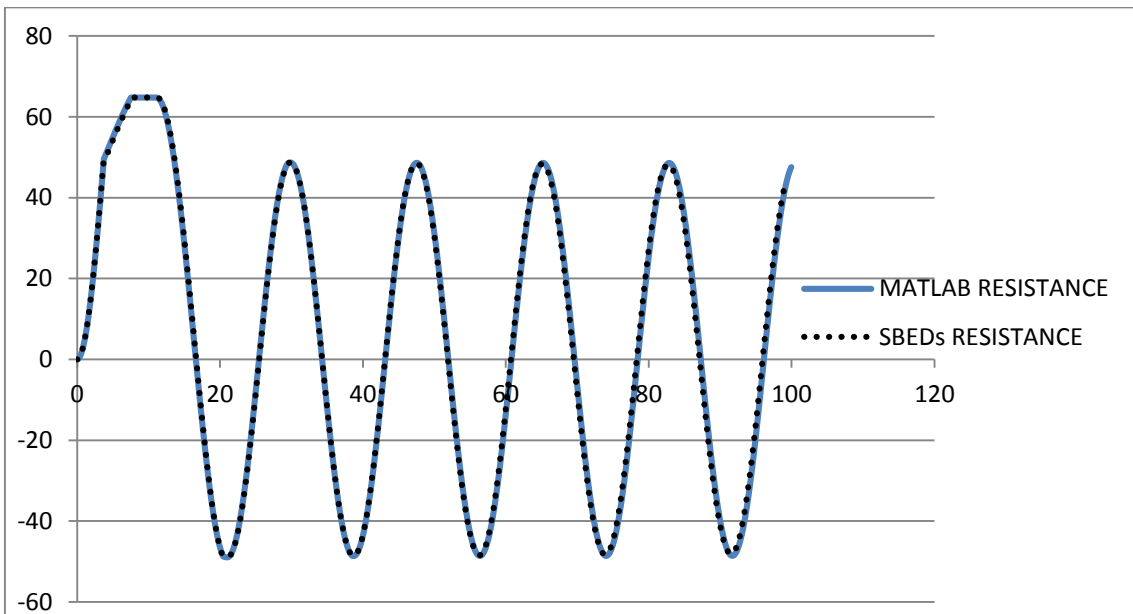
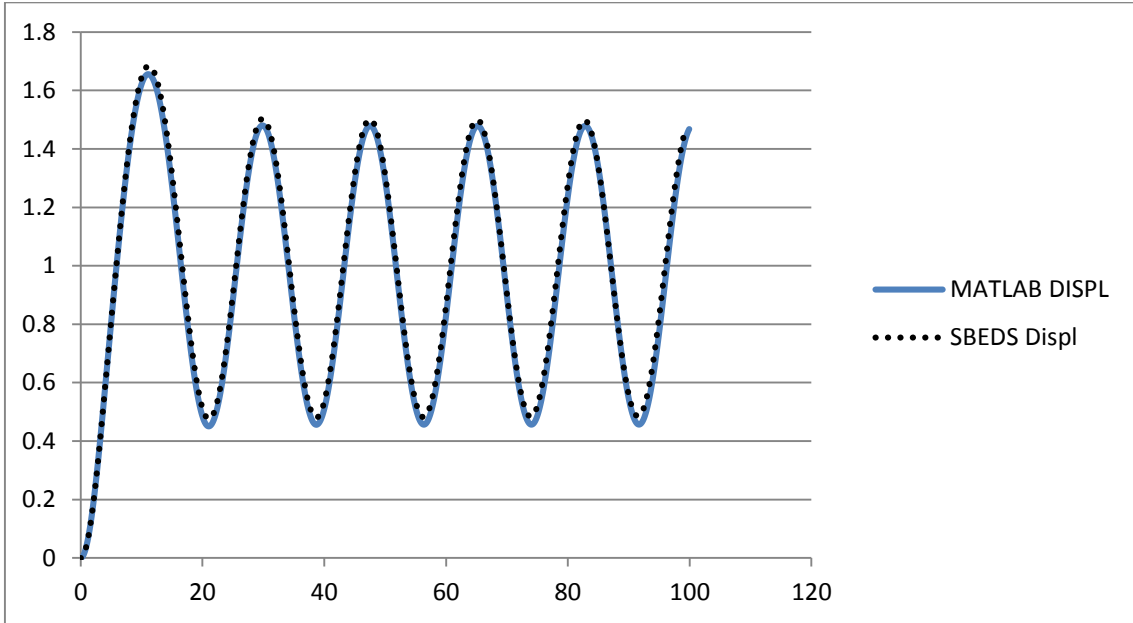


W10X12 Fixed-Fixed Point Load (70 psi in 17.8 ms) 5% Damping 30 psf Added Weight



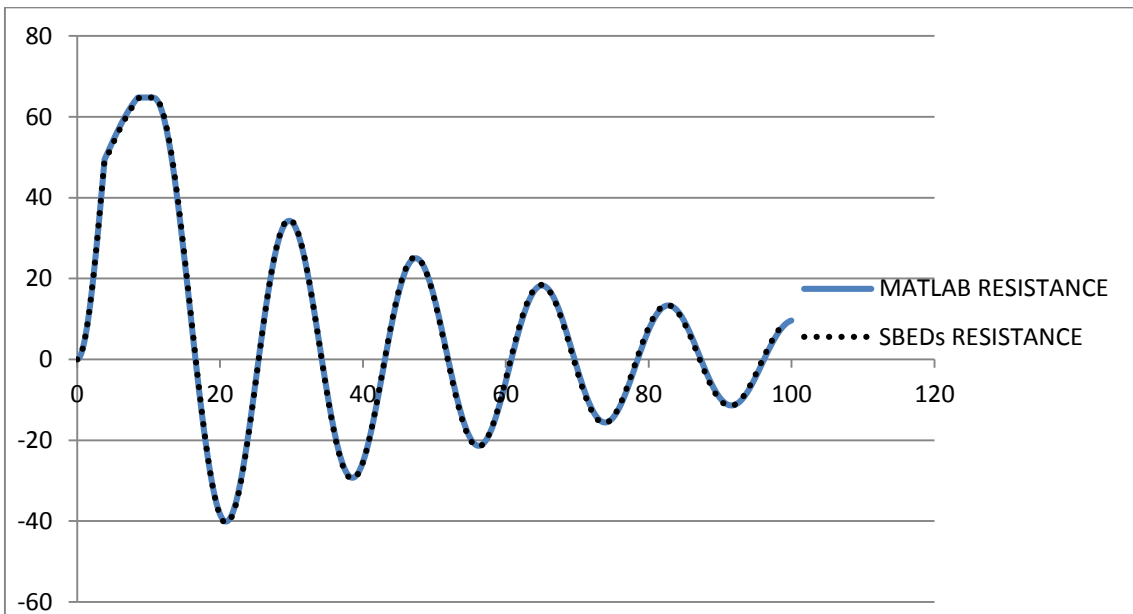
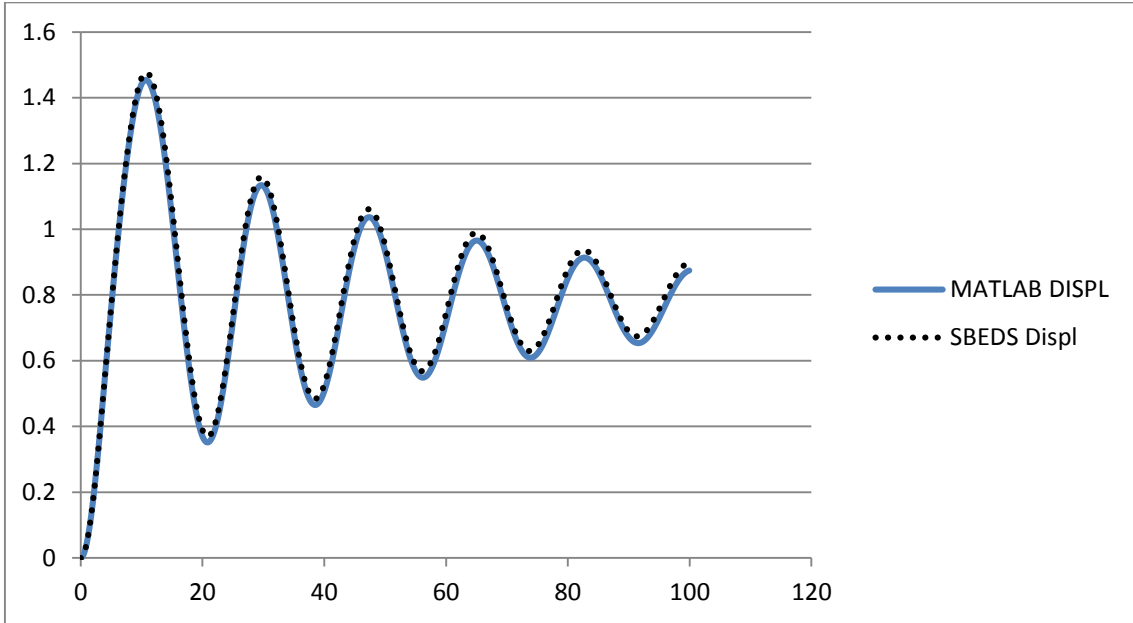
W10X12 Fixed-Fixed Uniform Load (70 psi in 17.8 ms) 0% Damping 30 psf Added

Weight

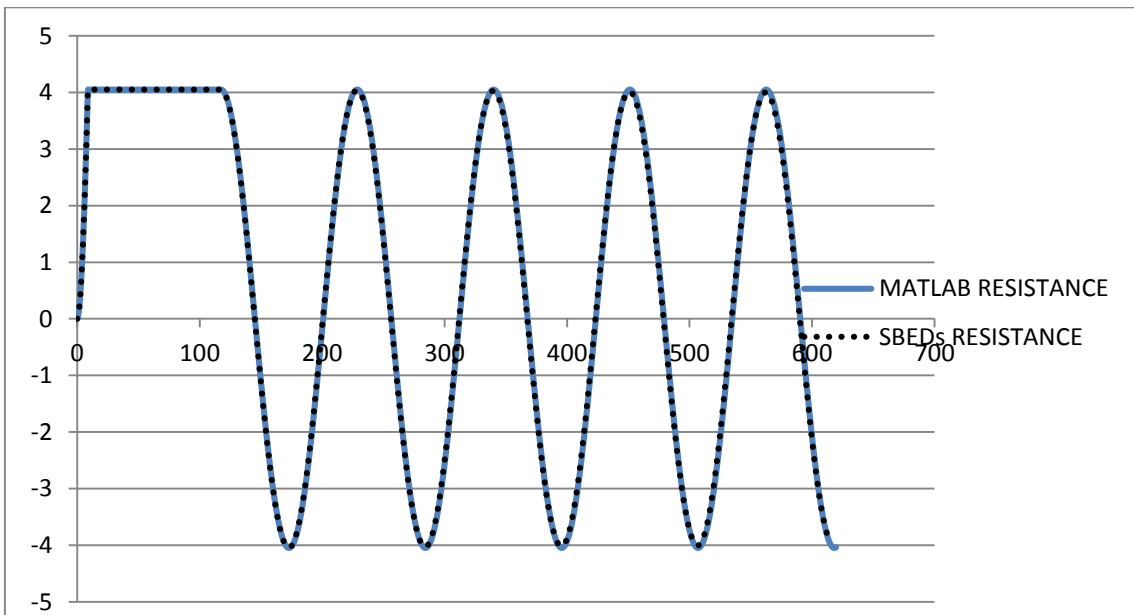
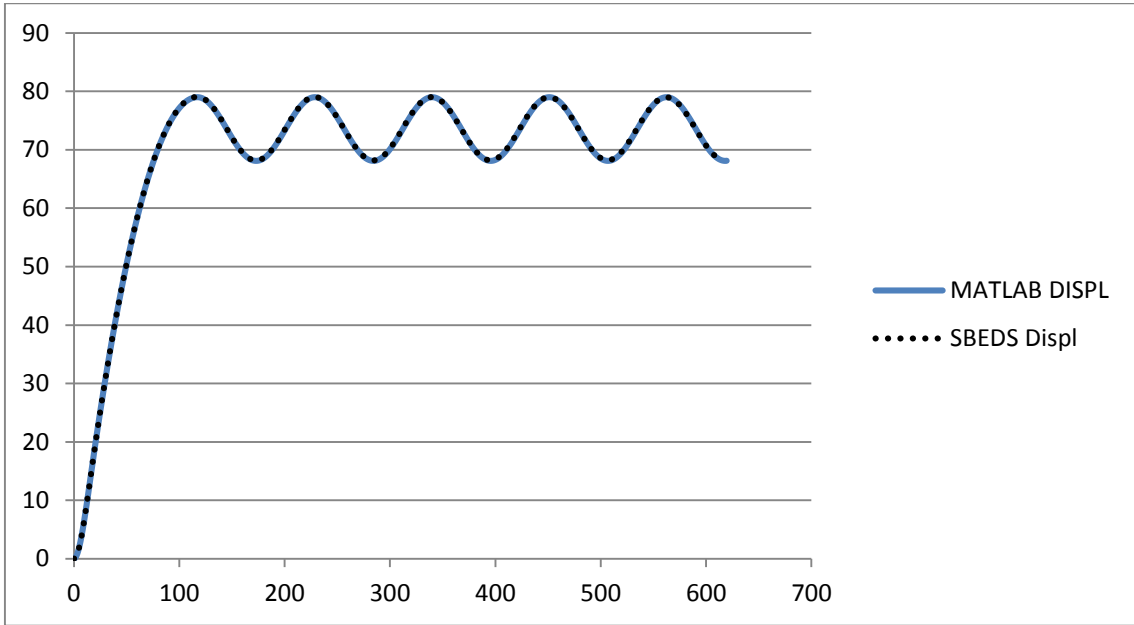


W10X12 Fixed-Fixed Uniform Load (70 psi in 17.8 ms) 5% Damping 30 psf Added

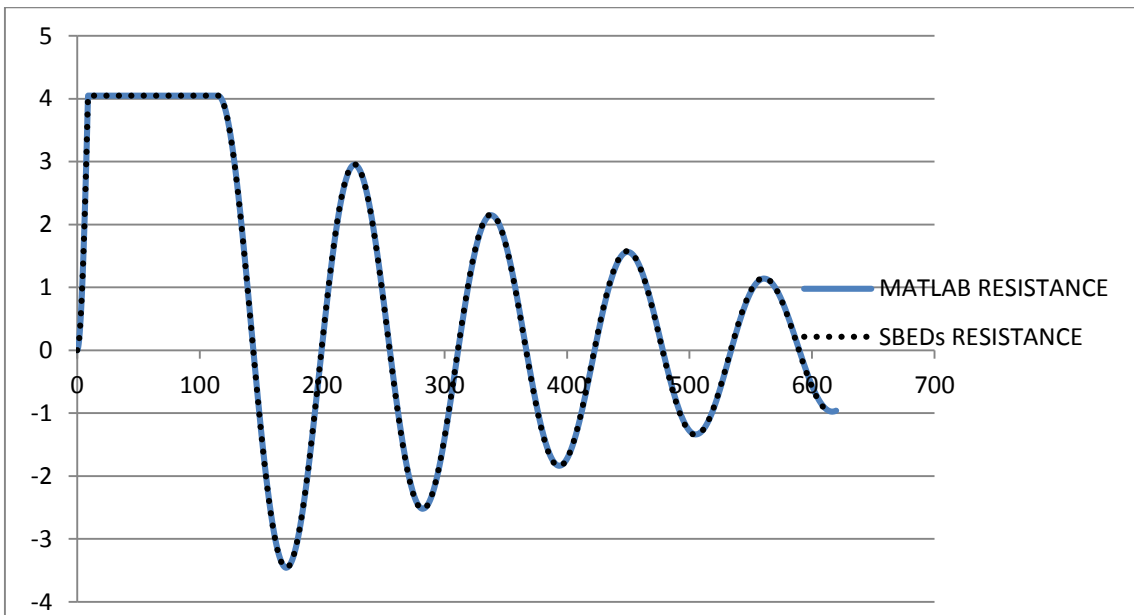
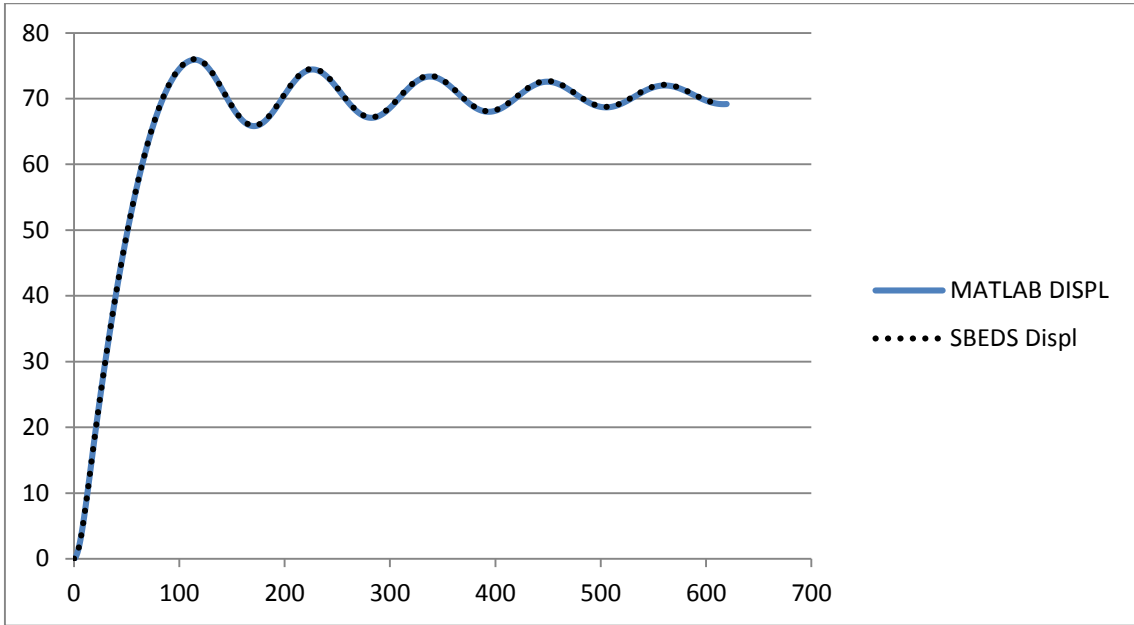
Weight



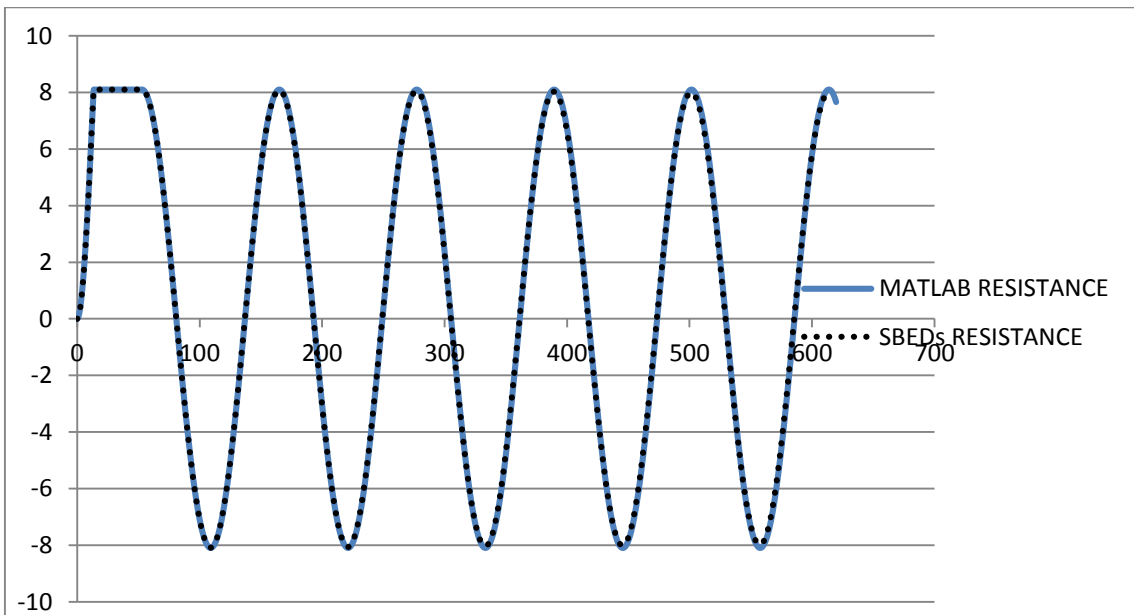
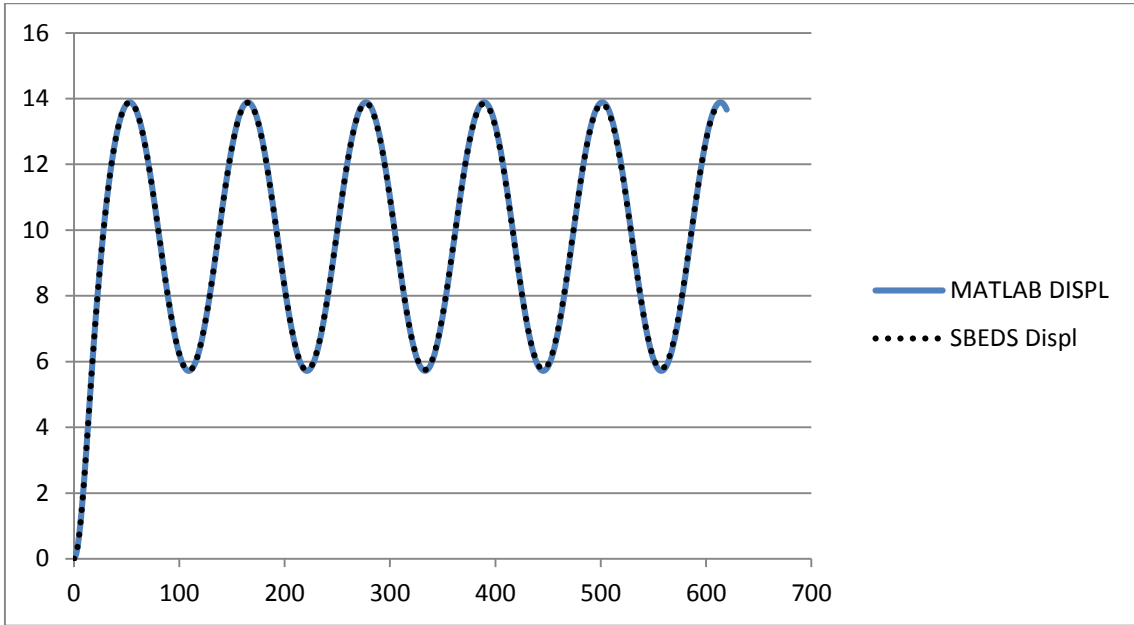
W10X12 Cantilever Point Load (40 psi in 17.8 ms) 0% Damping 30 psf Added Weight



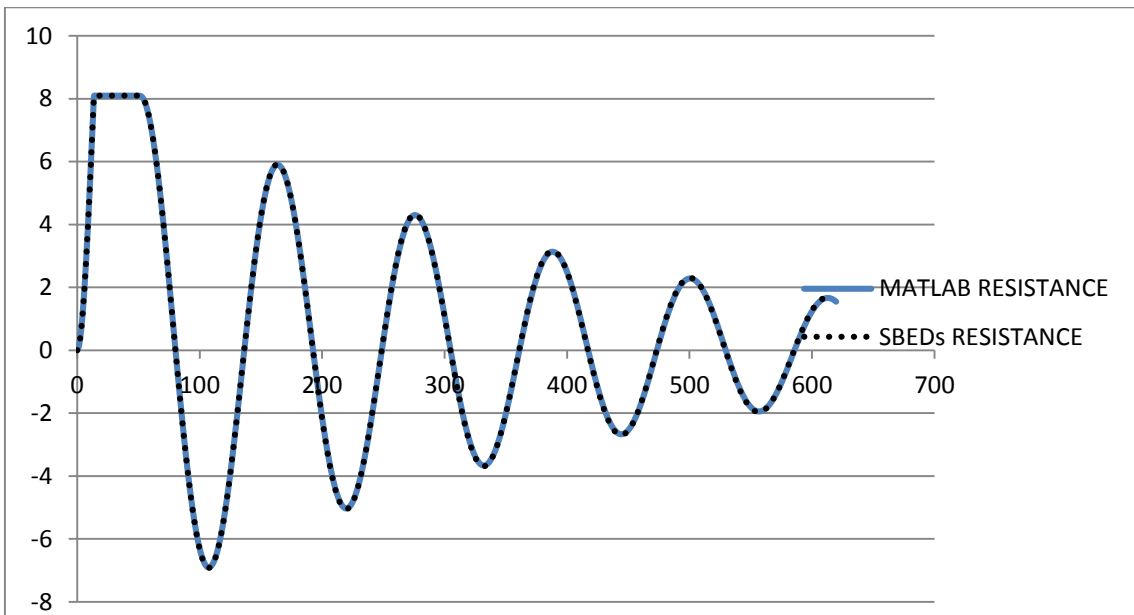
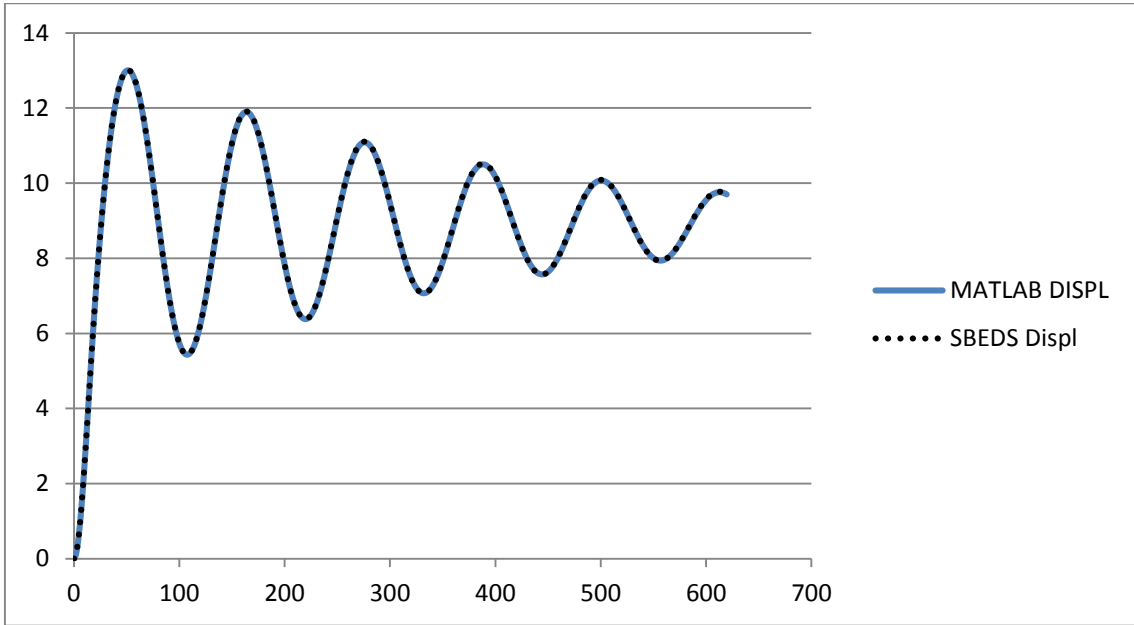
W10X12 Cantilever Point Load (40 psi in 17.8 ms) 5% Damping 30 psf Added Weight



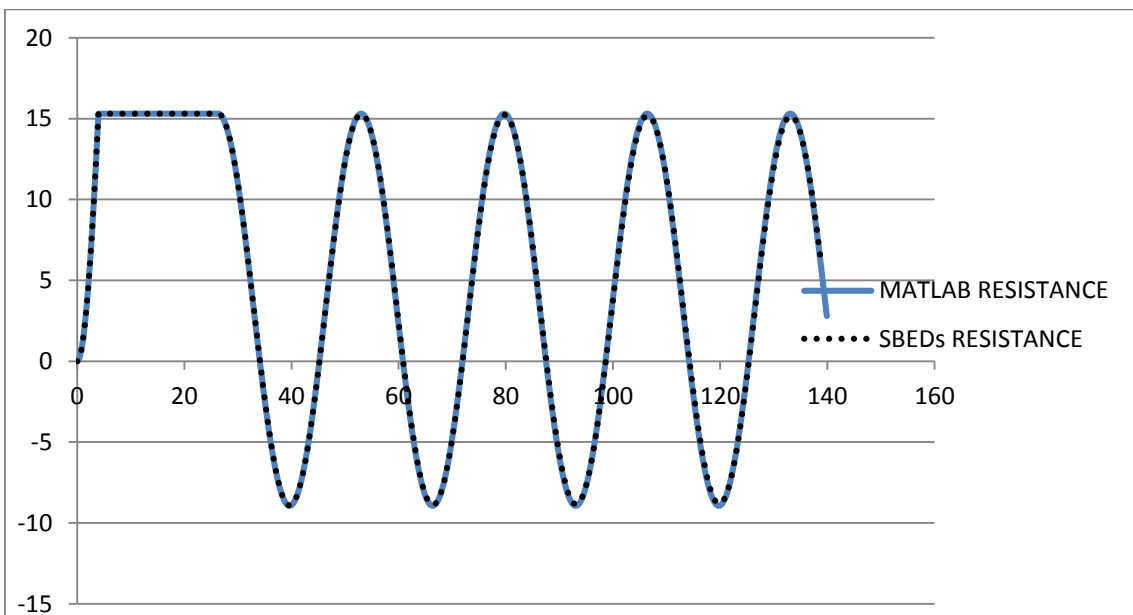
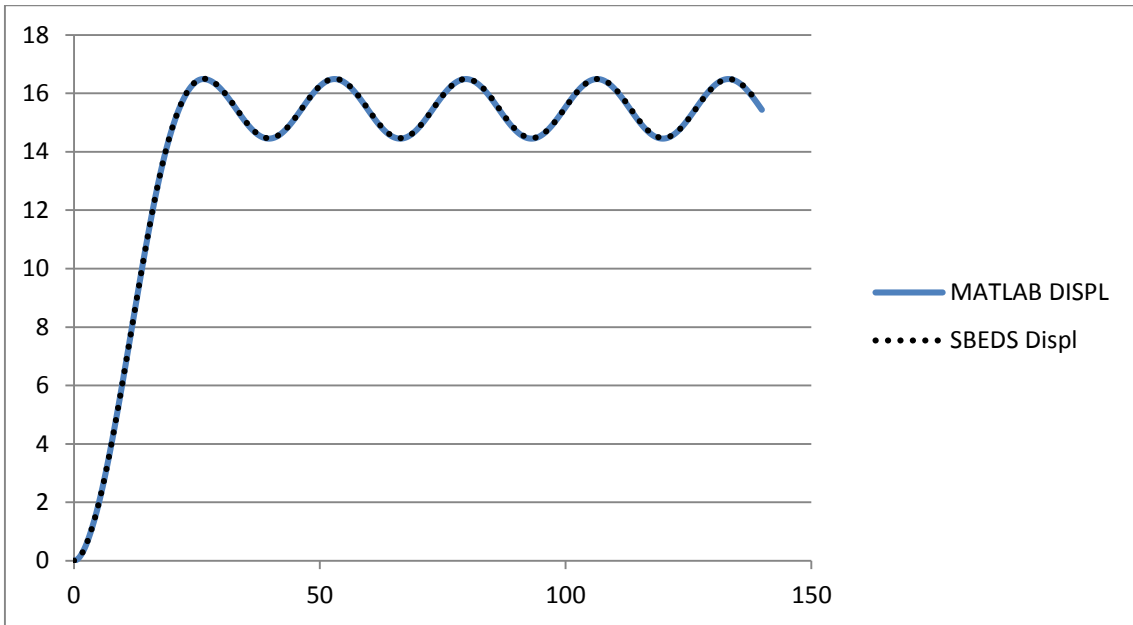
W10X12 Cantilever Uniform Load (40 psi in 17.8 ms) 0% Damping 30 psf Added Weight



W10X12 Cantilever Uniform Load (40 psi in 17.8 ms) 5% Damping 30 psf Added Weight

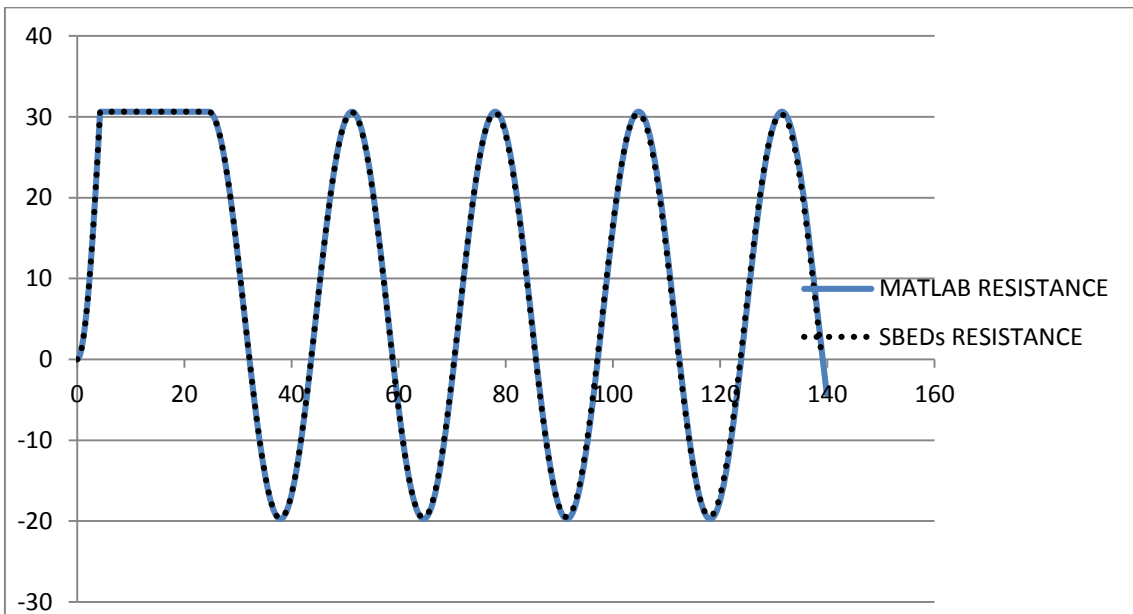
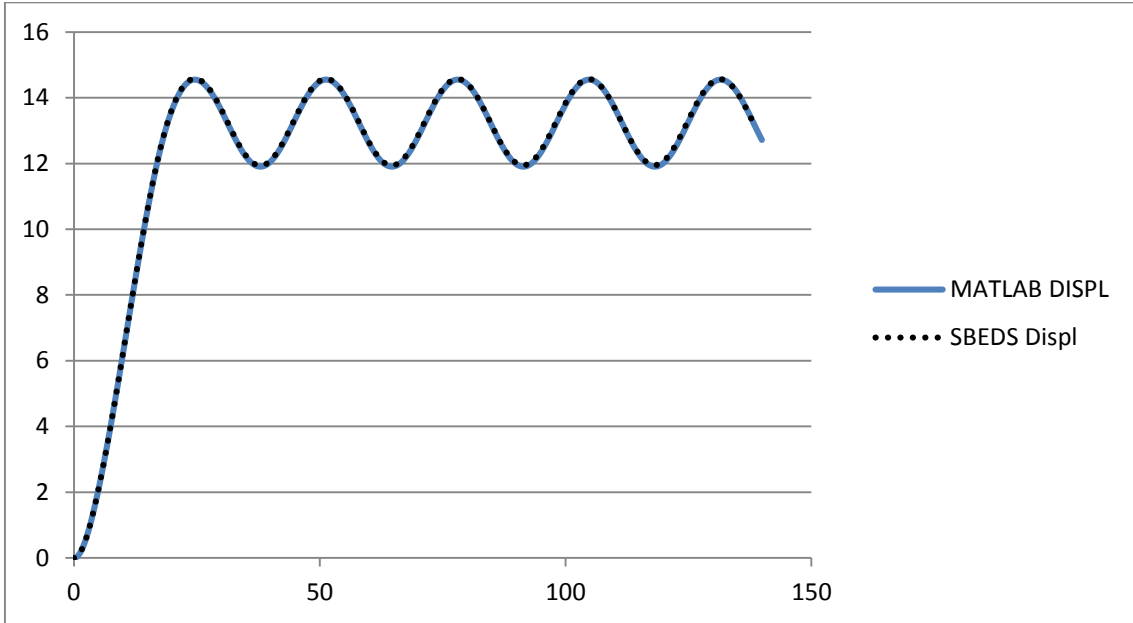


W10X12 Pinned-Pinned Point Load (40 psi in 17.8 ms) 0% Damping 10 kip Axial Load

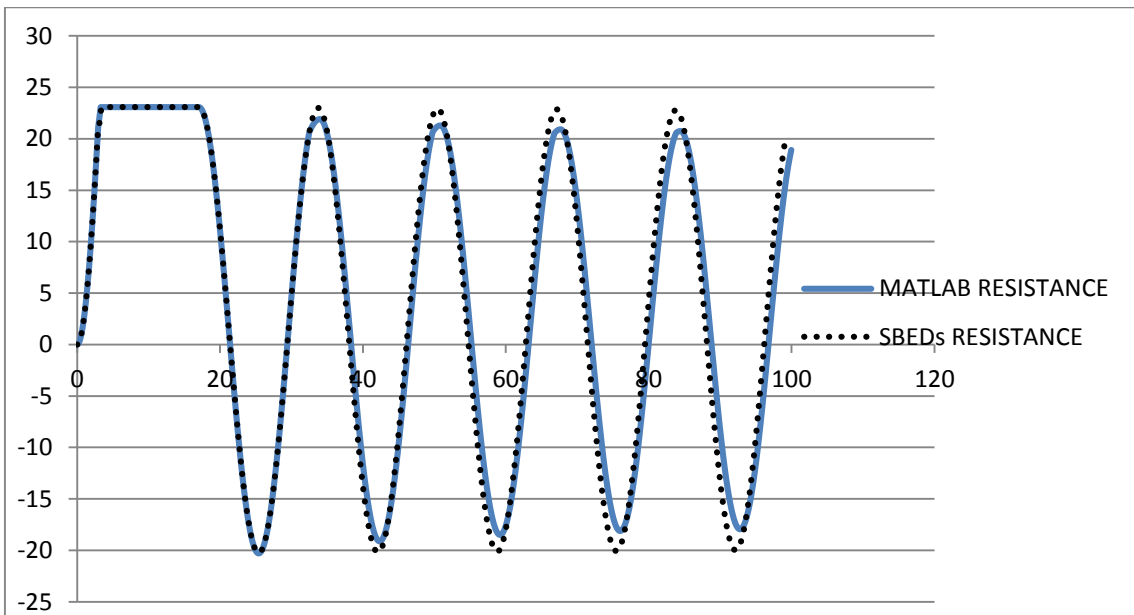
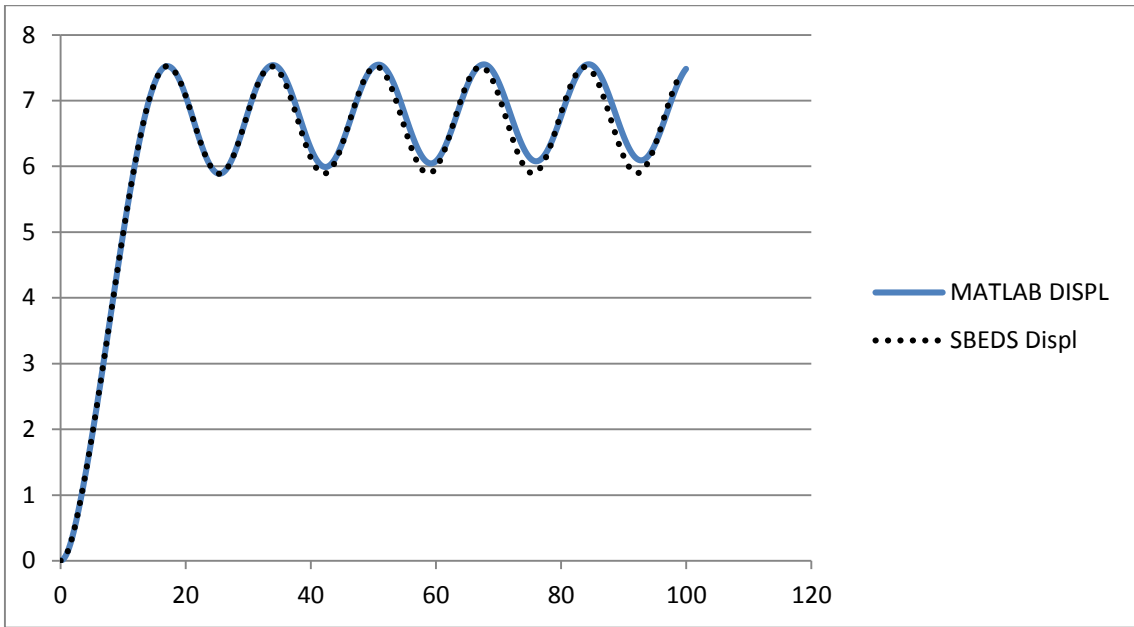


W10X12 Pinned-Pinned Uniform Load (70 psi in 17.8 ms) 0% Damping 10 kip Axial

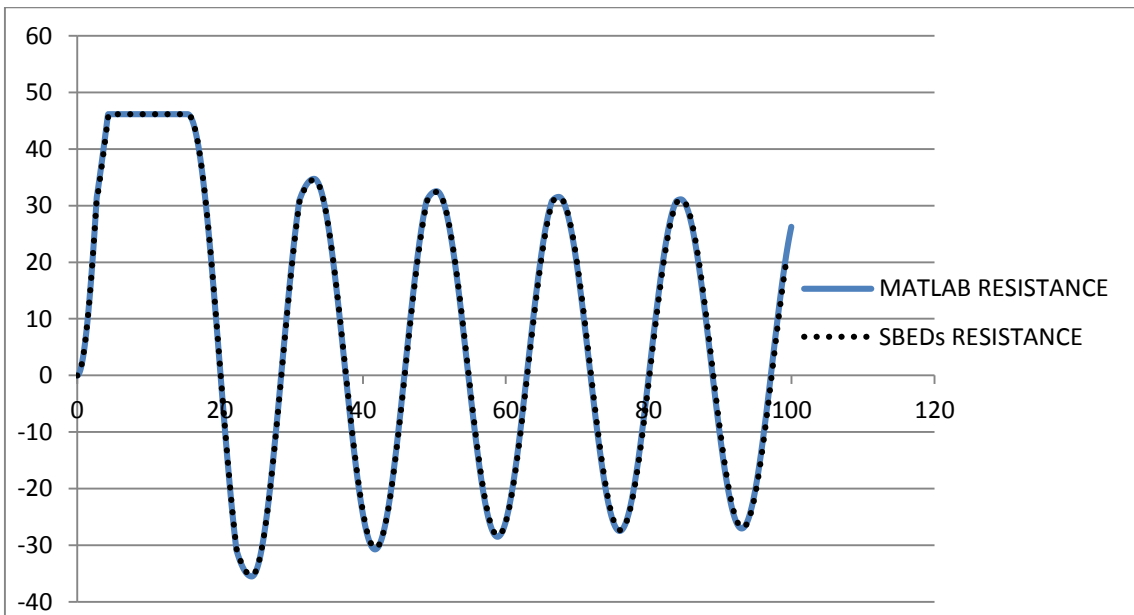
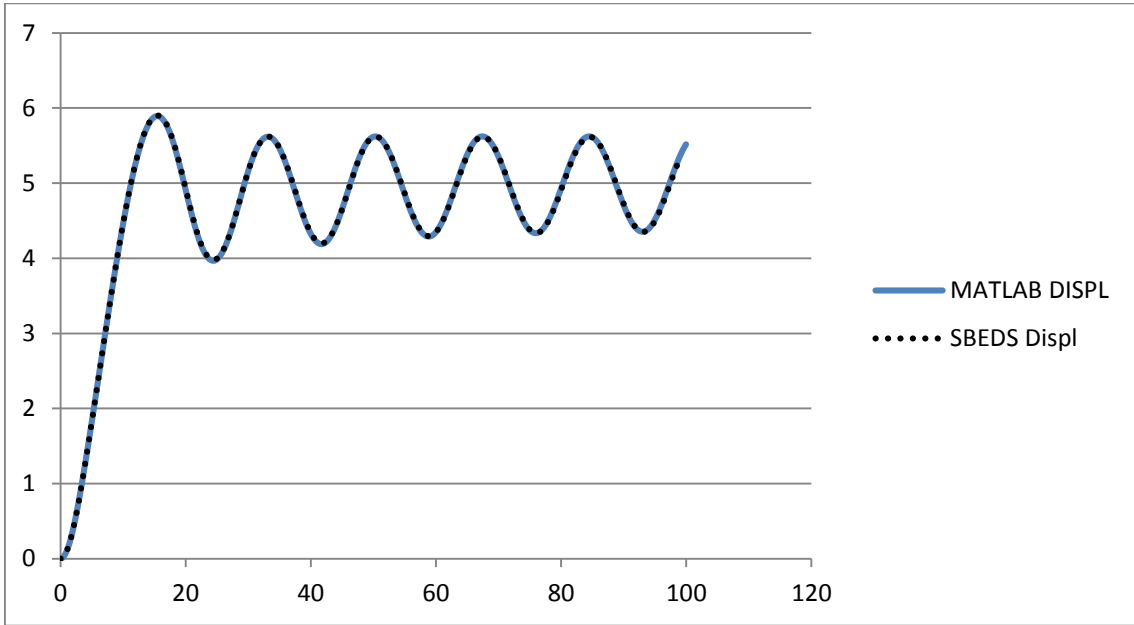
Load



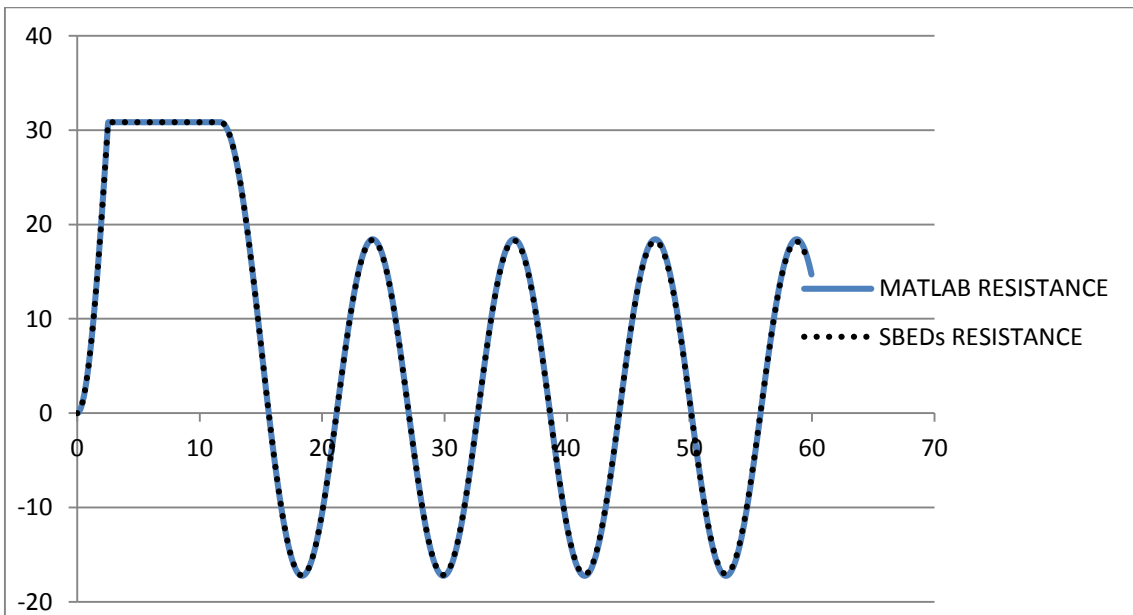
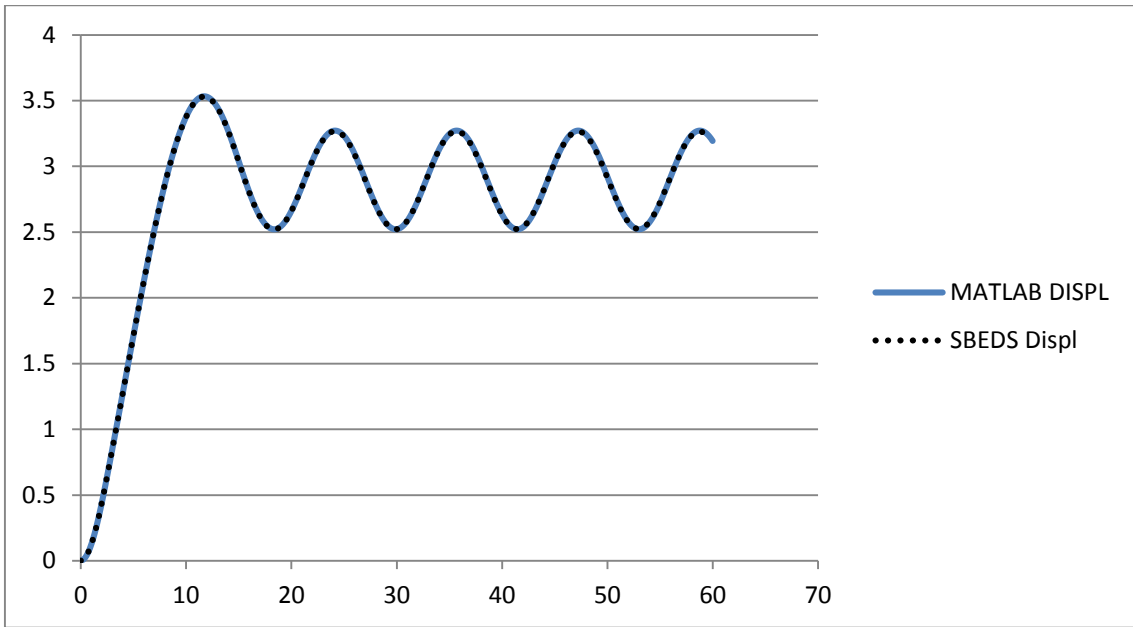
W10X12 Fixed-Pinned Point Load (40 psi in 17.8 ms) 0% Damping 10 kip Axial Load



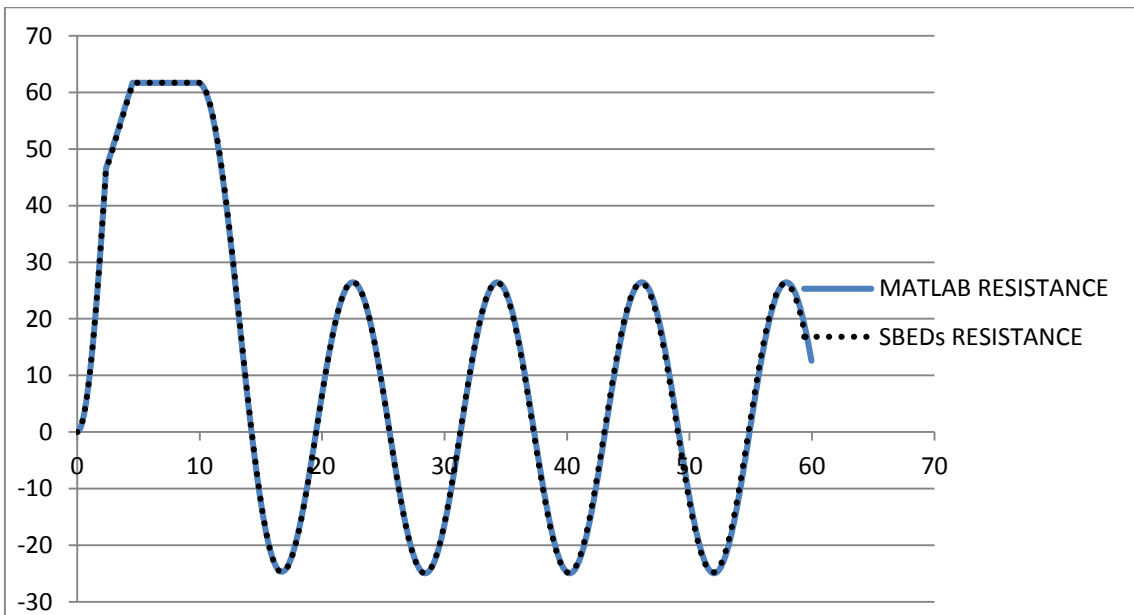
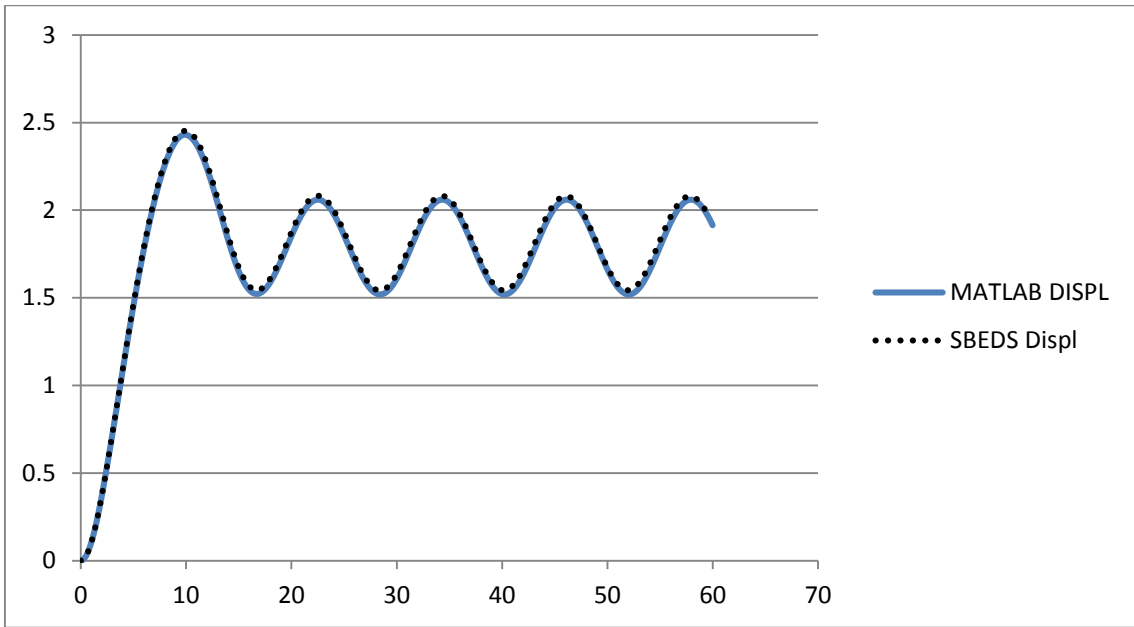
W10X12 Fixed-Pinned Uniform Load (70 psi in 17.8 ms) 0% Damping 10 kip Axial Load



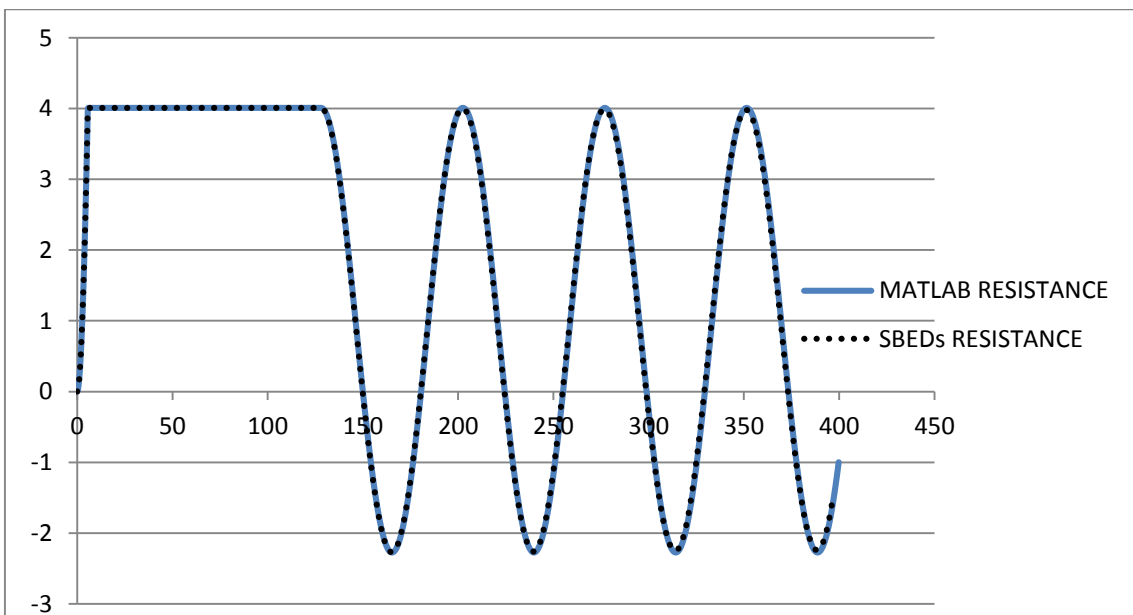
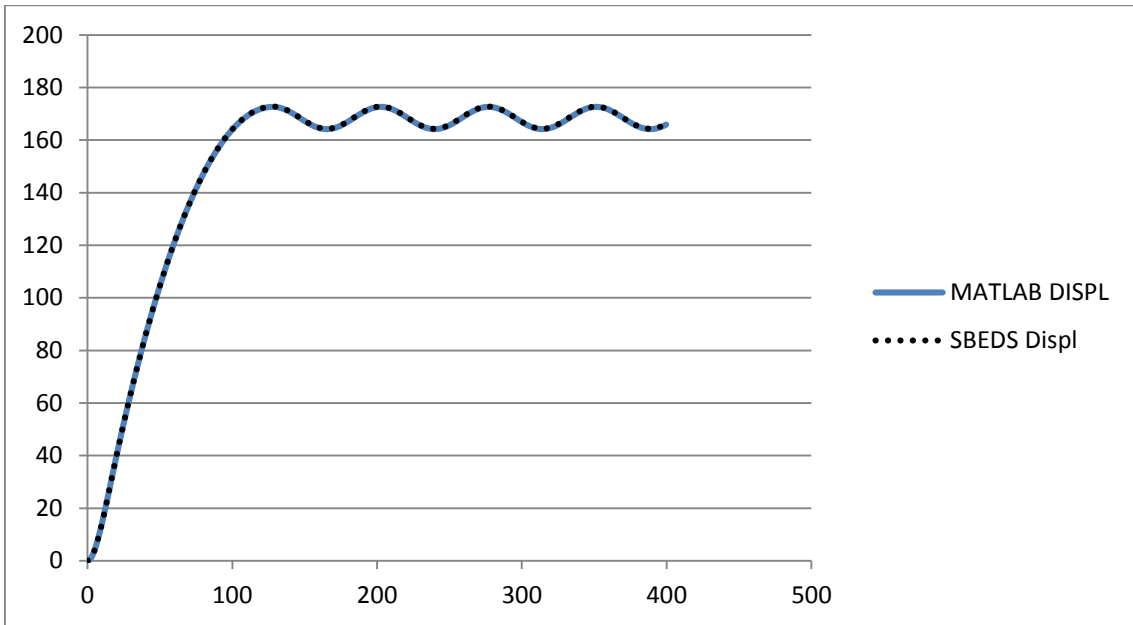
W10X12 Fixed-Fixed Point Load (40 psi in 17.8 ms) 0% Damping 10 kip Axial Load



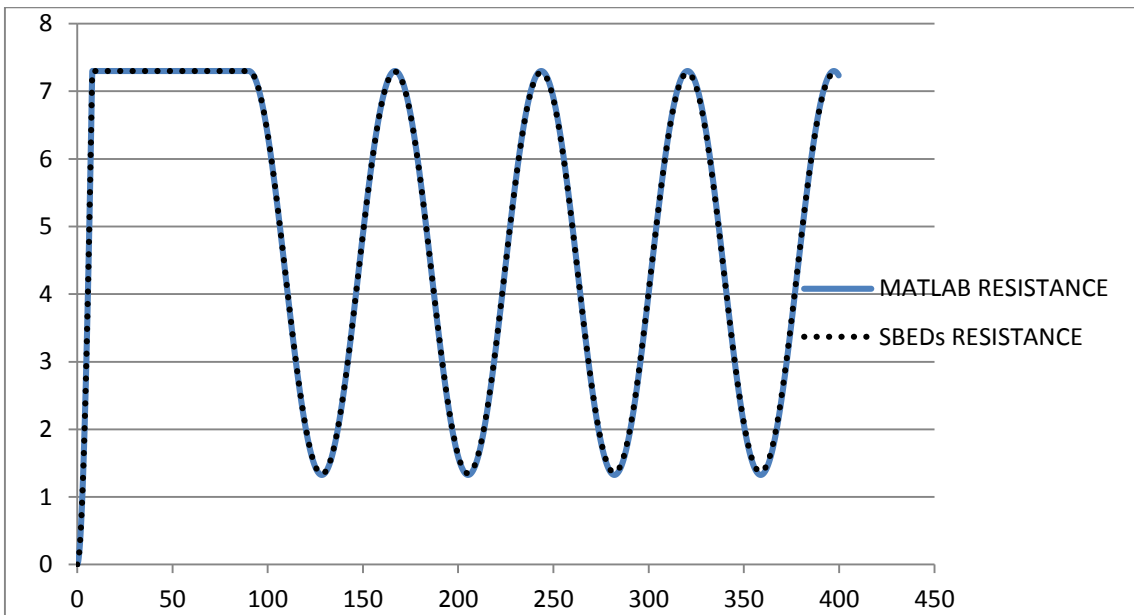
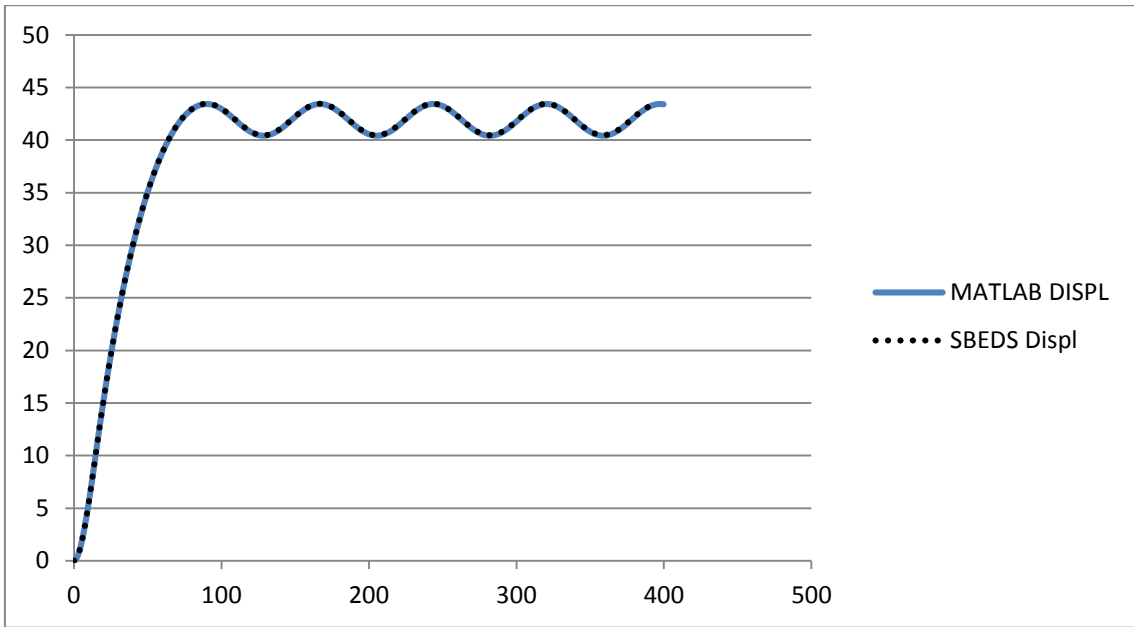
W10X12 Fixed-Fixed Uniform Load (70 psi in 17.8 ms) 0% Damping 10 kip Axial Load



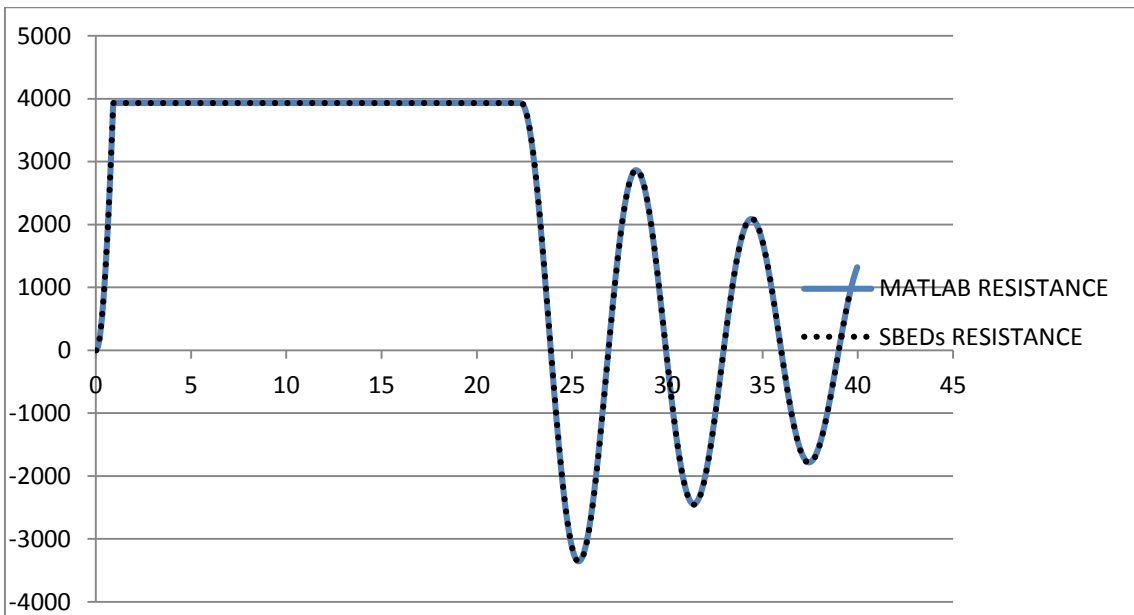
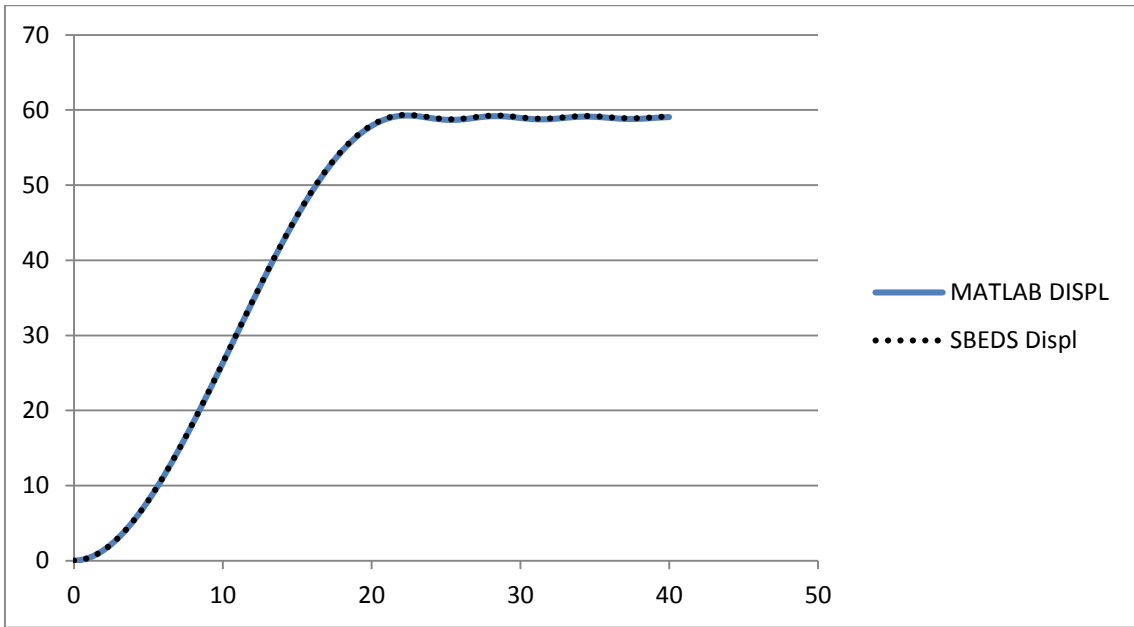
W10X12 Cantilever Point Load (40 psi in 17.8 ms) 0% Damping 1 kip Axial Load



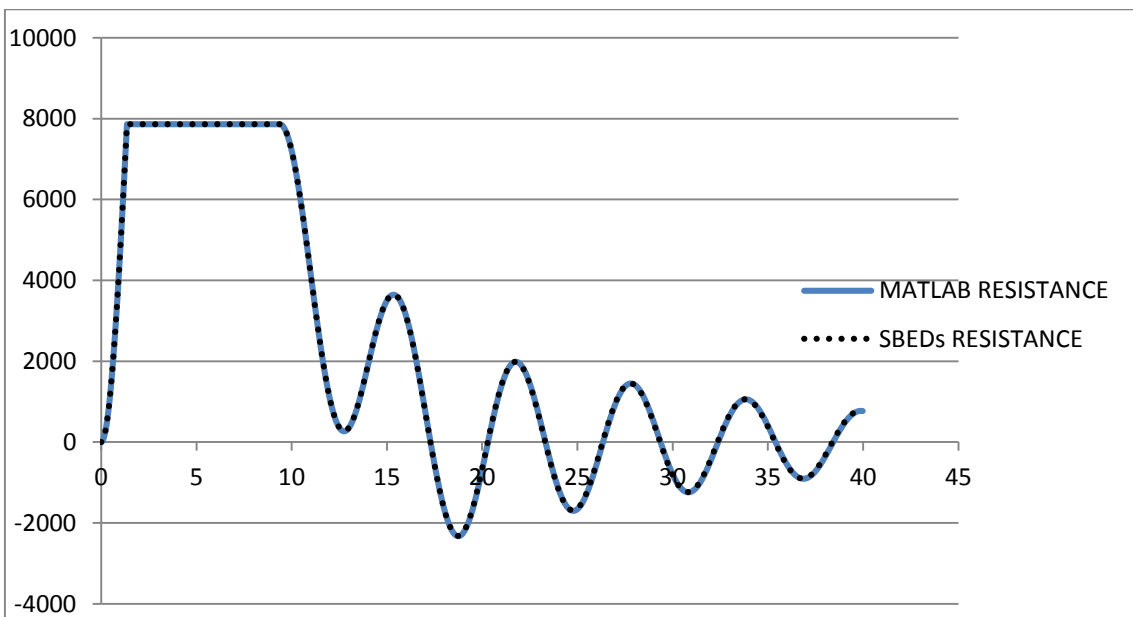
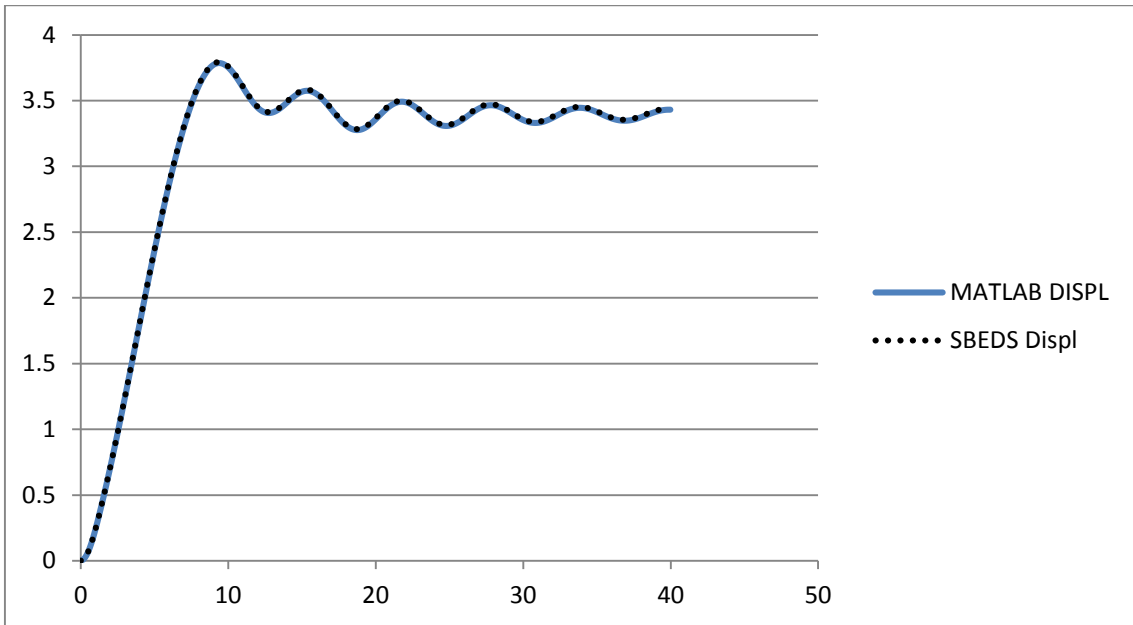
W10X12 Cantilever Uniform Load (40 psi in 17.8 ms) 0% Damping 10 kip Axial Load



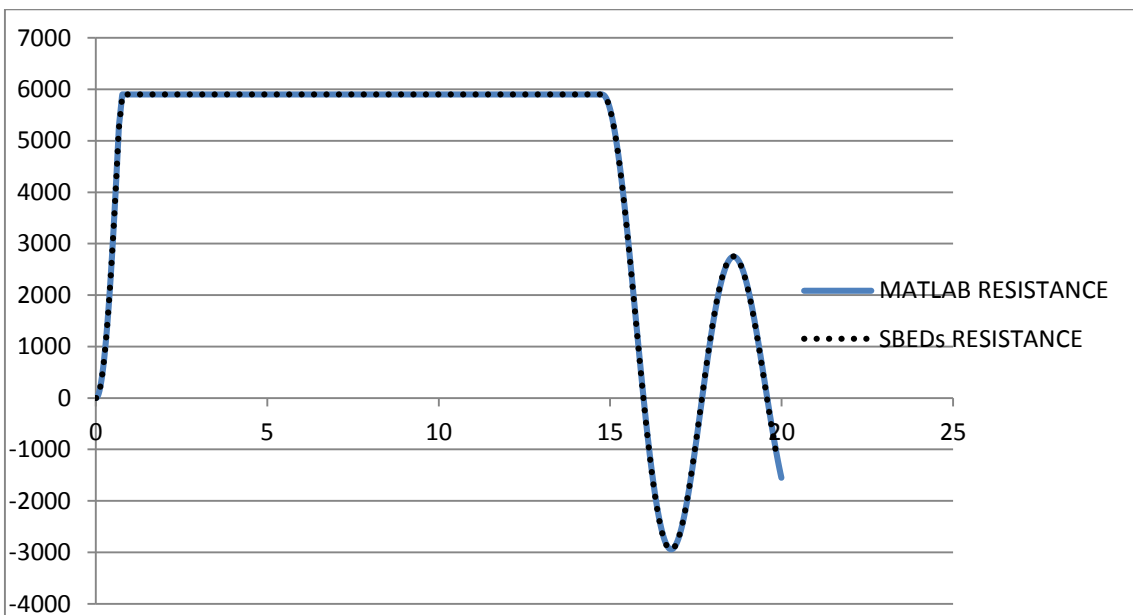
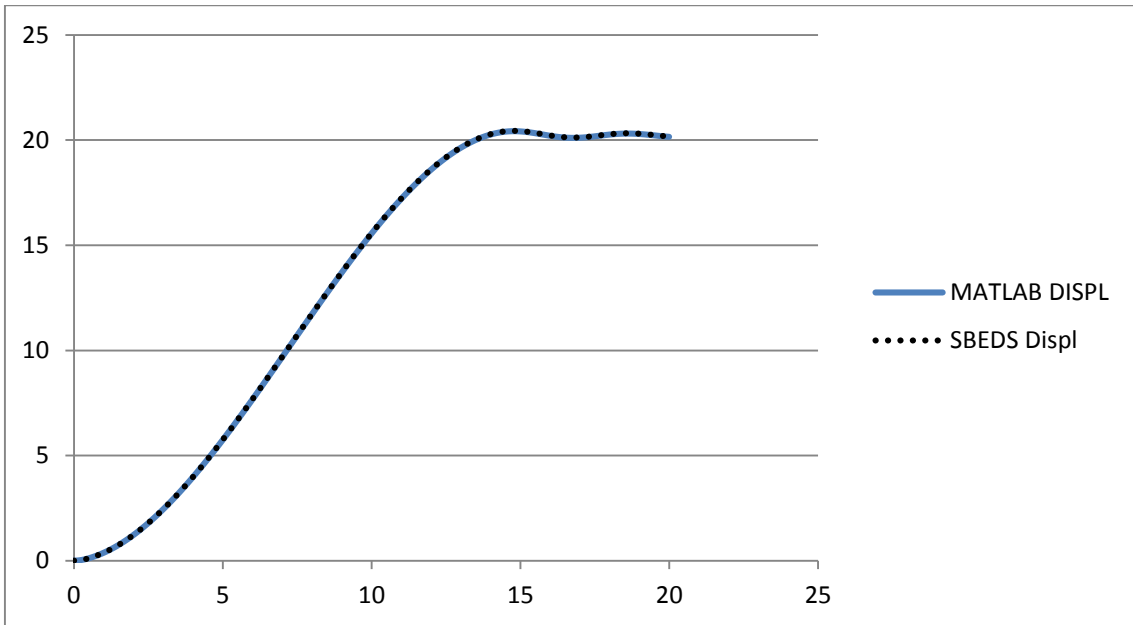
W40X65 Pinned-Pinned Point Load (10,000 psi in 17.8 ms) 5% Damping



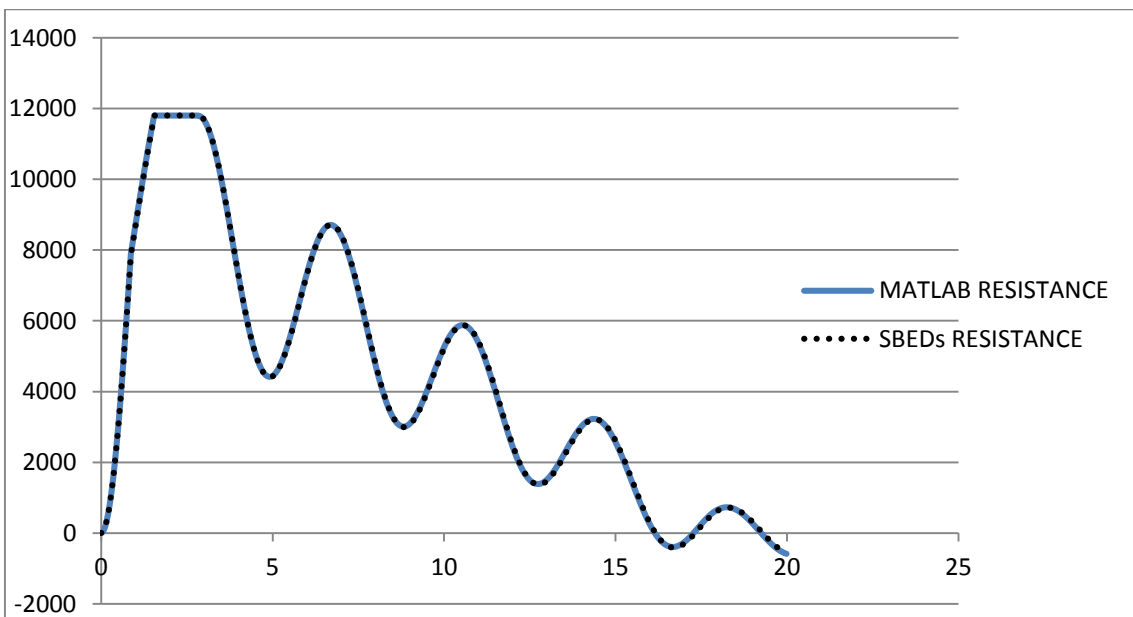
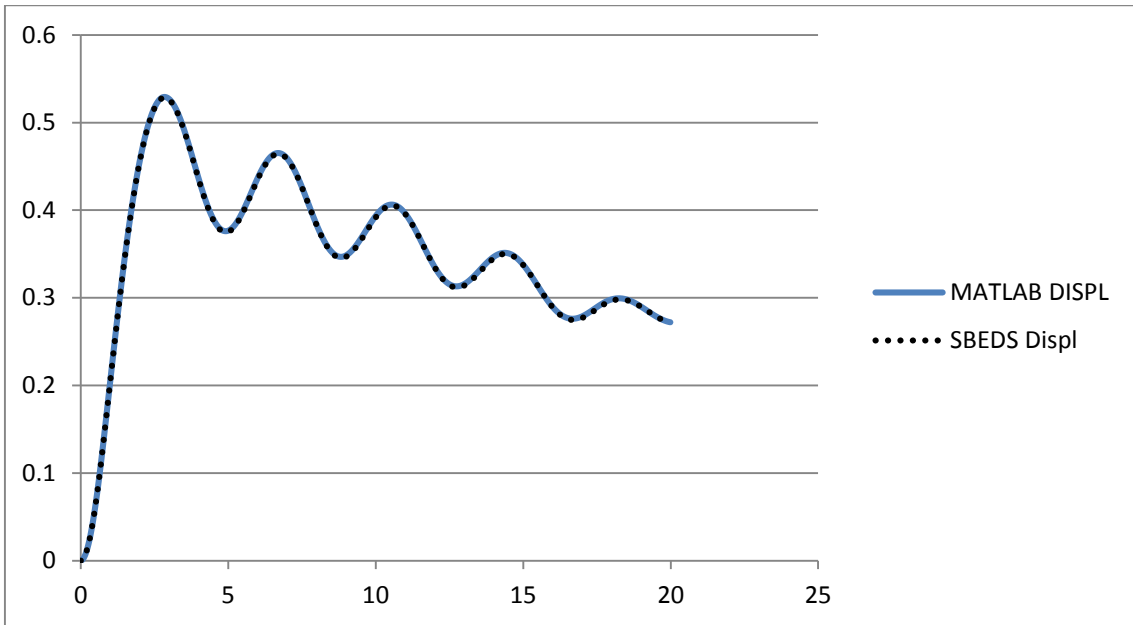
W40X655 Pinned-Pinned Uniform Load (10,000 psi in 17.8 ms) 5% Damping



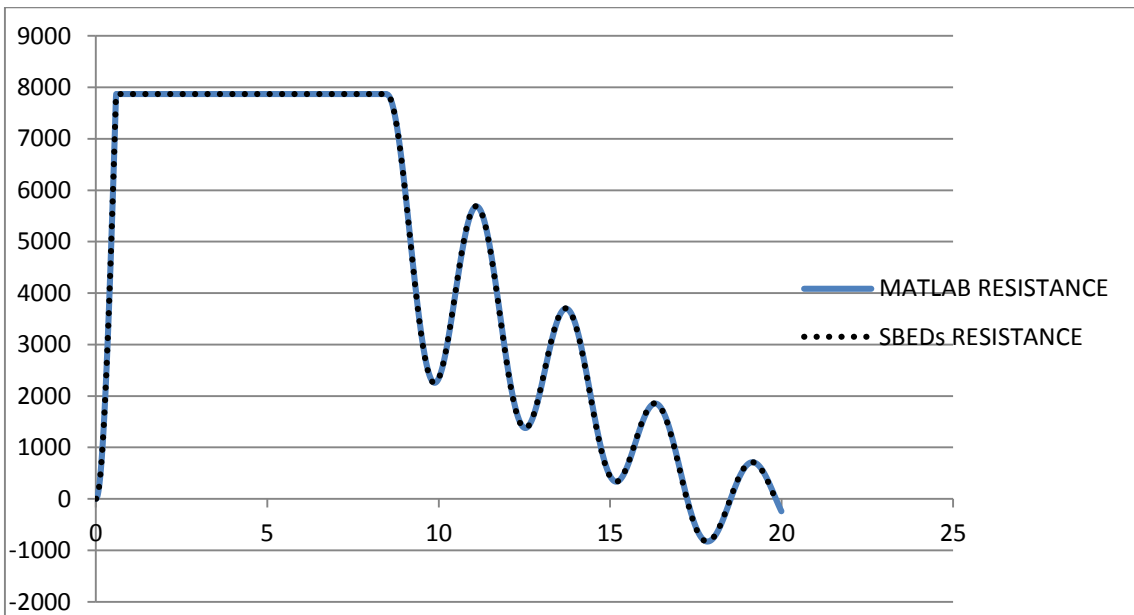
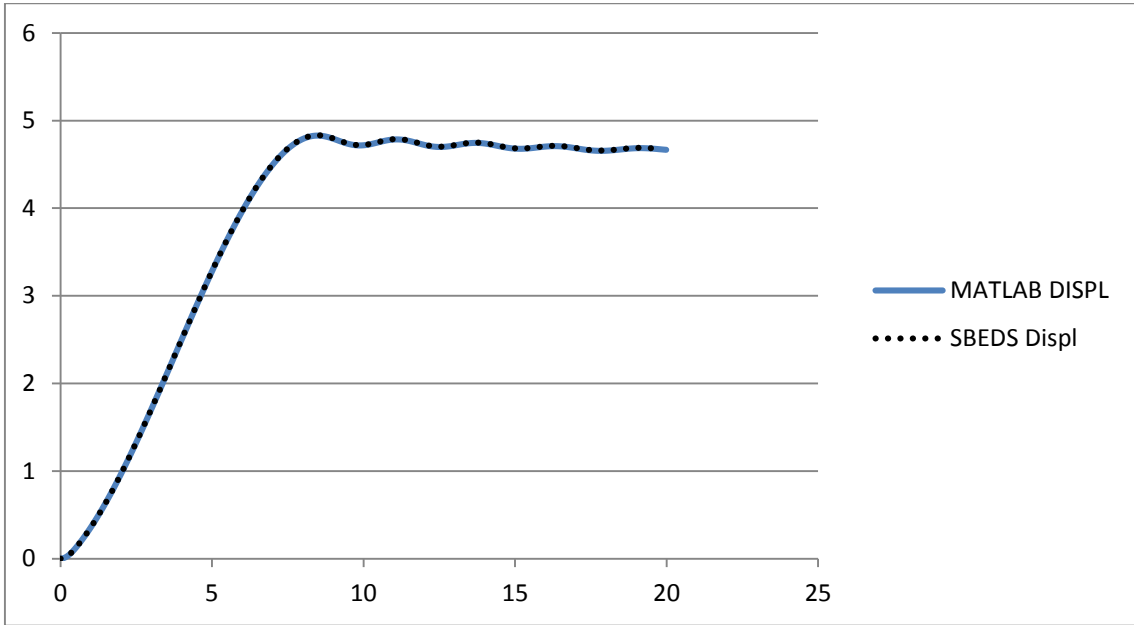
W40X65 Fixed-Pinned Point Load (10,000 psi in 17.8 ms) 5% Damping



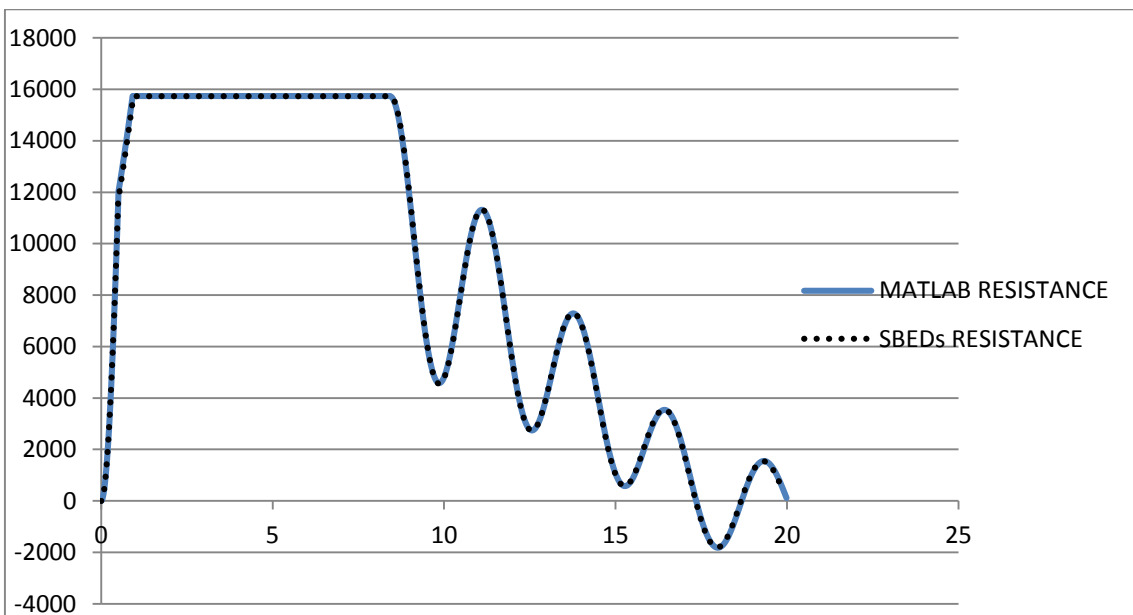
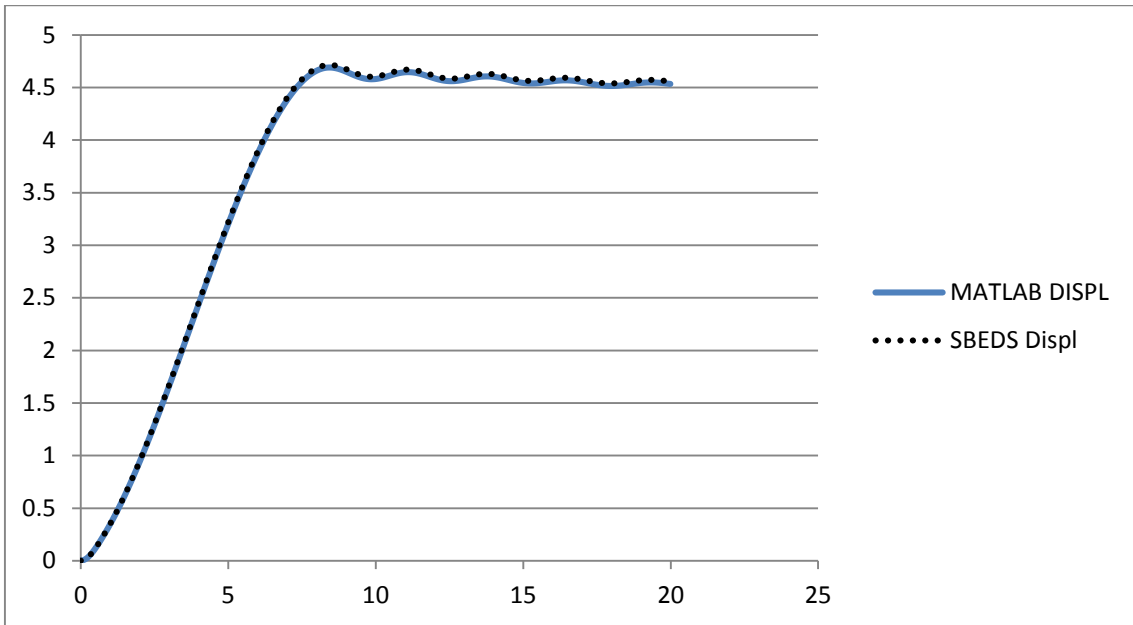
W40X65 Fixed-Pinned Uniform Load (10,000 psi in 17.8 ms) 5% Damping



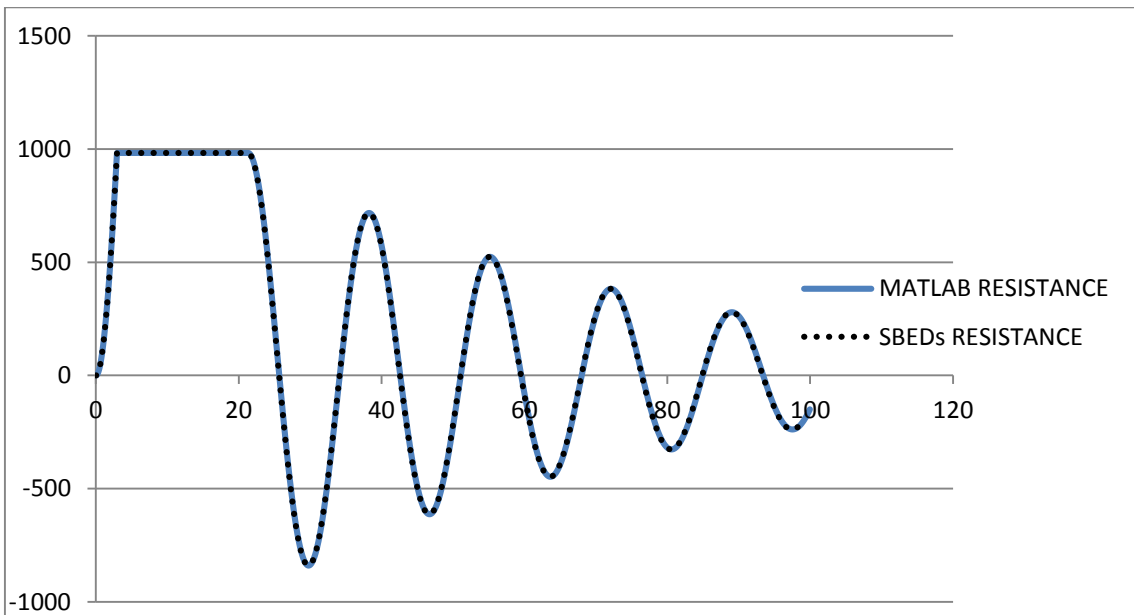
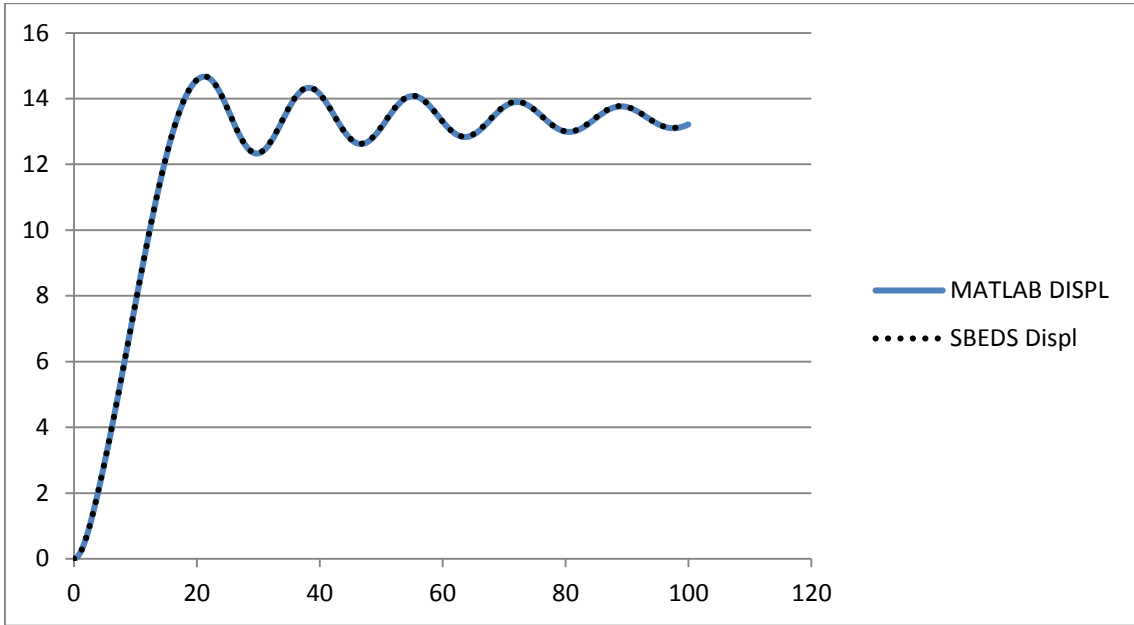
W40X65 Fixed-Fixed Point Load (10,000 psi in 17.8 ms) 5% Damping



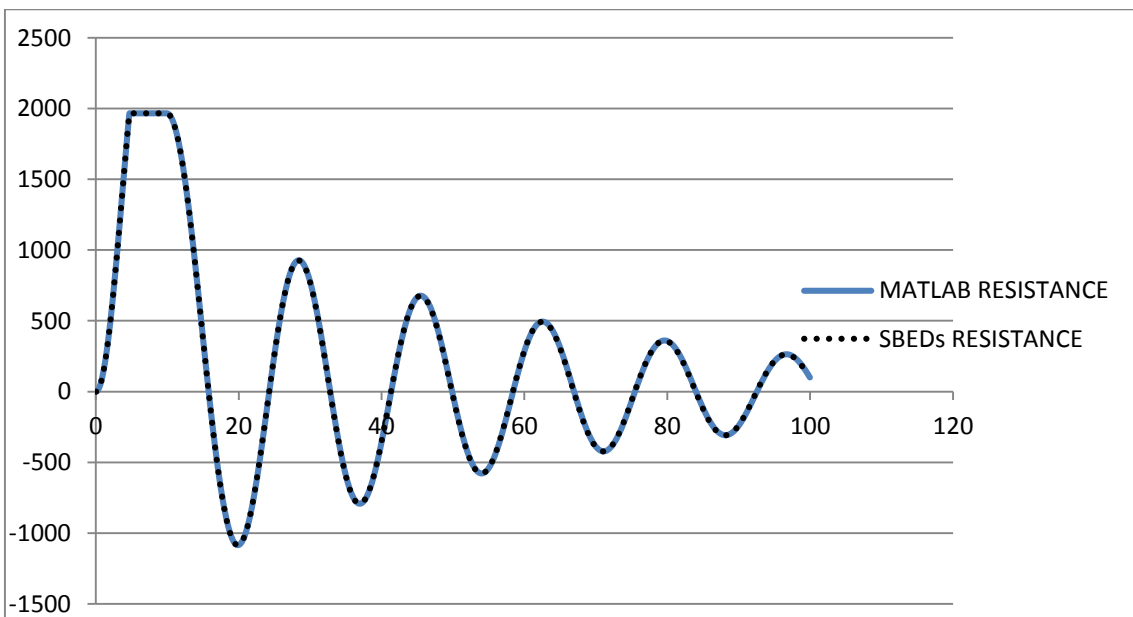
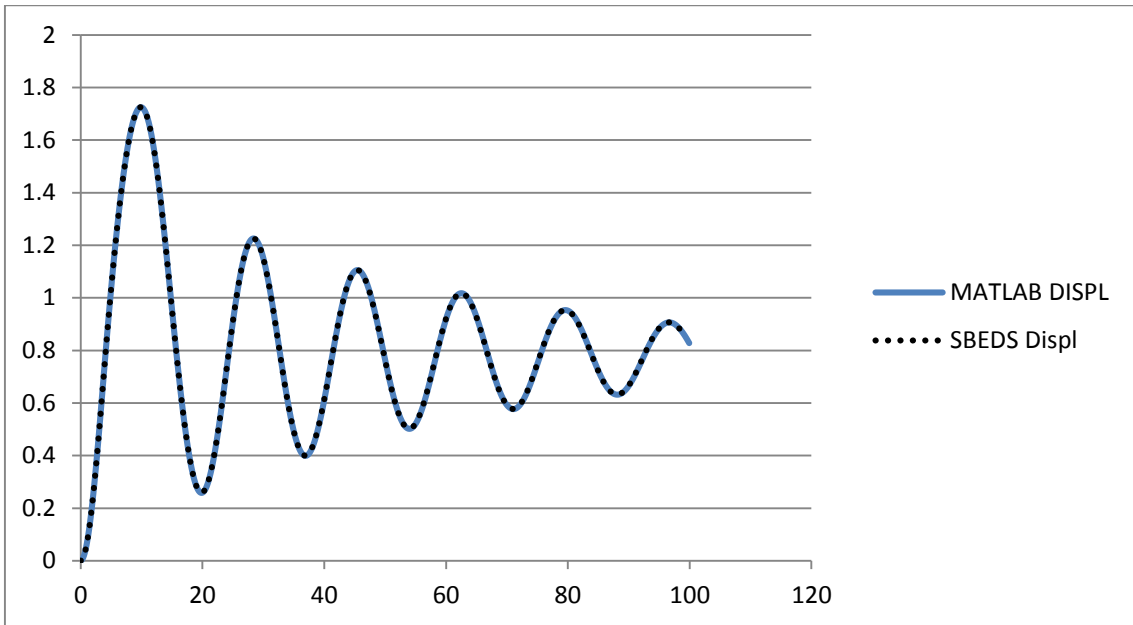
W40X65 Fixed-Fixed Uniform Load (20,000 psi in 17.8 ms) 5% Damping



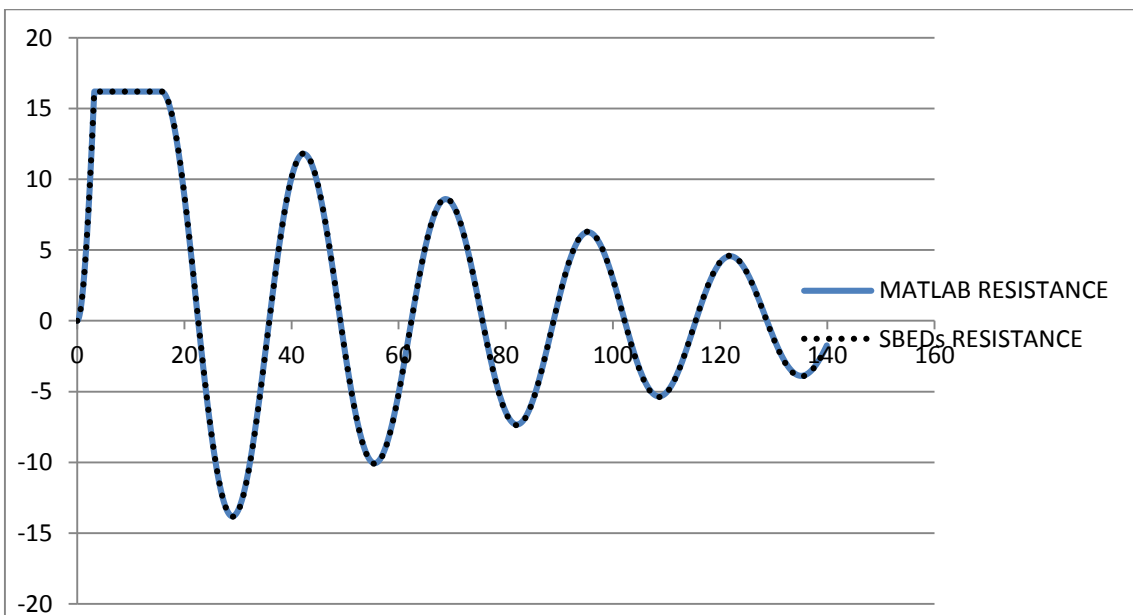
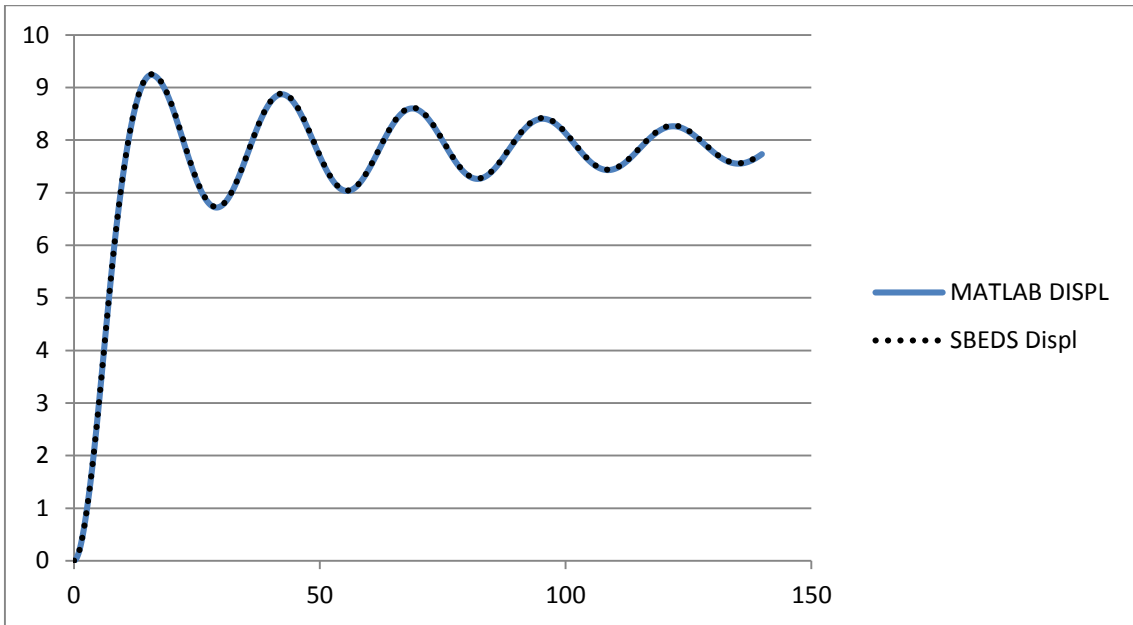
W40X655 Cantilever Point Load (2,000 psi in 17.8 ms) 5% Damping



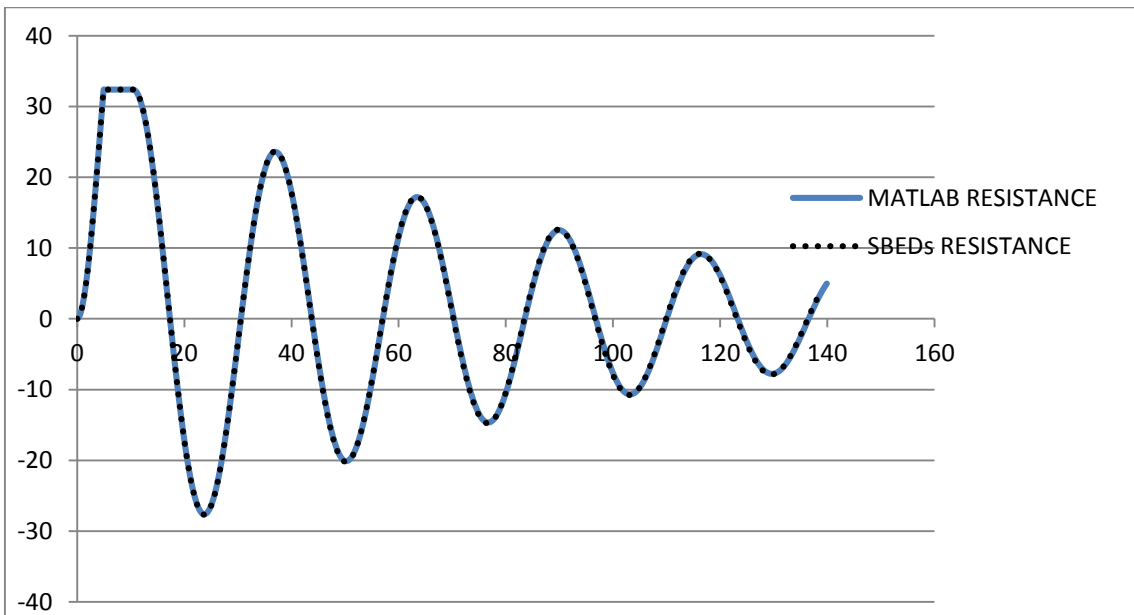
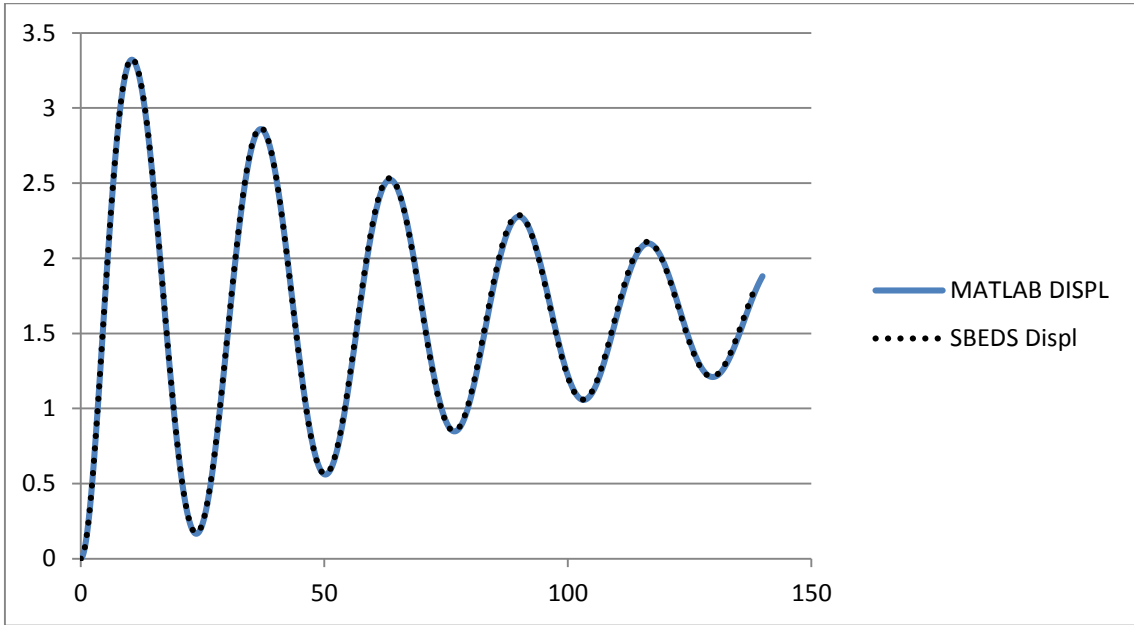
W40X65 Cantilever Uniform Load (2,000 psi in 17.8 ms) 5% Damping



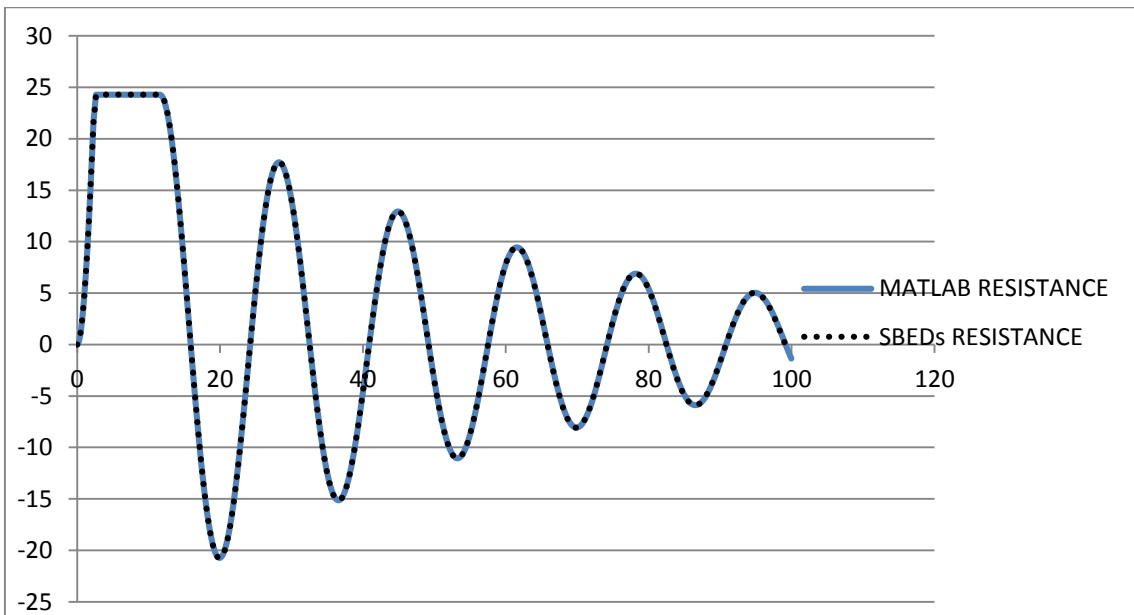
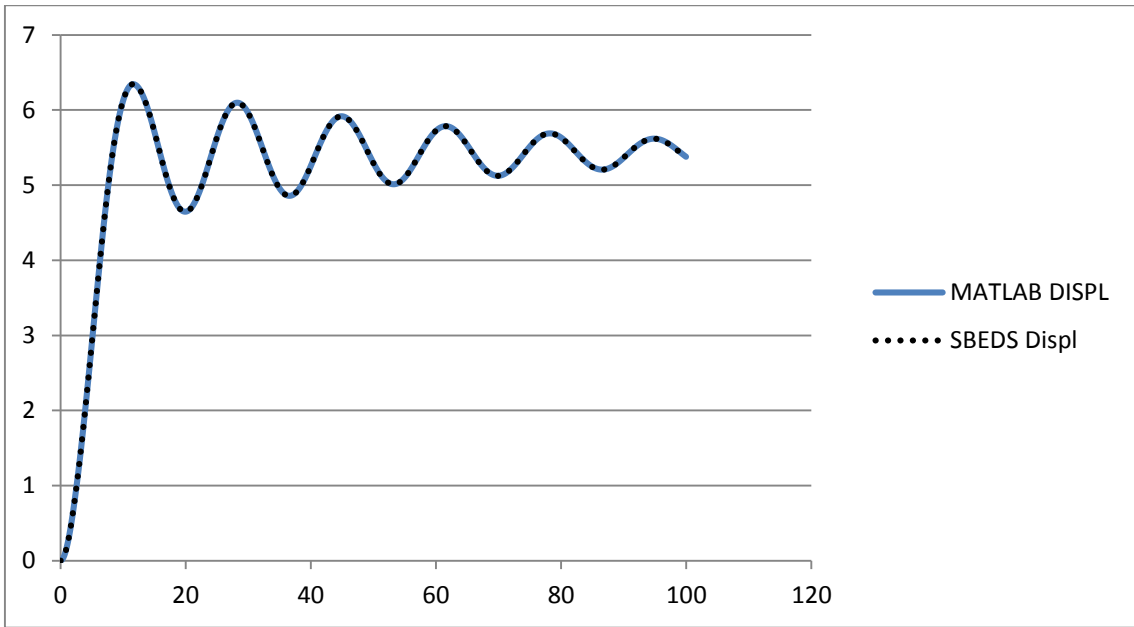
W10X12 Pinned-Pinned Point Load (70 psi in 8 ms) 5% Damping



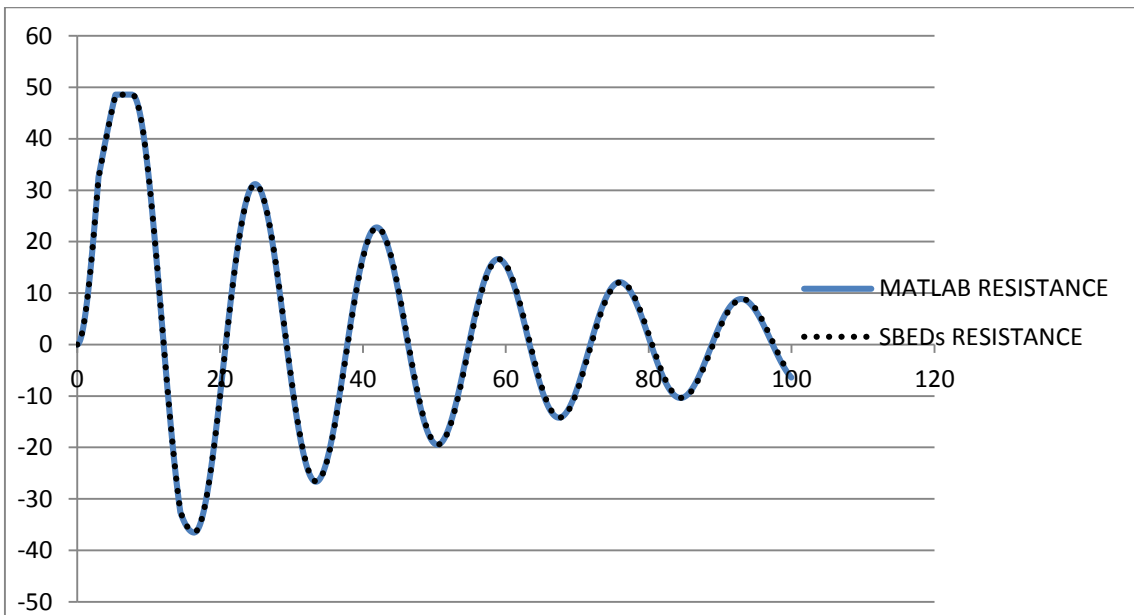
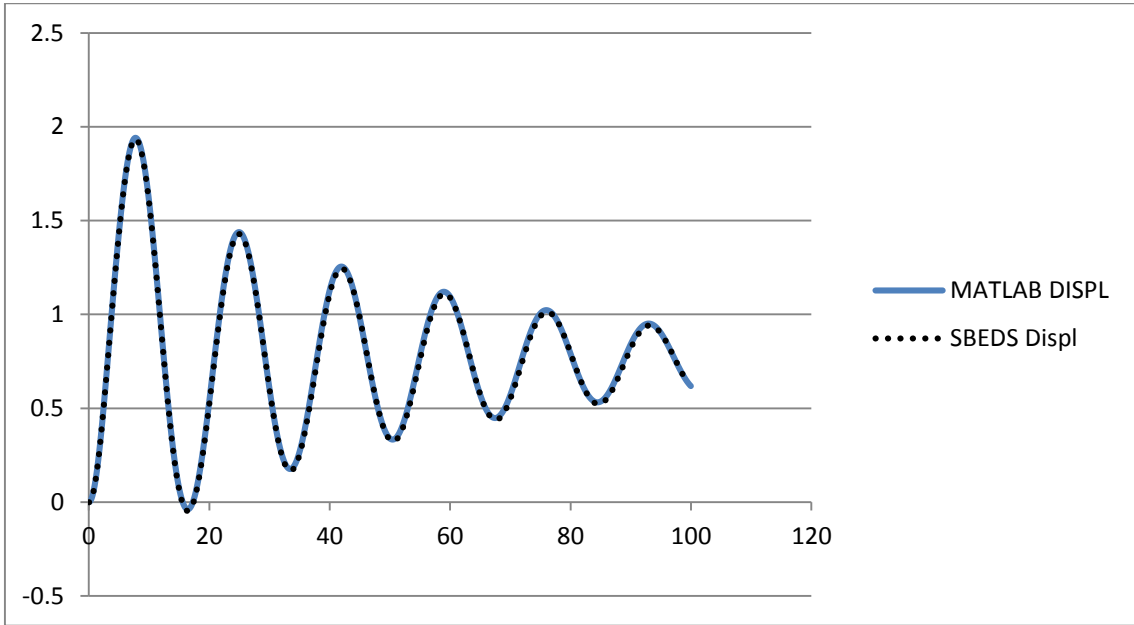
W10X12 Pinned-Pinned Uniform Load (70 psi in 8 ms) 5% Damping



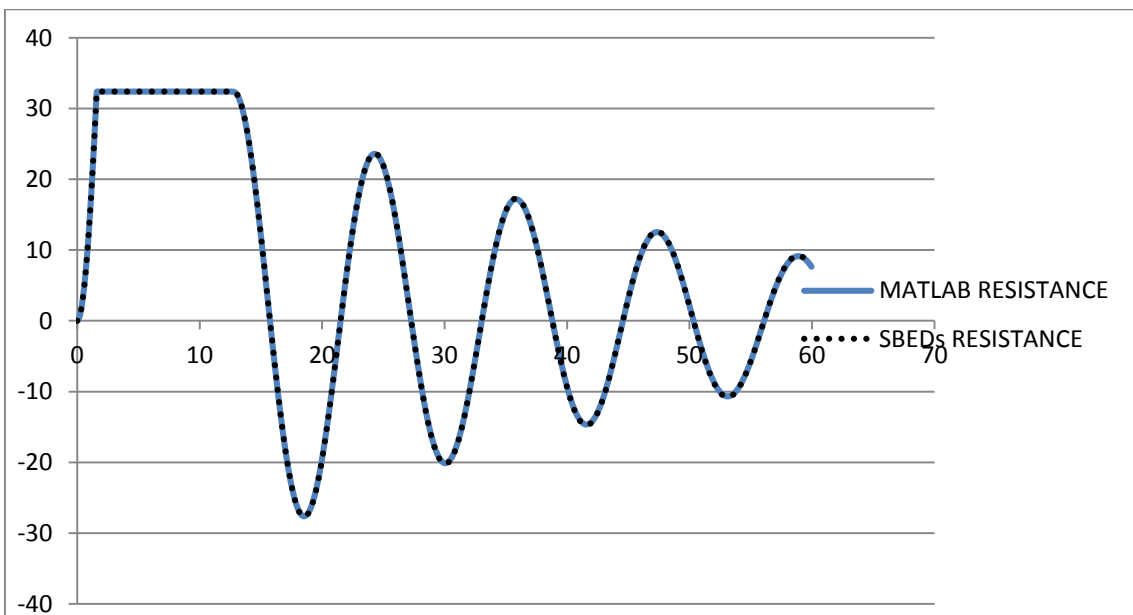
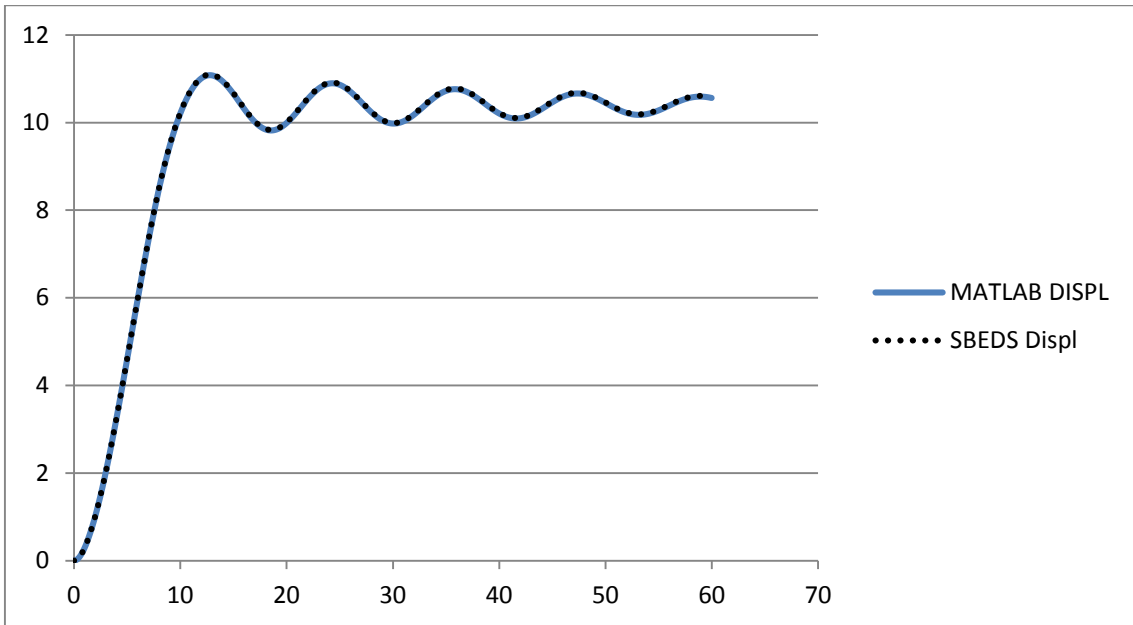
W10X12 Fixed-Pinned Point Load (70 psi in 8 ms) 5% Damping



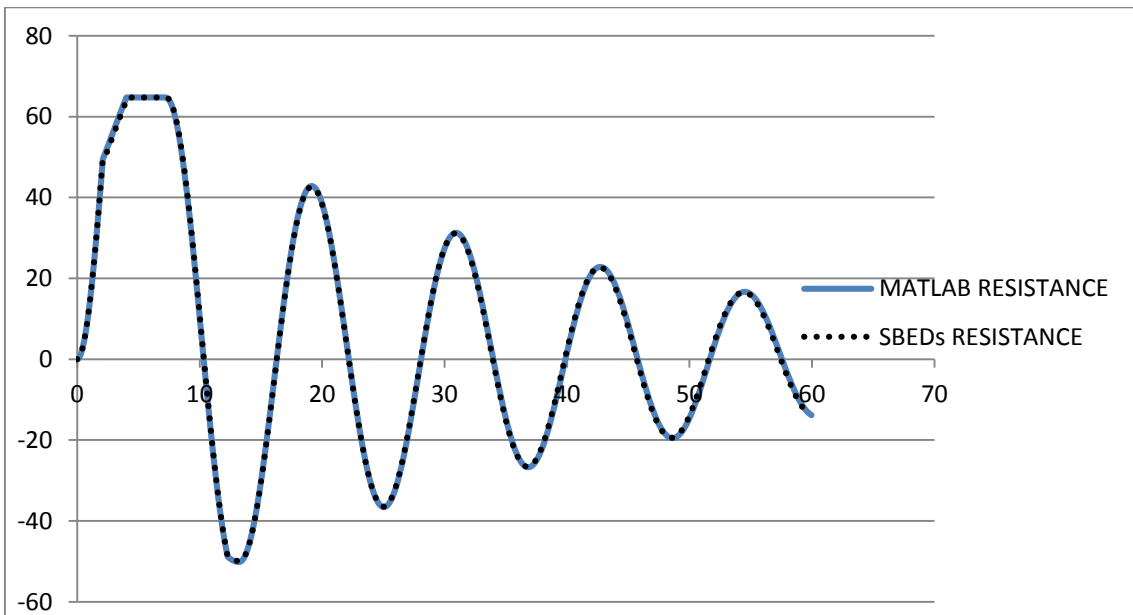
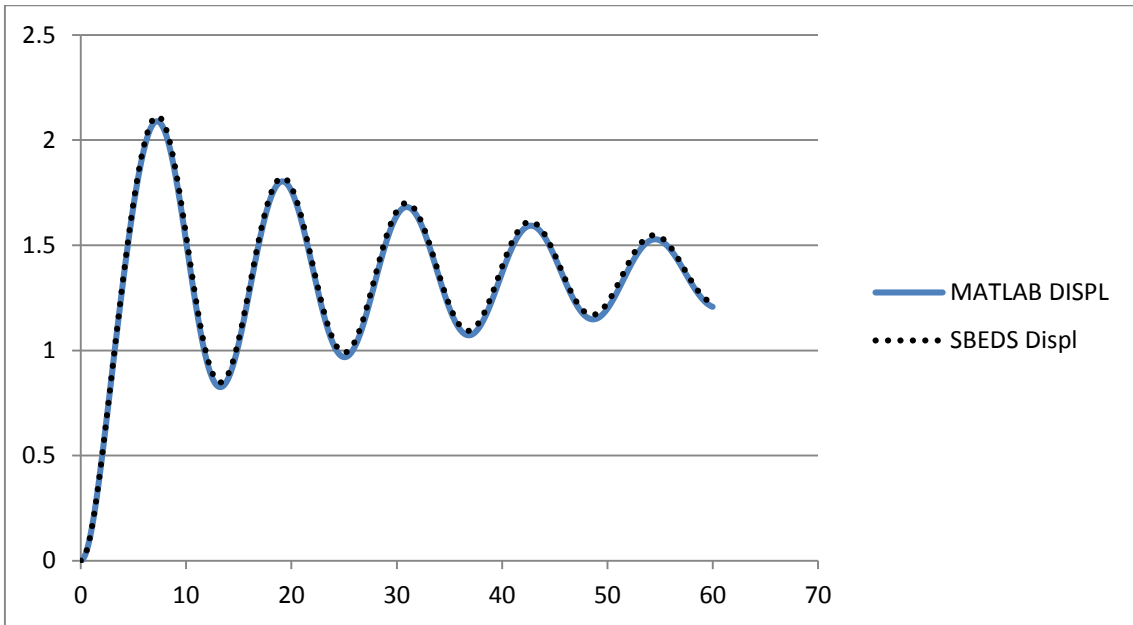
W10X12 Fixed-Pinned Uniform Load (70 psi in 8 ms) 5% Damping



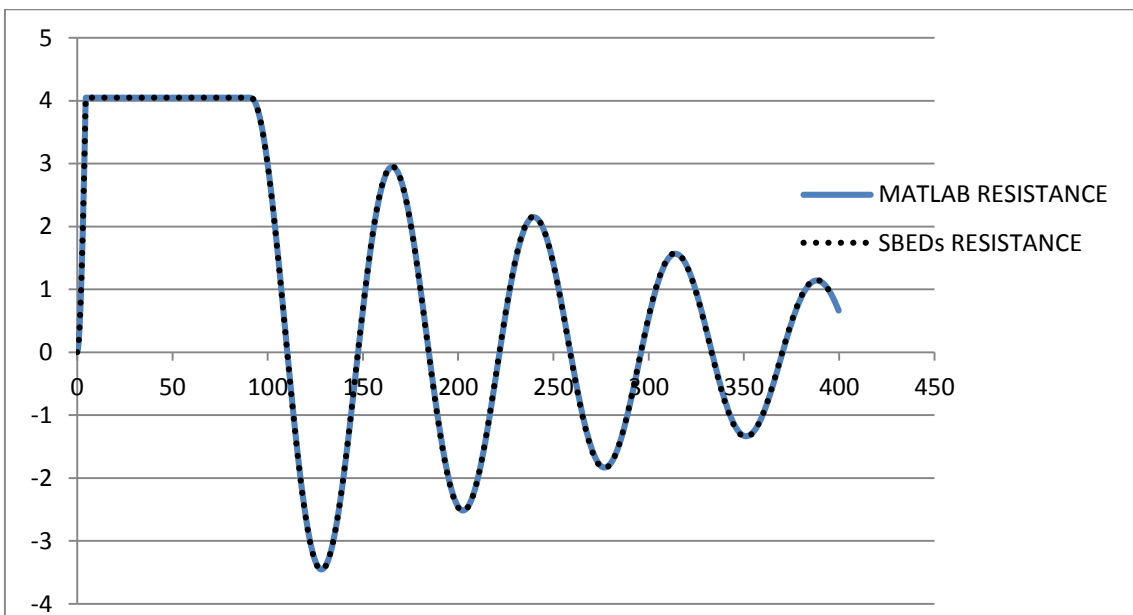
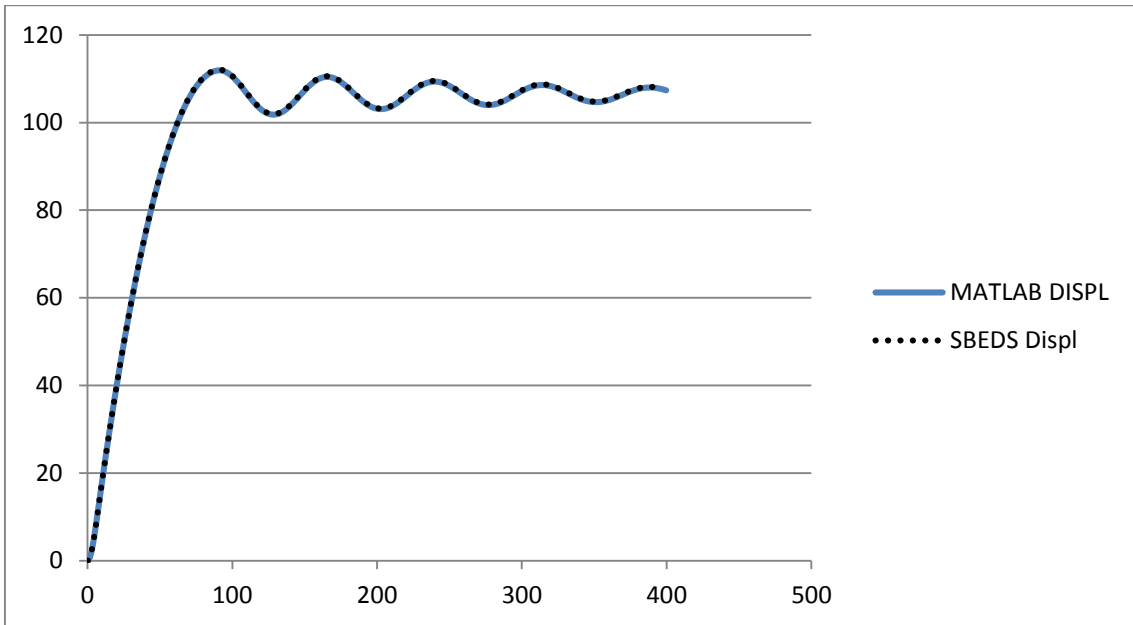
W10X12 Fixed-Fixed Point Load (100 psi in 8 ms) 5% Damping



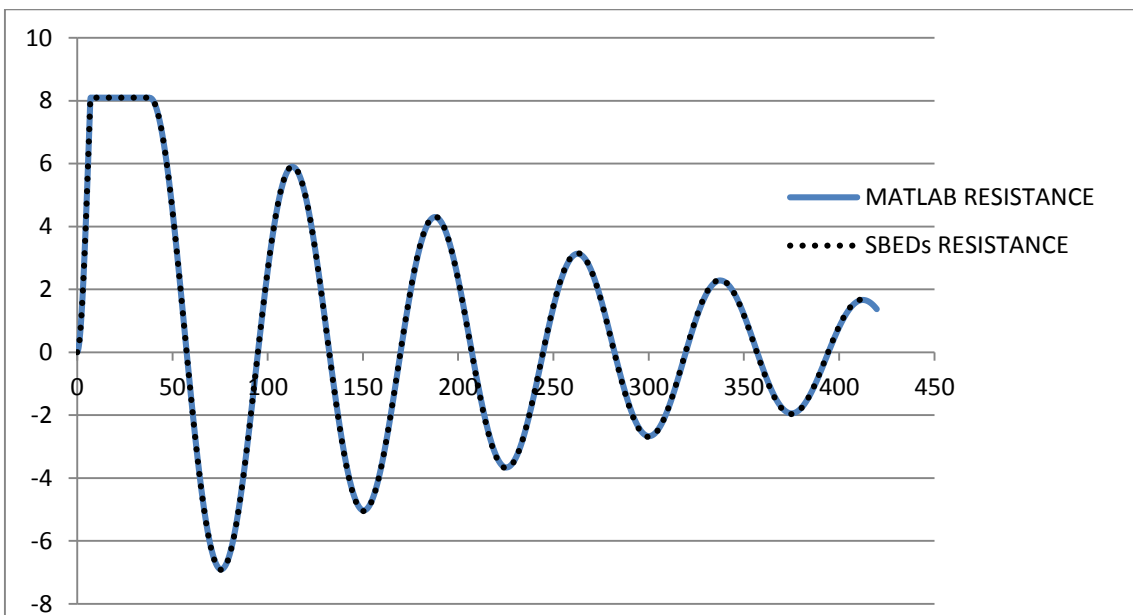
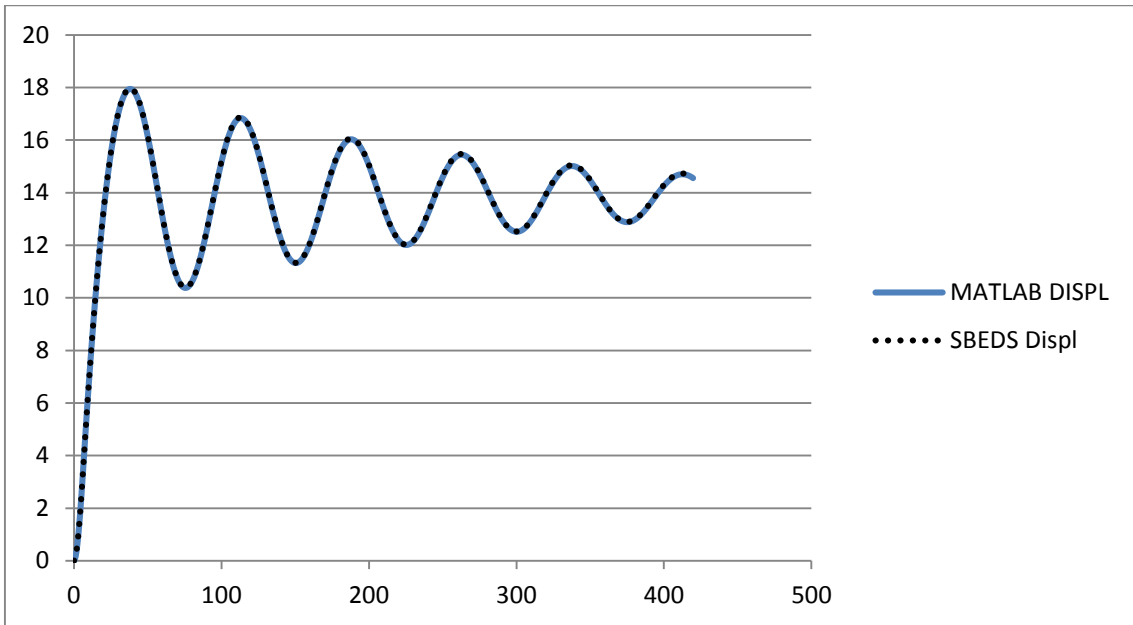
W10X12 Fixed-Fixed Uniform Load (100 psi in 8 ms) 5% Damping



W10X12 Cantilever Point Load (70 psi in 8 ms) 5% Damping



W10X12 Cantilever Uniform Load (70 psi in 8 ms) 5% Damping



Appendix C: MATLAB Code for More Accurate Model (MDOF Solver)

Main Program (Executable)

```
clc
clear

ii=sqrt(-1);

[Nodes Node_class ElementNodes ElementProp ElementSec BC Force
Force_distr, Axial_distr, Moment_distr analysis modes lumping
nonstr_mass gamma beta forcecurve del stopt ICu ICv TOL rayleigh axial
bc_type]=Input(); %Brings in data from input file

NumNodes=numel(Nodes(:,1)); %number of nodes

NumElem=numel(ElementNodes(:,1)); %number of elements

% preprocessing

sizBC=size(BC,1); %adds a row of 0's for BC, Force,
Axial_distr, Force_distr, Moment_distr, nonstr_mass, ICs
BC(sizBC+1,:)=zeros;

sizForce=size(Force,1);
Force(sizForce+1,:)=zeros;

sizAxial=size(Axial_distr,1);
Axial_distr(sizAxial+1,:)=zeros;

sizForce_d=size(Force_distr,1);
Force_distr(sizForce_d+1,:)=zeros;

sizMoment=size(Moment_distr,1);
Moment_distr(sizMoment+1,:)=zeros;

sizmass=size(nonstr_mass,1);
nonstr_mass(sizmass+1,:)=zeros;

sizICu=size(ICu,1);
ICu(sizICu+1,:)=zeros;

sizICv=size(ICv,1);
```

```

ICv(sizICv+1,:)=zeros;

siz1=size(Axial_distr,2); %number of columns of axial_distr
siz2=size(Force_distr,2); %number of columns of force_distr
siz3=size(Moment_distr,2); %number of columns of moment_distr

counter1=1; %counter to loop over Axial_distr
counter2=1; %counter to loop over Force_distr
counter3=1; %counter to loop over Moment_distr

degree1=siz1-1; %degree of polynomial for axial(x)
degree2=siz2-1; %degree of polynomial for force(x)
degree3=siz3-1; %degree of polynomial for moment(x)

for i=1:1:NumElem %preprocessing for elements (fills out full distr.
load)

    if i==Axial_distr(counter1,1) %for the size of the constants
(a's), loops and stores into temporary full load vector

        Axial_distrfull(i,1)=i;

        for count=2:1:siz1
            Axial_distrfull(i,count)=Axial_distr(counter1,count);
        end

        counter1=counter1+1;

    else

        Axial_distrfull(i,1)=i;

        for count=2:1:siz1
            Axial_distrfull(i,count)=0;
        end

    end

    if i==Force_distr(counter2,1) %for the size of the constants
(a's), loops and stores into temporary full load vector

        Force_distrfull(i,1)=i;

        for count=2:1:siz2
            Force_distrfull(i,count)=Force_distr(counter2,count);

```

```

        end

        counter2=counter2+1;

    else

        Force_distrfull(i,1)=i;

        for count=2:1:siz2
            Force_distrfull(i,count)=0;
        end

    end

    if i==Moment_distr(counter3,1) %for the size of the constants
(a's), loops and stores into temporary full load vector

        Moment_distrfull(i,1)=i;

        for count=2:1:siz3
            Moment_distrfull(i,count)=Moment_distr(counter3,count);
        end

        counter3=counter3+1;

    else

        Moment_distrfull(i,1)=i;

        for count=2:1:siz3
            Moment_distrfull(i,count)=0;
        end

    end

end

end

if degree1==0

    Axial_distrfull(:,2)=zeros;
end

if degree2==0

    Force_distrfull(:,2)=zeros;
end

if degree3==0

```

```

    Moment_distrfull(:,2)=zeros;
end

counter=1;      %counter to loop over Force
counter2=1;     %counter for BC's
counter3=1;     %counter for nonstr_mass
counter4=1;     %counter for ICu
counter5=1;     %counter for ICv

position=1;     %position in Force vector
position2=1;    %position in BC vector
position3=1;    %position in nonstr_mass vector
position4=1;    %position in ICu vector
position5=1;    %position in ICv vector

for i=1:1:NumNodes %preprocessing for nodes

    if i==Force(counter,1) %if the node i matches the node from Force

        Forcefull(position,1)=Force(counter,2);
        Forcefull(1+position,1)=Force(counter,3);
        Forcefull(2+position,1)=Force(counter,4);

        counter=counter+1;

    else %otherwise, put a 0

        Forcefull(position,1)=0;
        Forcefull(1+position,1)=0;
        Forcefull(2+position,1)=0;

    end

    position=position+3;

    if i==BC(counter2,1) %if the node i matches the node from BC

        BCfull(position2,1)=BC(counter2,2);
        BCfull(1+position2,1)=BC(counter2,3);
        BCfull(2+position2,1)=BC(counter2,4);

        counter2=counter2+1;

    else % otherwise put sqrt(-1) as placeholder

        BCfull(position2,1)=ii;
        BCfull(1+position2,1)=ii;
        BCfull(2+position2,1)=ii;

```

```

end

position2=position2+3;

if i==nonstr_mass(counter3,1) %if the node i matches the node
from nonstr_mass

    nonstr_massfull(position3,1)=nonstr_mass(counter3,2);
    nonstr_massfull(1+position3,1)=nonstr_mass(counter3,3);
    nonstr_massfull(2+position3,1)=nonstr_mass(counter3,4);

    counter3=counter3+1;

else % otherwise put 0 mass

    nonstr_massfull(position3,1)=0;
    nonstr_massfull(1+position3,1)=0;
    nonstr_massfull(2+position3,1)=0;

end

position3=position3+3;

if i==ICu(counter4,1) %if the node i matches the node from ICu

    ICufull(position4,1)=ICu(counter4,2);
    ICufull(1+position4,1)=ICu(counter4,3);
    ICufull(2+position4,1)=ICu(counter4,4);

    counter4=counter4+1;

else % otherwise put 0 IC

    ICufull(position4,1)=0;
    ICufull(1+position4,1)=0;
    ICufull(2+position4,1)=0;

end

position4=position4+3;

if i==ICv(counter5,1) %if the node i matches the node from ICv

    ICvfull(position5,1)=ICv(counter5,2);
    ICvfull(1+position5,1)=ICv(counter5,3);
    ICvfull(2+position5,1)=ICv(counter5,4);

```

```

        counter5=counter5+1;

else % otherwise put 0 IC

    ICvfull(position5,1)=0;
    ICvfull(1+position5,1)=0;
    ICvfull(2+position5,1)=0;

end

position5=position5+3;

end

BCfull_original=BCfull;

for i=1:1:NumElem %Loops over each element for lengths and material
properties

    %% lengths and angles

    node1=ElementNodes(i,1); %temporarily stores node 1 for element i
    node2=ElementNodes(i,2); %temporarily stores node 2 for element i

    x1=Nodes(node1,1); %temporarily stores x coordinate of node 1 for
element i
    y1=Nodes(node1,2); %temporarily stores y coordinate of node 1 for
element i
    x2=Nodes(node2,1); %temporarily stores x coordinate of node 2 for
element i
    y2=Nodes(node2,2); %temporarily stores y coordinate of node 2 for
element i

    [ang Length]=Lengths(x1, x2, y1, y2); %Function to calculate
lengths/angles

    angle(i)=ang;
    L(i)=Length;

    %% Material properties

    Mat_num=ElementProp(i,1);
    Type_num=ElementProp(i,2);

    Area=ElementSec(i,1);

```



```

Inert=ElementSec(i,2);
Flange=ElementSec(i,3);
Z=ElementSec(i,4);
d_l=ElementSec(i,5);
b_l=ElementSec(i,6);
f_l=ElementSec(i,7);
w_l=ElementSec(i,8);
nodeset=ElementNodes(i,:);

[EM GM DENS Ar Ar_sh Inertia Fy Mpl]=Data(Mat_num, Type_num, Area,
Inert, Flange, Z);

E(i)=EM;
G(i)=GM;
A(i)=Ar;
As(i)=Ar_sh;
I(i)=Inertia;
rho(i)=DENS;
Fyield(i)=Fy;
Mp(i)=Mpl;
Tp_num(i)=Type_num;

%uses first element as section for axial capacity
if i==1

    Py=Ar*Fy;
    axial=axial*Py

    if axial~=0    % P-M reduction in plastic moment (to match
SBEDS)

        rg=sqrt(Inertia/Ar);

        if bc_type=='cantil'
            klr_fact=2;
        elseif bc_type=='pinpin'
            klr_fact=1;
        elseif bc_type=='fixpin'
            klr_fact=.7;
        elseif bc_type=='fixfix'
            klr_fact=.5;
        end

        Klr=klr_fact*Length*NumElem/rg;

        Cc=Klr/pi*sqrt(Fy/EM);

        if Cc<=1.5
            Pcr=(.658^(Cc^2))*Fy*Ar;
        else

```

```

        Pcr=.877/Cc^2*Fy*Ar;
    end

    Mpset(1)=Fy*Z;

    Mpset(2)=(1-axial/Pcr)*Fy*Z;

    Mpset(3)=(1-axial/(Fy*Ar))*1.18*Z*Fy;

    Mp_red=min(Mpset)

    end

end

if axial~=0

    Mp(i)=Mp_red;
end

%
%
%   if axial~=0
%
%       Pboundary=w_l*(d_l-2*f_l)*Fy;
%
%       if axial>=Pboundary
%
%           NA=(2*b_l*f_l*Fy-axial+w_l*Fy*(d_l-2*f_l))/(2*b_l*Fy);
%
%           Mp(i)=b_l*NA*Fy*(d_l-NA);
%
%       elseif axial<Pboundary
%
%           NA=-(axial-d_l*w_l*Fy)/2/w_l/Fy;
%
%           Mp(i)=-Fy*(b_l*f_l^2-f_l^2*w_l+w_l*NA^2-
b_l*d_l*f_l+d_l*f_l*w_l-d_l*w_l*NA);
%           Mp(i)=557.3;
%           Mp(i)=511;
%           Mp(i)=473;
%       end
%
%   end
%
end

```

```

%% connectivity b's

[b_e] = connectivity(ElementNodes(i,:),NumNodes);

belement(:, :, i)=b_e;

%% Initializing internal forces as all 0 for first state
determination

Finternal(:, 1, i)=[0;0;0;0;0;0];
FsElement(:, 1, i)=[0;0;0;0;0;0];
FdElement_b(:, 1, i)=[0;0;0;0;0;0];
FdElement_a(:, 1, i)=[0;0;0;0;0;0];
FiElement(:, 1, i)=[0;0;0;0;0;0];

end

[F_t t steps] = Forcetime(forcecurve, del, stopt);

timecount=1;

%first timestep outside the loop

kglobal=[]; kglobal=zeros; %clears matrices every timestep
gglobal=[]; gglobal=zeros;
mglobal=[]; mglobal=zeros;
cglobal=[]; cglobal=zeros;
reducer=[]; reducer=zeros;

for i=1:1:NumElem %Loops over each element for stiffness, mass,
damping, and equivalent nodal loads

    node1=ElementNodes(i,1); %temporarily stores node 1 for element i
    node2=ElementNodes(i,2); %temporarily stores node 2 for element i
    nodeclass1=Node_class(node1,1);
    nodeclass2=Node_class(node2,1);

    moment_left=abs(FsElement(3,timecount,i));
    moment_right=abs(FsElement(6,timecount,i));

    if Type_num==6

        order=ElementProp(i,3);

```

```

        [N_e B_e C_e N_e_r B_e_r hinging] = NBC_element(L(i), E(i),
A(i), As(i), G(i), I(i), Tp_num(i), order);

    else

        N_e=0;
        B_e=0;
        C_e=0;
        N_e_r=0;
        B_e_r=0;

    end

    [k_e m_e r kms kss] = k_element(L(i), E(i), A(i), As(i), G(i),
I(i), Tp_num(i), B_e_r, C_e, N_e_r, lumping, analysis, rho(i),
FsElement(:,timecount,i), Mp(i), nodeclass1, nodeclass2);

    kelemento(:, :, i)=k_e;

    melemento(:, :, i)=m_e;

    if nodeclass1=='I' %Nonstructural masses block (adds half if
internal node, adds full if external node)

        melemento(1,1,i)=melemento(1,1,i)+0.5*nonstr_massfull(node1*3-
2);
        melemento(2,2,i)=melemento(2,2,i)+0.5*nonstr_massfull(node1*3-
1);

    melemento(3,3,i)=melemento(3,3,i)+0.5*nonstr_massfull(node1*3);

    elseif nodeclass1=='E'

        melemento(1,1,i)=melemento(1,1,i)+nonstr_massfull(node1*3-2);
        melemento(2,2,i)=melemento(2,2,i)+nonstr_massfull(node1*3-1);
        melemento(3,3,i)=melemento(3,3,i)+nonstr_massfull(node1*3);

    end

    if nodeclass2=='I'

        melemento(4,4,i)=melemento(4,4,i)+0.5*nonstr_massfull(node2*3-
2);
        melemento(5,5,i)=melemento(5,5,i)+0.5*nonstr_massfull(node2*3-
1);

    melemento(6,6,i)=melemento(6,6,i)+0.5*nonstr_massfull(node2*3);

    elseif nodeclass2=='E'

```

```

melemento(4,4,i)=melemento(4,4,i)+nonstr_massfull(node2*3-2);
melemento(5,5,i)=melemento(5,5,i)+nonstr_massfull(node2*3-1);
melemento(6,6,i)=melemento(6,6,i)+nonstr_massfull(node2*3);

end

%% Equivalent Nodal Loads

axial_nodal=Axial_distrfull(i,:);
forces_nodal=Force_distrfull(i,:);
moment_nodal=Moment_distrfull(i,:);

[g_e] = g_element(Length, degree1, degree2, degree3, axial_nodal,
forces_nodal, moment_nodal, N_e_r, Tp_num(i), r, kms, kss,
FsElement(:,timecount,i), Mp(i), nodeclass1, nodeclass2); %Equivalent
Load Function

gelemento(:,i)=g_e;

%% transformation matrices

c=cos(angle(i));
s=sin(angle(i));

transf=[c s 0 0 0 0
        -s c 0 0 0 0
        0 0 1 0 0 0
        0 0 0 c s 0
        0 0 0 -s c 0
        0 0 0 0 0 1];

kelement_tr(:,:,i)=transpose(transf)*kelemento(:,:,i)*transf;
%rotates elemental stiffness to global coordinates
gelement_tr(:,1,i)=transpose(transf)*gelemento(:,i); %rotates
elemental equivalent nodal loads to global coordinates
melement_tr(:,:,i)=transpose(transf)*melemento(:,:,i)*transf;
%rotates mass matrix

%% Assembly

kglobal=kglobal+transpose(belement(:,:,i))*kelement_tr(:,:,i)*belement
(:,:,i); %assembles k matrix, k=sum(b_eT*k_e*b_e)

gglobal=gglobal+transpose(belement(:,:,i))*gelement_tr(:,1,i);
%assembles g vector, k=sum(b_eT*g_e)

mglobal=mglobal+transpose(belement(:,:,i))*melement_tr(:,:,i)*belement
(:,:,i); %assembles global mass matrix m=sum(b_eT*m_e*b_e)

```

```

Fnet=gglobal+Forcefull; %calculates total load vector, which is
sum of nodal loads and equivalent nodal loads

% update BC vector for size of reducer matrix (and reduced
% stiffness, mass, damping matrices)

if nodeclass1=='E' %if a fixed end is yielding, creates different
boundary conditions

    if moment_left>=Mp(i) %Hinge at left node

        BCfull(node1*3,1)=ii;

    elseif moment_left<Mp(i)

        BCfull(node1*3,1)=BCfull_original(node1*3,1);

    end

end

if nodeclass2=='E' %if a fixed end is yielding, creates different
boundary conditions

    if moment_right>=Mp(i) %Hinge at left node

        BCfull(node2*3,1)=ii;

    elseif moment_right<Mp(i)

        BCfull(node2*3,1)=BCfull_original(node2*3,1);

    end

end

end

reduce_count=1;

for i=1:1:NumNodes*3 %calculates reducer matrix

    if BCfull(i,1)~=0

        reduzero(reduce_count,i)=1;
        reduce_count=reduce_count+1;

    else

        reduzero(:,i)=0;

```

```

        end

end

%applying B.C.s with reducer matrix

ko=reducero*kglobal*transpose(reducero);

go=reducero*gglobal;

mo=reducero*mglobal*transpose(reducero);

%elastic frequencies

modes_solve=min(modes,size(ko,1));

[phi w2]=eigs(ko,mo,modes_solve,'SM');

w=sqrt(w2);

for freq=1:1:size(w,1)

    T(freq,freq)=2*pi/w(freq,freq);

end

firstmode=rayleigh(1,1);
secondmode=rayleigh(2,1);
z1=rayleigh(1,2);
z2=rayleigh(2,2);

w1=w(firstmode,firstmode);
w2=w(secondmode,secondmode);

c_alpha=2*w1*w2/(w2^2-w1^2)*(w2*z1-w1*z2);
c_beta=2/(w2^2-w1^2)*(w2*z2-w1*z1);

c_alpha=0;
c_beta=.000002;

if axial~=0
    kglobal=[]; kglobal=zeros;
end

```

```

for i=1:1:NumElem %Loops over each element damping

    node1=ElementNodes(i,1); %temporarily stores node 1 for element i
    node2=ElementNodes(i,2); %temporarily stores node 2 for element i
    nodeclass1=Node_class(node1,1);
    nodeclass2=Node_class(node2,1);

    moment_left=abs(FsElement(3,timecount,i));
    moment_right=abs(FsElement(6,timecount,i));

    if Type_num==6

        order=ElementProp(i,3);

        [N_e B_e C_e N_e_r B_e_r hinging] = NBC_element(L(i), E(i),
A(i), As(i), G(i), I(i), Tp_num(i), order);

    else

        N_e=0;
        B_e=0;
        C_e=0;
        N_e_r=0;
        B_e_r=0;

    end

    damp_switch=1;

    [celemento_a(:, :, i) celemento_b(:, :, i)]=Damping(kelemento(:, :, i),
melemento(:, :, i), c_alpha, c_beta, damp_switch);

    celemento(:, :, i)=celemento_a(:, :, i)+celemento_b(:, :, i);

    if axial~=0 %adds geometric stiffness to elastic stiffness AFTER
calculating stiffness-proportional damping

        k_g = k_geom(L(i), axial, r);

        kgelemento(:, :, i)=k_g;

        kelemento(:, :, i)=kelemento(:, :, i)+k_g;
    end

    c=cos(angle(i));
    s=sin(angle(i));

```



```

transf=[c  s 0  0 0 0
        -s c 0  0 0 0
         0 0 1  0 0 0
         0 0 0  c s 0
         0 0 0 -s c 0
         0 0 0  0 0 1];

    celement_tr(:,:,i)=transpose(transf)*celemento(:,:,i)*transf;
%rotates damping matrix

    if axial~=0
        kelement_tr(:,:,i)=transpose(transf)*kelemento(:,:,i)*transf;
%rotates damping matrix
    end

    %% Assembly

cglobal=cglobal+transpose(belement(:,:,i))*celement_tr(:,:,i)*belement
(:,:,i); %assembles global damping matrix c=sum(b_eT*c_e*b_e)

    if axial~=0

kglobal=kglobal+transpose(belement(:,:,i))*kelement_tr(:,:,i)*belement
(:,:,i);
    end

end

co=reducero*cglobal*transpose(reducero);

%applying axial load statically before dynamic analysis

if axial~=0

    ko=reducero*kglobal*transpose(reducero);
    siz=size(ko,1);
    F_axial=zeros(siz,1);
    F_axial(1,1)=axial;
    u_axial=ko\F_axial;
    u_axial_full=reducero'*u_axial;
    ICu_full=u_axial_full;

end

% for i=3:3:3*NumNodes

```

```

%
%   if abs(Fnet(i,1))>Mp(1)
%
%       Fnet(i,1)=Fnet(i,1)/abs(Fnet(i,1))*Mp(1);
%
%   end
%
% end

Fexternal=reducero*Fnet;

% for first timecount, sets initial conditions

minvo=inv(mo);

Re(:,1)=Fexternal*F_t(1);

Ru(:,1)=Re(:,1)*0; %sets unbalanced to 0 initially

Ri(:,1)=Ru(:,1); %sets internal to 0 intially

u_t(:,1)=reducero*ICufull;

Fs(:,1)=ko*u_t(:,1);

v_t(:,1)=reducero*ICvfull;

Fd(:,1)=co*v_t(:,1);

a_t(:,1)=minvo*(Re(:,1)-Fd(:,1)-Fs(:,1));

Fi(:,1)=mo*a_t(:,1);

u_t_total(:,1)=ICufull;
v_t_total(:,1)=ICvfull;

count=1;

for i=1:1:NumNodes*3 %this loop adds back the boundary conditions to
initial a_t_total vector

    if BCfull(i,1)==ii

        a_t_total(i,1)=a_t(count,1);

        count=count+1;
    end
end

```

```

elseif BCfull(i,1)==0
    a_t_total(i,1)=0;
end

end

%stores the original (elastic) matrices
ao=gamma/beta*co+1/beta/del*mo;
bo=1/2/beta*mo+del*(gamma/2/beta-1)*co;
kbaro=1/del*ao;
kstaro=ko+kbaro;
invko=inv(kstaro);

for i=1:1:NumNodes % initializes
    yieldnodes(i,1)=0;
    yieldvalues(i,1)=0;
end

while (timecount<=steps) %Timestepping

    %% intial calcs done every timesetep

    Re(:,timecount+1)=Fexternal*F_t(timecount+1); %external applied
load of timestep

    %Re(:,timecount+1)=Re(:,timecount+1)+F_axial;

    dRe(:,timecount)=Re(:,timecount+1)-Re(:,timecount); %change in
applied load of timestep

    %The remainder of the timestepping is split into 2 sets of
solutions
    %The first set is the timestep solved elastically to determine
yielding
    %The second set is only calculated if there is yielding (using
hinged elements where required)

```

```

%% First set (elastic for yielding determination)

yielded='n'; %determines whether the second set is calculated

%   for i=1:1:NumNodes % stores all yielded nodes as unyielded
%
%       yieldnodes(i,timecount+1)=yieldnodes(i,timecount);
%   end

Ru(:,timecount+1)=dRe(:,timecount)+ao*v_t(:,timecount)+bo*a_t(:,timecount)+Ru(:,timecount); %unbalanced load

du(:,timecount)=Ru(:,timecount+1)*0; %initializes du

norm_check=TOL+1; %initializes the norm as greater than tolerance

while (norm_check>TOL) %NR iteration

    yielded='n';

    du(:,timecount)=du(:,timecount)+invko*Ru(:,timecount+1);

    dv(:,timecount)=gamma/beta/del*du(:,timecount)-
gamma/beta*v_t(:,timecount)+del*(1-gamma/2/beta)*a_t(:,timecount);

    da(:,timecount)=1/beta/del^2*du(:,timecount)-
1/beta/del*v_t(:,timecount)-1/2/beta*a_t(:,timecount);

    u_t(:,timecount+1)=u_t(:,timecount)+du(:,timecount);

    v_t(:,timecount+1)=v_t(:,timecount)+dv(:,timecount);

    a_t(:,timecount+1)=a_t(:,timecount)+da(:,timecount);

    count=1;
    for i=1:1:NumNodes*3 %this loop adds back all values to u, v,
a vectors

        if BCfull(i,1)==ii

            u_t_total(i,timecount+1)=u_t(count,timecount+1);
            du_total(i,timecount)=du(count,timecount);

            v_t_total(i,timecount+1)=v_t(count,timecount+1);
            dv_total(i,timecount)=dv(count,timecount);

            a_t_total(i,timecount+1)=a_t(count,timecount+1);

```

```

        da_total(i,timecount)=da(count,timecount);

        count=count+1;

elseif BCfull(i,1)==0

    u_t_total(i,timecount+1)=0;
    du_total(i,timecount)=0;

    v_t_total(i,timecount+1)=0;
    dv_total(i,timecount)=0;

    a_t_total(i,timecount+1)=0;
    da_total(i,timecount)=0;

end

end

Finternal_global=[]; %initialize internal as 0
Finternal_global=zeros;

for i=1:1:NumElem %Loops over each element for internal
forces

    set=size(kelemento(:, :, i), 2);

    element i
    node1=ElementNodes(i,1); %temporarily stores node 1 for
    element i
    node2=ElementNodes(i,2); %temporarily stores node 2 for
    element i

    nodeclass1=Node_class(node1,1);
    nodeclass2=Node_class(node2,1);

    du_t_element(:,timecount,i)=[du_total(node1*3-2,timecount)
%creates du for element i
    du_total(node1*3-1,timecount)
    du_total(node1*3,timecount)
    du_total(node2*3-2,timecount)
    du_total(node2*3-1,timecount)
    du_total(node2*3,timecount)];

    dv_t_element(:,timecount,i)=[dv_total(node1*3-2,timecount)
%creates dv for element i
    dv_total(node1*3-1,timecount)
    dv_total(node1*3,timecount)
    dv_total(node2*3-2,timecount)
    dv_total(node2*3-1,timecount)

```

```

        dv_total(node2*3,timecount)];

    da_t_element(:,timecount,i)=[da_total(node1*3-2,timecount)
%creates da for element i
    da_total(node1*3-1,timecount)
    da_total(node1*3,timecount)
    da_total(node2*3-2,timecount)
    da_total(node2*3-1,timecount)
    da_total(node2*3,timecount)];

    u_t_element(:,timecount+1,i)=[u_t_total(node1*3-
2,timecount+1) %creates u for element i
    u_t_total(node1*3-1,timecount+1)
    u_t_total(node1*3,timecount+1)
    u_t_total(node2*3-2,timecount+1)
    u_t_total(node2*3-1,timecount+1)
    u_t_total(node2*3,timecount+1)];

    v_t_element(:,timecount+1,i)=[v_t_total(node1*3-
2,timecount+1) %creates v for element i
    v_t_total(node1*3-1,timecount+1)
    v_t_total(node1*3,timecount+1)
    v_t_total(node2*3-2,timecount+1)
    v_t_total(node2*3-1,timecount+1)
    v_t_total(node2*3,timecount+1)];

    a_t_element(:,timecount+1,i)=[a_t_total(node1*3-
2,timecount+1) % creates a for element i
    a_t_total(node1*3-1,timecount+1)
    a_t_total(node1*3,timecount+1)
    a_t_total(node2*3-2,timecount+1)
    a_t_total(node2*3-1,timecount+1)
    a_t_total(node2*3,timecount+1)];

dFsElement(:,timecount,i)=kelemento(:, :, i)*du_t_element(:,timecount,i)
;

FsElement(:,timecount+1,i)=FsElement(:,timecount,i)+dFsElement(:,timec
ount,i); %Spring forces for element i (incremental)

FdElement_a(:,timecount+1,i)=celemento_a(:, :, i)*v_t_element(:,timecoun
t+1,i);

```

```

FdElement_b(:,timecount+1,i)=celemento_b(:, :, i)*v_t_element(:,timecount+1,i);

FdElement(:,timecount+1,i)=FdElement_a(:,timecount+1,i)+FdElement_b(:,timecount+1,i);

FiElement(:,timecount+1,i)=melemento(:, :, i)*a_t_element(:,timecount+1,i);

Finternal(:,timecount+1,i)=FsElement(:,timecount+1,i)+FdElement(:,timecount+1,i)+FiElement(:,timecount+1,i); %Internal forces for element i

        moment_right=abs(FsElement(6,timecount+1,i)); %right end
moment for element i
        moment_left=abs(FsElement(3,timecount+1,i));

        if moment_right>=Mp(i) %allows for recalculation using
end released elements after this section

        yielded='y'; %this allows the second set to be
calculated
        yieldnodes(node2,timecount+1)=1; %tracks which node
yielded
        yieldvalues(node2,timecount+1)=moment_right;

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%decrease timestep here and get
more accuracy for other elements spring forces

        %
        value=abs(FsElement(6,timecount,i));
%This scales the previous spring force to the yielded value so that
the second set has this value in the internal force
        %
        %
        %
        %
        scale=Mp(i)/value;
FsElement(:,timecount,i)=scale*FsElement(:,timecount,i);

else

        yieldnodes(node2,timecount+1)=0;
        yieldvalues(node2,timecount+1)=0;

end

```

```

        if nodeclass1=='E'

            if moment_left>=Mp(i)

                yielded='y';
                yieldnodes(node1,timecount+1)=1;
                yieldvalues(node1,timecount+1)=moment_left;

            else

                yieldnodes(node1,timecount+1)=0;
                yieldvalues(node1,timecount+1)=0;

            end

        end

        Finternal_global=Finternal_global+transpose(belement(:, :, i))*Finternal
        (:,timecount+1,i); %assembles the internal force vector

    end

    Ri(:,timecount+1)=reducero*Finternal_global; %reduces the
    internal force vector to the correct size

    Ru(:,timecount+1)=Re(:,timecount+1)-Ri(:,timecount+1);
    %calculates the unbalanced load

    norm_check=norm(Ru(:,timecount+1)); %restores the norm to be
    checked against tolerance

    end

    %% This is the second set (recalculation of all matrices and
    solution for timestep if yielding occurs)

    if yielded=='y' %only goes into this set if the yielded marker is
    'y'

        yield_values_sorted(:,timecount+1)=sort(yieldvalues(:,timecount+1),'de
        scend');

        loop='stay';
    end

```



```

        while (loop=='stay') %iterates this whole section until the
yielded nodes are correct

        a=[];
        b=[];
        kbar=[];
        kstar=[];
        invk=[];

        kglobal=[]; kglobal=zeros;
        gglobal=[]; gglobal=zeros;
        mglobal=[]; mglobal=zeros;
        cglobal=[]; cglobal=zeros;
        reducer=[]; reducer=zeros;

        for i=1:1:NumElem

                moment_left=abs(FsElement(3,timecount+1,i)); %These
moments were calculated in the first set so the yielded values will be
above Mp(i)
                moment_right=abs(FsElement(6,timecount+1,i));
                node1=ElementNodes(i,1); %temporarily stores node 1
for element i
                node2=ElementNodes(i,2); %temporarily stores node 2
for element i
                nodeclass1=Node_class(node1,1);
                nodeclass2=Node_class(node2,1);

                %reassemble k

                kelement(:, :, i)=kelemento(:, :, i);
                melement(:, :, i)=melemento(:, :, i);

                damp_switch=1;

                scale=0;

                if nodeclass1=='E'

                        if moment_left==yield_values_sorted(1,timecount+1)

                                [kelement(:, :, i)
Tr_k(:, :, i)]=Condenseleft(kelemento(:, :, i));%-kgelemento(:, :, i));
%creates hinged matrix
                                [melement(:, :, i)
Tr_m(:, :, i)]=Condenseleft(melemento(:, :, i)); %creates hinged matrix
                                %kelement(:, :, i)=kelement(:, :, i)+kgelemento(:, :, i);

```

```

        value=abs(FsElement(3,timecount,i)); %This
scales the previous spring force to the yielded value so that the
second set has this value in the internal force

        scale=Mp(i)/value;

FsElement(:,timecount,i)=scale*FsElement(:,timecount,i);

        if moment_right==Mp(i) %if the right end is
previously yielded, becomes bar
            'error- increase mesh to avoid fully
yielded element'
            break

        end

        elseif moment_left==Mp(i)

            [kelement(:, :, i)
Tr_k(:, :, i)]=Condenseleft(kelemento(:, :, i));%-kgelemento(:, :, i));
%creates hinged matrix

            [melement(:, :, i)
Tr_m(:, :, i)]=Condenseleft(melemento(:, :, i)); %creates hinged matrix

%kelement(:, :, i)=kelement(:, :, i)+kgelemento(:, :, i);

        end

        end

        if moment_right==yield_values_sorted(1,timecount+1)
%Hinge at right node for first node yielded

            [kelement(:, :, i)
Tr_k(:, :, i)]=Condense(kelemento(:, :, i));%-kgelemento(:, :, i)); %creates
hinged matrix

            if i~=NumElem
                [melement(:, :, i)
Tr_m(:, :, i)]=Condense(melemento(:, :, i)); %creates hinged matrix
            end

            end

%kelement(:, :, i)=kelement(:, :, i)+kgelemento(:, :, i);

```

```

        value=abs(FsElement(6,timecount,i)); %This scales
the previous spring force to the yielded value so that the second set
has this value in the internal force

        scale=Mp(i)/value;

FsElement(:,timecount,i)=scale*FsElement(:,timecount,i);

        if nodeclass1=='E' %if it is the left end member
and the left end is also previously yielded, becomes bar

            if moment_left==Mp(i)
                'error- increase mesh to avoid fully
yielded element'
                break
            end

        end

        end

        %damp_switch=10^-3;

        elseif moment_right==Mp(i) %any already yielded nodes
(takes value of Mp)

            [kelement(:, :, i)
Tr_k(:, :, i)]=Condense(kelemento(:, :, i)); %-kgelemento(:, :, i)); %creates
hinged matrix
            if i~=NumElem
                [melement(:, :, i)
Tr_m(:, :, i)]=Condense(melemento(:, :, i)); %creates hinged matrix
            end

            %kelement(:, :, i)=kelement(:, :, i)+kgelemento(:, :, i);
            %damp_switch=10^-3;

        end

        if moment_left>=Mp(i) %accounts for the other side of
each node yielded to decrease damping as well

            %damp_switch=10^-3;

            %value=abs(FsElement(3,timecount,i));

```

```

        %scale=Mp(i)/value;

%
FsElement(:,timecount,i)=scale*FsElement(:,timecount,i);

        end

        [celement_a(:, :, i)
celement_b(:, :, i)]=Damping(kelement(:, :, i), melement(:, :, i), c_alpha,
c_beta, damp_switch);
        celement(:, :, i)=celement_a(:, :, i)+celement_b(:, :, i);

%

        celement_a(:, :, i)=celemento_a(:, :, i);
        celement_b(:, :, i)=celemento_b(:, :, i);

        damp_switch_left=1;
        damp_switch_right=1;

        if moment_right>=Mp(i)

                damp_switch_right=10^-3;

        end

        if i==1

                moment_previous(i)=moment_left;
        end

        if moment_previous(i)>=Mp(i)

                damp_switch_left=10^-3;

        end

        celement_b(6, :, i)=celement_b(6, :, i)*damp_switch_right;
%"zeros" damping for right end rotational DOF of element i
        celement_b(:, 6, i)=celement_b(:, 6, i)*damp_switch_right;
        celement_b(6, 6, i)=celement_b(6, 6, i)/damp_switch_right;

        celement_b(3, :, i)=celement_b(3, :, i)*damp_switch_left;
%"zeros" damping for left end rotational DOF of adjacent member (same
DOF as above)
        celement_b(:, 3, i)=celement_b(:, 3, i)*damp_switch_left;
        celement_b(3, 3, i)=celement_b(3, 3, i)/damp_switch_left;

```

```

%
%
celement_b(:, :, i) = celement_b(:, :, i) * damp_switch_right;
%
%           if damp_switch_right == 1 && damp_switch_left ~ 1
%
%
celement_b(:, :, i) = celement_b(:, :, i) * damp_switch_left;
%
%           end

moment_previous(i+1) = moment_right;

%celement_b(:, :, i) = celement_b(:, :, i) * damp_switch;
celement(:, :, i) = celement_a(:, :, i) + celement_b(:, :, i);

%% Equivalent Nodal Loads

axial_nodal = Axial_distrfull(i, :);
forces_nodal = Force_distrfull(i, :);
moment_nodal = Moment_distrfull(i, :);

[g_e] = g_element(Length, degree1, degree2, degree3,
axial_nodal, forces_nodal, moment_nodal, N_e_r, Tp_num(i), r, kms,
kss, FsElement(:, timecount+1, i), Mp(i), nodeclass1, nodeclass2);
%Equivalent Load Function

gelement(:, i) = g_e;

%% transformation matrices

c = cos(angle(i));
s = sin(angle(i));

transf = [c  s  0  0  0  0
         -s  c  0  0  0  0
           0  0  1  0  0  0
           0  0  0  c  s  0
           0  0  0 -s  c  0
           0  0  0  0  0  1];

kelement_tr(:, :, i) = transpose(transf) * kelement(:, :, i) * transf; %rotates
elemental stiffness to global coordinates
gelement_tr(:, 1, i) = transpose(transf) * gelement(:, i);
%rotates elemental equivalent nodal loads to global coordinates

melement_tr(:, :, i) = transpose(transf) * melement(:, :, i) * transf; %rotates
mass matrix

celement_tr(:, :, i) = transpose(transf) * celement(:, :, i) * transf; %rotates
damping matrix

```

```

%% Assembly

kglobal=kglobal+transpose(belement(:, :, i))*kelement_tr(:, :, i)*belement
(:, :, i); %assembles k matrix, k=sum(b_eT*k_e*b_e)

gglobal=gglobal+transpose(belement(:, :, i))*gelement_tr(:, 1, i);
%assembles g vector, k=sum(b_eT*g_e)

mglobal=mglobal+transpose(belement(:, :, i))*melement_tr(:, :, i)*belement
(:, :, i); %assembles global mass matrix m=sum(b_eT*m_e*b_e)

cglobal=cglobal+transpose(belement(:, :, i))*celement_tr(:, :, i)*belement
(:, :, i); %assembles global damping matrix c=sum(b_eT*c_e*b_e)

Fnet=gglobal+Forcefull; %calculates total load vector,
which is sum of nodal loads and equivalent nodal loads

% update BC vector for size of reducer matrix (and
reduced % stiffness, mass, damping matrices)

end

%applying B.C.s with reducer matrix

k=reducero*kglobal*transpose(reducero);

g=reducero*gglobal;

m=reducero*mglobal*transpose(reducero);

c=reducero*cglobal*transpose(reducero);

%Newmark Beta solution

Fexternal=reducero*Fnet;

Re(:, timecount+1)=Fexternal*F_t(timecount+1); %external
applied load of timestep

%Re(:, timecount+1)=Re(:, timecount+1)+F_axial;

dRe(:, timecount)=Re(:, timecount+1)-Re(:, timecount);
%change in applied load of timestep

```

```

a=gamma/beta*c+1/beta/del*m;

b=1/2/beta*m+del*(gamma/2/beta-1)*c;

kbar=1/del*a;

kstar=k+kbar;

invk=inv(kstar);

Ru(:,timecount+1)=dRe(:,timecount)+a*v_t(:,timecount)+b*a_t(:,timecount)+Ru(:,timecount);

du(:,timecount)=Ru(:,timecount+1)*0;

norm_check=TOL+1;

while (norm_check>TOL) %NR iteration

du(:,timecount)=du(:,timecount)+invk*Ru(:,timecount+1);

dv(:,timecount)=gamma/beta/del*du(:,timecount)-
gamma/beta*v_t(:,timecount)+del*(1-gamma/2/beta)*a_t(:,timecount);

da(:,timecount)=1/beta/del^2*du(:,timecount)-
1/beta/del*v_t(:,timecount)-1/2/beta*a_t(:,timecount);

u_t(:,timecount+1)=u_t(:,timecount)+du(:,timecount);

v_t(:,timecount+1)=v_t(:,timecount)+dv(:,timecount);

a_t(:,timecount+1)=a_t(:,timecount)+da(:,timecount);

count=1;
for i=1:1:NumNodes*3 %this loop adds back the
boundary conditions to u vector

if BCfull(i,1)==i

u_t_total(i,timecount+1)=u_t(count,timecount+1);
du_total(i,timecount)=du(count,timecount);

v_t_total(i,timecount+1)=v_t(count,timecount+1);
dv_total(i,timecount)=dv(count,timecount);

```

```

a_t_total(i,timecount+1)=a_t(count,timecount+1);
    da_total(i,timecount)=da(count,timecount);

    count=count+1;

elseif BCfull(i,1)==0

    u_t_total(i,timecount+1)=0;
    du_total(i,timecount)=0;

    v_t_total(i,timecount+1)=0;
    dv_total(i,timecount)=0;

    a_t_total(i,timecount+1)=0;
    da_total(i,timecount)=0;

end

end

Finternal_global=[];
Finternal_global=zeros;

for i=1:1:NumElem %Loops over each element for
internal forces

    set=size(kelement(:, :, i), 2);

    node1=ElementNodes(i,1); %temporarily stores node
1 for element i
    node2=ElementNodes(i,2); %temporarily stores node
2 for element i

    nodeclass1=Node_class(node1,1);
    nodeclass2=Node_class(node2,1);

    du_t_element(:,timecount,i)=[du_total(node1*3-
2,timecount) %creates du for element i
    du_total(node1*3-1,timecount)
    du_total(node1*3,timecount)
    du_total(node2*3-2,timecount)
    du_total(node2*3-1,timecount)
    du_total(node2*3,timecount)];

```



```

        dv_t_element(:,timecount,i)=[dv_total(node1*3-
2,timecount) %creates dv for element i
        dv_total(node1*3-1,timecount)
        dv_total(node1*3,timecount)
        dv_total(node2*3-2,timecount)
        dv_total(node2*3-1,timecount)
        dv_total(node2*3,timecount)];

        da_t_element(:,timecount,i)=[da_total(node1*3-
2,timecount) %creates da for element i
        da_total(node1*3-1,timecount)
        da_total(node1*3,timecount)
        da_total(node2*3-2,timecount)
        da_total(node2*3-1,timecount)
        da_total(node2*3,timecount)];

        u_t_element(:,timecount+1,i)=[u_t_total(node1*3-
2,timecount+1) %creates u for element i
        u_t_total(node1*3-1,timecount+1)
        u_t_total(node1*3,timecount+1)
        u_t_total(node2*3-2,timecount+1)
        u_t_total(node2*3-1,timecount+1)
        u_t_total(node2*3,timecount+1)];

        v_t_element(:,timecount+1,i)=[v_t_total(node1*3-
2,timecount+1) %creates v for element i
        v_t_total(node1*3-1,timecount+1)
        v_t_total(node1*3,timecount+1)
        v_t_total(node2*3-2,timecount+1)
        v_t_total(node2*3-1,timecount+1)
        v_t_total(node2*3,timecount+1)];

        a_t_element(:,timecount+1,i)=[a_t_total(node1*3-
2,timecount+1) % creates a for element i
        a_t_total(node1*3-1,timecount+1)
        a_t_total(node1*3,timecount+1)
        a_t_total(node2*3-2,timecount+1)
        a_t_total(node2*3-1,timecount+1)
        a_t_total(node2*3,timecount+1)];

dFsElement(:,timecount,i)=kelement(:, :, i)*du_t_element(:,timecount,i);

FsElement(:,timecount+1,i)=FsElement(:,timecount,i)+dFsElement(:,timec
ount,i); %Spring forces for element i (incremental)

```

```

        moment_left=abs(FsElement(3,timecount+1,i));
%These moments were calculated in the first set so the yielded values
will be above Mp(i)
        moment_right=abs(FsElement(6,timecount+1,i));

%           if moment_left>=Mp(i)
%
%
FsElement(3,timecount+1,i)=abs(FsElement(3,timecount+1,i))/FsElement(3
,timecount+1,i)*Mp(i);
%           end

FdElement_a(:,timecount+1,i)=celement_a(:, :, i)*v_t_element(:,timecount
+1,i);

FdElement_b(:,timecount+1,i)=celement_b(:, :, i)*v_t_element(:,timecount
+1,i);

FdElement(:,timecount+1,i)=FdElement_a(:,timecount+1,i)+FdElement_b(:,
timecount+1,i);

FiElement(:,timecount+1,i)=melement(:, :, i)*a_t_element(:,timecount+1,i
);

Finternal(:,timecount+1,i)=FsElement(:,timecount+1,i)+FdElement(:,time
count+1,i)+FiElement(:,timecount+1,i); %Internal forces for element i

Finternal_global=Finternal_global+transpose(belement(:, :, i))*Finternal
(:,timecount+1,i);

        moment_right=abs(FsElement(6,timecount+1,i));
%again this value will be over Mp(i) if yielded
        moment_left=abs(FsElement(3,timecount+1,i));

        if moment_right>=Mp(i)

            yieldnodes(node2,timecount+1)=1;
            yieldvalues(node2,timecount+1)=moment_right;

        else

            yieldnodes(node2,timecount+1)=0;
            yieldvalues(node2,timecount+1)=0;

        end

        if nodeclass1=='E' %leftmost node

```

```

        if moment_left >= Mp(i)
            yieldnodes(nodel,timecount+1)=1;
yieldvalues(nodel,timecount+1)=moment_left;

        else

            yieldnodes(nodel,timecount+1)=0;
            yieldvalues(nodel,timecount+1)=0;

        end

    end

end

end

Ri(:,timecount+1)=reducero*Finternal_global;
Ru(:,timecount+1)=Re(:,timecount+1)-Ri(:,timecount+1);
norm_check=norm(Ru(:,timecount+1));

end

yield_values_sorted(:,timecount+1)=sort(yieldvalues(:,timecount+1),'de
scend');

loop='exit';

    for i=1:1:NumNodes %if there is a value that isn't scaled
to Mp, it stays in the loop

        j=i; %moment check is associated with the left node of
the element

        if i==NumNodes

            j=i-1; %moment check is associated with right
moment for last node

        end

        if yield_values_sorted(i,timecount+1)~=0 &&
yield_values_sorted(i,timecount+1)~=Mp(j)

            loop='stay';

```

```

        end
    end

    end %end of yieldcheck convergence (converges to the correct
number of yielded nodes

    end %end of yielded='y' section

    %% increase timesetep for next timestep

    timecount=timecount+1;

end

DOFs=NumNodes*3;

midpoint=(DOFs+1)/2;

figure(1)

subplot(4,1,1)
plot(t,F_t)
title('Force function')
ylabel('Forcing shape with max=1 for scaling')
grid on

t(timecount+1)=[];

subplot(4,1,2)
plot(t,u_t_total(midpoint,:))
title('midspan deflection')
grid on

subplot(4,1,3)
plot(t,v_t_total(midpoint,:))
title('midspan velocity')
grid on

subplot(4,1,4)
plot(t,a_t_total(midpoint,:))
title('midspan acceleration')
grid on

F1=Finternal(:, :, 1);
F2=Finternal(:, :, 2);
F3=Finternal(:, :, 3);
F4=Finternal(:, :, 4);

```

```

Fs1=FsElement(:, :, 1);
Fs2=FsElement(:, :, 2);
Fs3=FsElement(:, :, 3);
Fs4=FsElement(:, :, 4);

Fd1=FdElement(:, :, 1);
Fd2=FdElement(:, :, 2);
Fd3=FdElement(:, :, 3);
Fd4=FdElement(:, :, 4);

Fi1=FiElement(:, :, 1);
Fi2=FiElement(:, :, 2);
Fi3=FiElement(:, :, 3);
Fi4=FiElement(:, :, 4);

u1=u_t_element(:, :, 1);
u2=u_t_element(:, :, 2);
u3=u_t_element(:, :, 3);
u4=u_t_element(:, :, 4);

v1=v_t_element(:, :, 1);
v2=v_t_element(:, :, 2);
v3=v_t_element(:, :, 3);
v4=v_t_element(:, :, 4);

a1=a_t_element(:, :, 1);
a2=a_t_element(:, :, 2);
a3=a_t_element(:, :, 3);
a4=a_t_element(:, :, 4);

text1='Element ';

text3=' left moment';

text4=' right moment';

%figure(2)

% counter=1;
%
% for i=1:1:NumElem
%
%     text2=num2str(i);
%
%     lefttext=strcat(text1, text2, text3);
%     righttext=strcat(text1, text2, text4);
%
%     subplot(NumElem, 2, counter)
%     plot(t, Finternal(3, :, i));
%     title(lefttext)
%     grid on
%
%
```

```

%
%     subplot (NumElem,2,counter+1)
%     plot (t,Finternal (6,:,i));
%     title (righttext)
%     grid on
%
%
%
%     counter=counter+2;
%
% end

spring='spring';

figure(3)

counter=1;

for i=1:1:NumElem

    text2=num2str(i);

    lefttext=strcat(text1,text2,spring,text3);
    righttext=strcat(text1,text2,spring,text4);

    subplot (NumElem,2,counter)
    plot (t,FsElement (3,:,i));
    title (lefttext)
    grid on

    subplot (NumElem,2,counter+1)
    plot (t,FsElement (6,:,i));
    title (righttext)
    grid on

    counter=counter+2;

end

%
%
%
%
%
%
% spring='damping';
%
% figure(4)
%
% counter=1;
%
% for i=1:1:NumElem
%

```

```

% text2=num2str(i);
%
% lefttext=strcat(text1,text2, spring, text3);
% righttext=strcat(text1, text2, spring, text4);
%
% subplot(NumElem,2,counter)
% plot(t,FdElement(3, :, i));
% title(lefttext)
% grid on
%
%
% subplot(NumElem,2,counter+1)
% plot(t,FdElement(6, :, i));
% title(righttext)
% grid on
%
%
% counter=counter+2;
% end
%
%
% spring='inertial';
%
% figure(5)
% counter=1;
% for i=1:1:NumElem
%
% text2=num2str(i);
%
% lefttext=strcat(text1, text2, spring, text3);
% righttext=strcat(text1, text2, spring, text4);
%
% subplot(NumElem,2,counter)
% plot(t,FiElement(3, :, i));
% title(lefttext)
% grid on
%
%
% subplot(NumElem,2,counter+1)
% plot(t,FiElement(6, :, i));
% title(righttext)
% grid on
%
%
% counter=counter+2;
% end
%
%
```

```

%
% text3=' left';
%
% text4=' right';
%
% spring='displ';
%
% figure(6)
%
% counter=1;
%
% for i=1:1:NumElem
%
%     text2=num2str(i);
%
%     leftttext=strcat(text1,text2,spring,text3);
%     rightttext=strcat(text1,text2,spring,text4);
%
%     subplot(NumElem,2,counter)
%     plot(t,u_t_element(3,:,i));
%     title(leftttext)
%     grid on
%
%
%     subplot(NumElem,2,counter+1)
%     plot(t,u_t_element(6,:,i));
%     title(rightttext)
%     grid on
%
%
%     counter=counter+2;
%
% end
%
%
% spring='vel';
%
% figure(7)
%
% counter=1;
%
% for i=1:1:NumElem
%
%     text2=num2str(i);
%
%     leftttext=strcat(text1,text2,spring,text3);
%     rightttext=strcat(text1,text2,spring,text4);
%
%     subplot(NumElem,2,counter)
%     plot(t,v_t_element(3,:,i));
%     title(leftttext)
%     grid on
%
%
%

```



```

% subplot(NumElem,2,counter+1)
% plot(t,v_t_element(6,:,i));
% title(righttext)
% grid on
%
% counter=counter+2;
%
% end
%
% spring='acc';
%
% figure(8)
%
% counter=1;
%
% for i=1:1:NumElem
%
% text2=num2str(i);
%
% lefttext=strcat(text1,text2,spring,text3);
% righttext=strcat(text1,text2,spring,text4);
%
% subplot(NumElem,2,counter)
% plot(t,a_t_element(3,:,i));
% title(lefttext)
% grid on
%
% subplot(NumElem,2,counter+1)
% plot(t,a_t_element(6,:,i));
% title(righttext)
% grid on
%
% counter=counter+2;
%
% end
%
% text3=' left shear';
%
% text4=' right shear';
%
% figure(3)
%
% counter=1;
%
% for i=1:1:NumElem
%
% text2=num2str(i);
%

```

```

% lefttext=strcat(text1,text2,text3);
% righttext=strcat(text1,text2,text4);
%
% subplot(NumElem,2,counter)
% plot(t,Finternal(2,:,i));
% title(lefttext)
% grid on
%
%
% subplot(NumElem,2,counter+1)
% plot(t,Finternal(5,:,i));
% title(righttext)
% grid on
%
%
% counter=counter+2;
%
% end

```

```

t=transpose(t);
u_t=transpose(u_t);
Fs2=-transpose(Fs2);
Fs3=transpose(Fs3);
Fs4=transpose(Fs4);

```

```

a_output(:,1)=t;
a_output(:,2)=u_t(:,7);
a_output(:,3)=Fs2(:,6);
a_output(:,4)=-Fs4(:,6);

```

```

%if BC has left fixed end

```

```

%
% a_output(:,1)=t;
% a_output(:,2)=u_t(:,6);
% a_output(:,3)=Fs2(:,6);
% a_output(:,4)=-Fs4(:,6);

```

```

% %
% % % if cantilever
%
% a_output(:,1)=t;
% a_output(:,2)=u_t(:,2);
% a_output(:,3)=-Fs4(:,6);
% a_output(:,4)=-Fs4(:,6);

```

```

% Fs3=transpose(Fs3);
%
%
% Fs1=transpose(-Fs1);
%
% a_outputspr(:,1)=t;

```

```

% a_outputspr(:,2)=Fs1(:,3);
%
% mcheck=melemento(:, :, 1);

aaimpose(1)=c_alpha;
aaimpose(2)=c_beta;
aaimpose(3)=axial;
aaimpose(4)=axial*L(1)*NumElem/A(1)/E(1);

min(u_t_total(8, :));

```

Input Function

```

%% This is the input file. Enter in units of inches, seconds, and kips

```

```

function [Nodes Node_class ElementNodes ElementProp ElementSec BC
Force Force_distr, Axial_distr, Moment_distr, analysis, modes,
lumping, nonstr_mass gamma beta forcecurve del stopt ICu ICv TOL
rayleigh axial bc_type] = Input

```

```

%% Input Section

```

```

i=sqrt(-1);

```

```

% enter type of analysis
% 1 for static analysis
% 2 for modal/dynamics analysis

```

```

analysis=2;

```

```

%NR iteration unbalanced load tolerance

```

```

TOL=10^-5;

```

```

% If modal analysis: enter number of modes to calculate, otherwise
enter

```

```

% any. Enter type of mass matrix. otherwise, enter any.
% 1: Consistent mass matrix
% 2: Particle Lumping (SAP)
% 3: Row Summing
% 4: HRZ Lumping

```

```

modes=1000;
lumping=2;

```

```

type=5; %mbeam type (4 bernoulli, 5 timoshenko)

% If modal analysis: gamma and beta values for newmark time stepping

beta=1/4;
gamma=1/2;

% If modal analysis, list linear endpoints of force time curve
% Scale so that largest point is 1 since you will be entering in the
values
% of the scaling later

% example: triangle function with 0 to 15k in 0.3 sec and back down
is:
% [0 0
% .3 1
% .6 0]

%enter delta t for timestepping (must be dividable into each
%segment)
%Also, enter time to stop analysis (delta t must be dividable)

del=0.0001;

ts=.0178;

% forcecurve=[0 0
% .01-del 0
% .01 1
% .01+ts 0
% .1 0 ];

forcecurve=[0 0
.0099 0
.01 1
.01+ts 0
.1 0 ];

% forcecurve=[0 1
% ts 0];

stopt=.08;

% enter pinpin, fixpin, fixfix, or cantil

```

```

bc_type='pinpin';

midforce=-40; %this gets put into the force vector at midspan (see
line 211)
midforce=0;

distr_load=-.45;      % k/in this gets put into the distr load vector
(see line 258)
%distr_load=0;

axial=0; %enter axial load as proportion of axial cap

% Enter initial conditions in node order
% for nodes with nonzero ICs, list conditions
% example ICu=[1 5 4 0
%              6 -2 0 0];
% ICu is the displacement initial conditions (node, ux0, uy0, r0)
% ICv is the velocity initial conditions (node, vx0, vy0, rdot0,

ICu=[];

ICv=[];

%enter 2 modes followed by damping value as proportion

rayleigh=[1  .02
          2  .02];

%List total length of beam (inches), and number of elements (must be
even)

ltotal=120;
nElem=4; %even number
nNodes=nElem+1;

lElem=ltotal/nElem;

```

```

%{
for below,

material:
    1: 50 ksi steel with 1.05 static increase, 1.19 dynamic
        increase
    2. Unit Material(E=1, G=1, density=1, fy=1

type:
    1: bar (1d or 2d) (Can only have constant distributed
loads)

    4: Bernoulli Beam with decoupled axial (Can only have
constant distributed loads)
    5: Timoshenko Beam with decoupled axial (Can only have
constant distributed loads)
    6: Polynomial Finite Beam (Can have any polynomial
distributed load

% order (for type 6 elements only, if not type 6, enter 0 here):

    1: linear
    2: quadratic
    3: cubic
    etc...

%}

matr1=1;
tp=type;
ord=0;

%Section info

Area=3.54;
Inertia=53.8;
shear_area=9.87*.19;
Z=12.6;
d_1=9.87;
b_1=3.96;
f_1=0.21;
w_1=0.19;

```

```

%% Node Looping

for n=1:1:nNodes

    Nodes(n,1)=(n-1)*lElem; %Creates coordinates of each node
    (constrained to a beam)
    Nodes(n,2)=0;

    if n==1 || n==nNodes %Labels node as exterior or interior

        Node_class(n,1)='E';

    else

        Node_class(n,1)='I';
    end

end

%% Element Looping

for n=1:1:nElem

    ElementNodes(n,1)=n; %Connectivity info (orders nodes left to
    right)
    ElementNodes(n,2)=n+1;

    ElementProp(n,1)=matr1;
    ElementProp(n,2)=tp;
    ElementProp(n,3)=ord;

    ElementSec(n,1)=Area;
    ElementSec(n,2)=Inertia;
    ElementSec(n,3)=shear_area;
    ElementSec(n,4)=Z;
    ElementSec(n,5)=d_1;
    ElementSec(n,6)=b_1;
    ElementSec(n,7)=f_1;
    ElementSec(n,8)=w_1;

end

```

```

%% More Direct Inputs

%list boundary conditions in node order (node, BCx, BCy, BCrot)
%put 0 for constrained, i for free,

if bc_type=='pinpin'

    BC=[1 i 0 i
        nNodes 0 0 i];

elseif bc_type=='fixpin'

    BC=[1 i 0 i
        nNodes 0 0 0];

elseif bc_type=='fixfix'

    BC=[1 i 0 0
        nNodes 0 0 0];

elseif bc_type=='cantil'

    BC=[1 i i i
        nNodes 0 0 0];

end

%list applied nodal forces in order by node (node, x force, y force,
moment)

Force=[

];

Force=    [(1+nNodes)/2 0 midforce 0    %for point load at midspan
           ];

if bc_type=='cantil'

    Force=[];

    Force=[1 0 midforce 0];

```



```

end

%If modal analysis: Enter non-structural masses to nodes by listing
nodes in order, then mass in x, y, and rotation
% (node, x-direction mass, y-direction mass, rotational inertia)

nonstr_mass= [1 1.619171e-4 1.619171e-4 0
              2 3.238342e-4 3.238342e-4 0
              3 3.238342e-4 3.238342e-4 0
              4 3.238342e-4 3.238342e-4 0
              5 1.619171e-4 1.619171e-4 0

              ];

nonstr_mass=[];

%list all distributed forces on elements in order by element(element,
values of constants)
%for values of constants list a's , ao + a1x + a2x^2 + .....)

Axial_distr=      [

                  ];

%list all distributed forces on elements in order by element(element,
values of constants)
%for values of constants list a's , ao + a1x + a2x^2 + .....)

Force_distr=      [

                  ];

for nn=1:1:nElem

    Force_distr(nn,1)=nn;
    Force_distr(nn,2)=distr_load;

end

```

```

%list all distributed forces on elements in order by element(element,
values of constants)
%for values of constants list a's , ao + a1x + a2x^2 + .....)

Moment_distr=      [

                                ];

end

```

Lengths Function

```

%% This function calculates the length and angle for each element sent
to it

function [ang Length] = Lengths(x1, x2, y1, y2)

    Lx=x2-x1; % x component of length
    Ly=y2-y1; % y component of length

    ang=atan(Ly/Lx); %computes angle

    Length=sqrt(Lx^2+Ly^2); %computes length

end

```

Force Time History Function

```

%% This function creates force vs time lines for dynamic analysis

function [F_t t steps] = Forcetime(forcecurve, del, stopt)

ii=sqrt(-1);

steps=stopt/del;

F_t(1)=forcecurve(1,2);

t(1)=forcecurve(1,1);

curvecount=1.0;

```

```

siz=size(forcecurve,1);

forcecurve(siz+1,:)=ii; %adds row of zeros at end

for i=1:1:steps+1

    t(i+1)=t(i)+del;

    if forcecurve(curvecount+1,1)~=ii

        if t(i+1)>=real(forcecurve(curvecount+1,1))+10^-13

            curvecount=curvecount+1;

            end

            slope=(forcecurve(curvecount+1,2)-
forcecurve(curvecount,2))/(forcecurve(curvecount+1,1)-
forcecurve(curvecount,1));
            F_t(i+1)=F_t(i)+del*slope;
            end

            if forcecurve(curvecount+1,1)==ii

                F_t(i+1)=0;

            end

        end

    end

    t=real(t);
    F_t=real(F_t);

end

```

Connectivity Function

```

%% this function calculates the b matrix for each element
(connectivity)

function [b_e] = connectivity(Nodes,NumNodes)

```

```

b_e=zeros(6,NumNodes*3); %first fills with zeros of size, 6 by
Nodes*3

for i=1:1:NumNodes %loops over the nodes

    if Nodes(1,1)==i %if element node 1 is global node of
iteration

        b_e(1,3*i-2)=1;
        b_e(2,3*i-1)=1;
        b_e(3,3*i)=1;

    end

    if Nodes(1,2)==i %if element node 2 is global node of
iteration

        b_e(4,3*i-2)=1;
        b_e(5,3*i-1)=1;
        b_e(6,3*i)=1;

    end

end
end

```

Data Function

```

%% This function contains all the data for materials

function [EM GM DENS Ar Ar_sh Inertia Fy Mpl] = Data(Mat_num,
Type_num, Area, Inert, Flange, Z)

% list of materials E, G, density, fy

materials=[29000 29000/2/(1+.29) .284/386.0886/1000 50*1.05*1.19 %ksi,
ksi, kip*s^2/in^3, ksi
          1 1 1 1];

EM=materials(Mat_num,1); %assigns E to element sent to function
GM=materials(Mat_num,2); %assigns G to element sent to function
DENS=materials(Mat_num,3); %assigns density to element sent to
function
Fy=materials(Mat_num,4);

```

```

Ar=Area;

Ar_sh=Flange; %Shear area

Inertia=Inert;

Mpl=Fy*Z;

end

```

Elemental NBC Function

```

% This function calculates N, B, and C matrices for each element sent
to it

```

```

function [N_e B_e C_e N_e_r B_e_r] = NBC_element (Length, EM, Ar,
Ar_sh, GM, Inertia, Type_num, order)

```

```

E=EM;
L=Length;
A=Ar;
As=Ar_sh;
G=GM;
I=Inertia;

```

```

syms x
z=2/L*x-1; %zeta for shape functions

```

```

%{

```

```

if Type_num==1

```

```

    N=[.5*(1-z)
        .5*(1+z)];

```

```

    N_e=[N(1,1) 0 0 N(2,1) 0 0
          0      0 0 0      0 0
          0      0 0 0      0 0];

```

```

    B_e=diff(N_e);

```

```

    C_e=[E*A 0 0
          0 As*G 0
          0 0 E*I];

```

```

elseif Type_num==4

N=[1/4*(1-z)^2*(2+z)
   1/4*(1-z)^2*(z+1)
   1/4*(1+z)^2*(2-z)
   1/4*(1+z)^2*(z-1)];

Nx=[.5*(1-z)
     .5*(1+z)];

N_e=[Nx(1,1) 0 0 Nx(2,1) 0 0
      0 N(1,1) L/2*N(2,1) 0 N(3,1) L/2*N(4,1)
      0 diff(N(1,1),x) L/2*diff(N(2,1),x) 0 diff(N(3,1),x)
L/2*diff(N(4,1),x)];

B_e=diff(N_e,x);

B_e(2,:)=zeros;

C_e=[E*A 0 0
      0 0 0
      0 0 E*I];

elseif Type_num==5

N=[.5*(1-z)
     .5*(1+z)];

N_e=[N(1,1) 0 0 N(2,1) 0 0
      0 N(1,1) 0 0 N(2,1) 0
      0 0 N(1,1) 0 0 N(2,1)];

B_e=diff(N_e);
B_e(2,3)=-N(1,1);
B_e(2,6)=-N(2,1);

C_e=[E*A 0 0
      0 As*G 0
      0 0 E*I];

%}

n=order+1;

space=2/order;

```

```

zee(1)=-1;

for i=2:1:n
    zee(i)=zee(i-1)+space;
end

for k=1:1:n %calculates lagrange functions
    num1=1;
    for i=1:1:k-1
        num1=num1*(zee(i)-z);
    end
    num2=1;
    for i=k+1:1:n
        num2=num2*(zee(i)-z);
    end
    den1=1;
    for i=1:1:k-1
        den1=den1*(zee(i)-zee(k));
    end
    den2=1;
    for i=k+1:1:n
        den2=den2*(zee(i)-zee(k));
    end
    N(k,1)=num1*num2/(den1*den2);
end

```

```

count=1;

for i=1:3:order*3+1

    N_e(1,i)=N(count,1);
    N_e(2,i+1)=N(count,1);
    N_e(3,i+2)=N(count,1);

    count=count+1;

end

%reorders N matrix so that external nodes are first 2 nodes

for i=1:1:3

    N_e_r(i,i)=N_e(i,i);

end

for i=1:1:3

    N_e_r(i,3+i)=N_e(i, 3*(order)+i);

end

for i=4:3:order*3

    N_e_r(1,i+3)=N_e(1,i);
    N_e_r(2,i+4)=N_e(2,i+1);
    N_e_r(3,i+5)=N_e(3,i+2);

end

B_e=diff(N_e,x);

B_e_r=diff(N_e_r,x);

```



```
for i=1:3:order*3+1 %adds the -1 part for B matrices
```

```
    B_e(2,i+2)=-N_e(1,i);  
    B_e_r(2,i+2)=-N_e_r(1,i);
```

```
end
```

```
C_e=[E*A 0 0  
     0 As*G 0  
     0 0 E*I];
```

```
end
```

K_Element Function (Elemental Stiffness and Mass Matrices)

```
% This calculates k matrix (and mass if modal) for each element sent  
to it
```

```
function [k_e m_e r kms kss] = k_element(Length, EM, Ar, Ar_sh, GM,  
Inertia, Type_num, B_e_r, C_e, N_e_r, lumping, analysis, DENS,  
Finternal, Mp, nodeclass1, nodeclass2) %Stiffness Matrix Function
```

```
    L=Length;  
    E=EM;  
    A=Ar;  
    As=Ar_sh;  
    G=GM;  
    I=Inertia;  
    type=Type_num;  
    density=DENS;
```

```
if type~=6 %If not polynomial finite element
```

```
    if analysis==1 %if not modal, set mass equal to 0
```

```
        m_e=0;  
    end
```

```
    r=12*E*I/L^2/G/As; %rho for timoshenko  
    beta=r/12; %beta for timoshenko mass
```

```

%block that defines the type of stiffness matrix
  if type==1 %if bar element

      I=0;
      r=0;

      elseif type==4 %if bernoulli element

          r=0;

      end

      k_e1=E*A/L*[1 -1;-1,1]; %element stiffness for bar

      if analysis==2 %if modal, element mass for bar

          mass_e=density*L*A;

          m_e1=mass_e/6*[2 1; 1 2];

          if lumping==2

              m_e_new=zeros;

              for i=[1 2]

                  m_e_new(i,i)=density*L*A/2;

              end

              m_e1=m_e_new;

          end

          if type==1 %if bar, zeros mass so following calcs are zero for
bending

              mass_e=0;

          end

      end

      moment_left=Finternal(3,1);
      moment_right=Finternal(6,1);

      %initially sets element stiffness as having no hinges

```

```

k_e2=E*I/L^3/(1+r)*[12      6*L      -12      6*L %element stiffness
for beam
                        6*L      L^2*(4+r)  -6*L      L^2*(2-r)
-12      -6*L      12      -6*L
6*L      L^2*(2-r)  -6*L      L^2*(4+r)];

if nodeclass2=='I' %any member except rightmost
member

if abs(moment_right)>=Mp %Hinge at right node

k_e2=Condense(k_e2);

end

end

if analysis==2 %if modal, element mass for beam

if type==5 %if timoshenko element

comp1=[13/35+7/10*r+1/3*r^2
(11/210+11/120*r+1/24*r^2)*L      9/70+3/10*r+1/6*r^2      -
(13/420+3/40*r+1/24*r^2)*L
(11/210+11/120*r+1/24*r^2)*L
(1/105+1/60*r+1/120*r^2)*L^2      (13/420+3/40*r+1/24*r^2)*L      -
(1/140+1/60*r+1/120*r^2)*L^2
9/70+3/10*r+1/6*r^2
(13/420+3/40*r+1/24*r^2)*L      13/35+7/10*r+1/3*r^2      -
(11/210+11/120*r+1/24*r^2)*L
- (13/420+3/40*r+1/24*r^2)*L      -
(1/140+1/60*r+1/120*r^2)*L^2      - (11/210+11/120*r+1/24*r^2)*L
(1/105+1/60*r+1/120*r^2)*L^2];

radius=sqrt(I/A);

comp2=(radius/L)^2*[6/5      (1/10+1/2*r)*L
-6/5      (1/10-1/2*r)*L
(1/10+1/2*r)*L
(2/15+1/6*r+1/3*r^2)*L^2      (-1/10+1/2*r)*L      (-1/30-
1/6*r+1/6*r^2)*L^2

```

```

        -6/5          (-1/10+1/2*r)*L
6/5          (-1/10+1/2*r)*L          (1/10-1/2*r)*L  (-1/30-
1/6*r+1/6*r^2)*L^2  (-1/10+1/2*r)*L  (2/15+1/6*r+1/3*r^2)*L^2];

    m_e2=density*A*L/(1+r)^2*(comp1+comp2);

elseif type==4 %if bernoulli element

    beta=0;

    m_e2=mass_e*[48*beta^2+42/5*beta+13/35 -(-6*beta^2-
11/10*beta-11/210)*L 24*beta^2+18/5*beta+9/70 -
(6*beta^2+9/10*beta+13/420)*L
        -(-6*beta^2-11/10*beta-11/210)*L
(6/5*beta^2+1/5*beta+1/105)*L^2 -(-6*beta^2-9/10*beta-13/420)*L (-
6/5*beta^2-1/5*beta-1/140)*L^2
        24*beta^2+18/5*beta+9/70 -(-6*beta^2-
9/10*beta-13/420)*L 48*beta^2+42/5*beta+13/35 -
(6*beta^2+11/10*beta+11/210)*L
        -(6*beta^2+9/10*beta+13/420)*L (-
6/5*beta^2-1/5*beta-1/140)*L^2 -(6*beta^2+11/10*beta+11/210)*L
(6/5*beta^2+1/5*beta+1/105)*L^2];

    end

    if lumping==2

        for i=[1 3] %fills translationals

            m_e_new(i,i)=density*L*A/2;

        end

        for i=[2 4] %fills rotationals

            m_e_new(i,i)=density*L*A*L^2/24;

        end

        m_e2=m_e_new;

    end

    if nodeclass2=='I' %any member except rightmost member

```

```

        if abs(moment_right)>=Mp %Hinge at right node

            m_e2=Condense(m_e2);

        end

    end

end

    k_e(2,:)=k_e2(1,:); %shifting the k_e2 matrix with zeros in
axial spots
    k_e(3,:)=k_e2(2,:);
    k_e(5,:)=k_e2(3,:);
    k_e(6,:)=k_e2(4,:);
    k_e(4,:)=zeros;
    k_e(1,:)=zeros;

    k_e(:,6)=k_e(:,4);
    k_e(:,5)=k_e(:,3);
    k_e(:,4)=zeros;
    k_e(:,3)=k_e(:,2);
    k_e(:,2)=k_e(:,1);
    k_e(:,1)=zeros;

    if analysis==2 %if modal, shifting the m_e2 matrix with zeros
in axial spots

        m_e(2,:)=m_e2(1,:);
        m_e(3,:)=m_e2(2,:);
        m_e(5,:)=m_e2(3,:);
        m_e(6,:)=m_e2(4,:);
        m_e(4,:)=zeros;
        m_e(1,:)=zeros;

        m_e(:,6)=m_e(:,4);
        m_e(:,5)=m_e(:,3);
        m_e(:,4)=zeros;
        m_e(:,3)=m_e(:,2);
        m_e(:,2)=m_e(:,1);
        m_e(:,1)=zeros;

    end

```

```

for i=[1,4] %fills stiffness matrix for axial components
    for j=[1,4]
        k_e(i,j)=k_e1(sqrt(i),sqrt(j));

        if analysis==2 %if modal
            m_e(i,j)=m_e1(sqrt(i),sqrt(j));
        end
    end
end

end

end

kms=0;
kss=0;

elseif type==6 %if polynomial finite element

    if analysis==1 %if not modal, set mass equal to 0
        m_e=0;
    end

    r=0;
    syms x

    indef=int(transpose(B_e_r)*C_e*B_e_r);

    smaller=subs(indef,x,0);

    larger=subs(indef,x,L);

    k_e=larger-smaller;

    k_e=double(k_e);

    if analysis==2 %if modal

        sizeM=size(N_e_r,2);

```

```

for i=1:3:sizeM-2

    N_e_m(i,1)=N_e_r(1,i);
    N_e_m(i+1,1)=N_e_r(2,i+1);
    N_e_m(i+2,1)=N_e_r(3,i+2);

end

indefM=int(N_e_m*transpose(N_e_m));

smallerM=subs(indefM,x,0);

largerM=subs(indefM,x,L);

m_e=DENS*A*(largerM-smallerM)
end

sizek=size(k_e,1);

if sizek>6

    %static condensation to get rid of internal nodes

    for i=1:1:6 % loop for kmm matrix

        for j=1:1:6

            kmm(i,j)=k_e(i,j);

        end

    end

end

for i=1:1:sizek-6 %loop for kss matrix

    for j=1:1:sizek-6

        kss(i,j)=k_e(i+6,j+6);

        if analysis==2

            mss(i,j)=m_e(i+6,j+6);

        end

    end

end

```

```

end

for i=1:1:sizek-6 %loop for ksm matrix
    for j=1:1:6
        ksm(i,j)=k_e(i+6,j);
        if analysis==2
            msm(i,j)=m_e(i+6,j);
        end
    end
end

end

kms=transpose(ksm);
k_e=kmm-kms*inv(kss)*ksm;
if analysis==2
    mbot=-inv(mss)*msm;
    sizembot=size(mbot,2);
    Ident=eye(sizembot);
    TransfM=vertcat(Ident,mbot);
    rows0=size(TransfM,1);
    cols0=size(mbot,2);
    for i=cols0+1:1:rows0
        TransfM(:,i)=0;
    end
end

m_e=transpose(TransfM)*m_e*TransfM;
end

```



```

else
    kms=0;
    kss=0;
end

```

```
end
```

```
end
```

K_geometric Function (Geometric Stiffness Matrices)

```

%% This calculates k matrix (and mass if modal) for each element sent
to it

```

```

function [k_g] = k_geom(L,P,r) %Stiffness Matrix Function

```

```

k_g=-P/L/(1+r)^2*[1 0          0          -1 0
0          0 6/5+2*r+r^2  L/10          %geometric stiffness
2*r-r^2  L/10          0 L/10          2*L^2/15+L^2*r/6+L^2*r^2/12 0 -L/10
-L^2/30-L^2*r/6-L^2*r^2/12 -1 0          0          1 0
0          0 -6/5-2*r-r^2 -L/10          0
6/5+2*r+r^2 -L/10          0 L/10          -L^2/30-L^2*r/6-L^2*r^2/12 0 -L/10
2*L^2/15+L^2*r/6+L^2*r^2/12];

```

```
end
```

G_Element Function (Equivalent Nodal Loads)

```
%% This function calculates g for each element

function [g_e] = g_element(Length, degree1, degree2, degree3,
axial_nodal, forces_nodal, moment_nodal, N_e_r, Type_num, r, kms, kss,
Finternal, Mp, nodeclass1, nodeclass2) %Equivalent Load Function

L=Length;

Pbeam=forces_nodal(1,2);
Pbar=axial_nodal(1,2);
Pmoment=moment_nodal(1,2);

Loadvect=[Pbar
          Pbeam
          Pmoment];

if Type_num==6

    g=int(transpose(N_e_r)*Loadvect,x);

    smaller=subs(g,x,0);

    larger=subs(g,x,L);

    g_e=larger-smaller;

    g_e=double(g_e);

    sizeg=size(g_e,1);

    if sizeg>6

        %static condensation to get rid of internal nodes

        for i=1:1:6 %loop for gm vector

            gm(i,1)=g_e(i,1);

        end

        for i=1:1:sizeg-6 %loop for gs vector
```

```

        gs(i,1)=g_e(i+6,1);

    end

    g_e=gm-kms*inv(kss)*gs;

end

else

moment_left=Finternal(3,1);
moment_right=Finternal(6,1);

%%initially sets equivalent load vector as having no hinges

a=[0 0
   -L/2 1/(1+r)
   -L^2/12 -L*r/2/(1+r)
   0 0
   -L/2 -1/(1+r)
   L^2/12 -L*r/2/(1+r)];

b=[Pbeam
   Pmoment];

g_e=-a*b;

if abs(moment_right)>=Mp %Hinge at right node

    a=[0 0
       -5*L/8 1/(1+r)
       -L^2/8 -L*r/2/(1+r)
       0 0
       -3*L/8 -1/(1+r)
       0 -L*r/2/(1+r)];
    %

    a=[0 0
       -7*L/12 1/(1+r)
       -L^2/12 -L*r/2/(1+r)
       0 0
       -5*L/12 -1/(1+r)
       0 -L*r/2/(1+r)];
    % %

    %

    b=[Pbeam

```

```

        Pmoment];

    g_e=-a*b;

end

if nodeclass1=='E'

    if abs(moment_left)>=Mp %hinge at leftmost node

        a=[0 0
            -3*L/8 1/(1+r)
            0 -L*r/2/(1+r)
            0 0
            -5*L/8 -1/(1+r)
            L^2/8 -L*r/2/(1+r)];

        a=[0 0
            -5*L/12 1/(1+r)
            0 -L*r/2/(1+r)
            0 0
            -7*L/12 -1/(1+r)
            L^2/12 -L*r/2/(1+r)];

        %% %%
        %% %%
        %% %%
        %% %%
        %% %%
        %% %%

        b=[Pbeam
            Pmoment];

        g_e=-a*b;

    end
end

g_e(1,1)=Pbar*L/2;

g_e(4,1)=Pbar*L/2;

end

end

```

Damping Function

```

%% This function creates Rayleigh Damping Matrix

function [c_matrix_alpha c_matrix_beta] = Damping(kglobal_BC2,
mglobal_BC2, c_alpha, c_beta, damp_switch)

m=mglobal_BC2;

k=kglobal_BC2;

a=c_alpha;

b=c_beta;

c_matrix_alpha=m*a;
c_matrix_beta=k*b*damp_switch;

end

```

Condense Function (Static Condensation for Right Rotation)

```

%% This function condenses the right rotation out of the matrix

function [ksend Tr] = Condense(k)

for i=1:1:5

    kab(i,1)=k(i,6);
end

kbb=k(6,6);

kinput=-kbb^-1*transpose(kab);

Id=[1 0 0 0 0;0 1 0 0 0; 0 0 1 0 0; 0 0 0 1 0; 0 0 0 0 1];

coll1=vertcat(Id,kinput);

col2=[0;0;0;0;0;0];

T=horzcat(coll1,col2);

ksend=transpose(T)*k*T;

Tr=kinput;

```

```
end
```

Condense Left Function (Static Condensation for Left Rotation)

```
%% This function condenses the right rotation out of the matrix
```

```
function [ksend Tr] = Condenseleft(k)
```

```
for i=1:1:5
```

```
    kab(i,1)=k(i,6);
```

```
end
```

```
kbb=k(6,6);
```

```
kinput=-kbb^-1*transpose(kab);
```

```
Id=[1 0 0 0 0;0 1 0 0 0; 0 0 1 0 0; 0 0 0 1 0; 0 0 0 0 1];
```

```
coll=vertcat(Id,kinput);
```

```
col2=[0;0;0;0;0;0];
```

```
T=horzcat(coll,col2);
```

```
kcond=transpose(T)*k*T;
```

```
kint=kcond;
```

```
kint(:,3)=kcond(:,6);
```

```
kint(:,6)=kcond(:,3);
```

```
kint2=kint;
```

```
kint2(3,:)=kint(6,:);
```

```
kint2(6,:)=kint(3,:);
```

```
ksend=kint2;
```

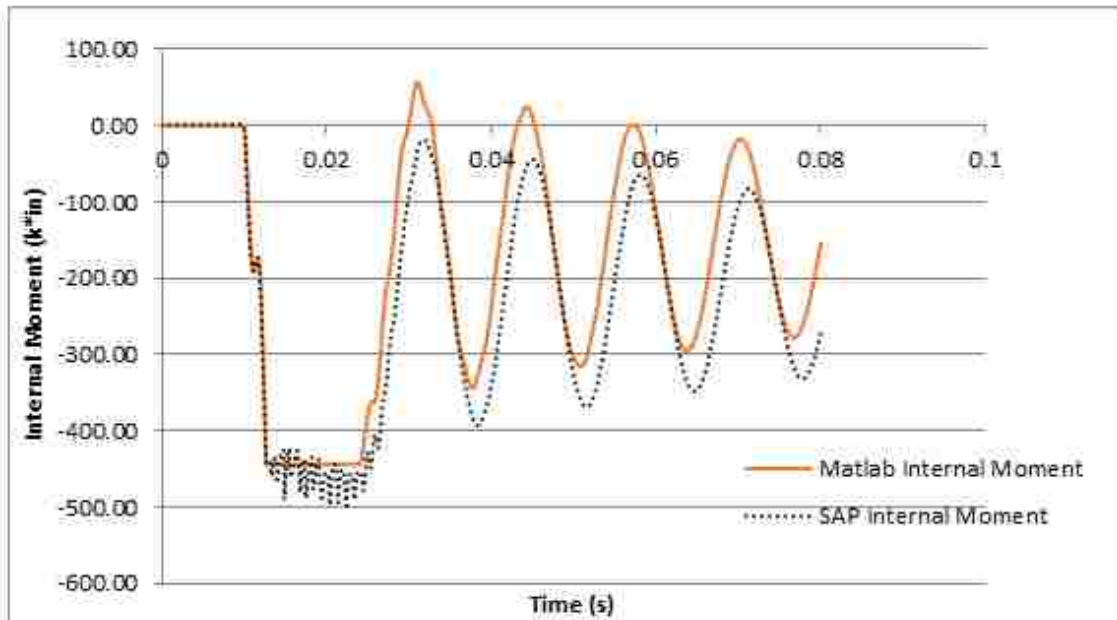
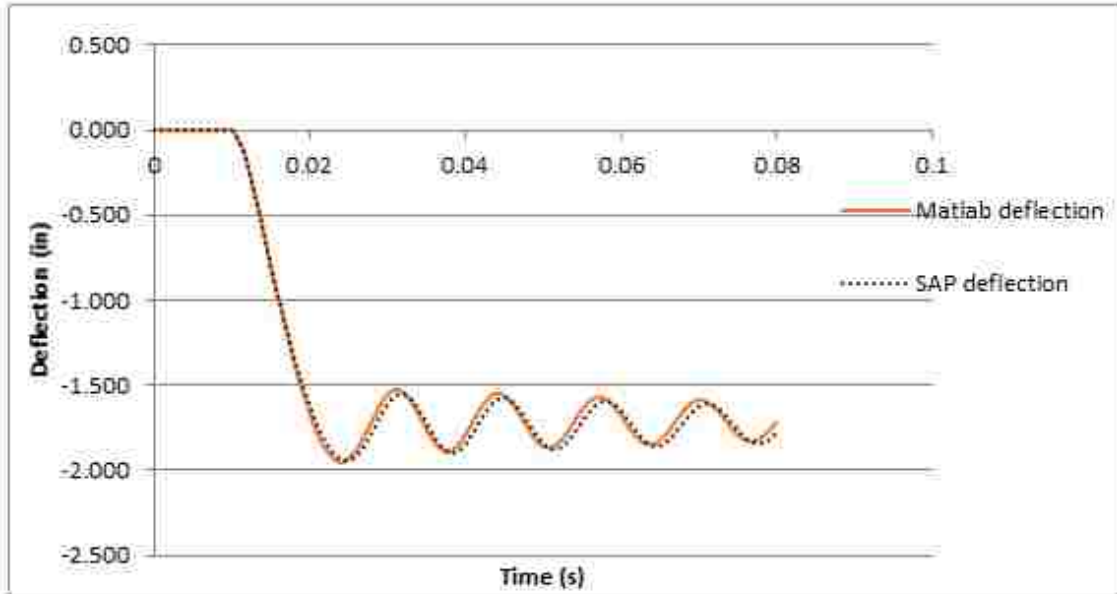
```
Tr=kinput;
```

```
end
```

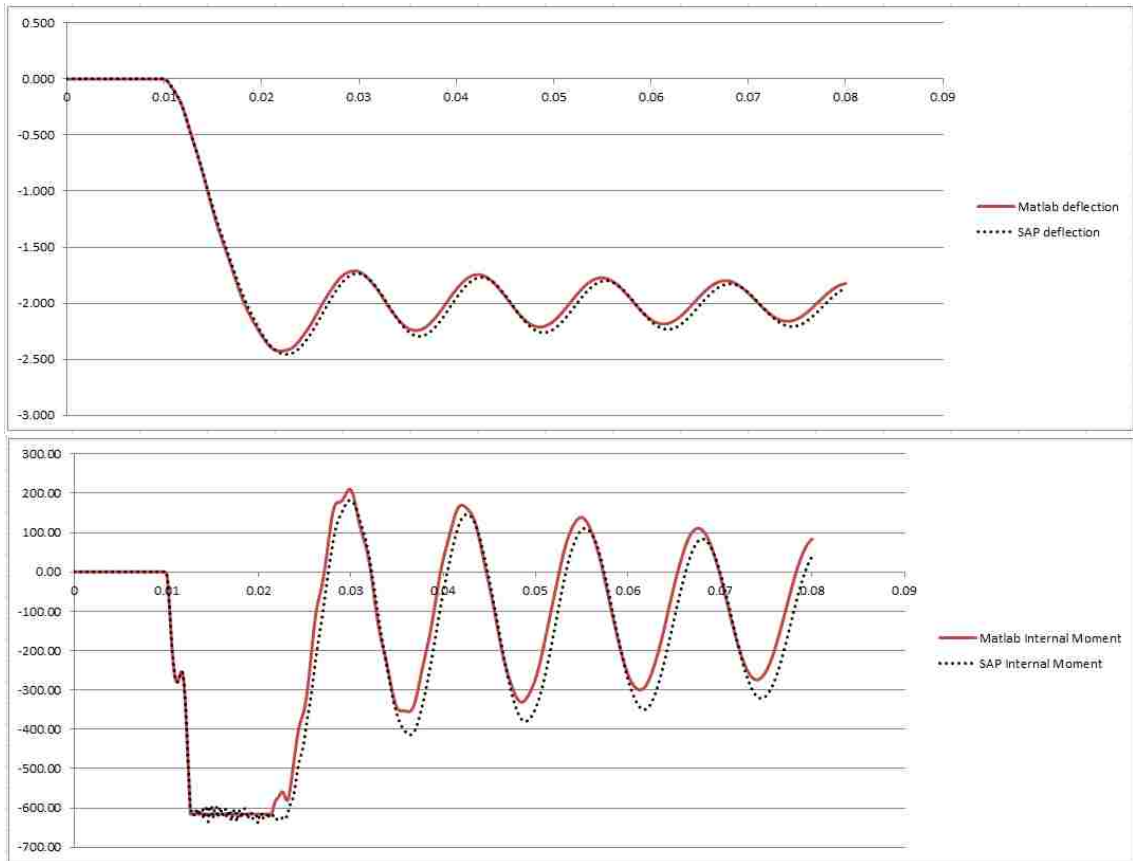
Appendix D: MDOF Solver Verifications (Time-History Plots)

All verifications use 15 ft Length, 4 Elements, W10X12, 2% damping in first two modes unless specified otherwise

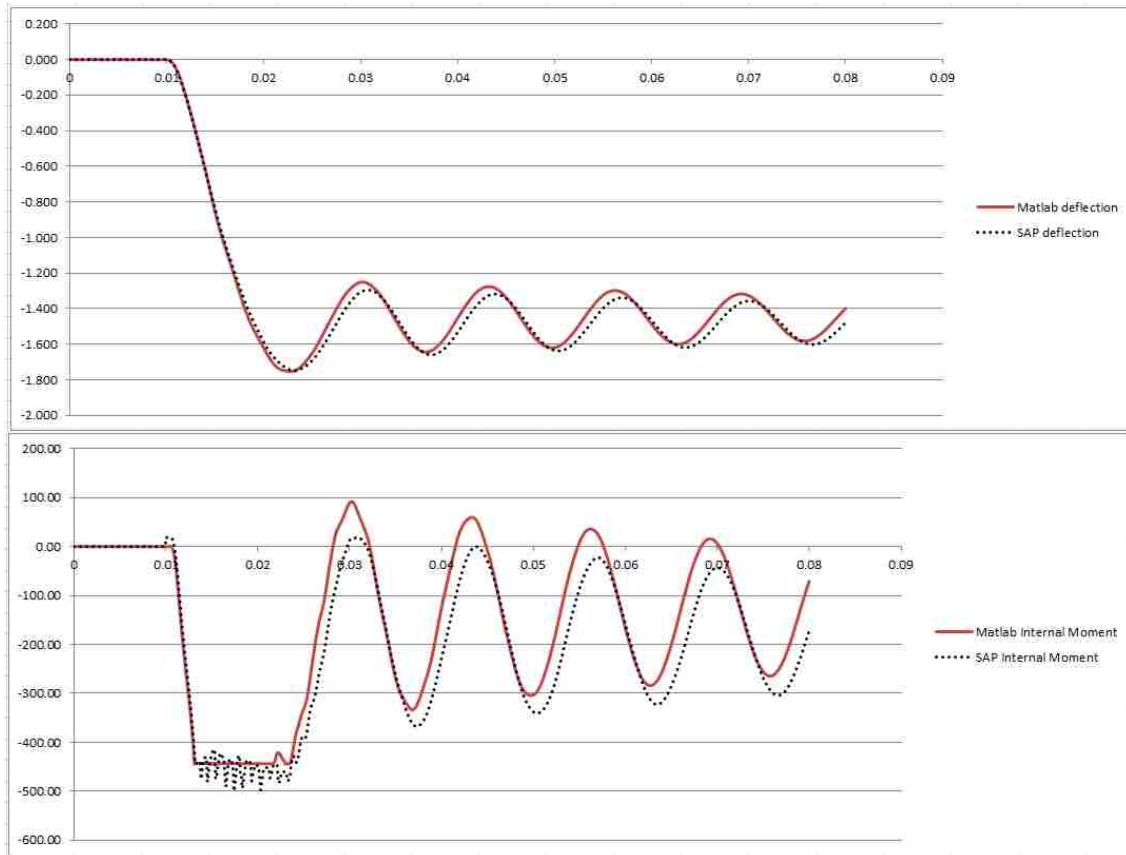
Pinned-Pinned Point Load (17 k in 17.8 ms) 40% Axial Capacity Applied



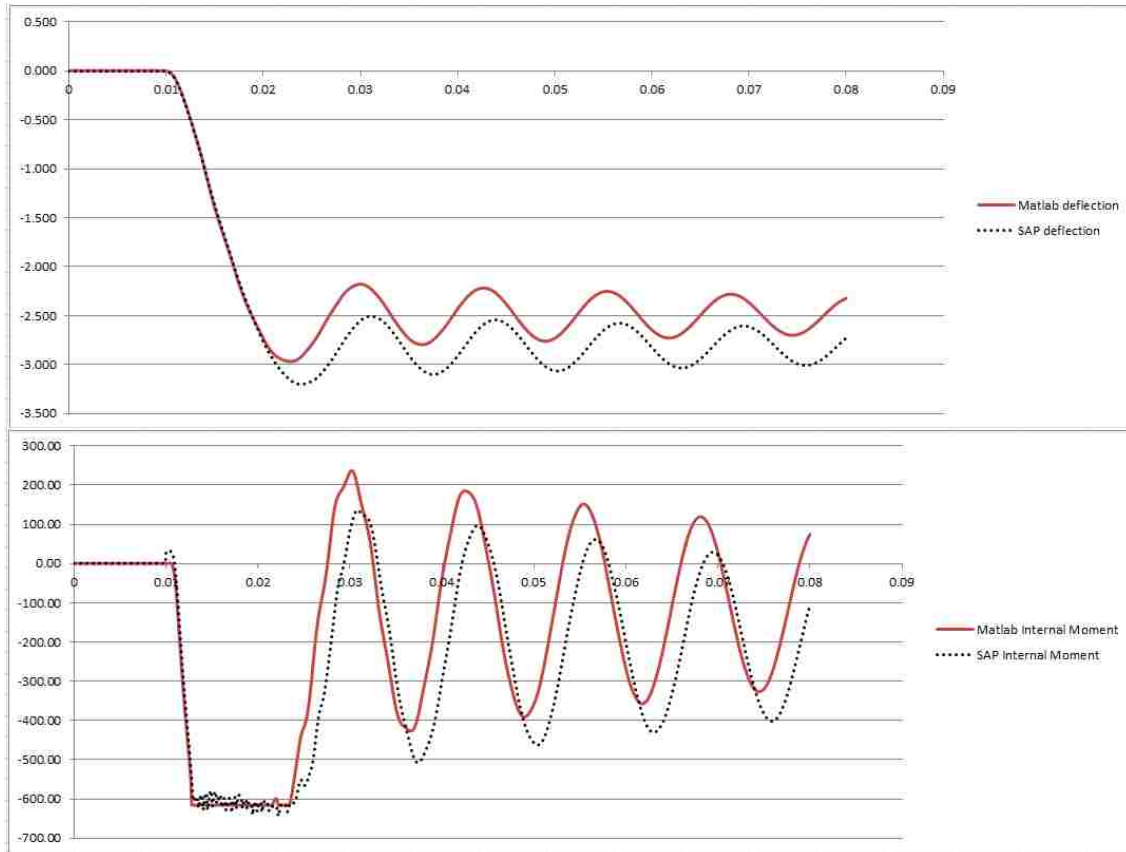
Pinned-Pinned Point Load (30 k in 17.8 ms) 20% Axial Capacity Applied



Pinned-Pinned Uniform Load (0.25 k/in in 17.8 ms) 40% Axial Capacity Applied



Pinned-Pinned Uniform Load (0.4 k/in in 17.8 ms) 20% Axial Capacity Applied

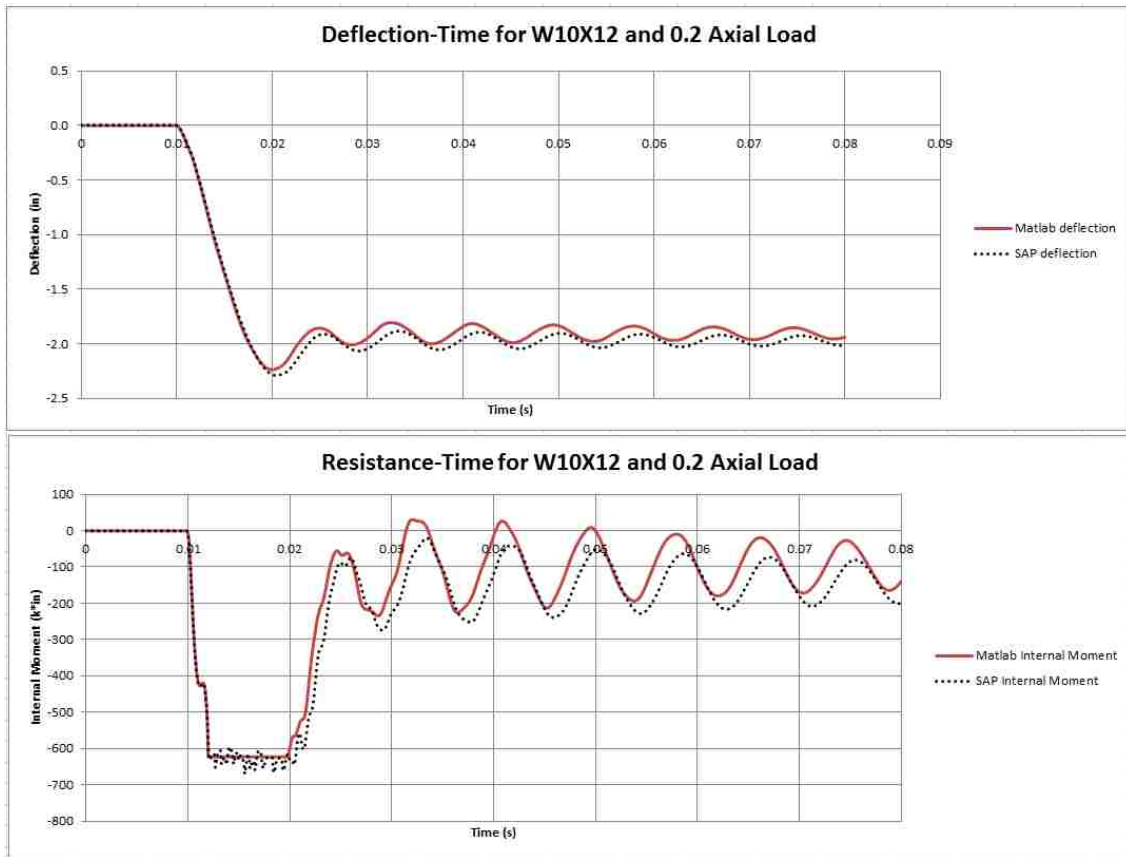


Large deviation in deflection curve (7%) likely due to poorly adjusted yield stress in SAP2000, causing a smaller reduced plastic moment capacity

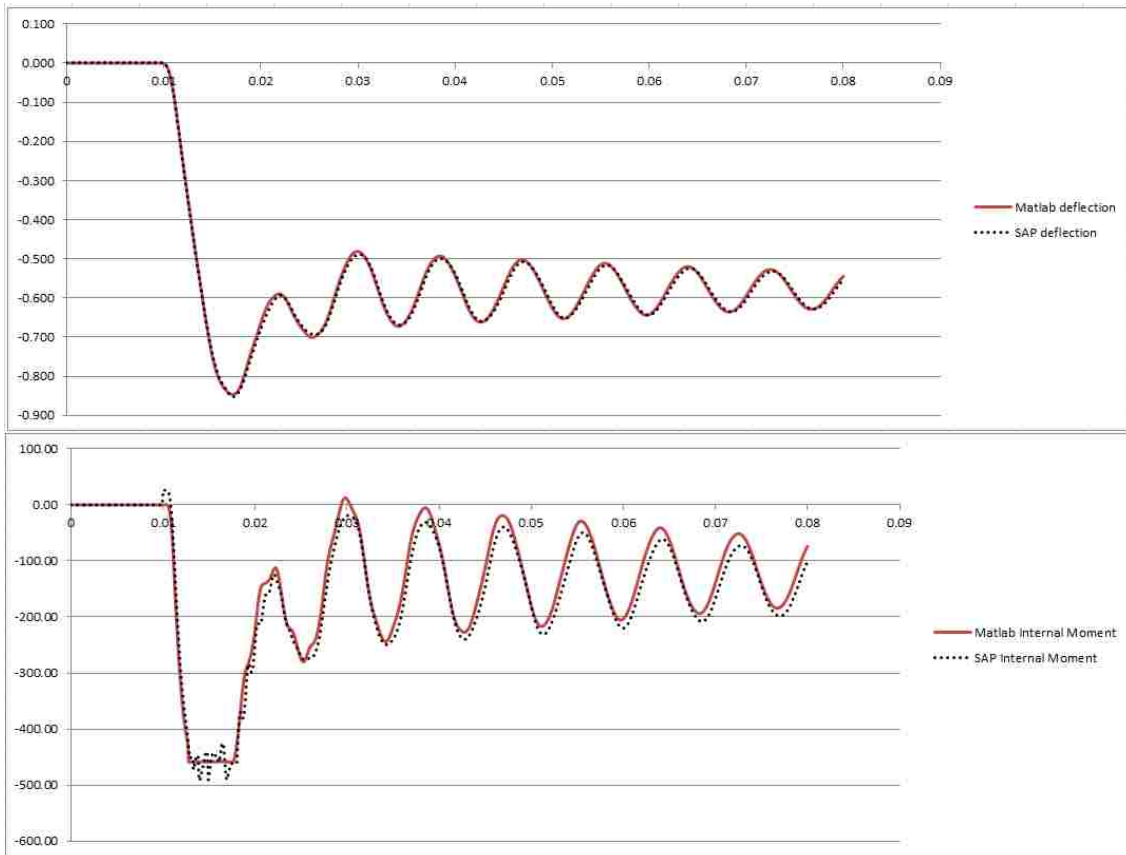
Fixed-Pinned Point Load (25 k in 17.8 ms) 40% Axial Capacity Applied



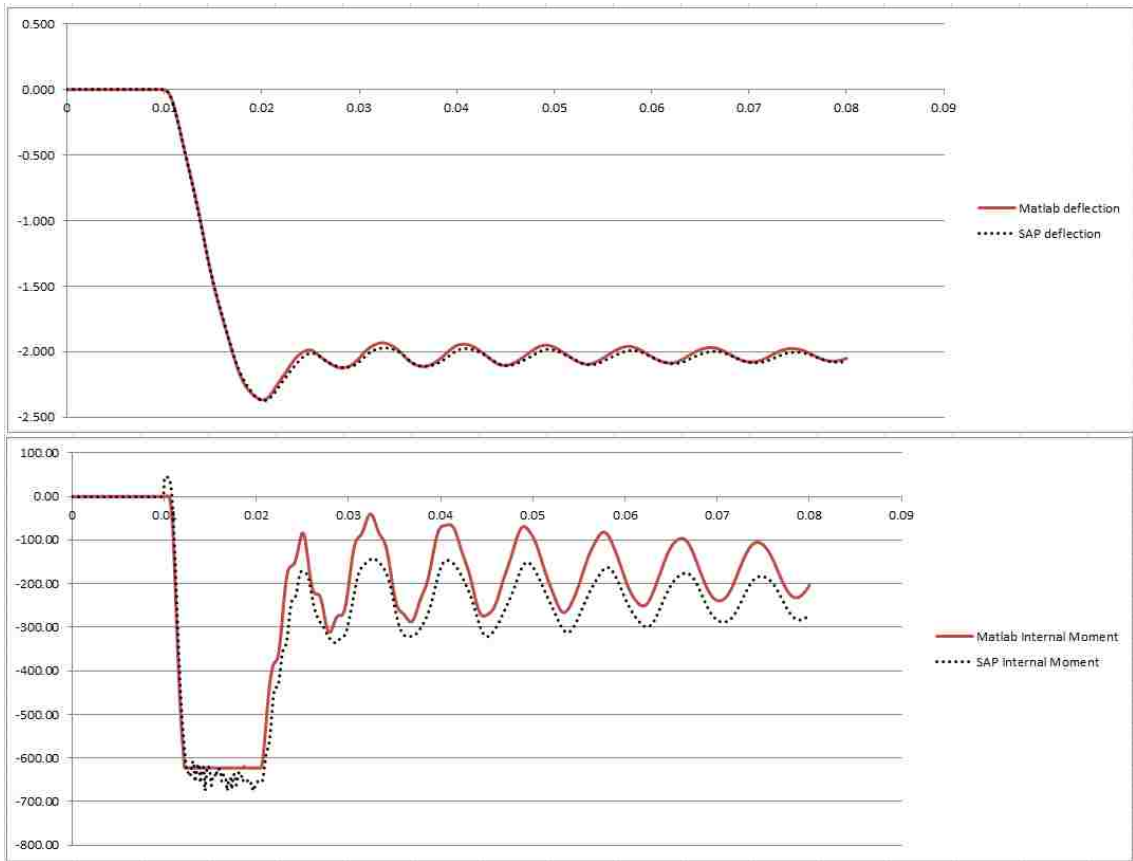
Fixed-Pinned Point Load (37.5 k in 17.8 ms) 20% Axial Capacity Applied



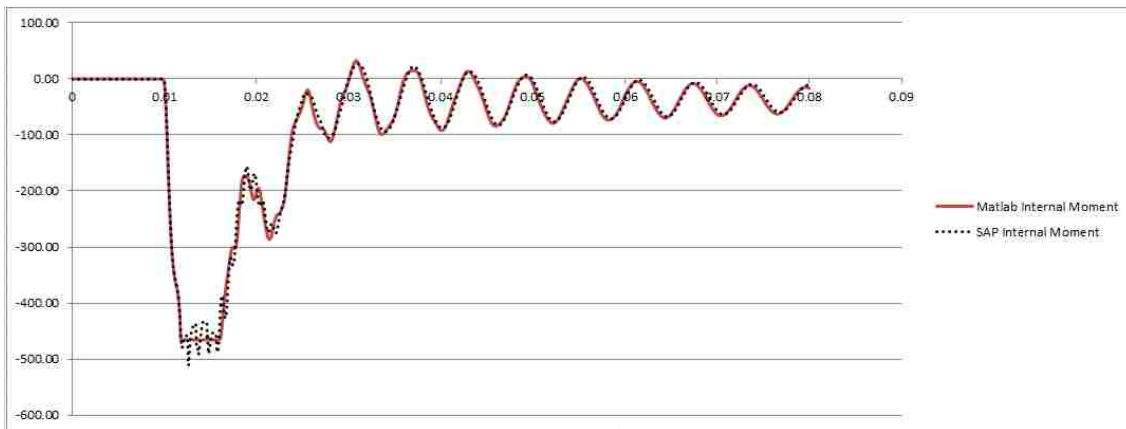
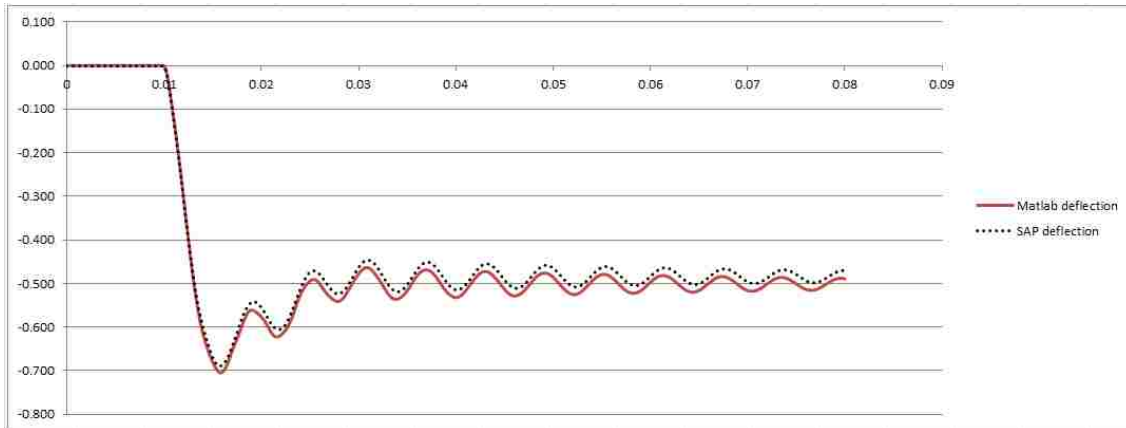
Fixed-Pinned Uniform Load (0.35 k/in in 17.8 ms) 40% Axial Capacity Applied



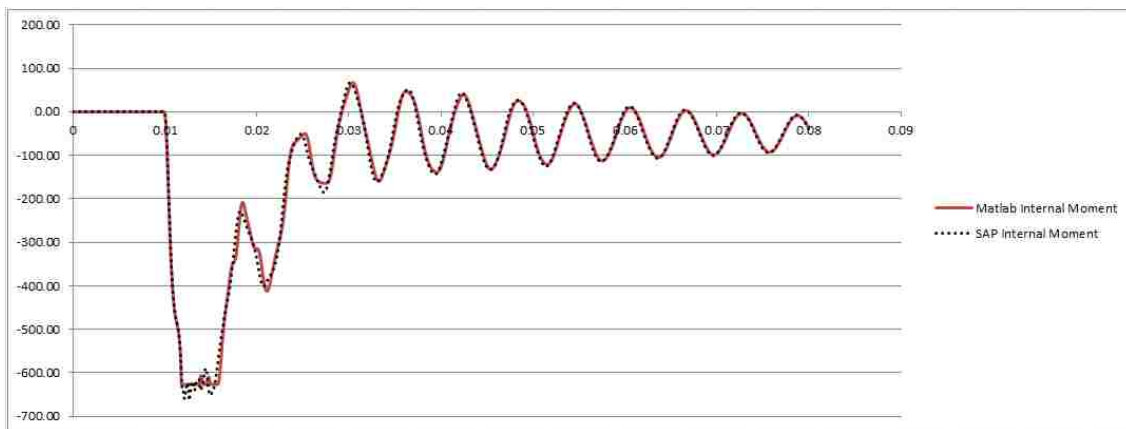
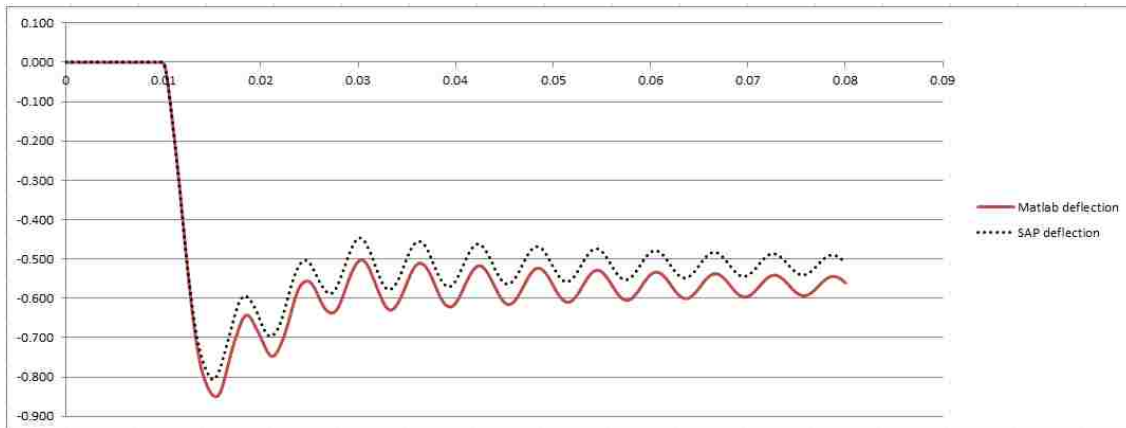
Fixed-Pinned Uniform Load (0.6 k/in in 17.8 ms) 20% Axial Capacity Applied



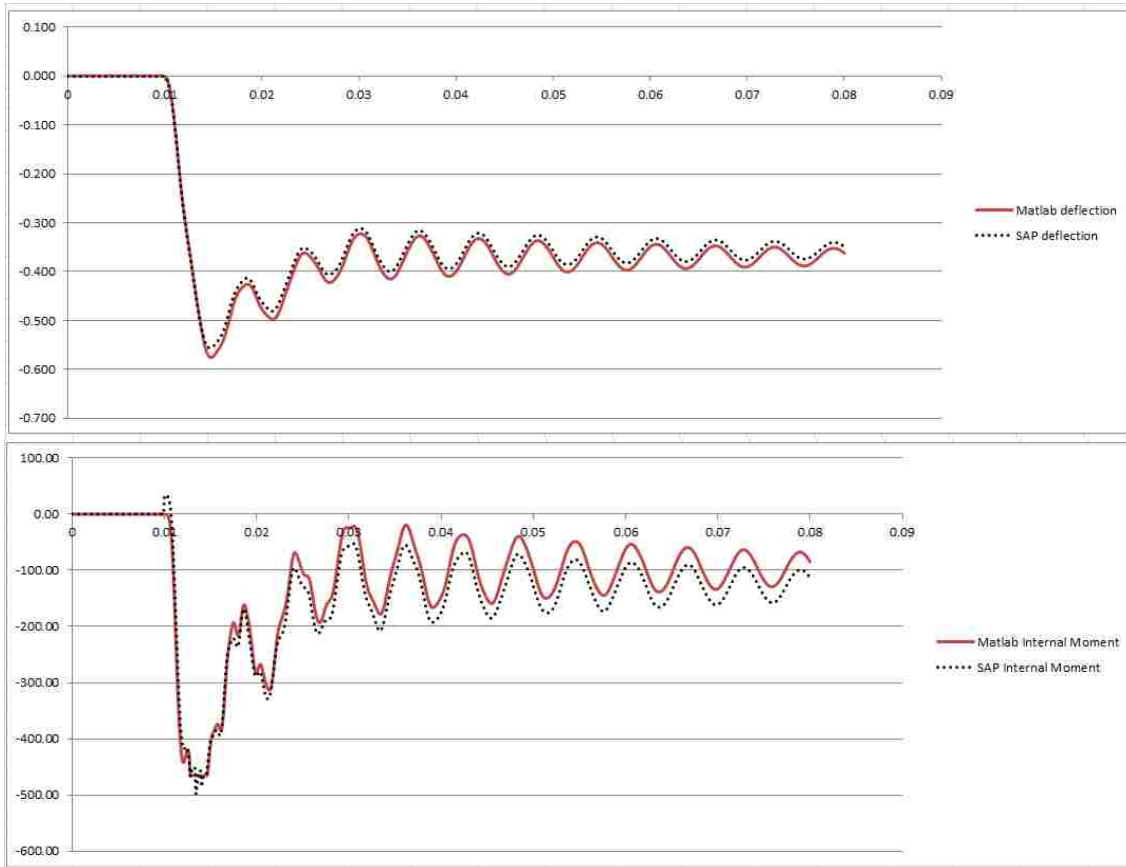
Fixed-Fixed Point Load (30 k in 17.8 ms) 40% Axial Capacity Applied



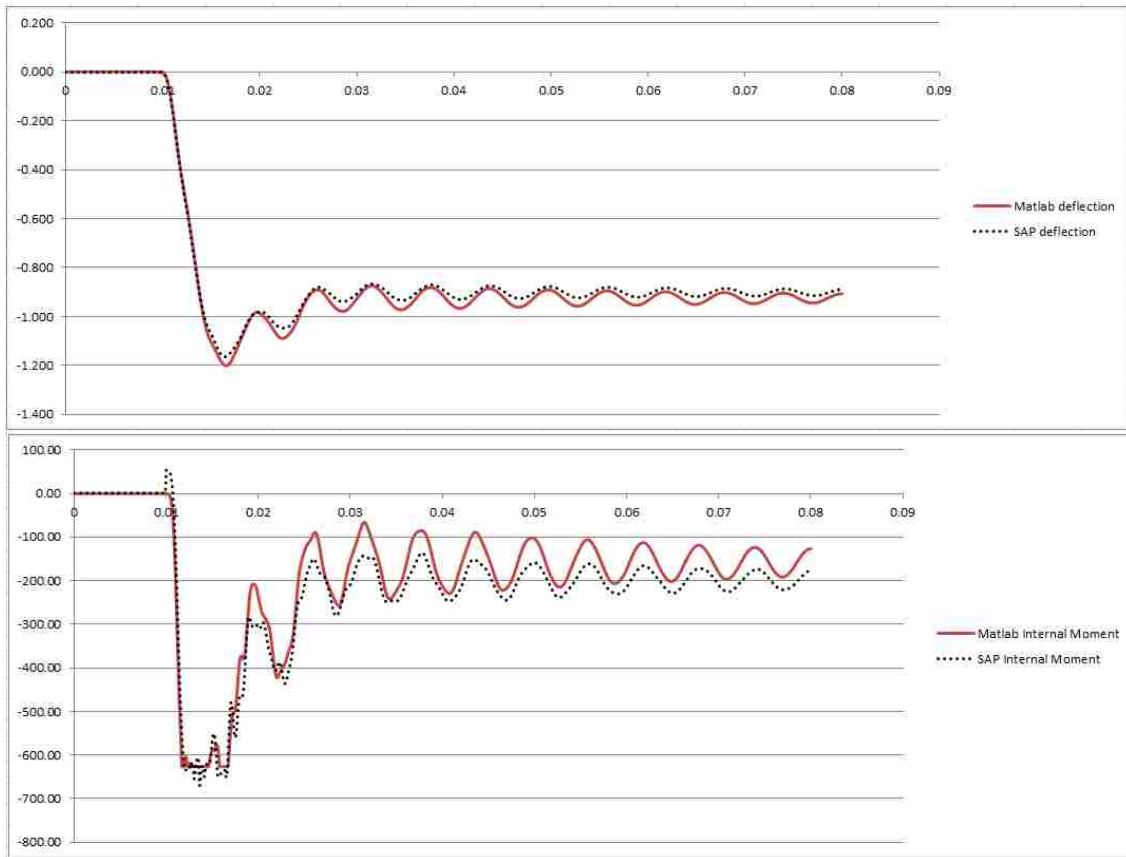
Fixed-Fixed Point Load (40 k in 17.8 ms) 20% Axial Capacity Applied



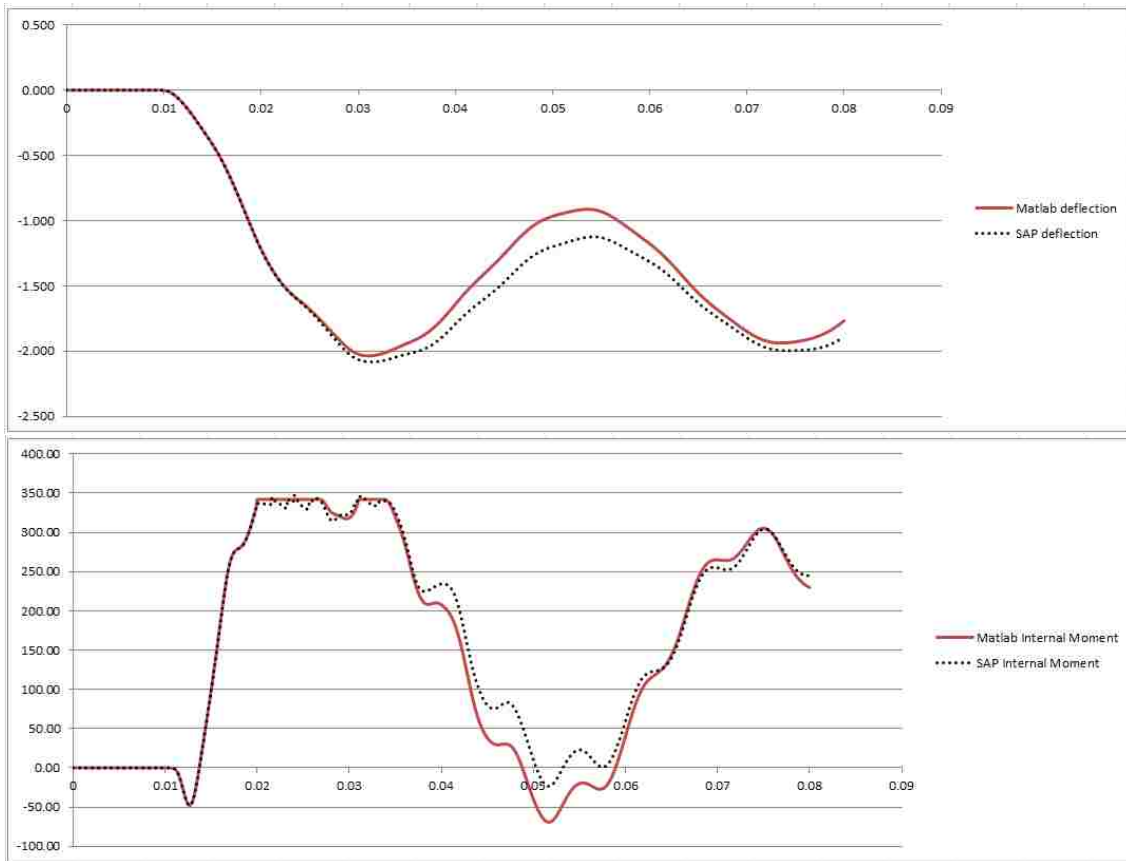
Fixed-Fixed Uniform Load (0.45 k/in in 17.8 ms) 40% Axial Capacity Applied



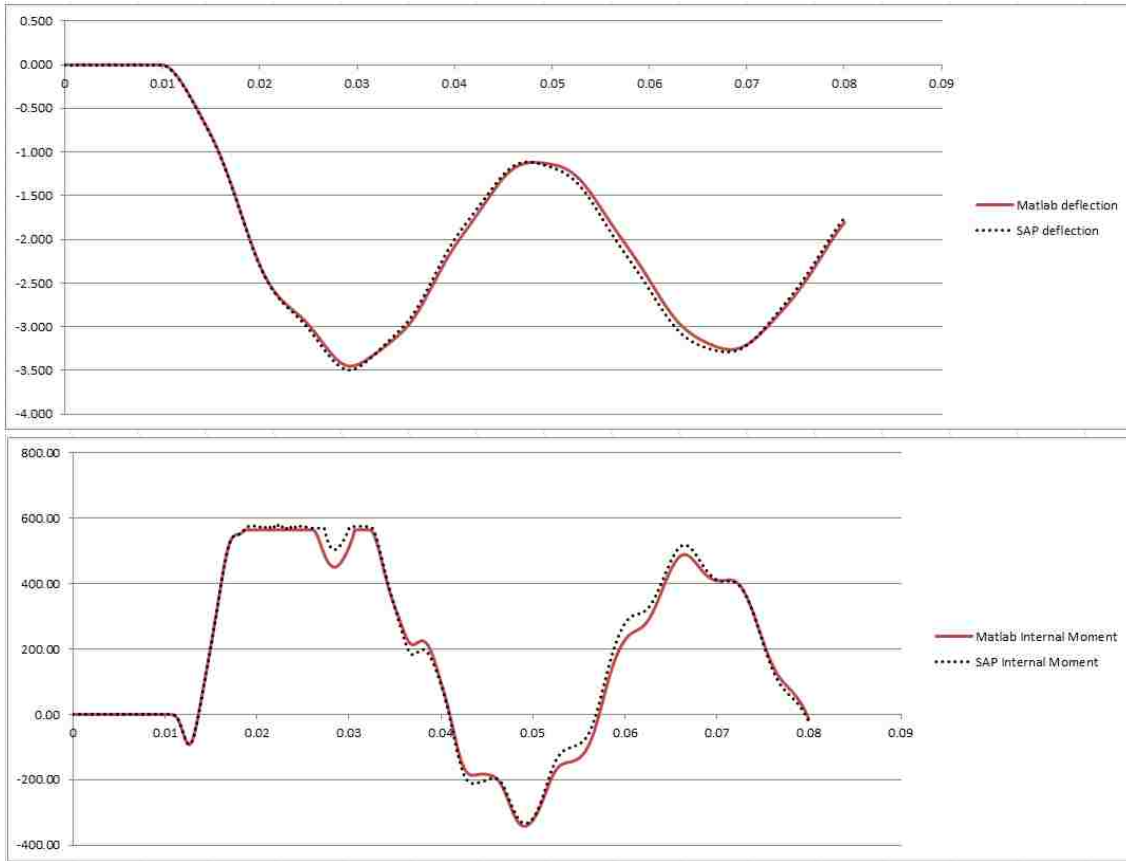
Fixed-Fixed Uniform Load (0.7 k/in in 17.8 ms) 20% Axial Capacity Applied



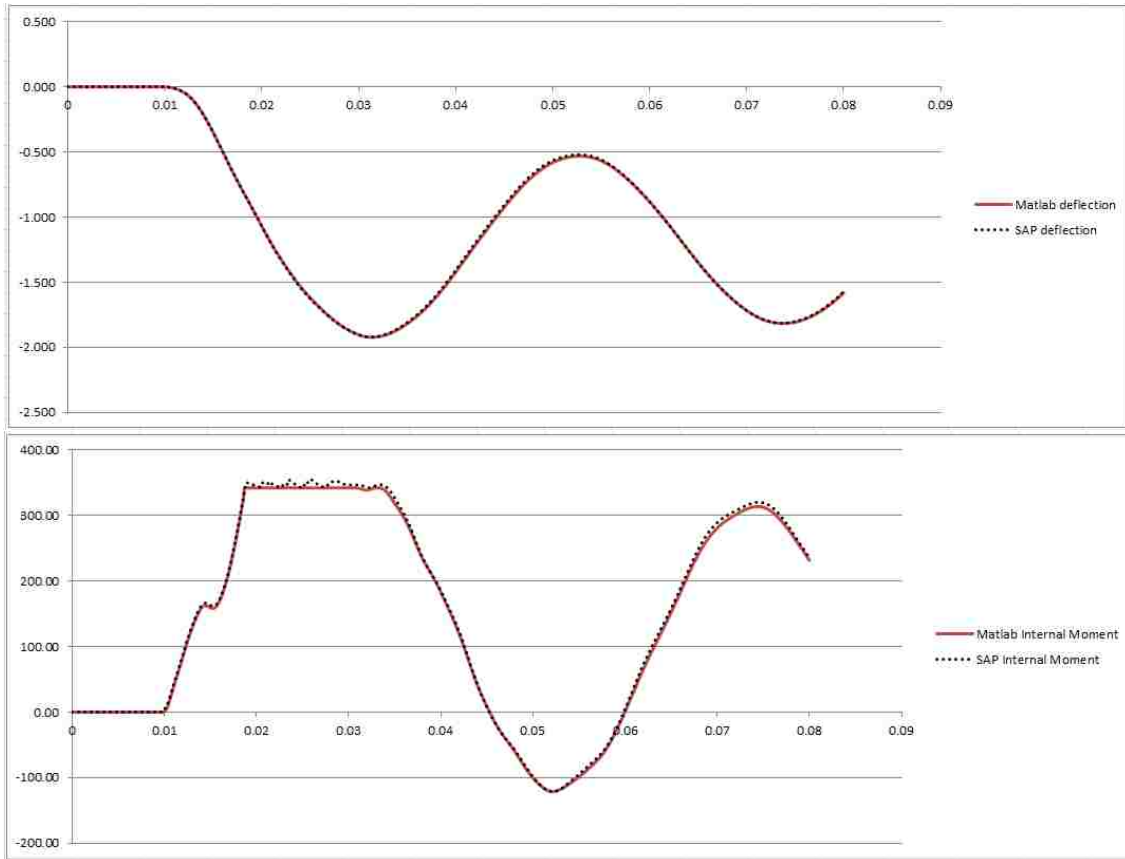
Cantilever Point Load (3 k in 17.8 ms) 40% Axial Capacity Applied



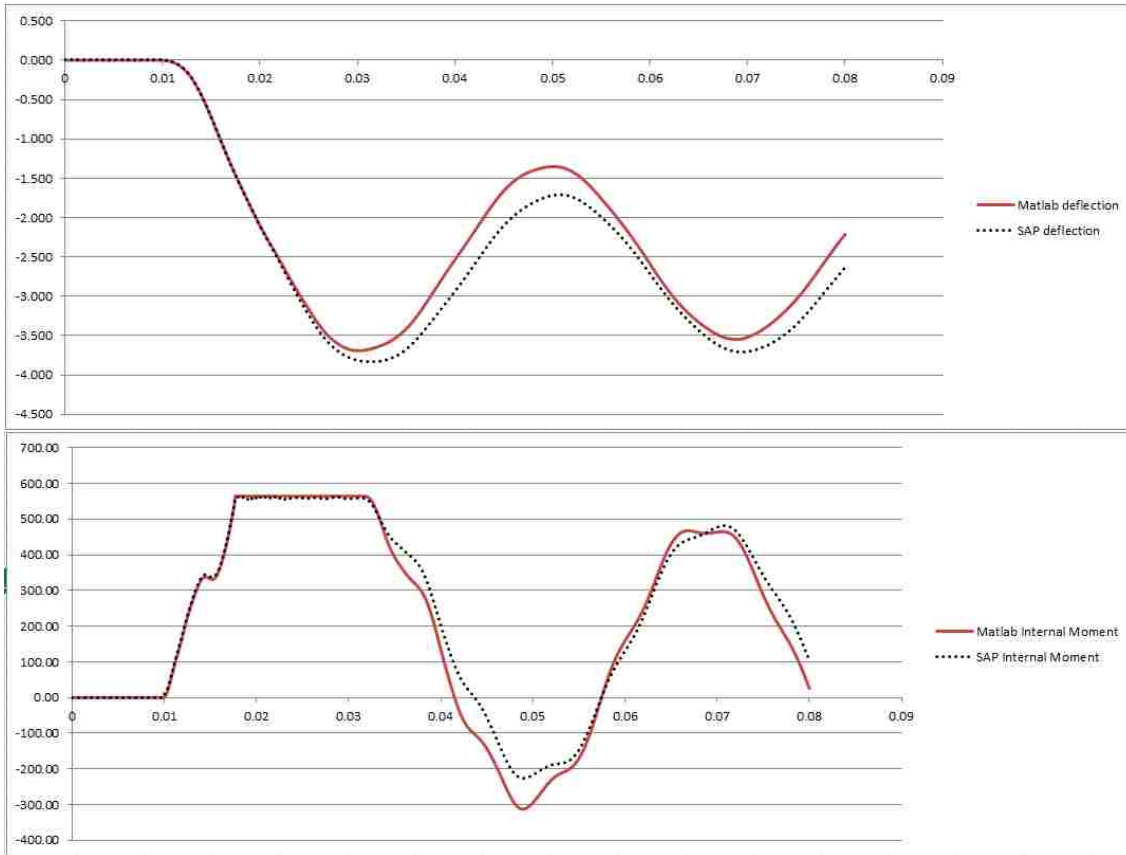
Cantilever Point Load (6 k in 17.8 ms) 20% Axial Capacity Applied



Cantilever Uniform Load (0.06 k/in in 17.8 ms) 40% Axial Capacity Applied

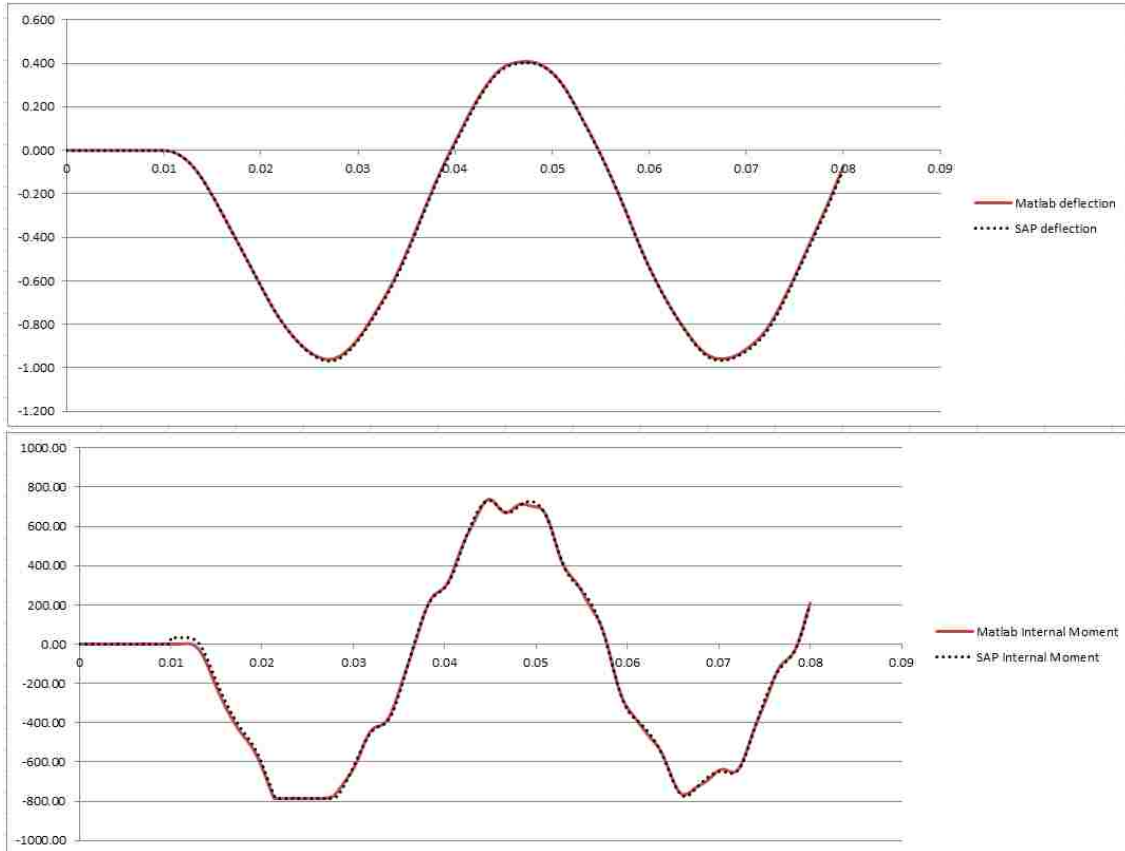


Cantilever Uniform Load (0.125 k/in in 17.8 ms) 20% Axial Capacity Applied



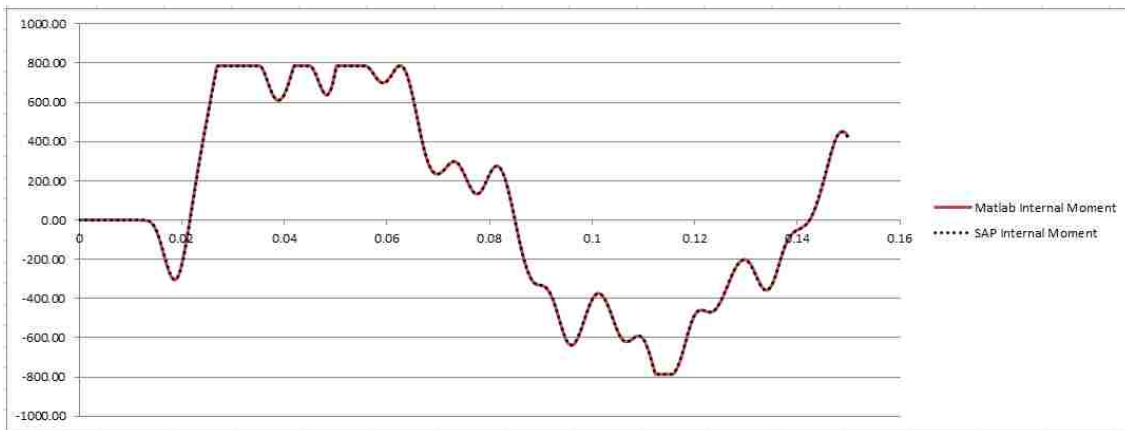
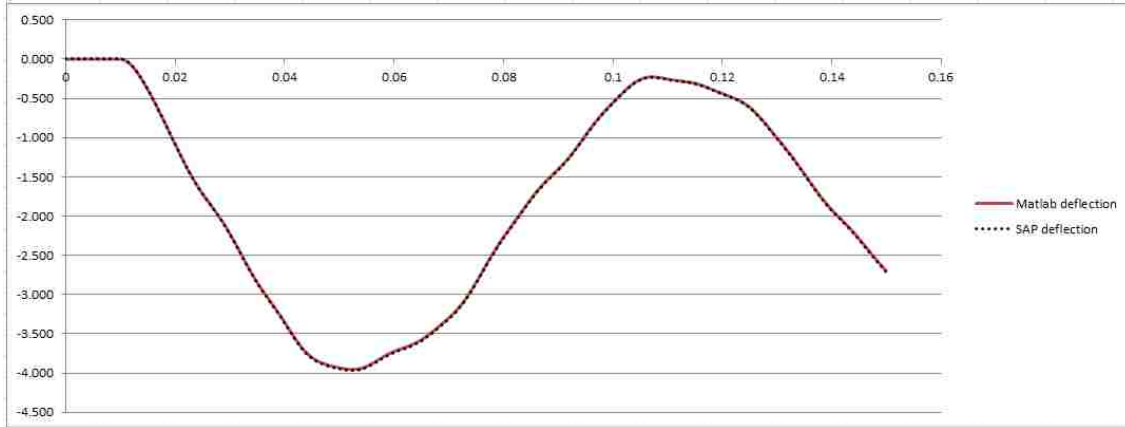
Pinned-Pinned Uniform Load (0.45 k/in in 17.8 ms) with Added Mass 10X Self Weight,

$\alpha=0$ $\beta=0.0000002$ (damping)



Cantilever Point Load (20 k in 17.8 ms) with Added Mass 10X Self Weight

$\alpha=0$ $\beta=0.0000002$ (damping)

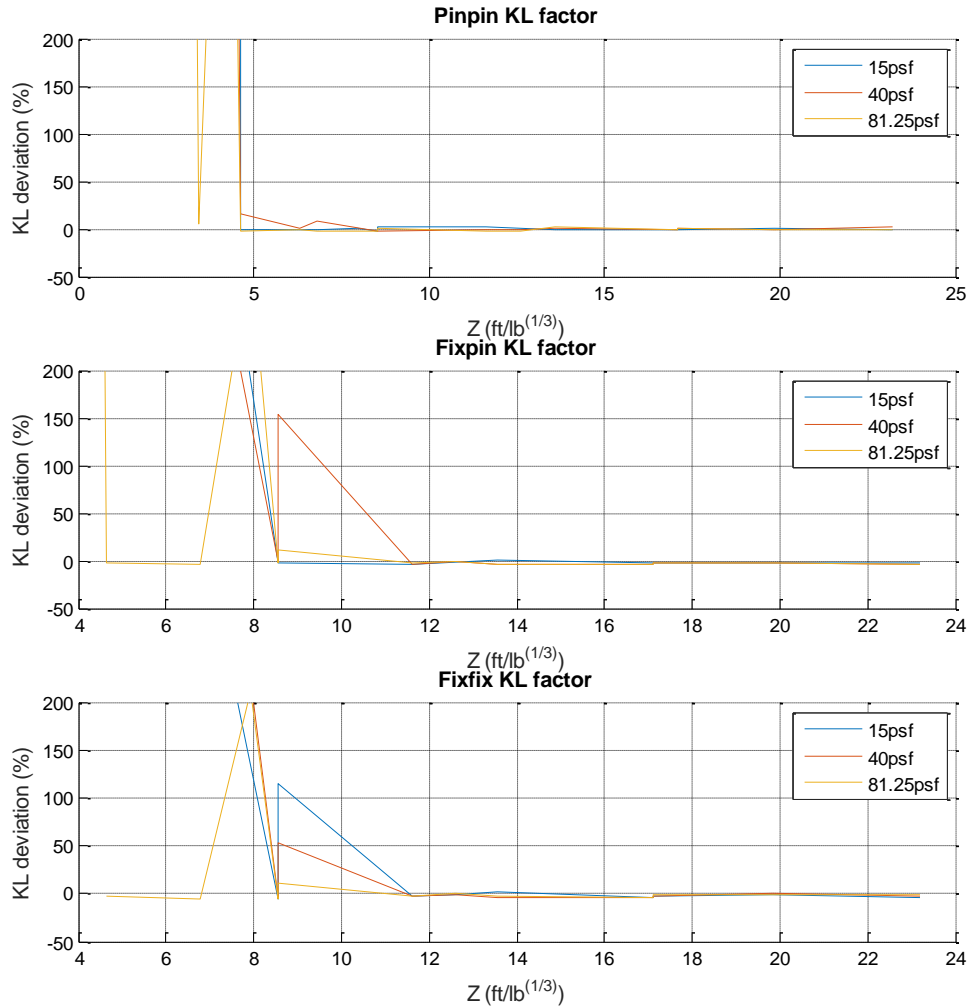


Appendix E: First Data Set Plots

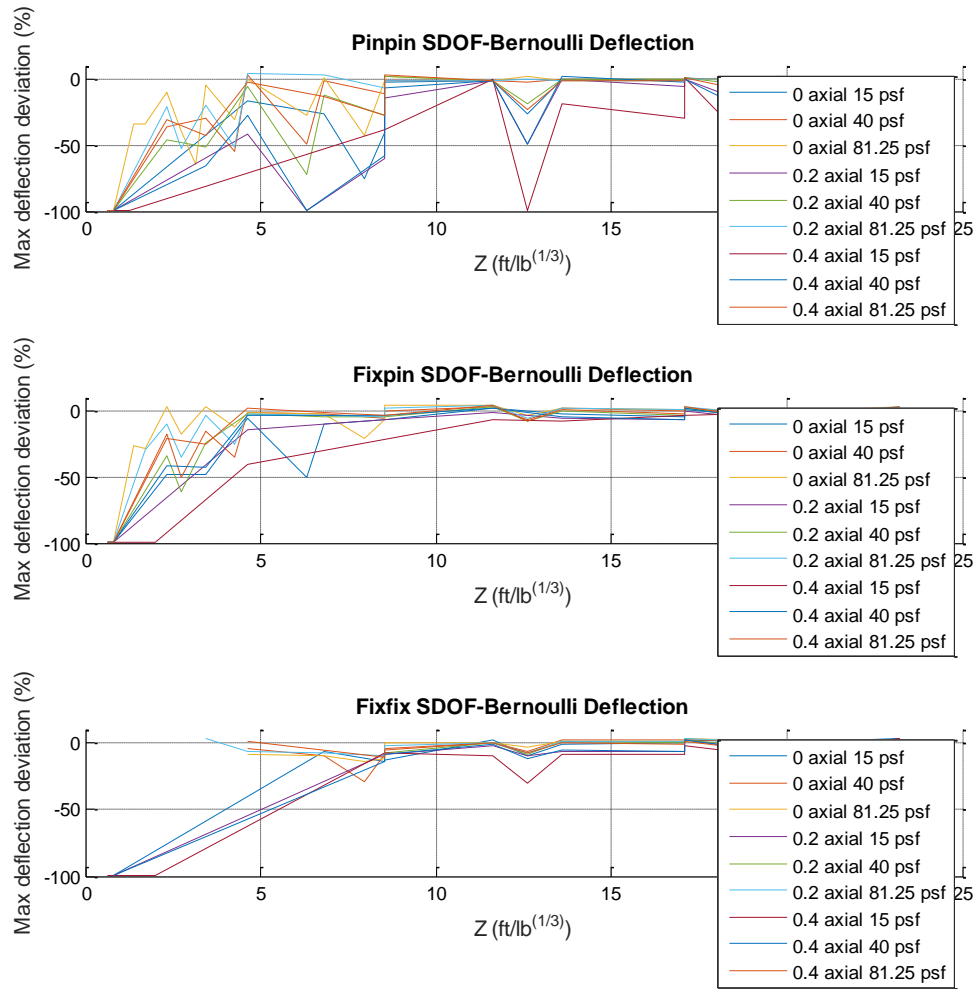
Various Sections With 2% Damping in First and Second Modes

W14X48 Section

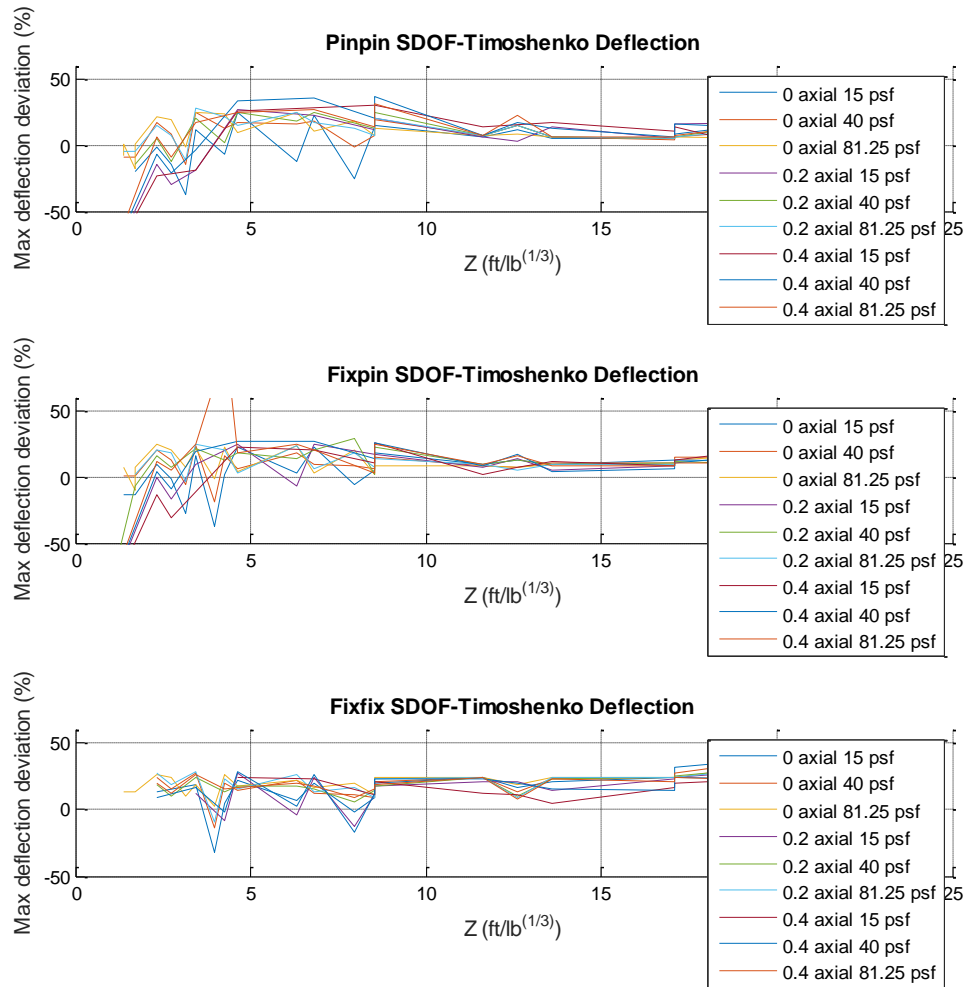
KL factor



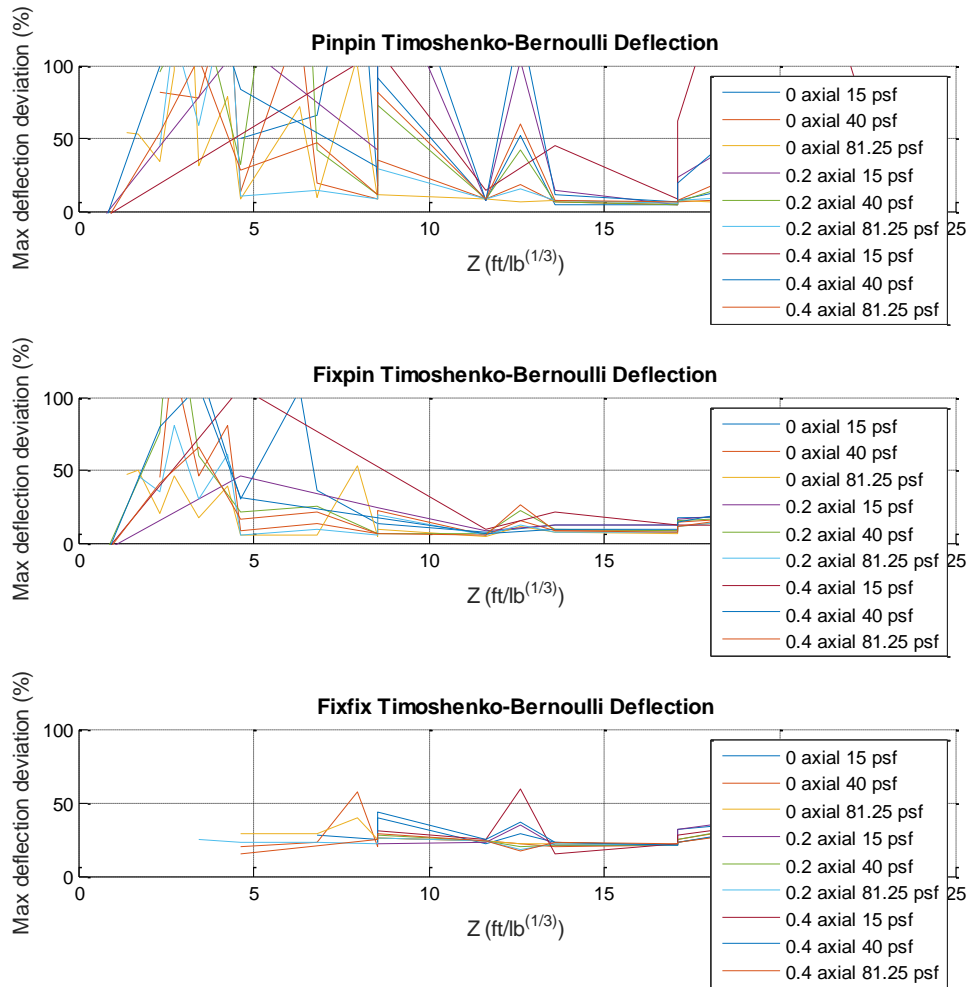
SDOF-Bernoulli



SDOF Timoshenko

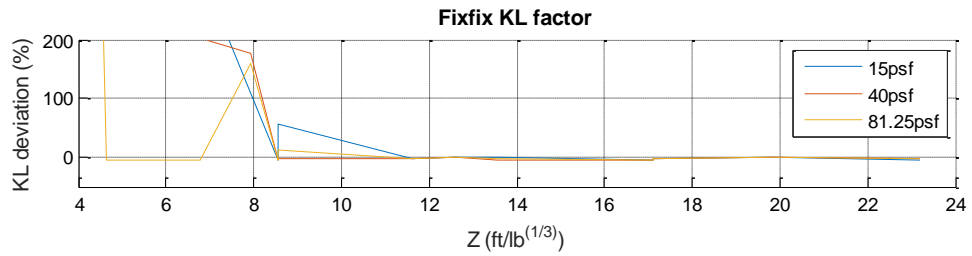
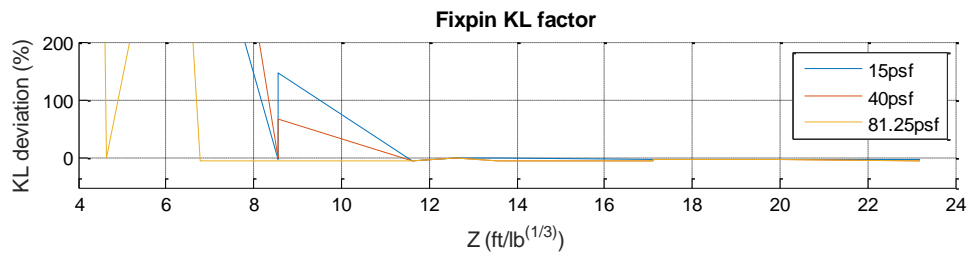
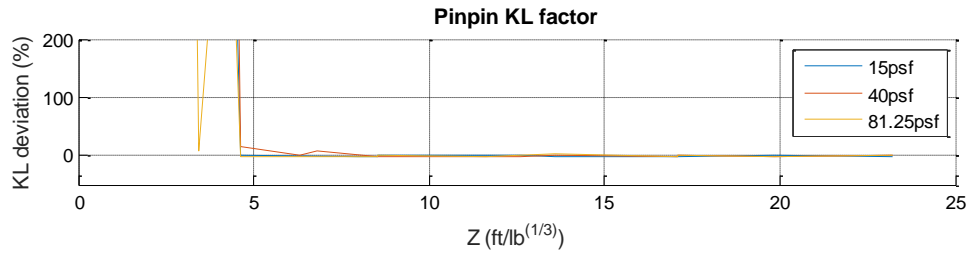


Bernoulli Timoshenko

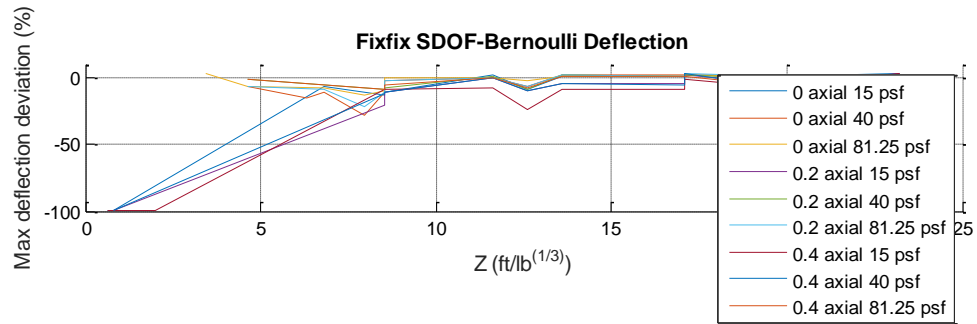
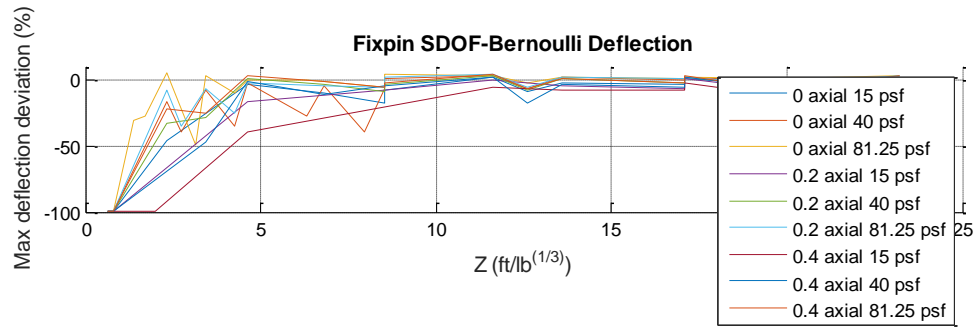
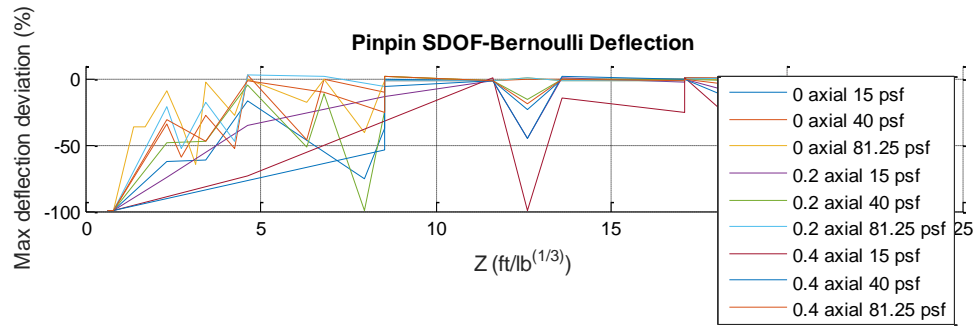


W14X53:

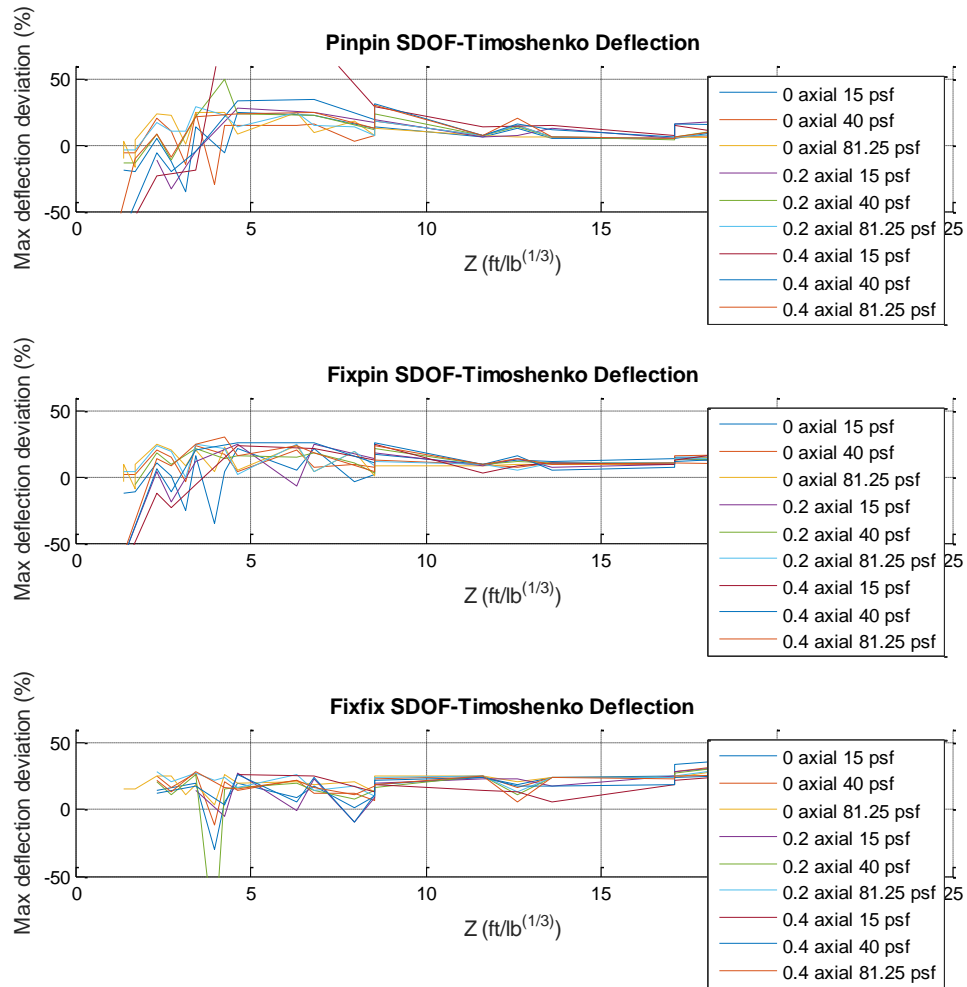
KL factor



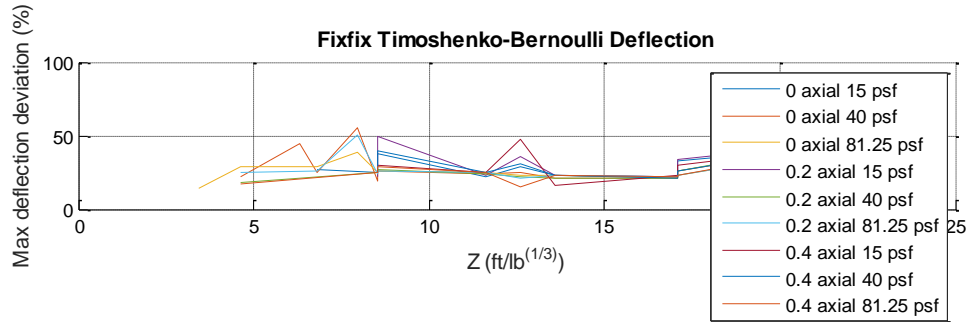
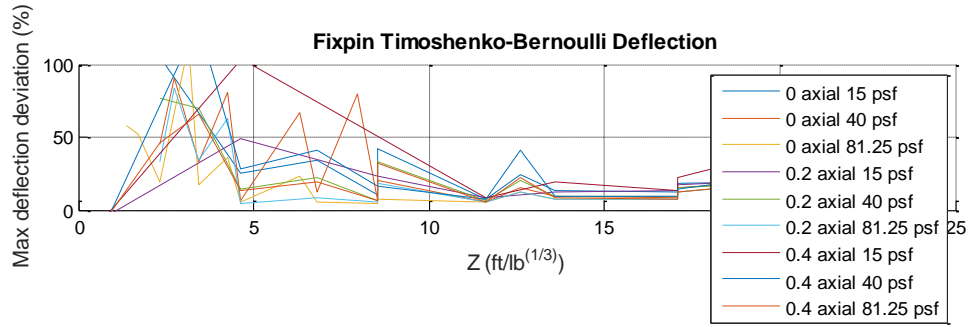
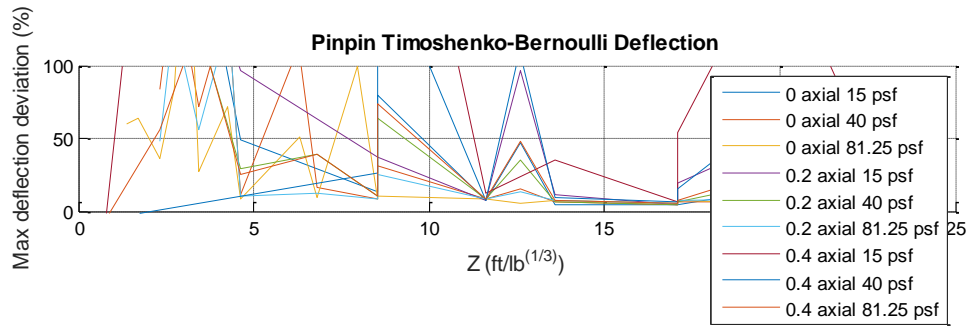
SDOF Bernoulli



SDOF Timoshenko

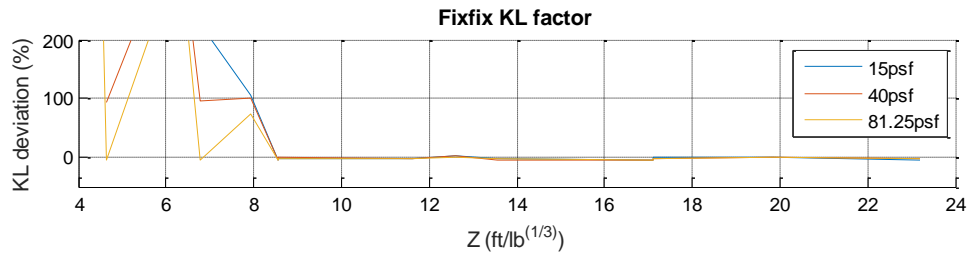
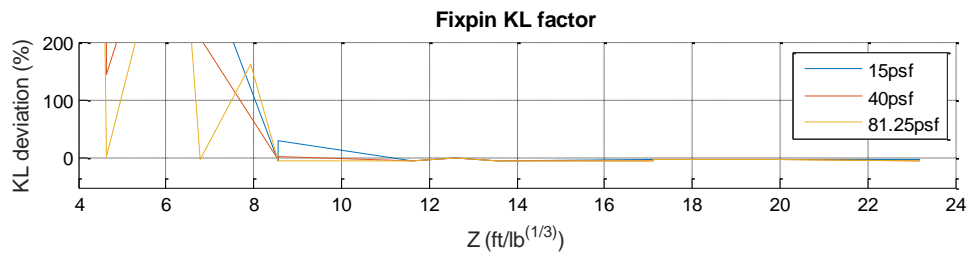
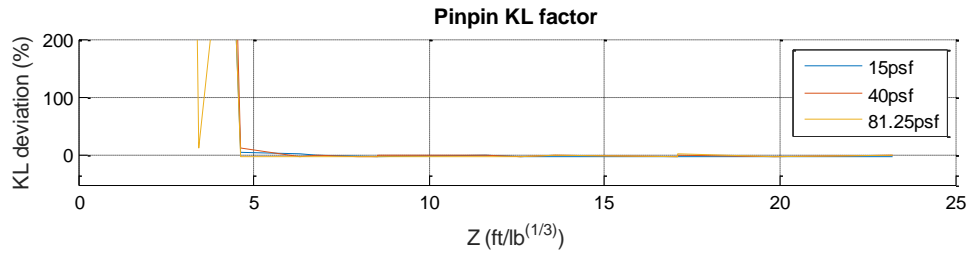


Bernoulli Timoshenko

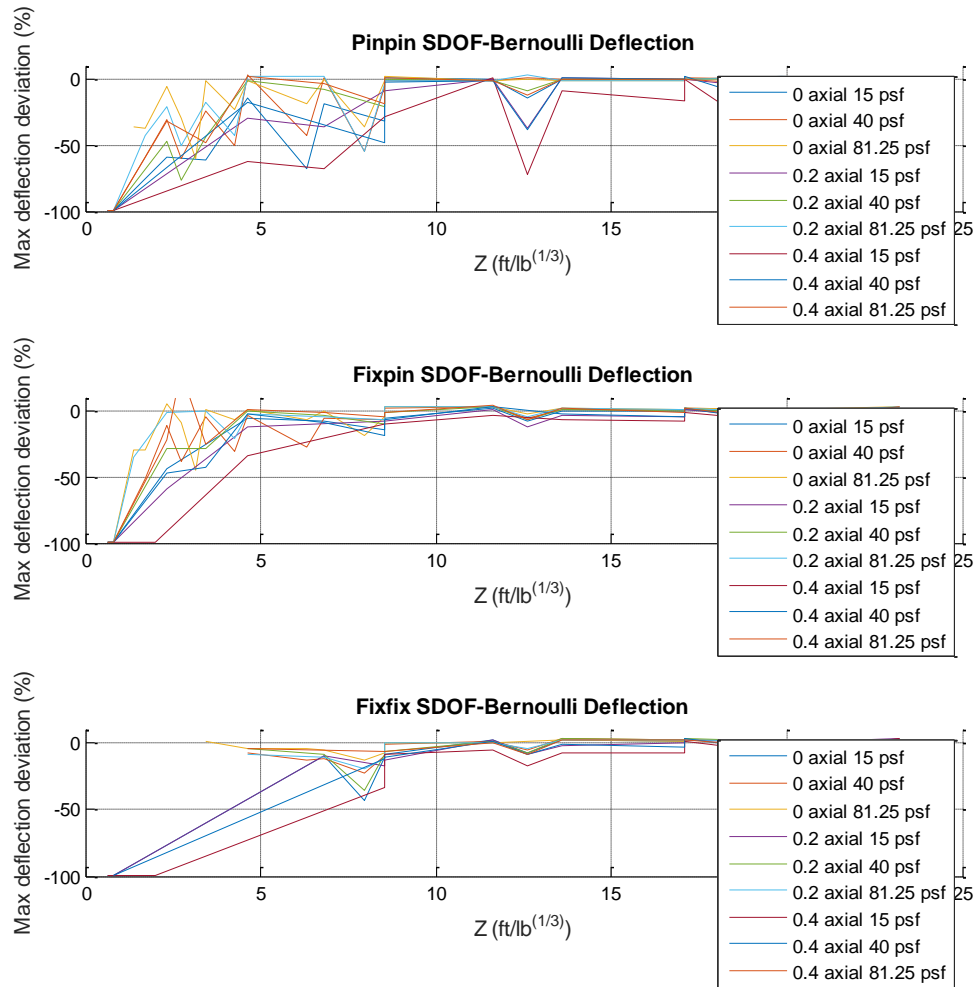


W14X61

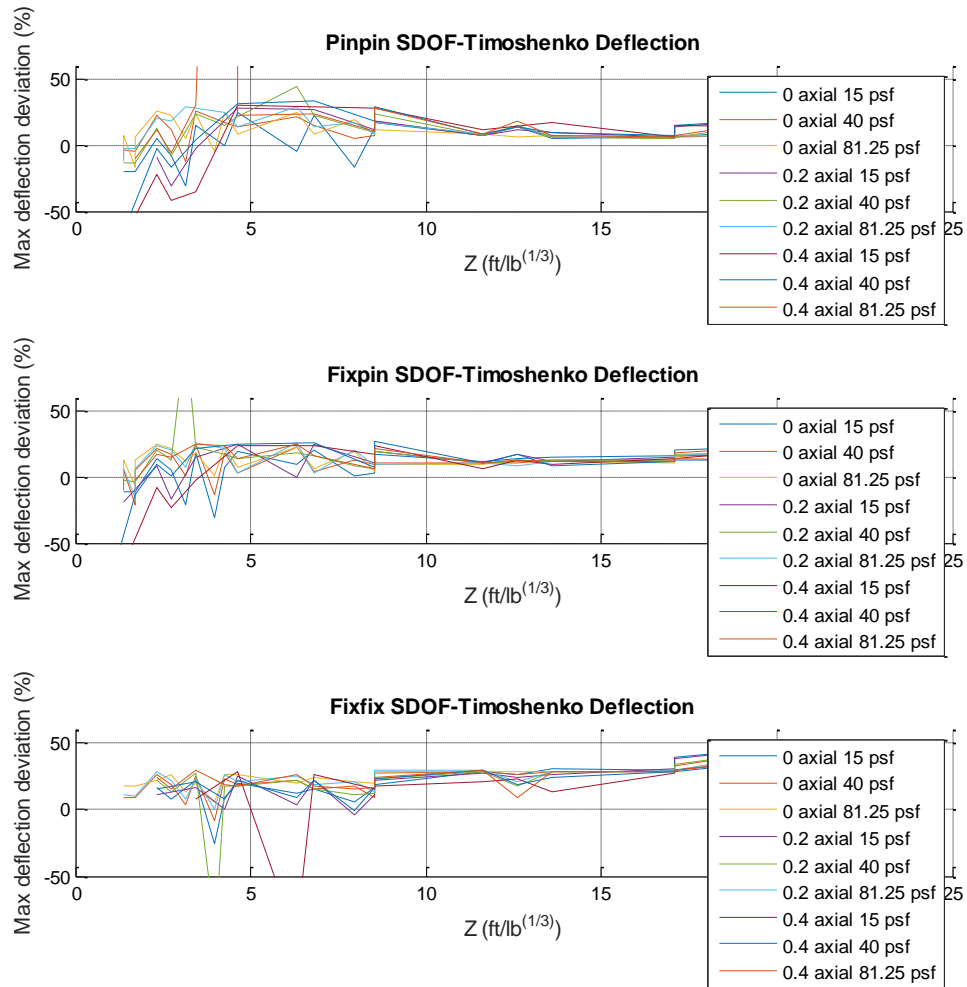
KL Factor



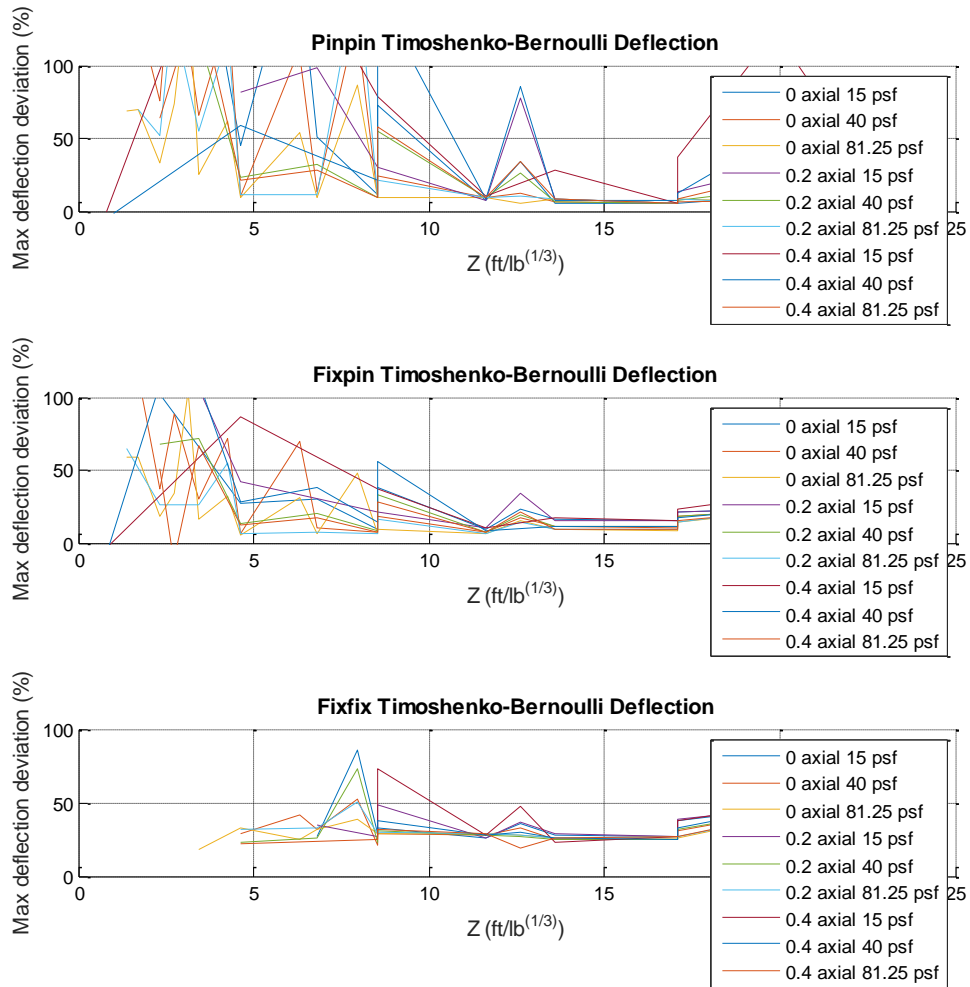
SDOF Bernoulli



SDOF Timoshenko

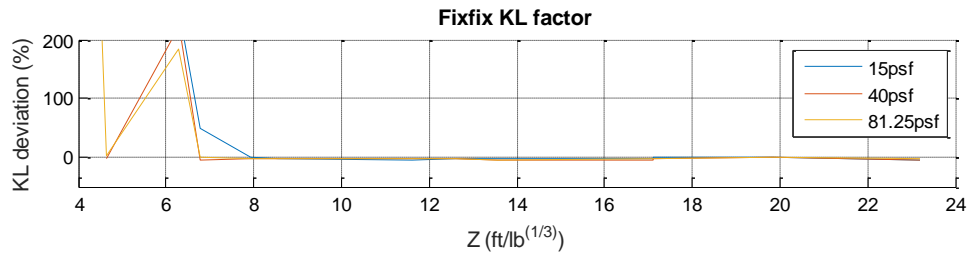
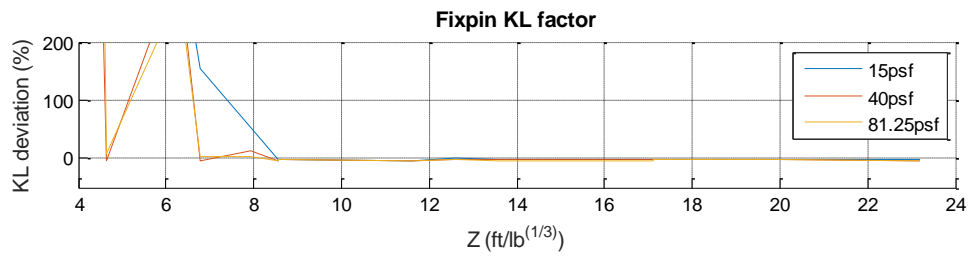
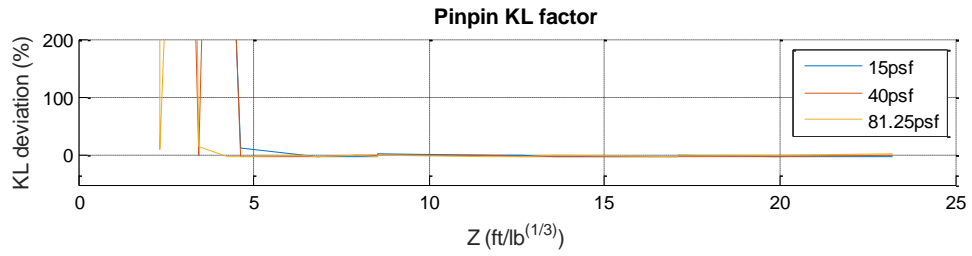


Bernoulli Timoshenko

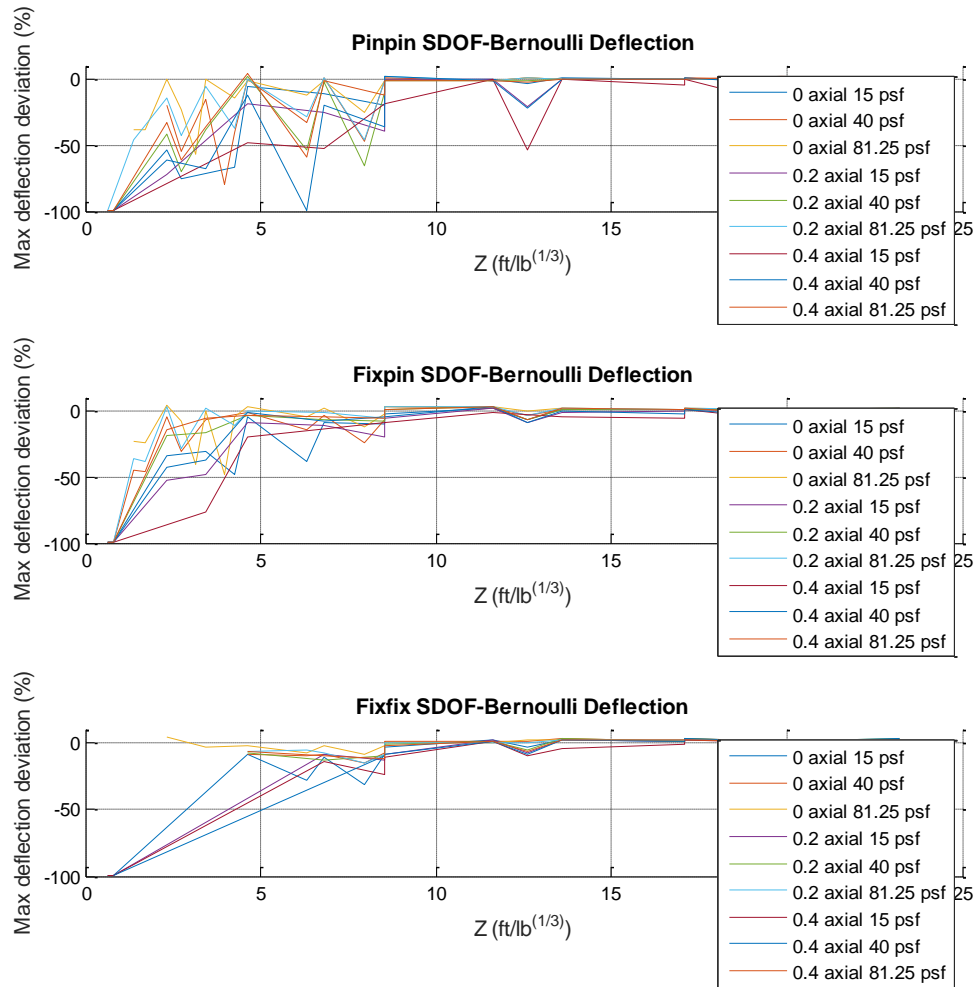


W14X82

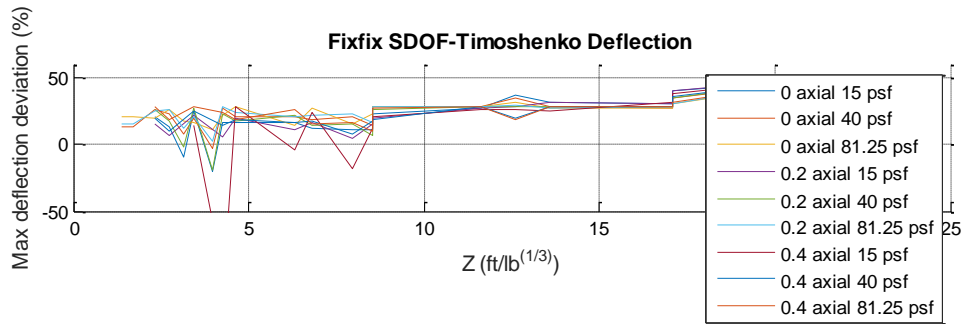
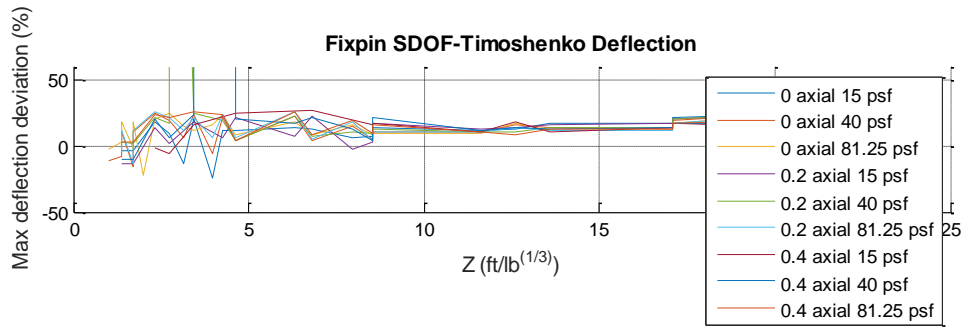
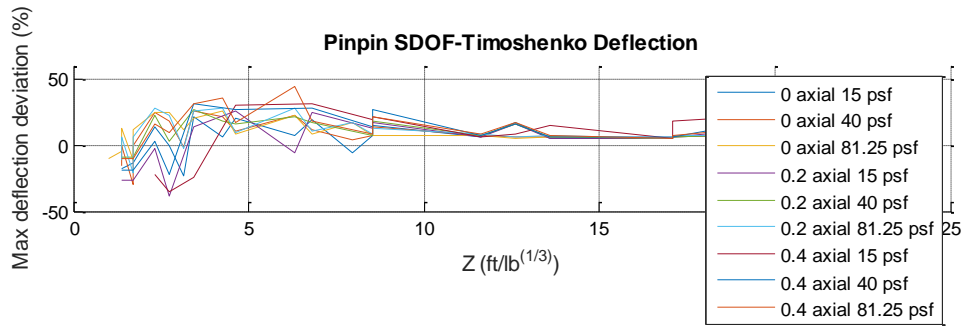
KL Factor



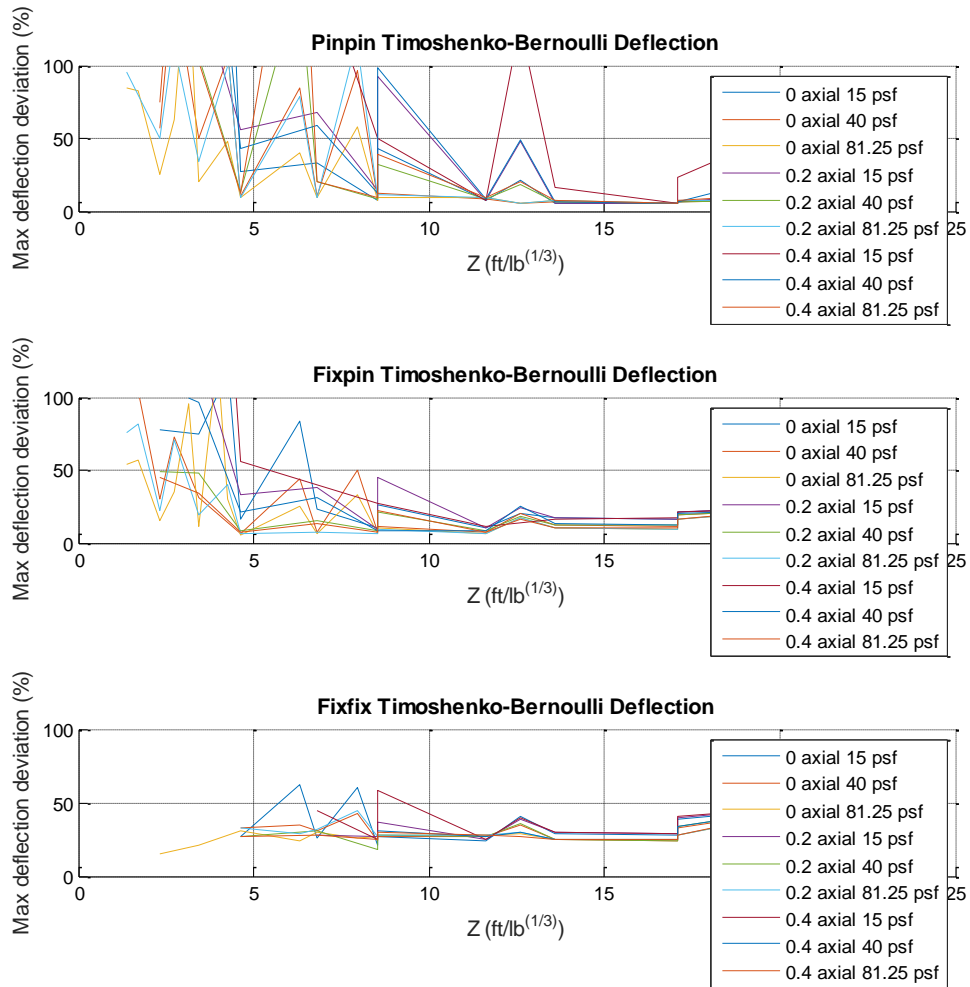
SDOF Bernoulli



SDOF Timoshenko

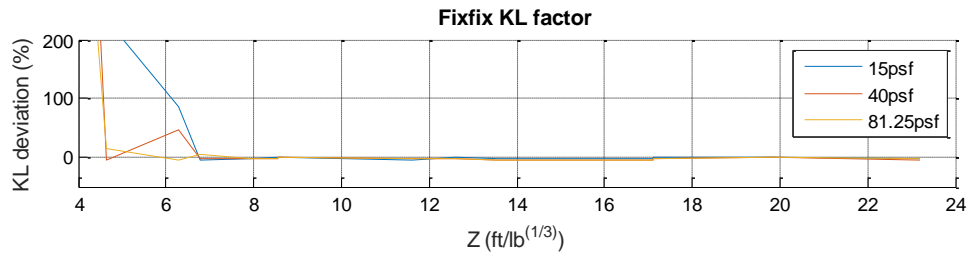
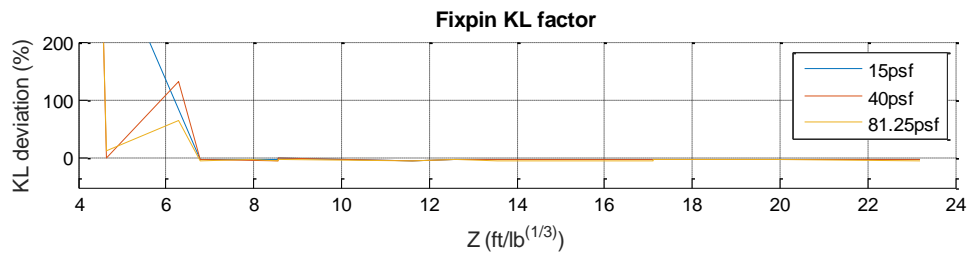
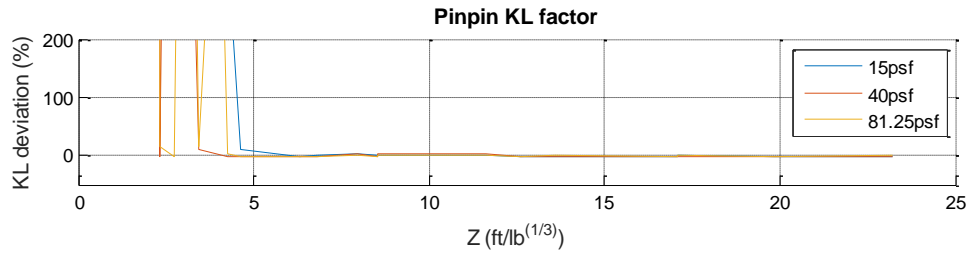


Bernoulli Timoshenko

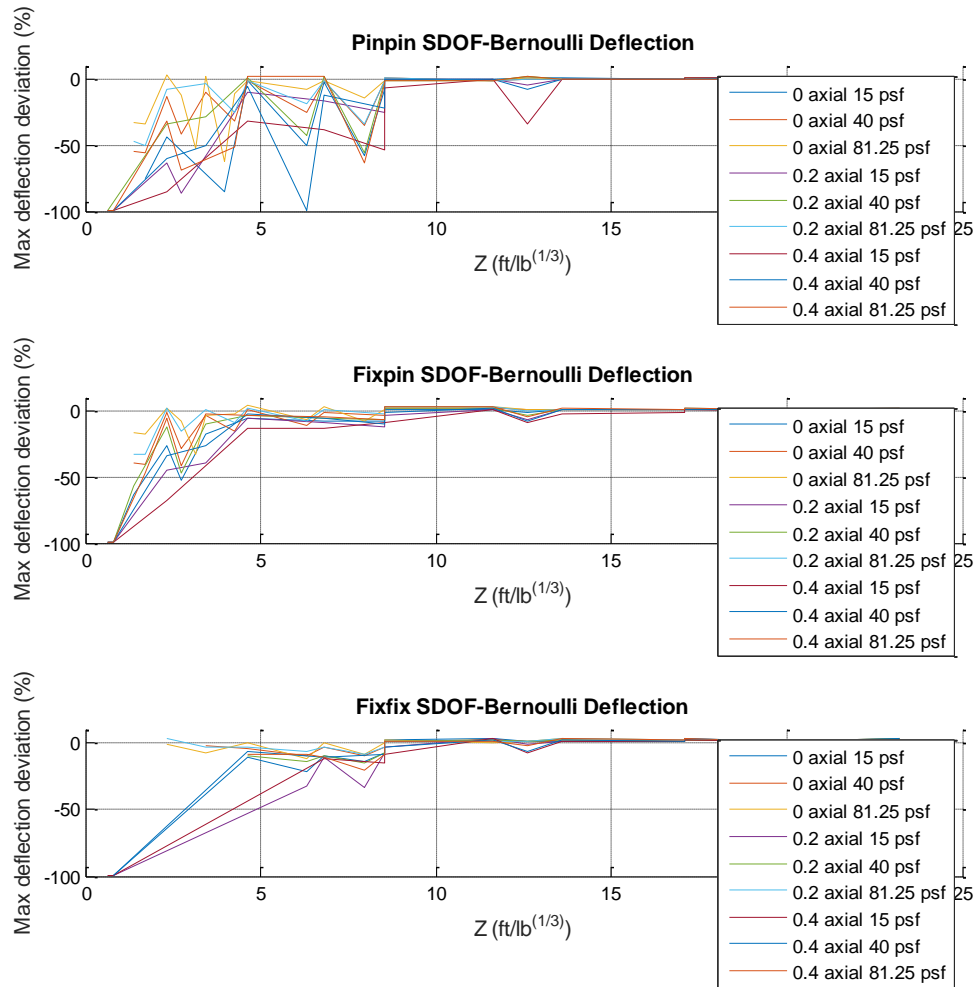


W14X109

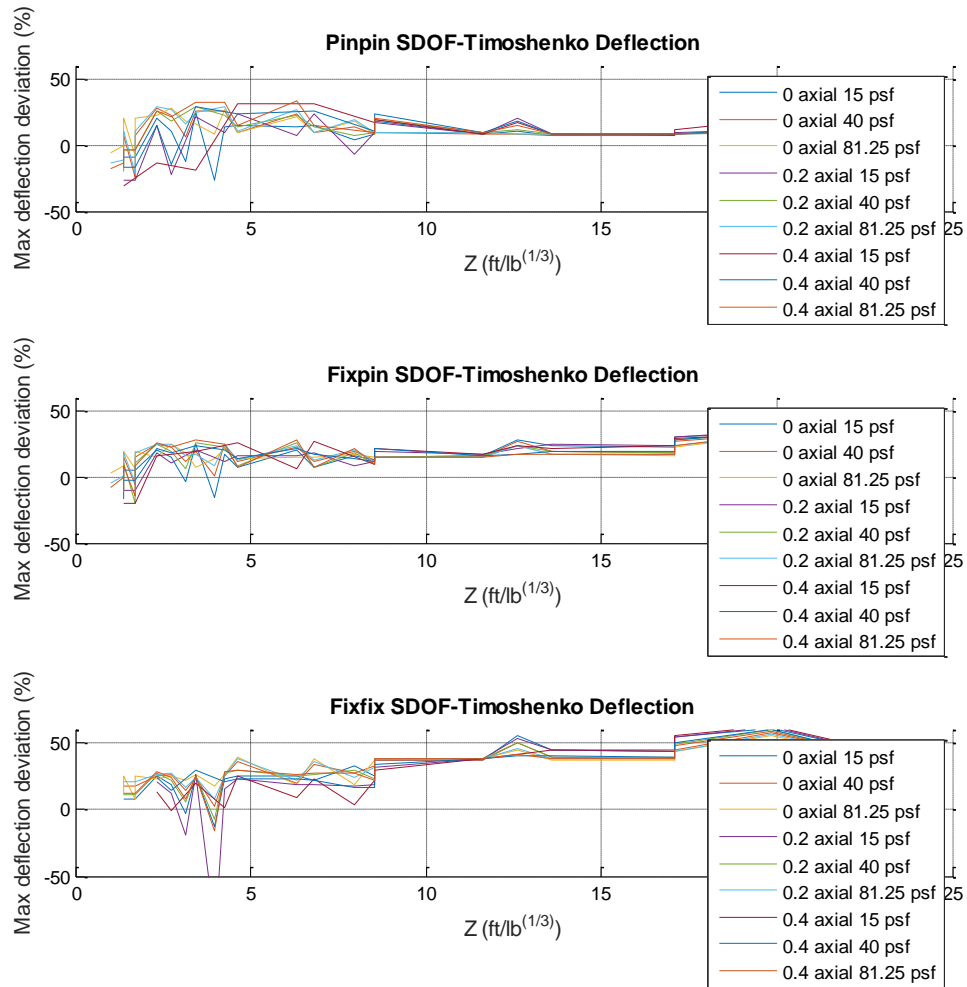
KL Factor



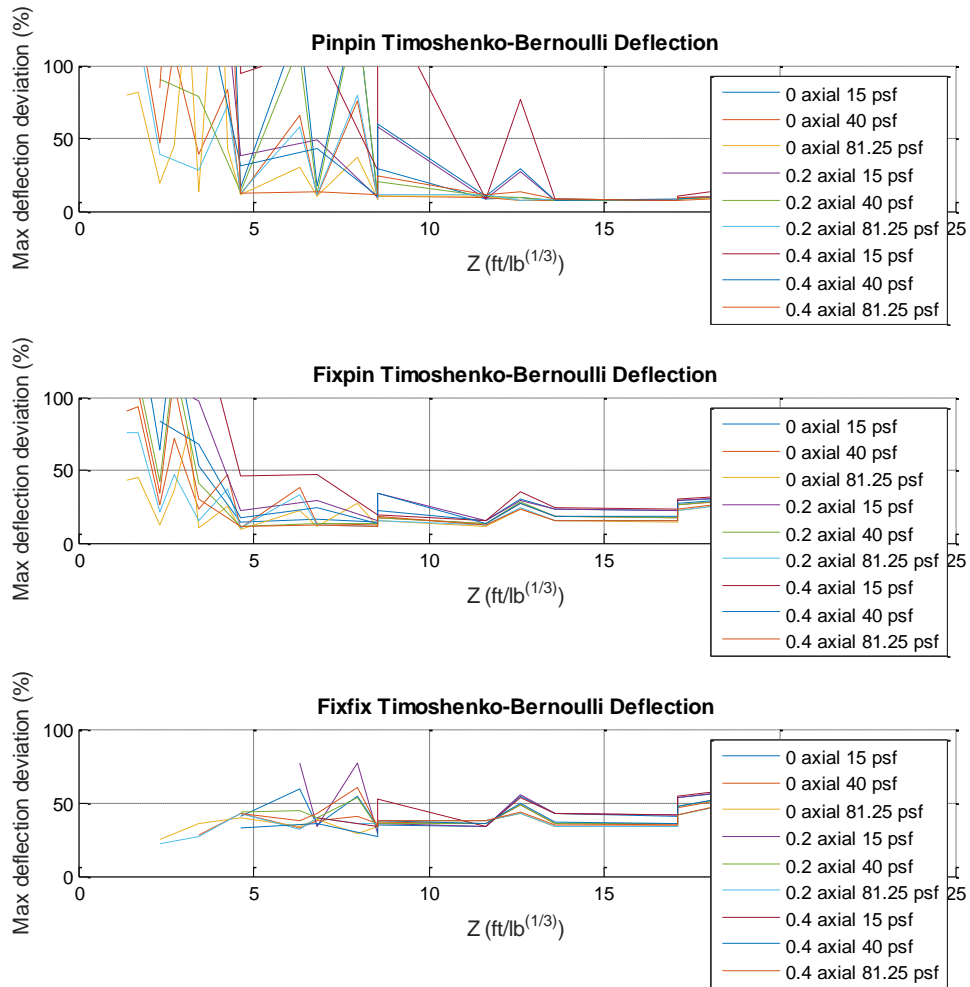
SDOF Bernoulli



SDOF Timoshenko

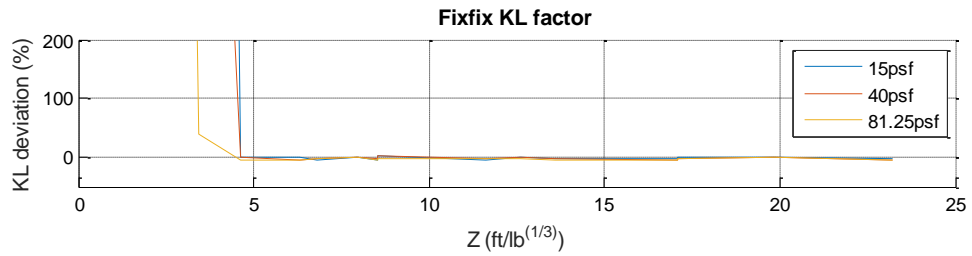
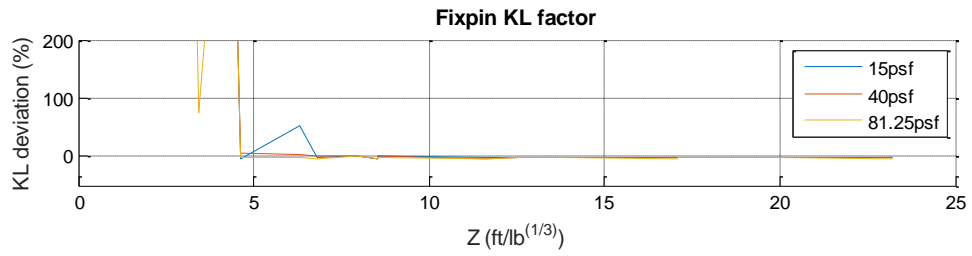
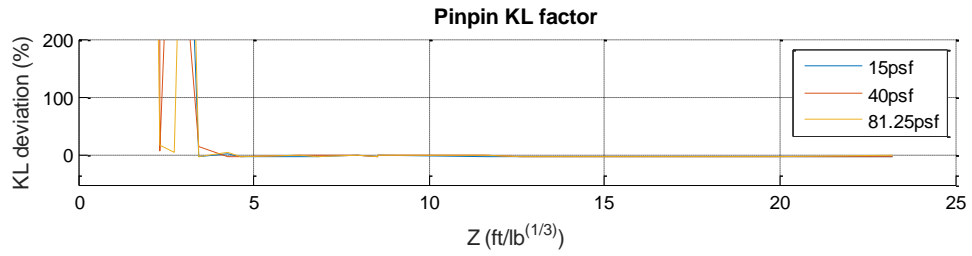


Bernoulli Timoshenko

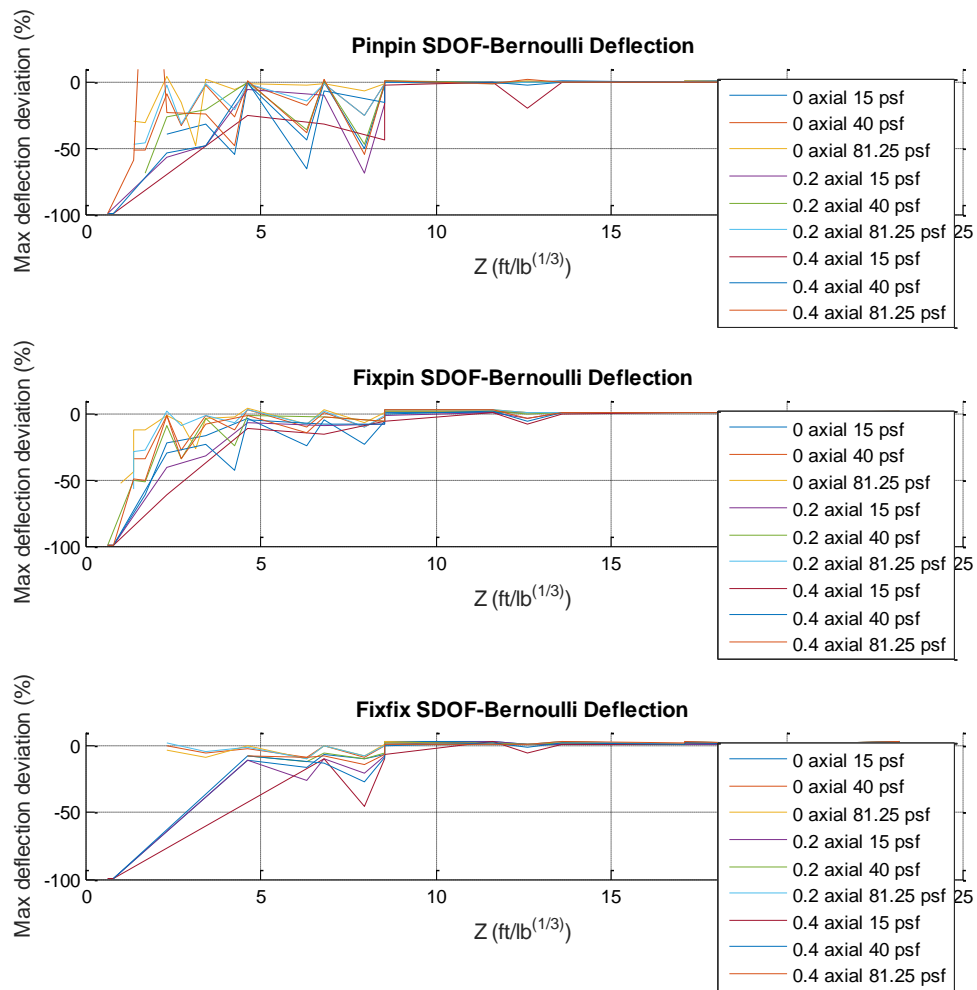


W14X132

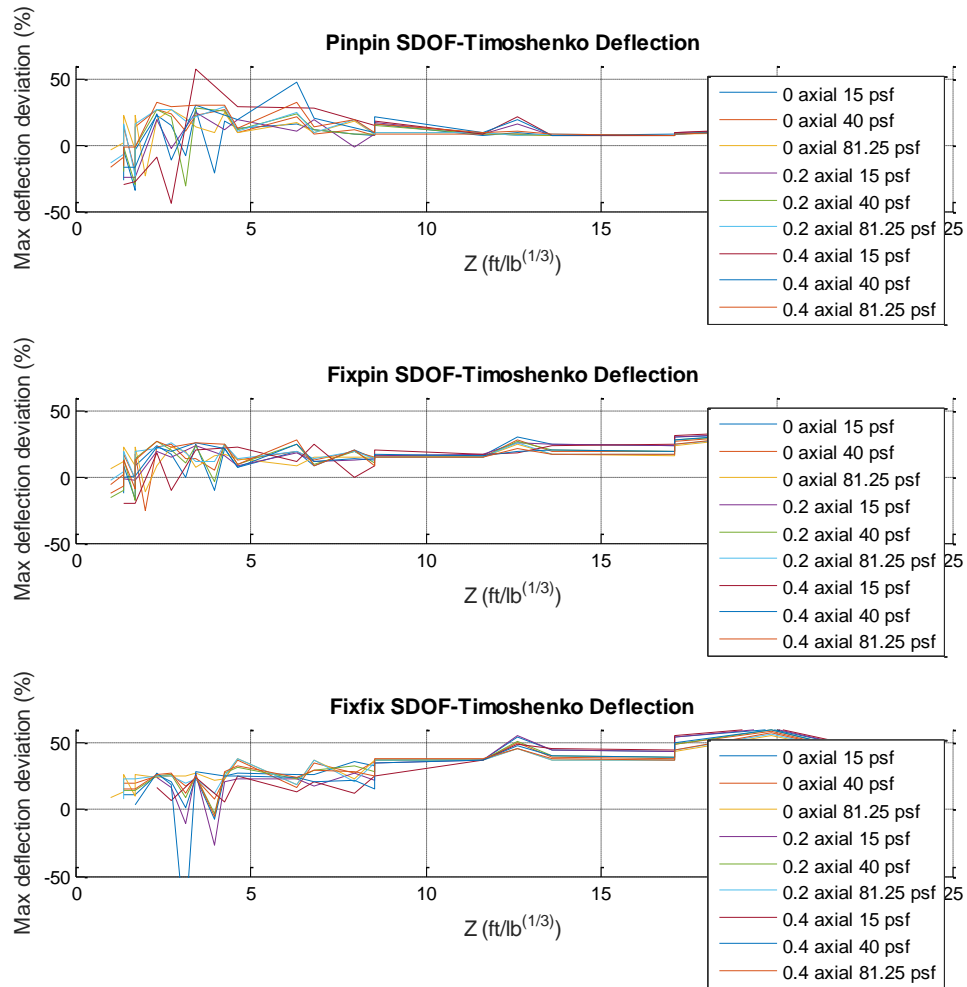
KL Factor



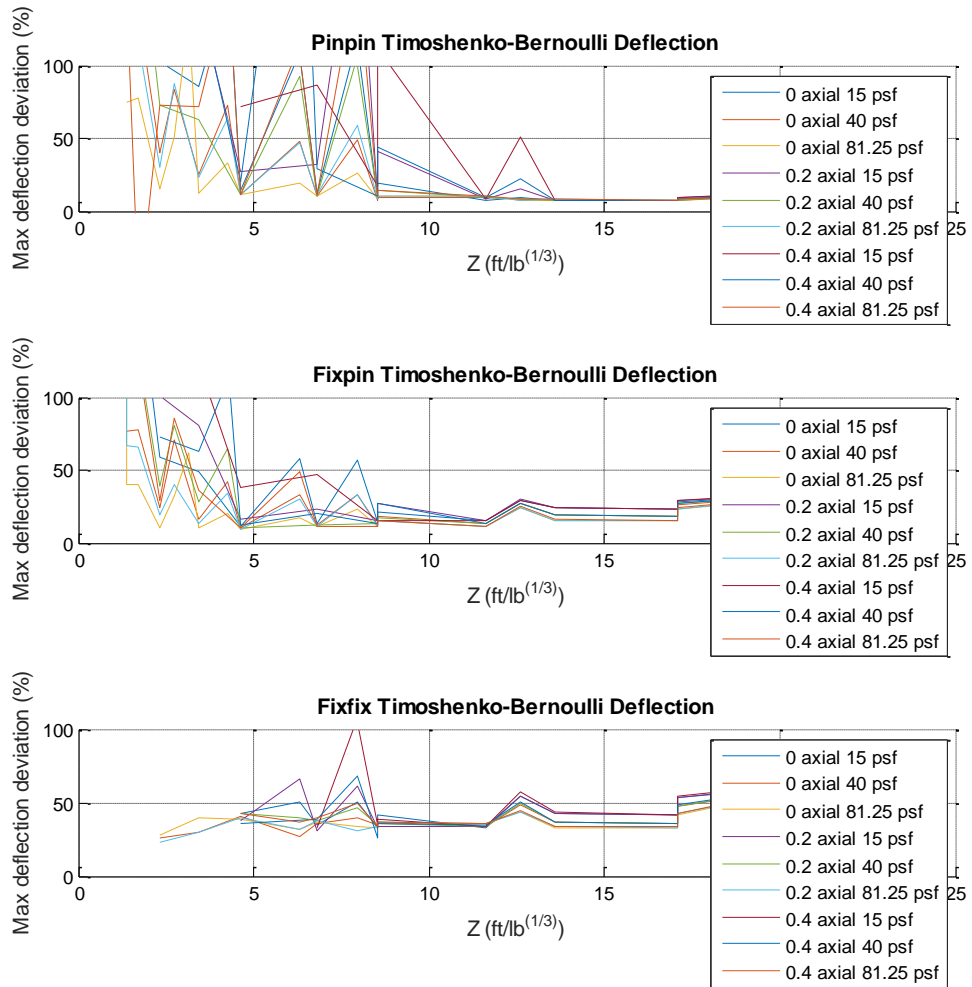
SDOF Bernoulli



SDOF Timoshenko

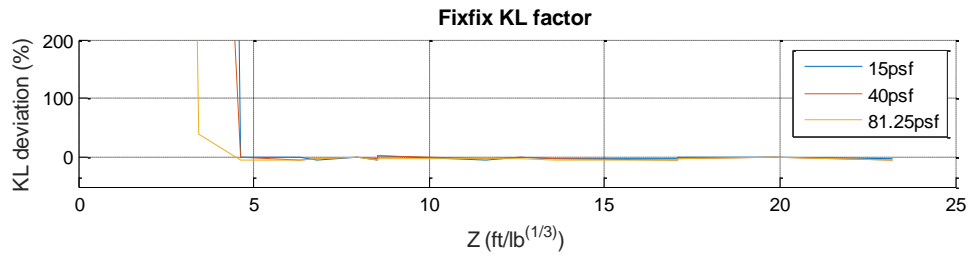
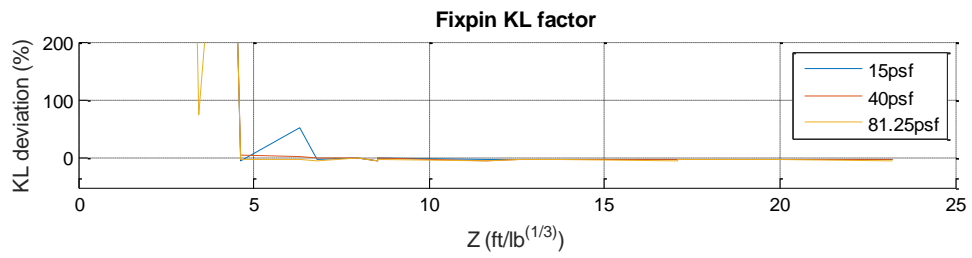
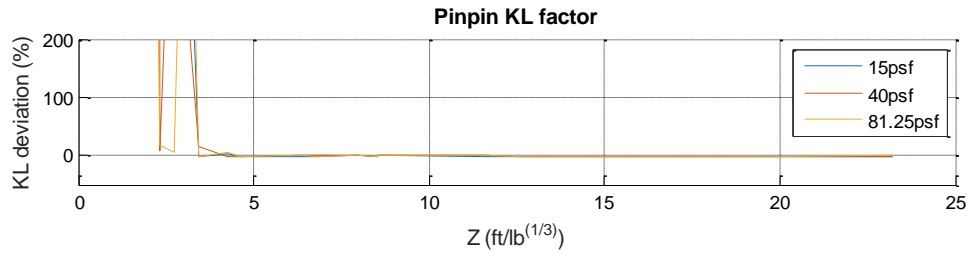


Bernoulli Timoshenko

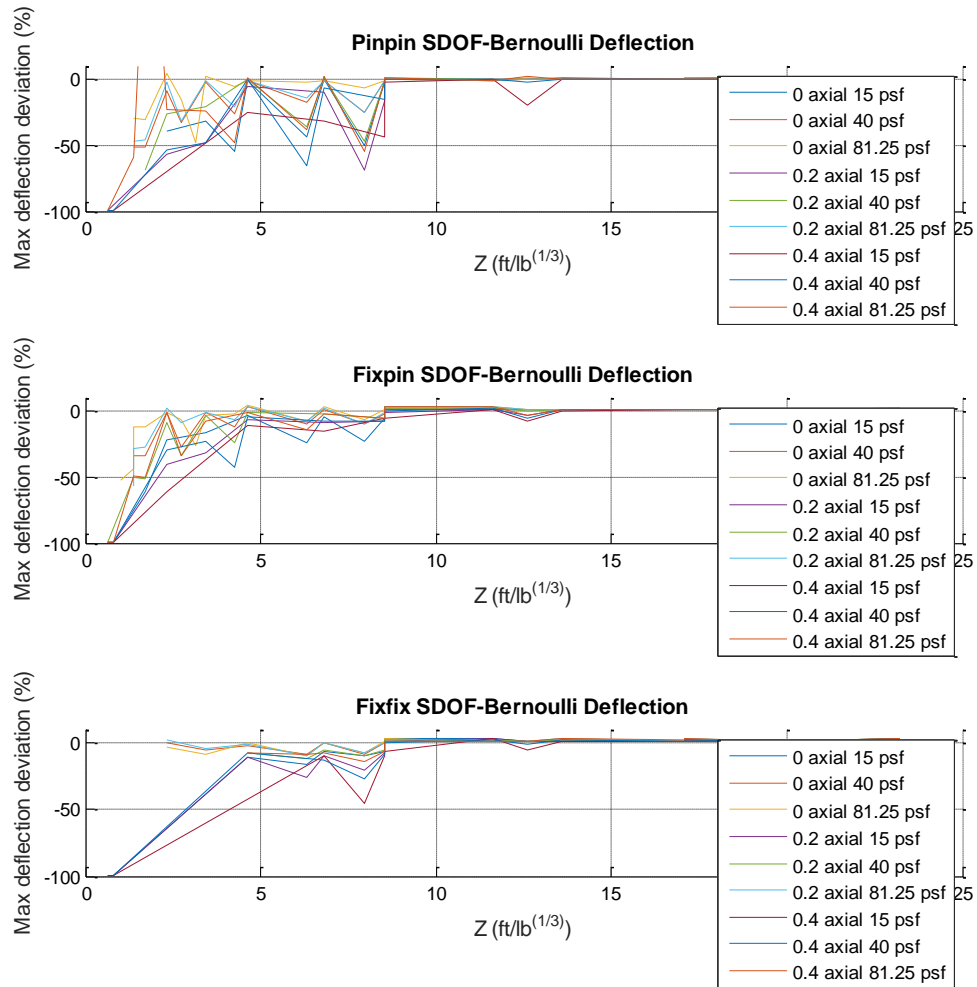


W14X132

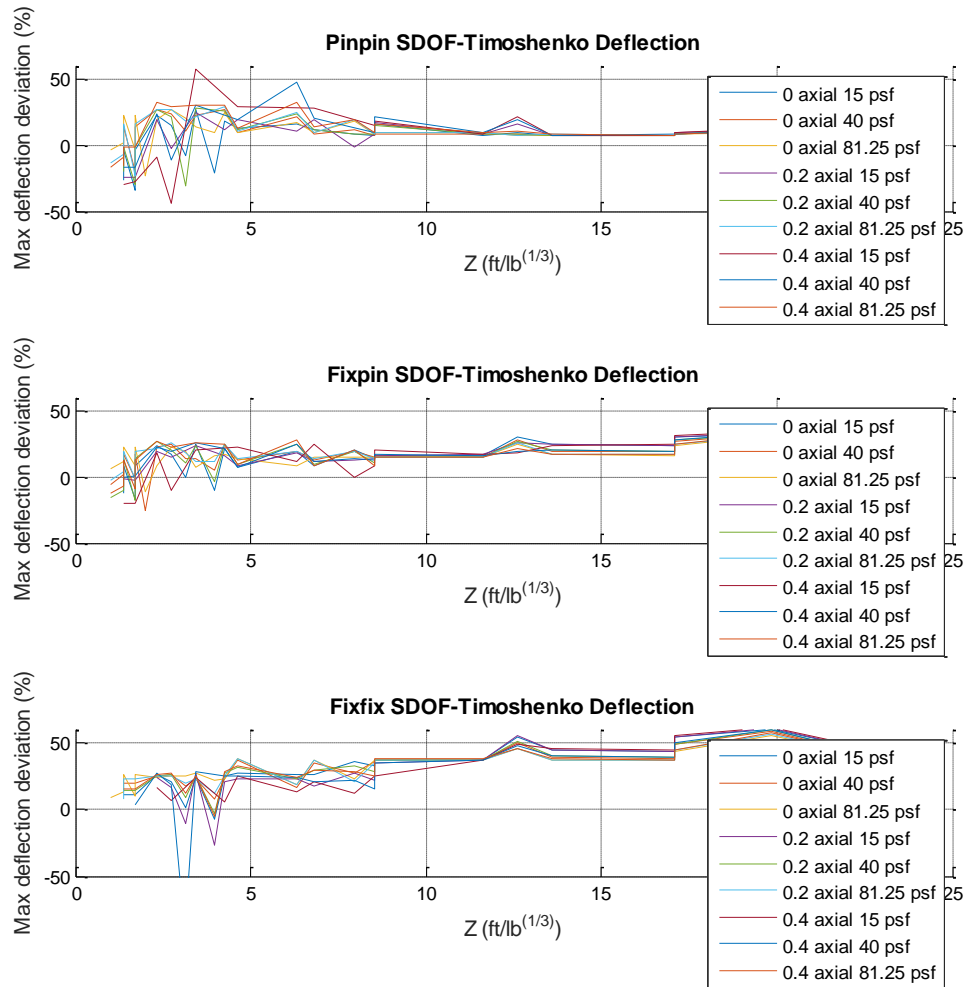
KL Factor



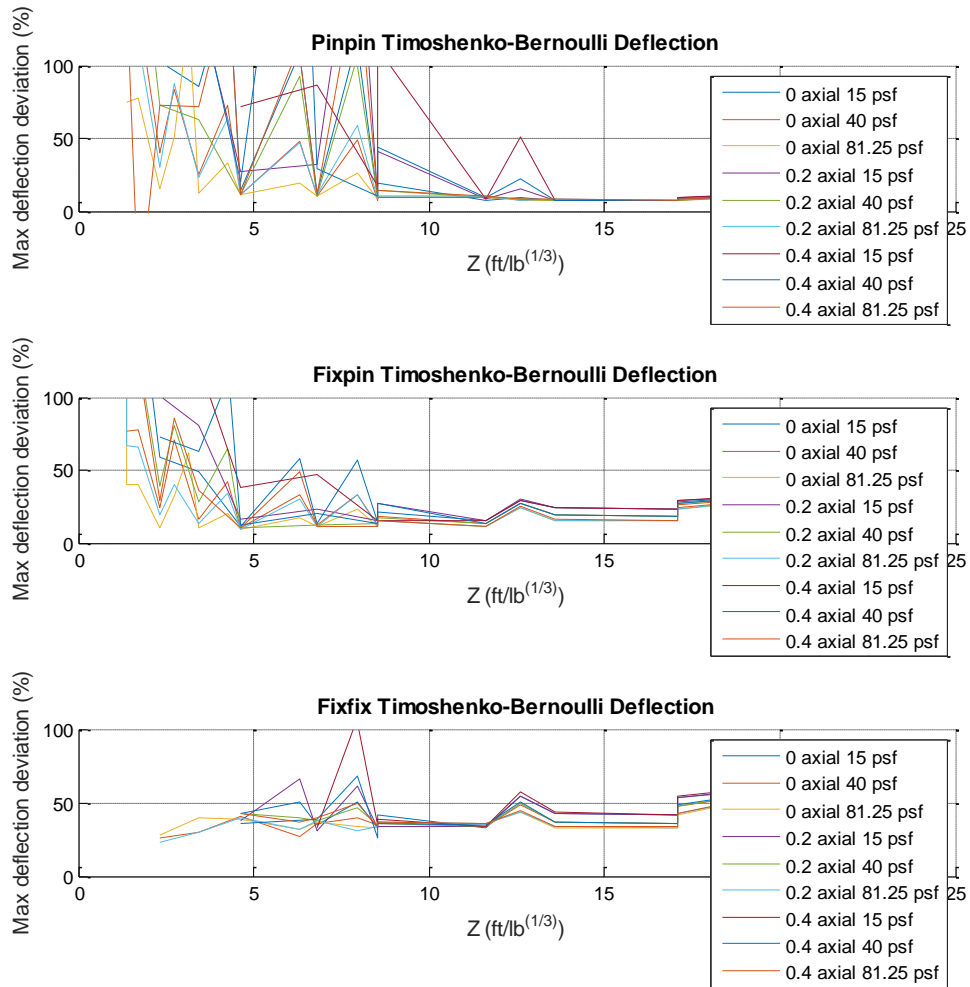
SDOF Bernoulli



SDOF Timoshenko

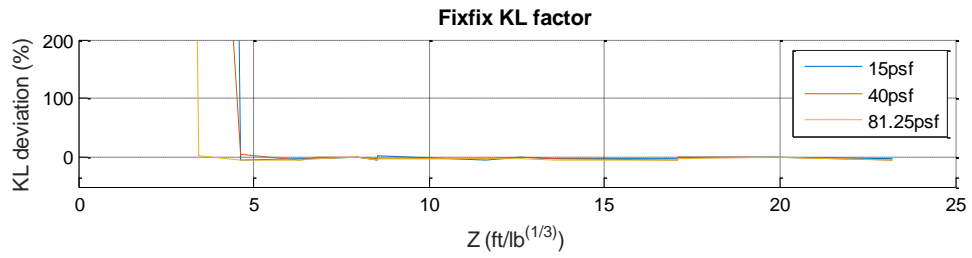
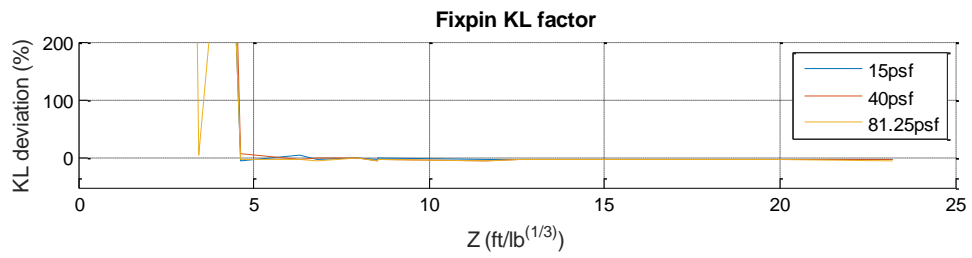
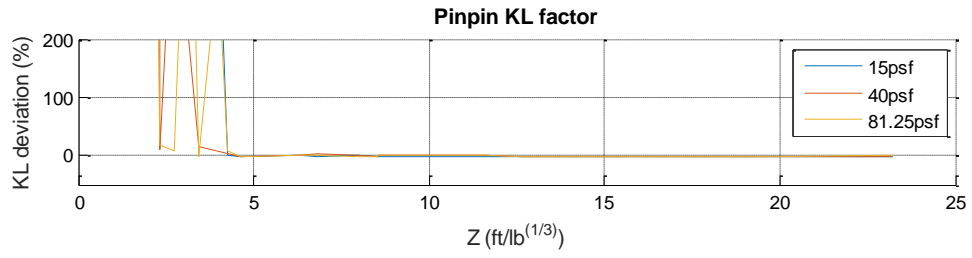


Bernoulli Timoshenko

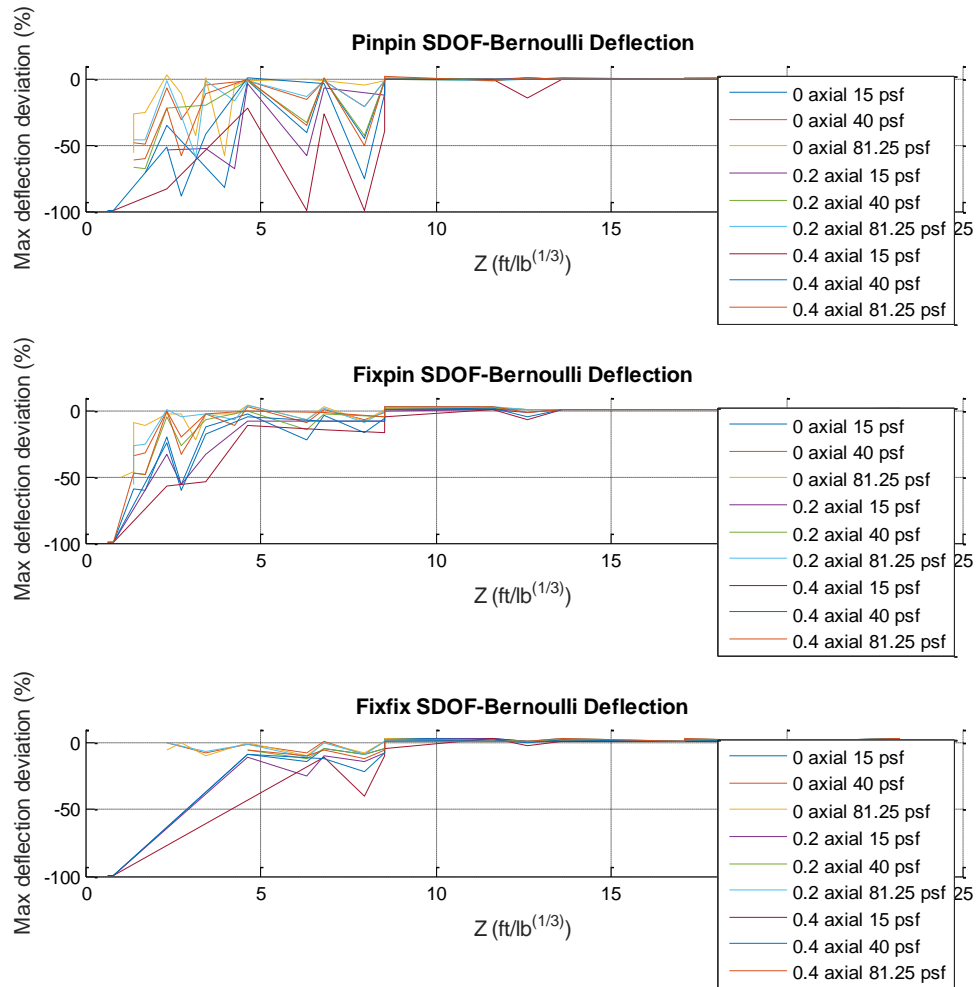


W14X145

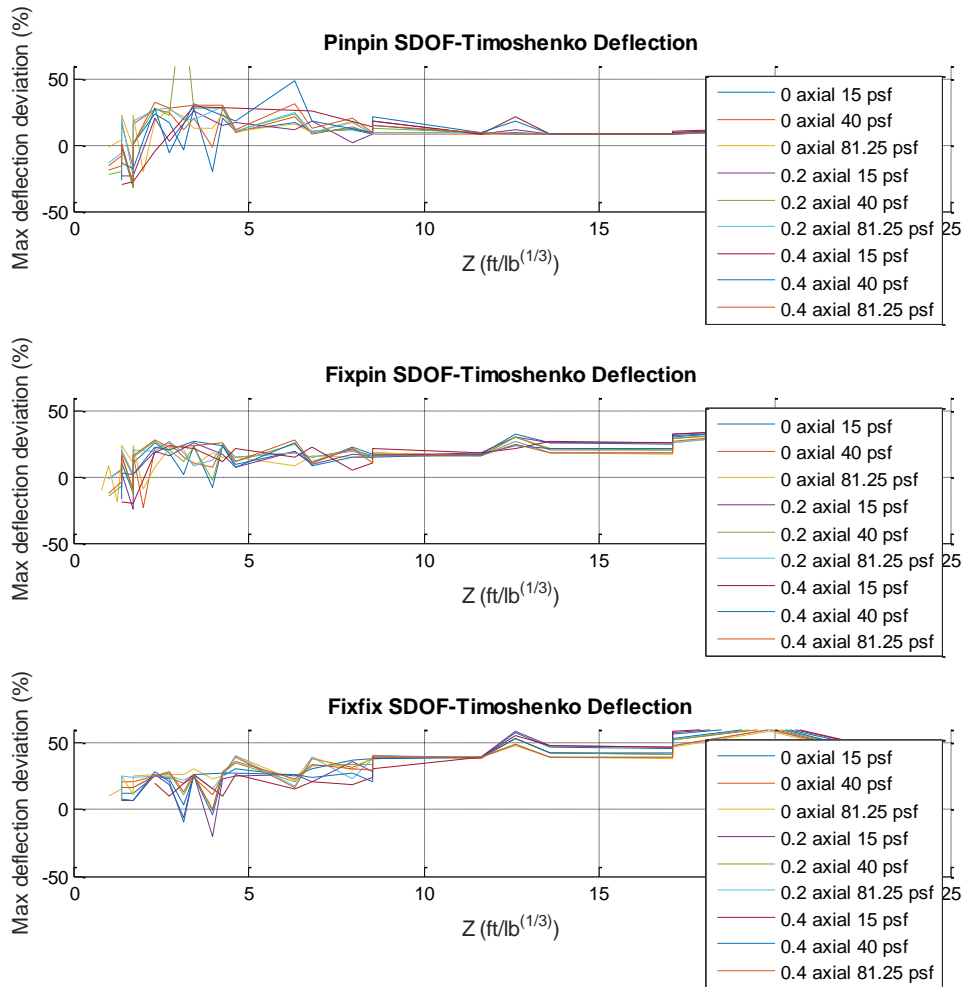
KL Factor



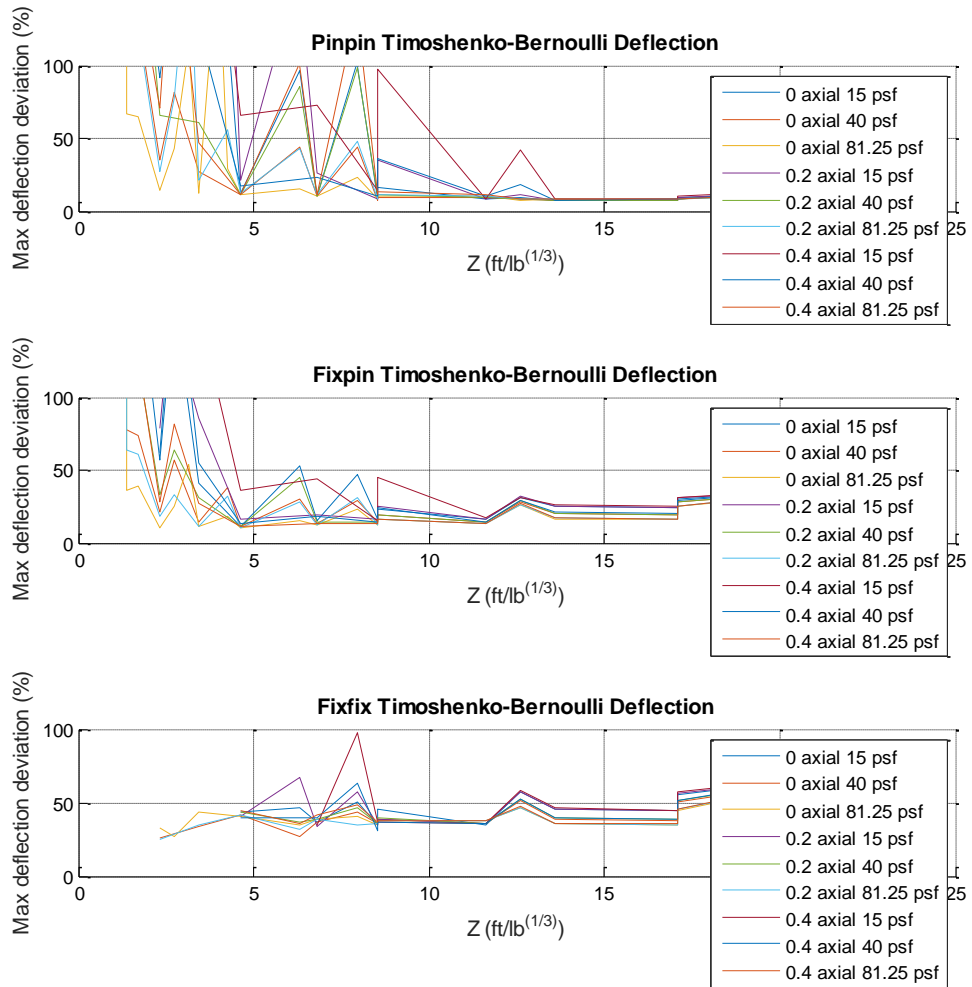
SDOF Bernoulli



SDOF Timoshenko

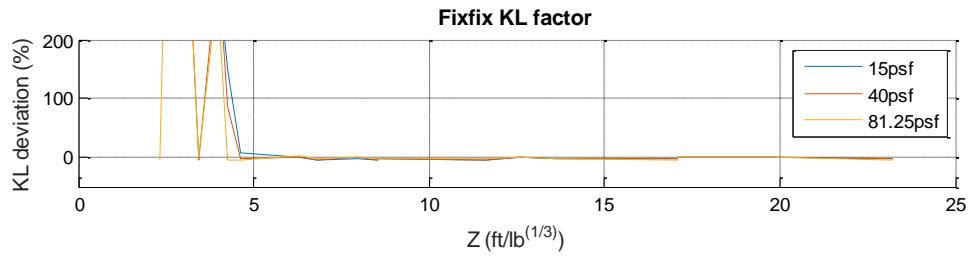
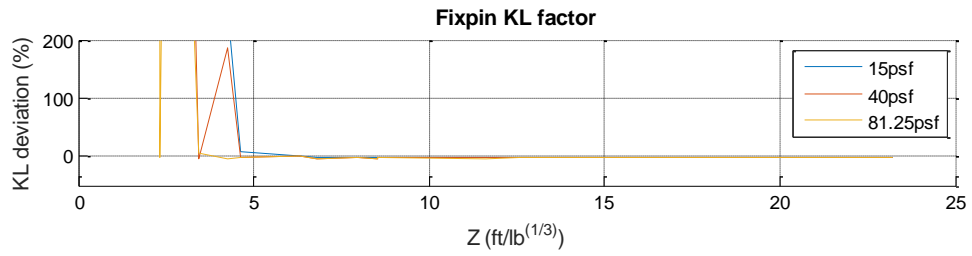
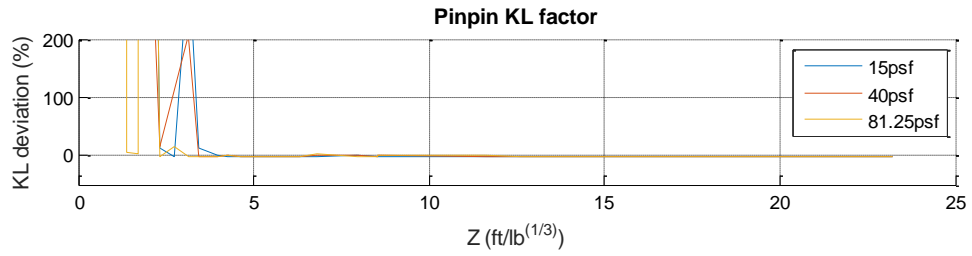


Bernoulli Timoshenko

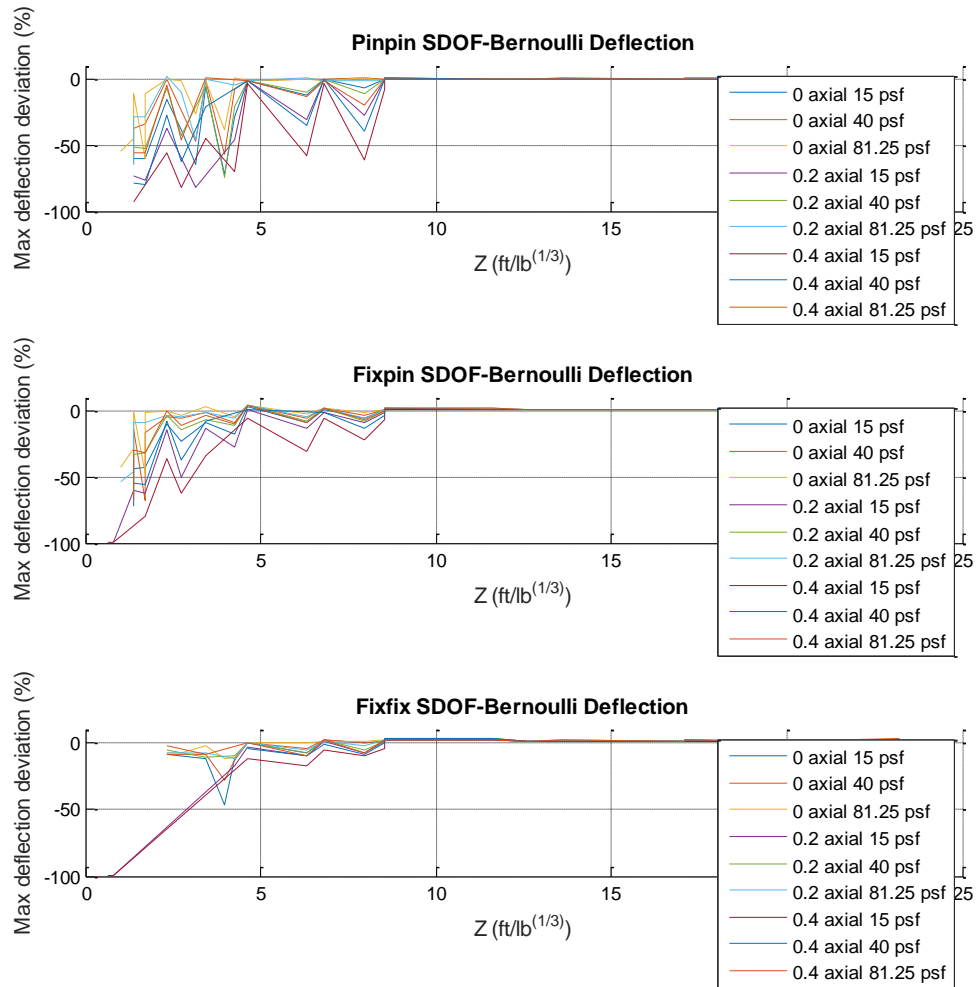


W14X257

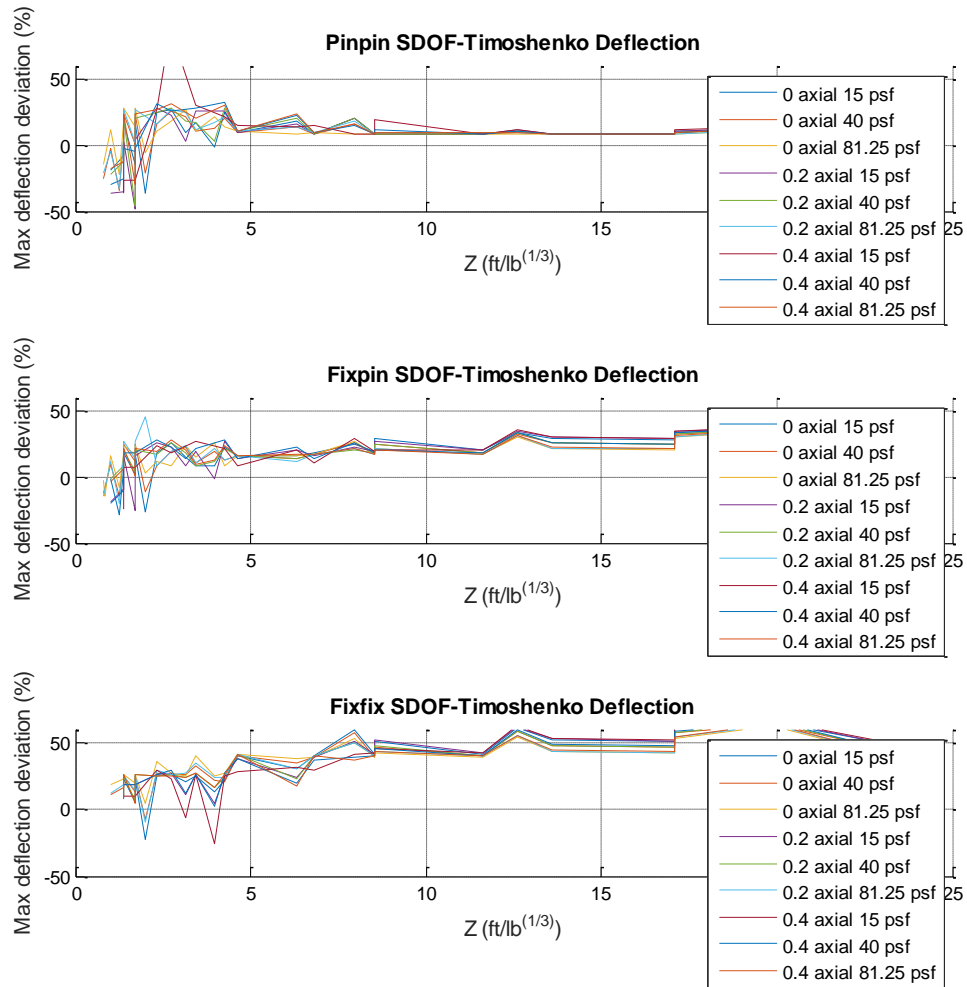
KL Factor



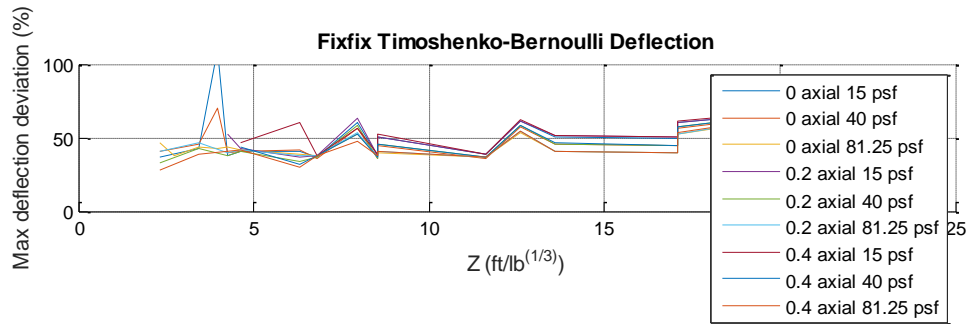
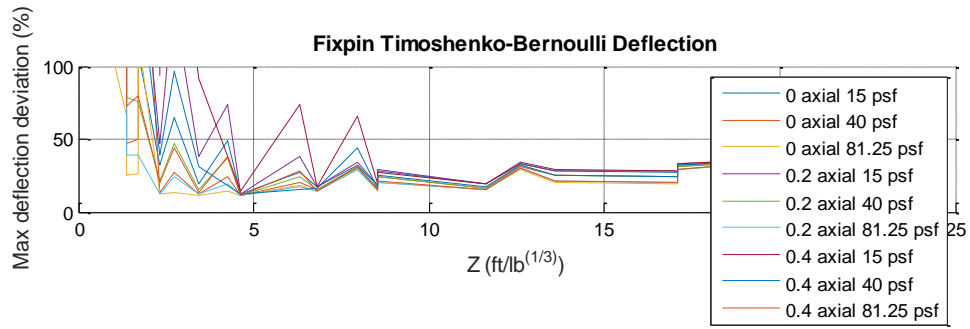
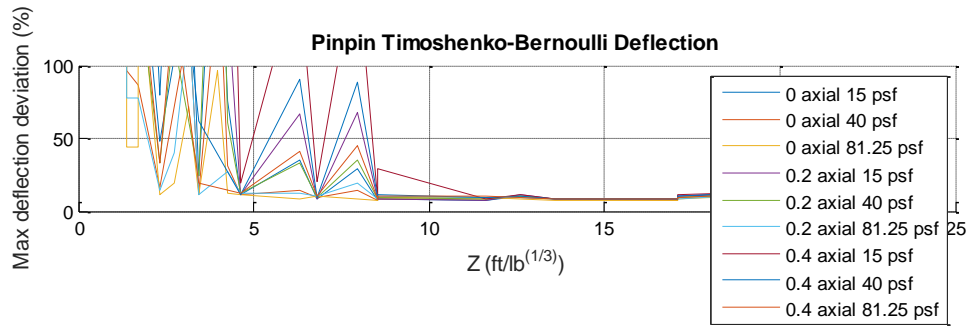
SDOF Bernoulli



SDOF Timoshenko



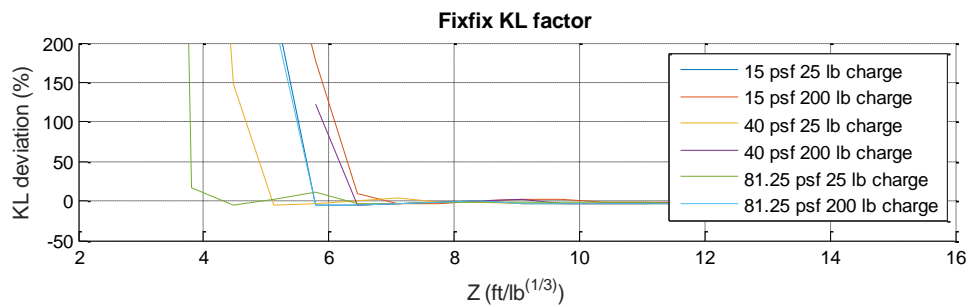
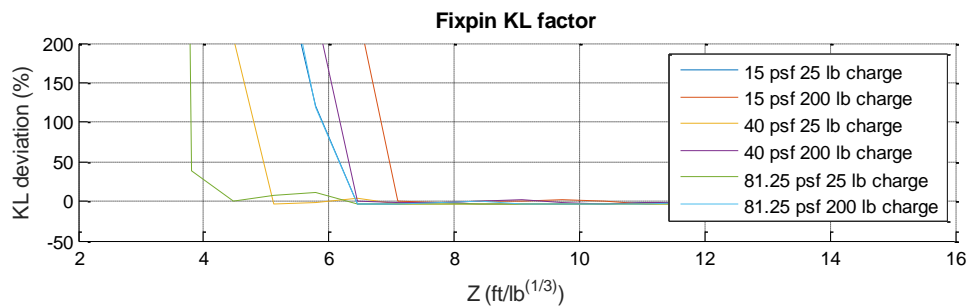
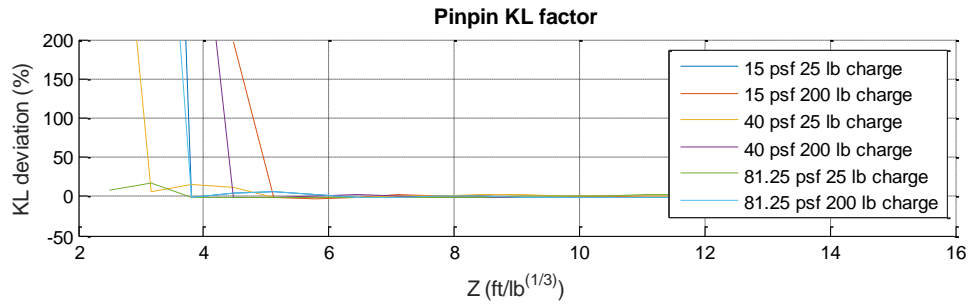
Bernoulli Timoshenko



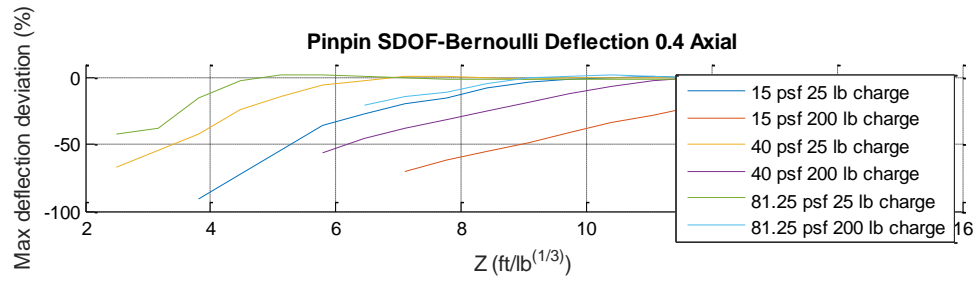
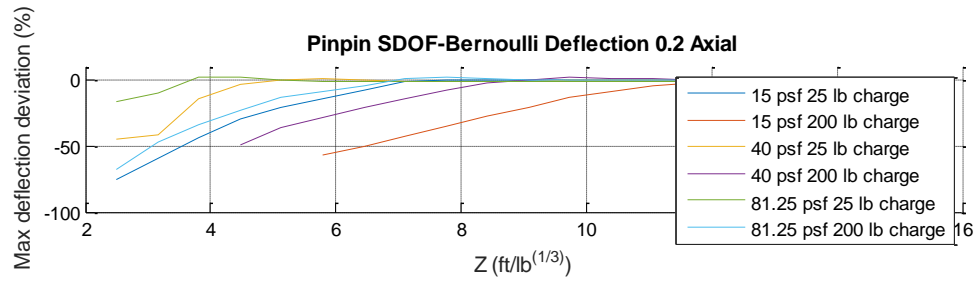
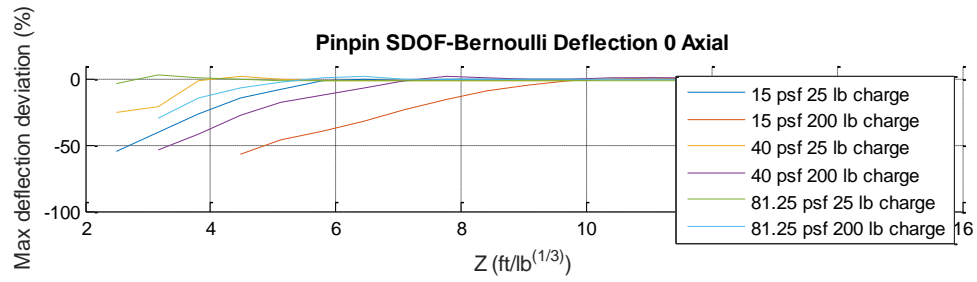
Appendix F: Second Data Set Plots

W14X109 With 2% Damping in First and Second Modes

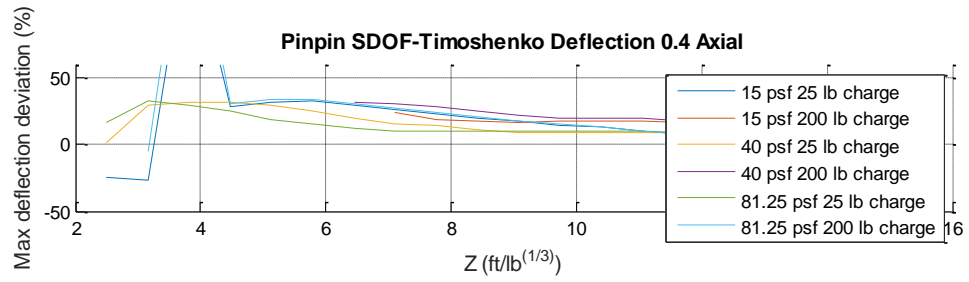
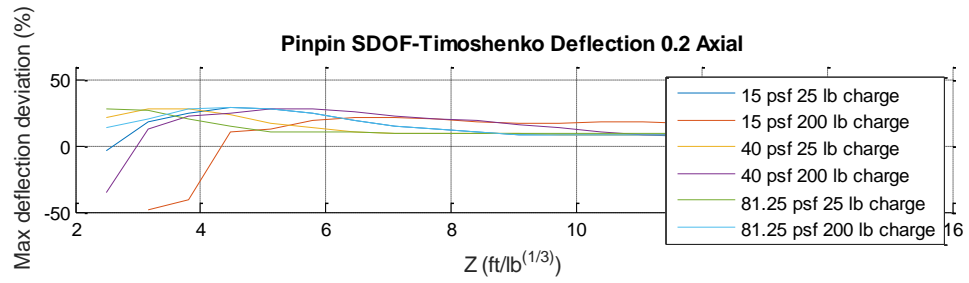
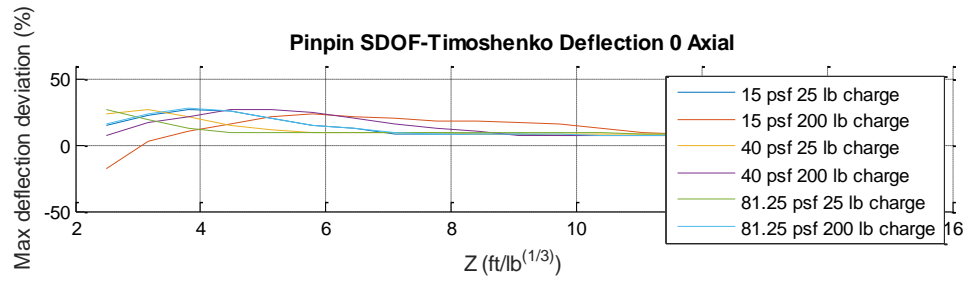
KL factor



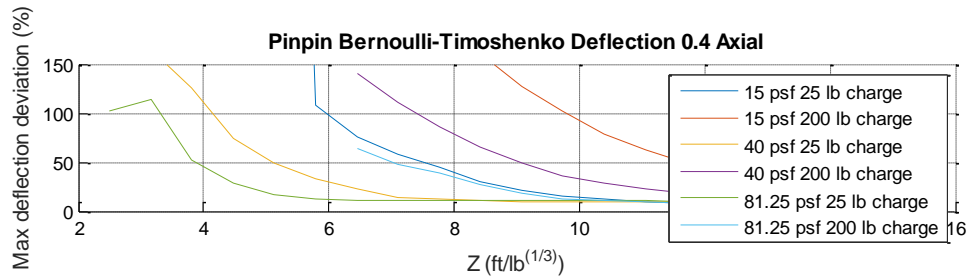
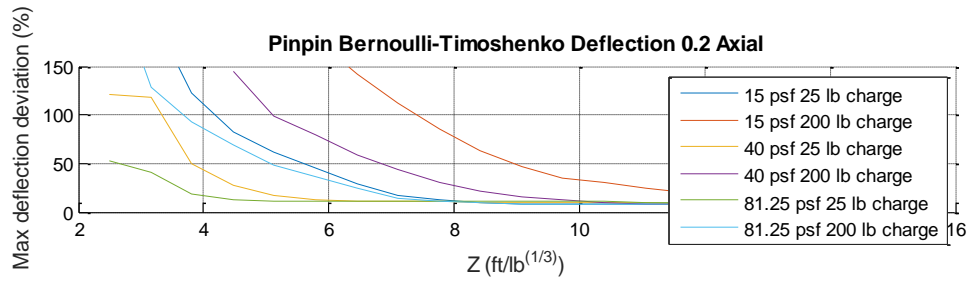
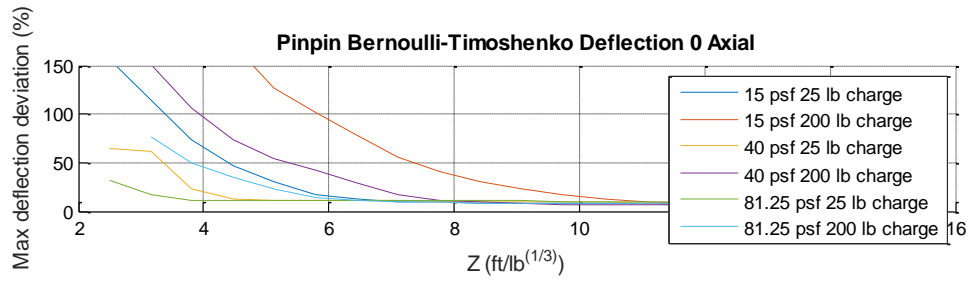
Pinned-Pinned SDOF Bernoulli



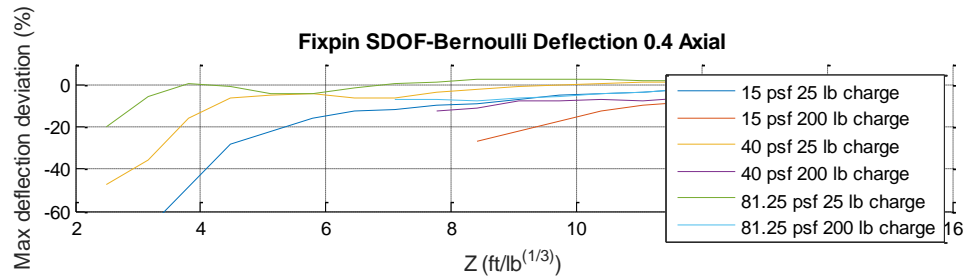
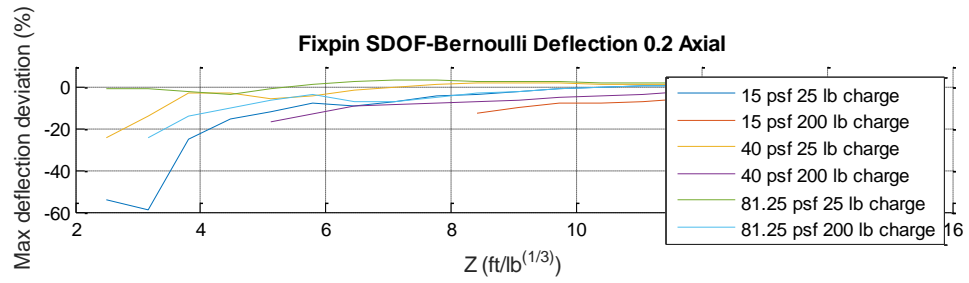
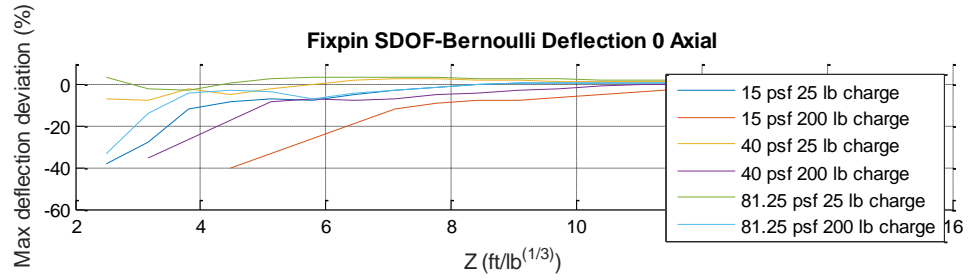
Pinned-Pinned SDOF Timoshenko



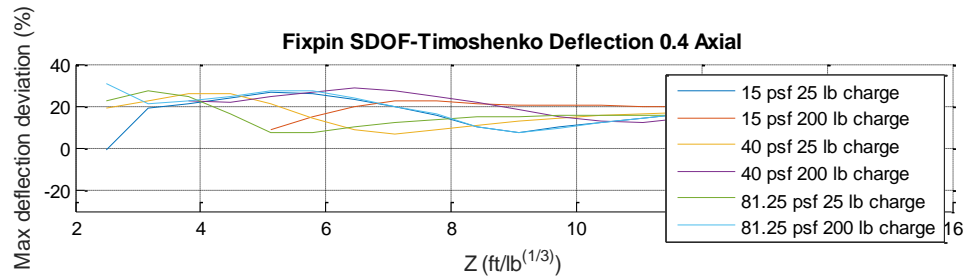
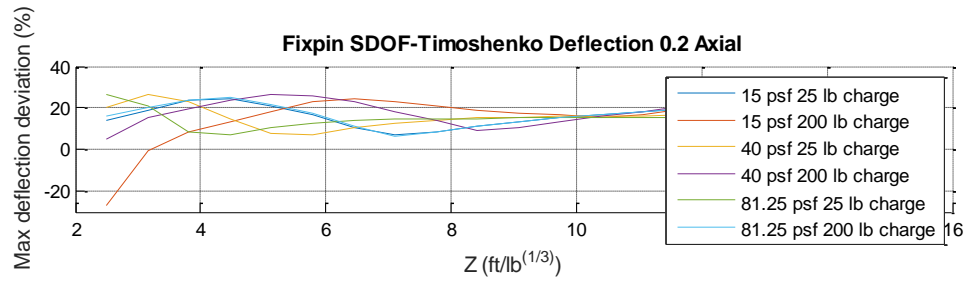
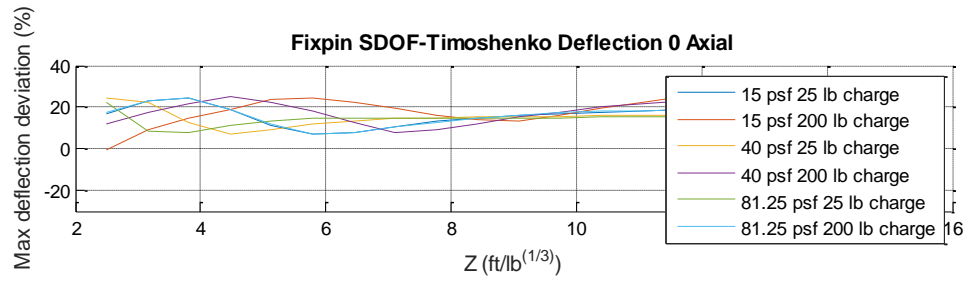
Pinned-Pinned Bernoulli Timoshenko



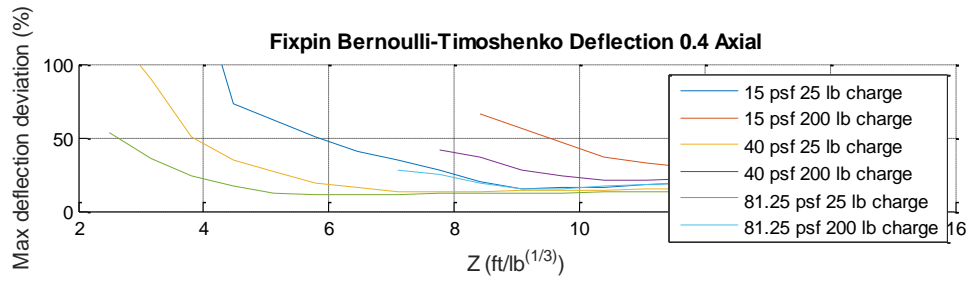
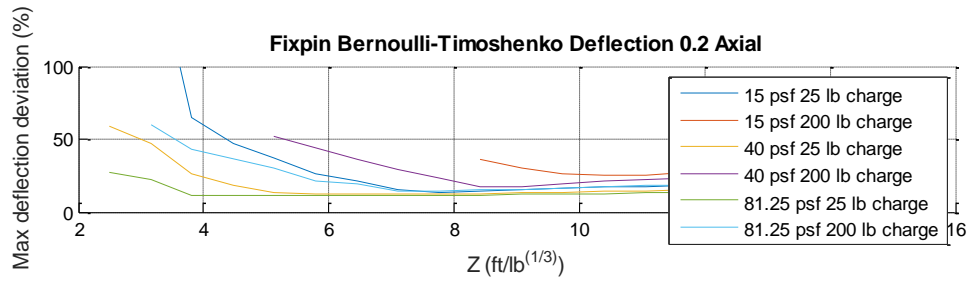
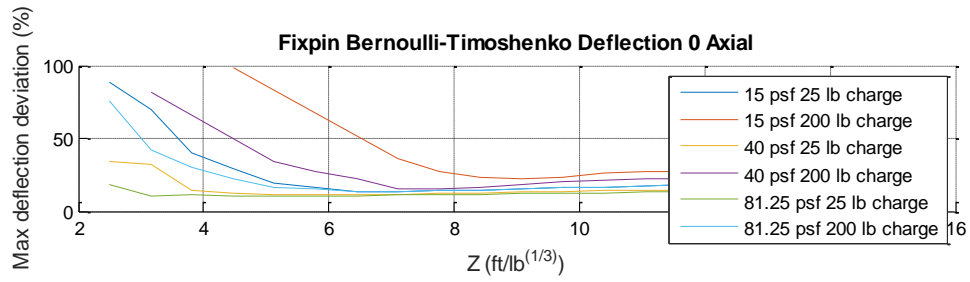
Fixed-Pinned SDOF Bernoulli



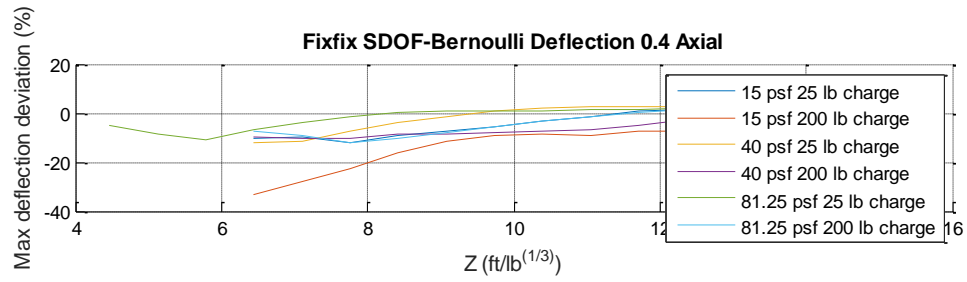
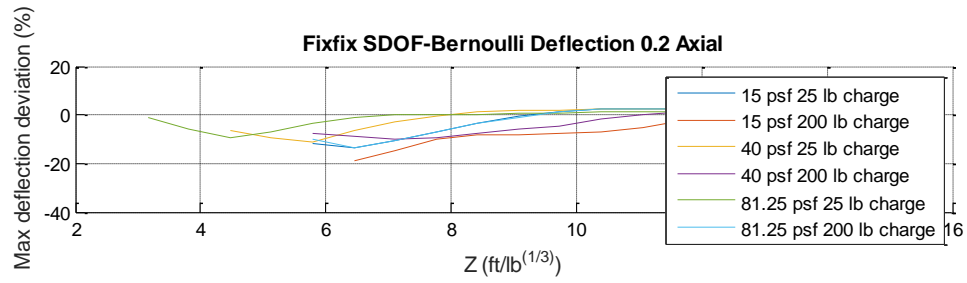
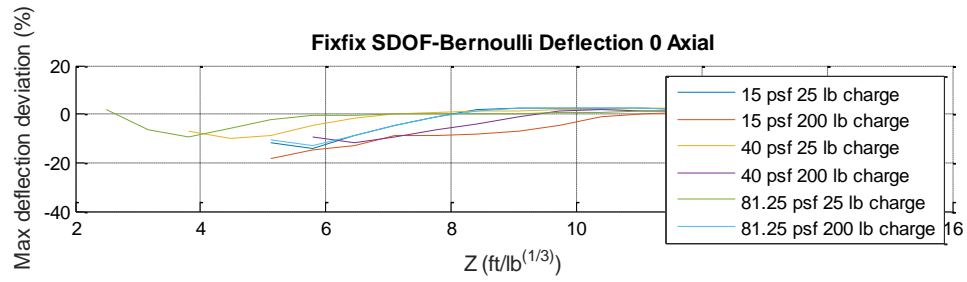
Fixed-Pinned SDOF Timoshenko



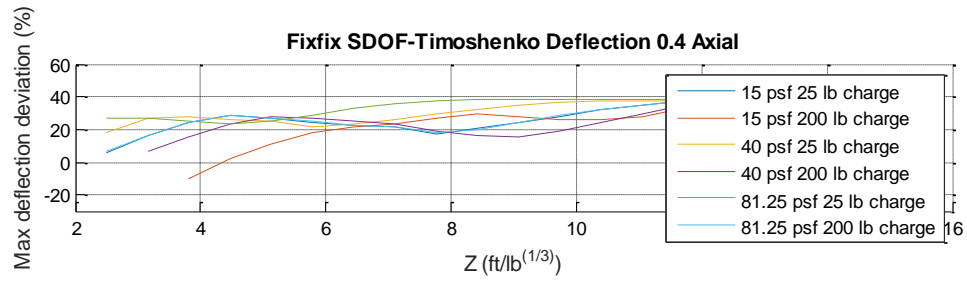
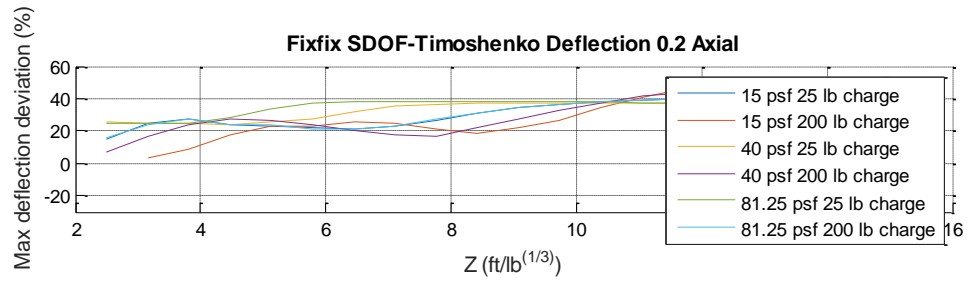
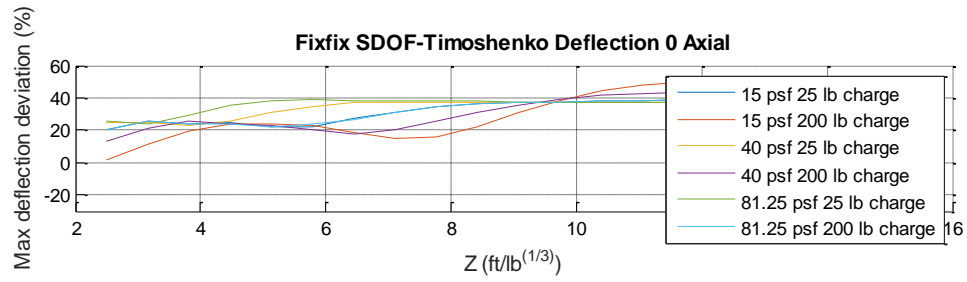
Fixed-Pinned Bernoulli Timoshenko



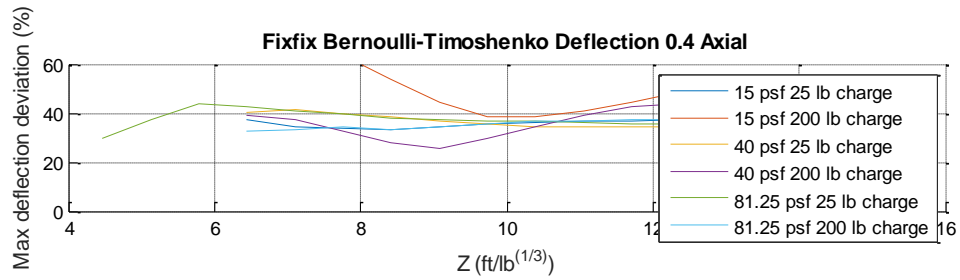
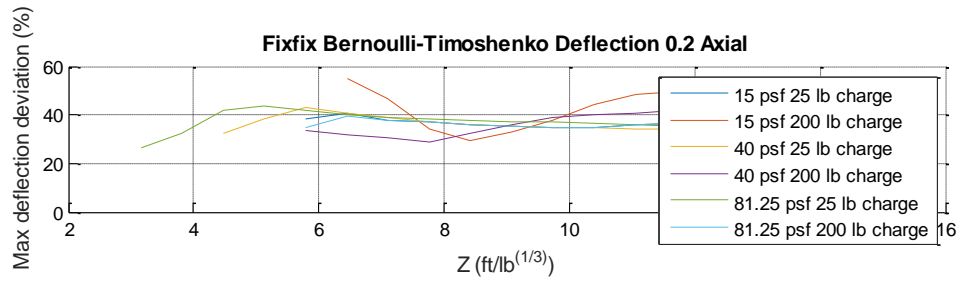
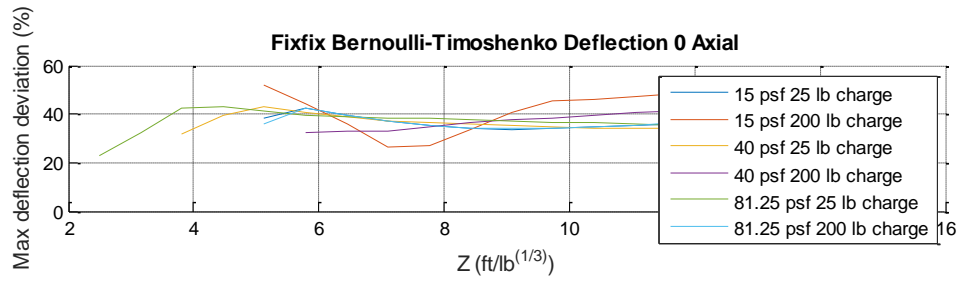
Fixed-Fixed SDOF Bernoulli



Fixed-Fixed SDOF Timoshenko



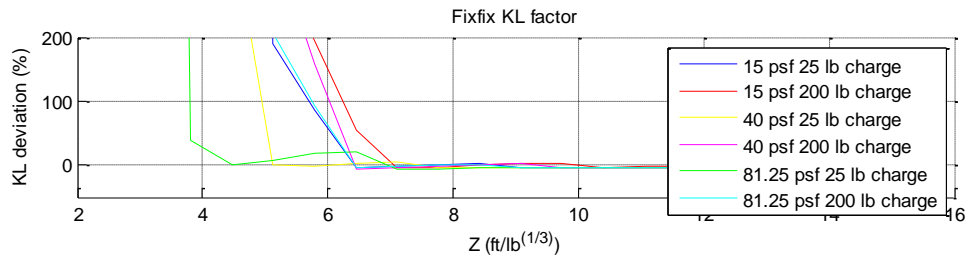
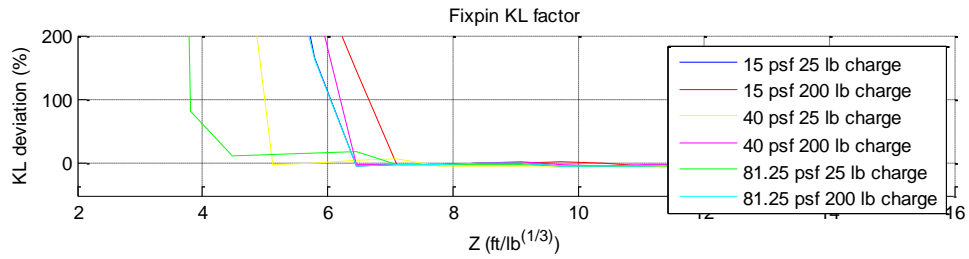
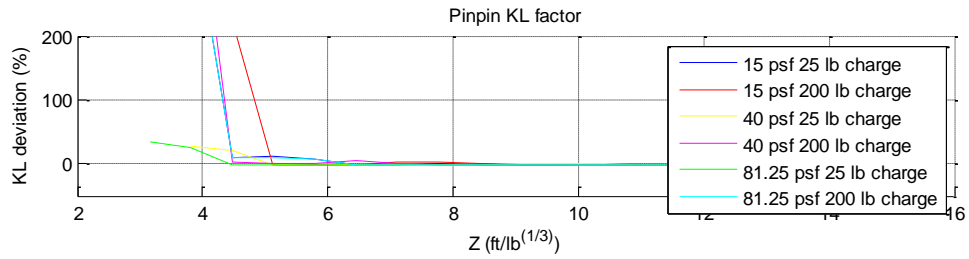
Fixed-Fixed Bernoulli Timoshenko



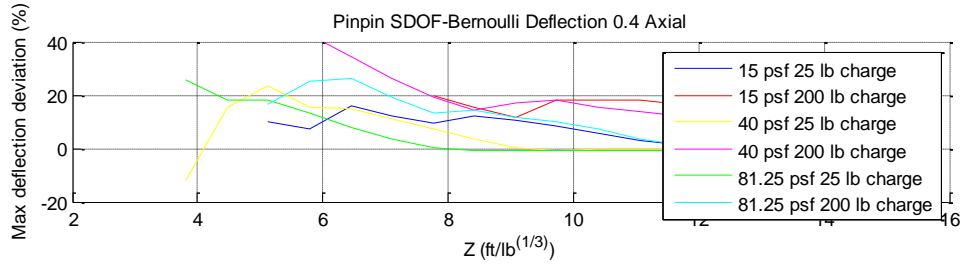
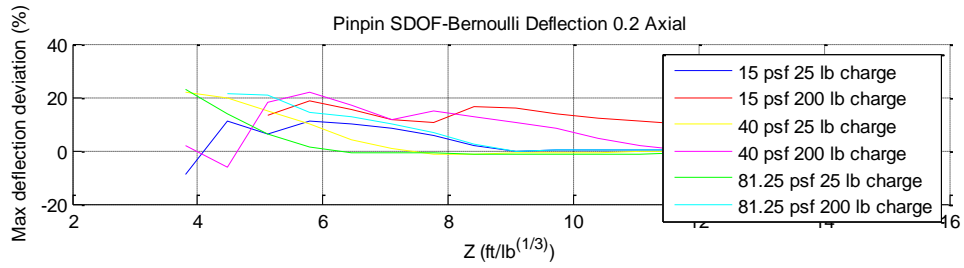
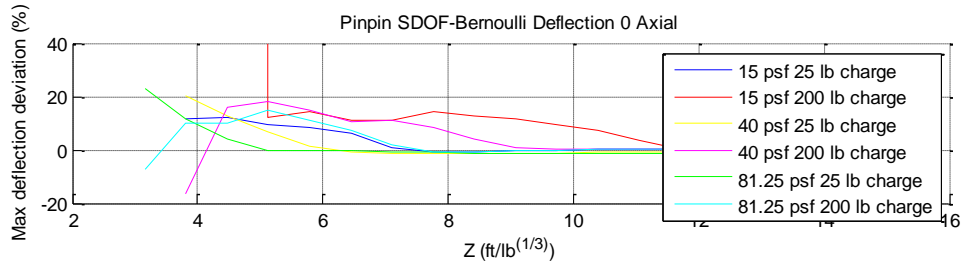
Appendix G: Third Data Set Plots

W14X109 With 2% Damping in First and Last Modes

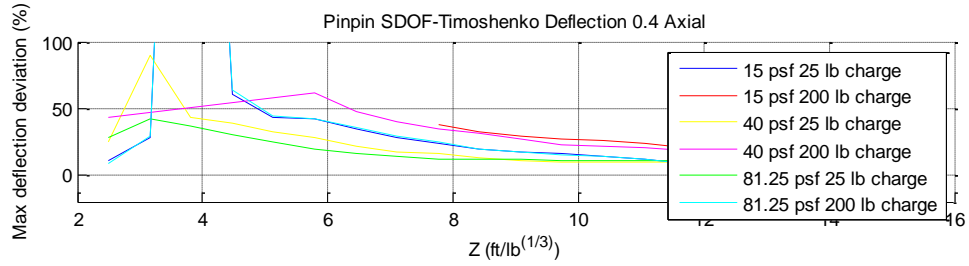
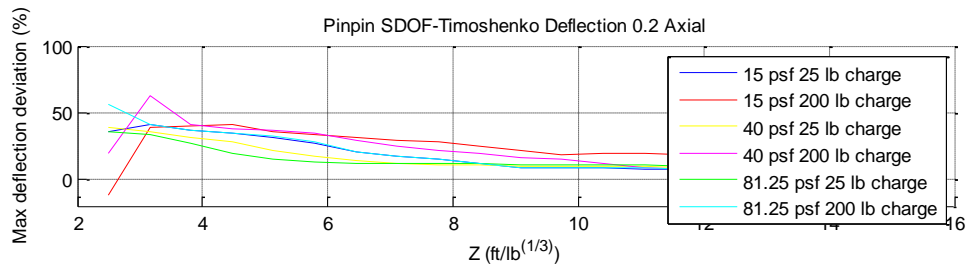
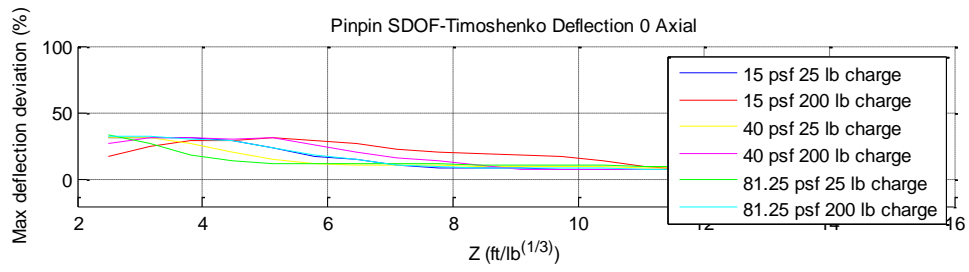
KL factor



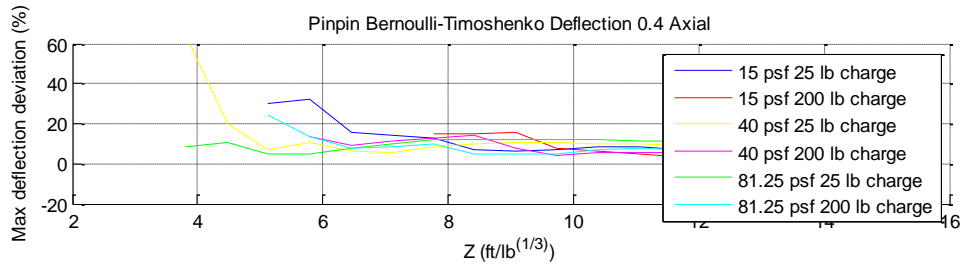
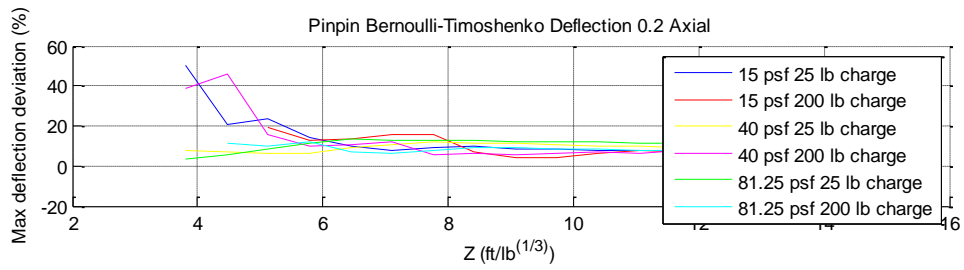
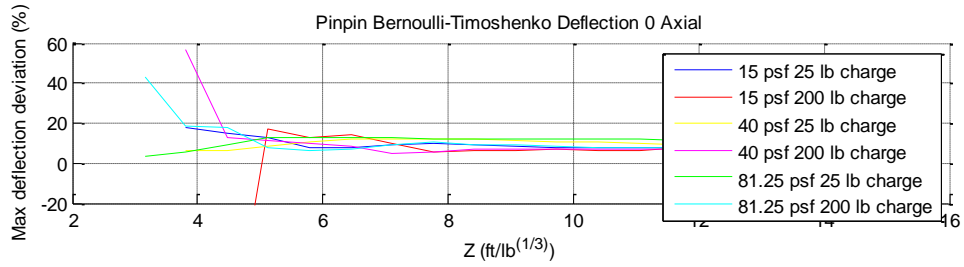
Pinned-Pinned SDOF Bernoulli



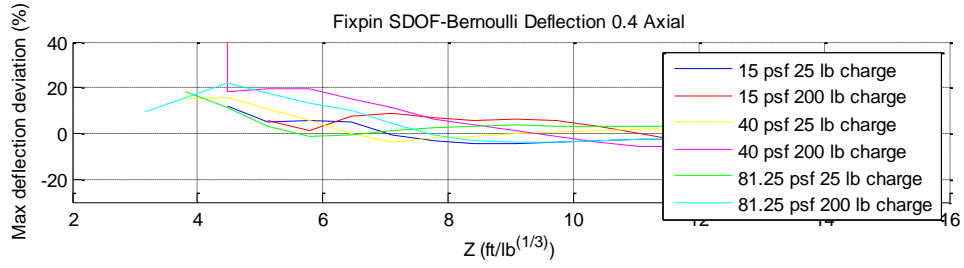
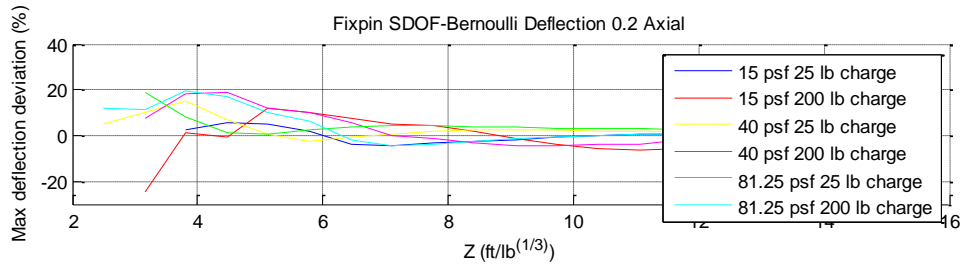
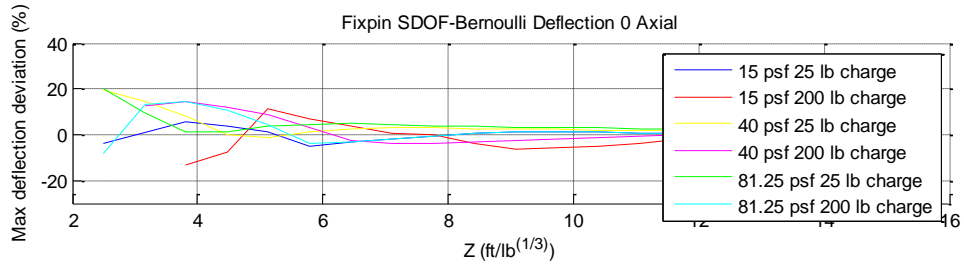
Pinned-Pinned SDOF Timoshenko



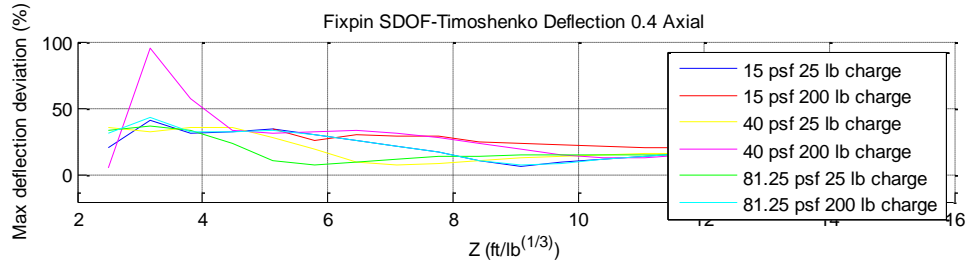
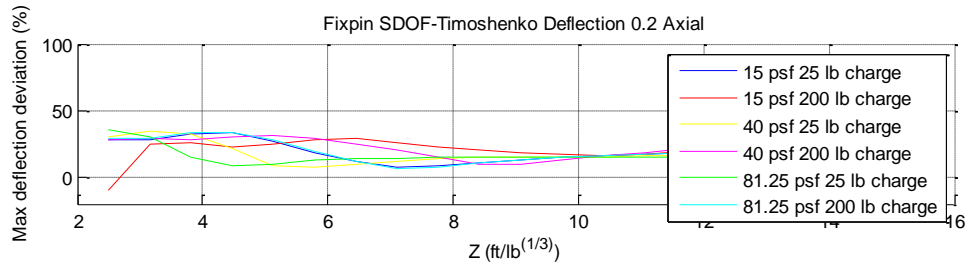
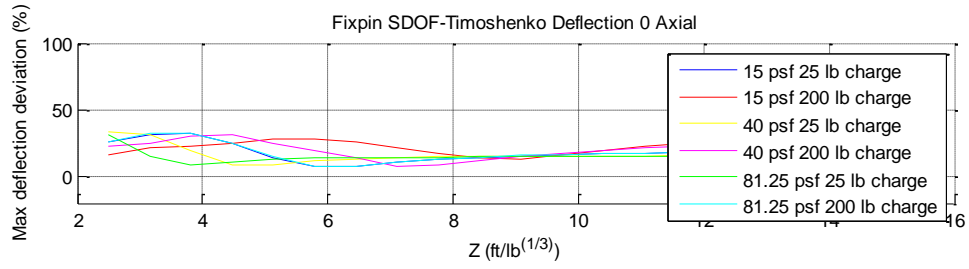
Pinned-Pinned Bernoulli Timoshenko



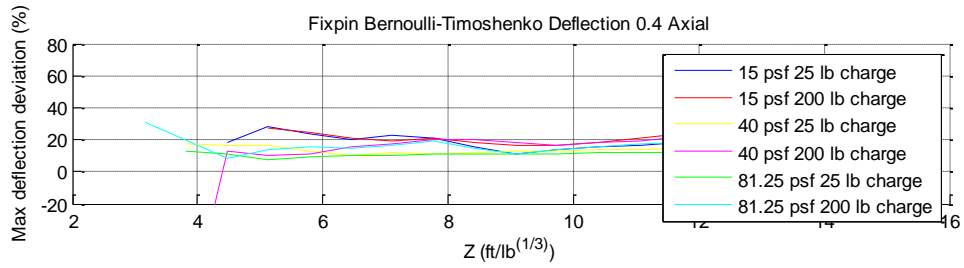
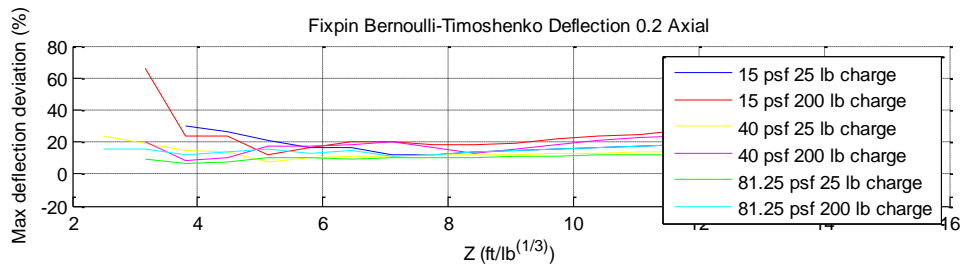
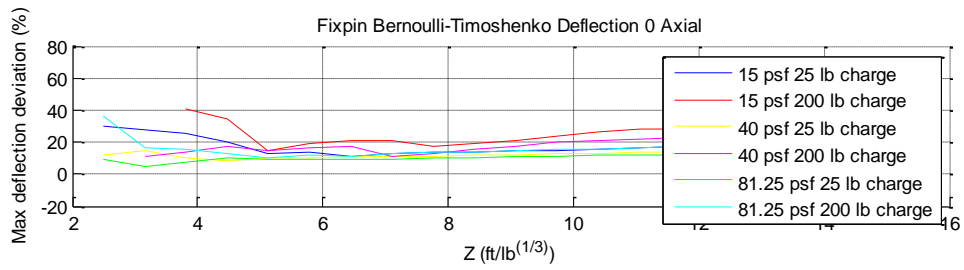
Fixed-Pinned SDOF Bernoulli



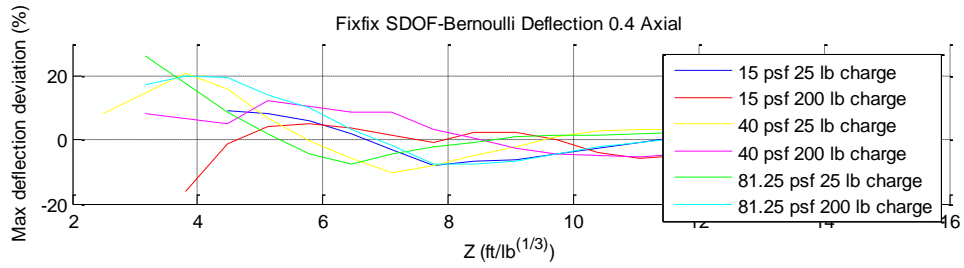
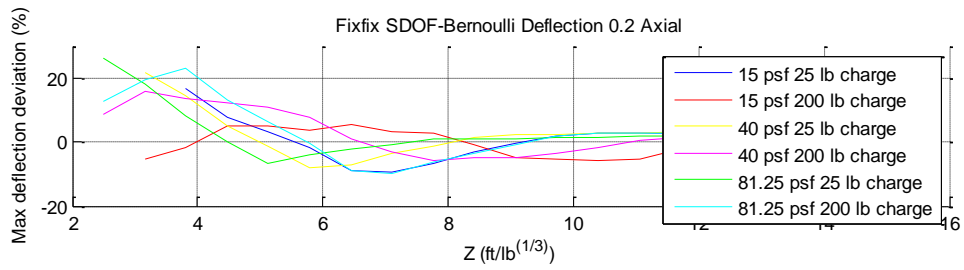
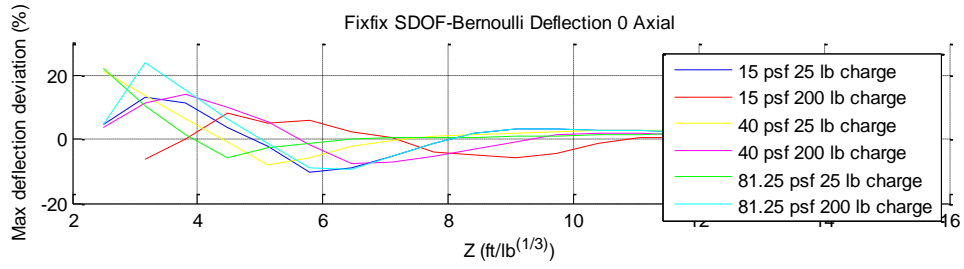
Fixed-Pinned SDOF Timoshenko



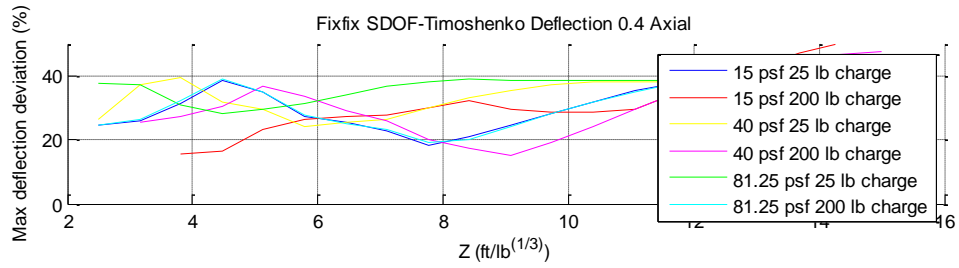
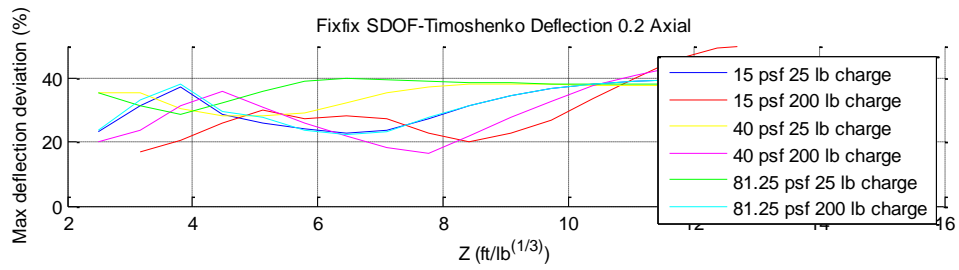
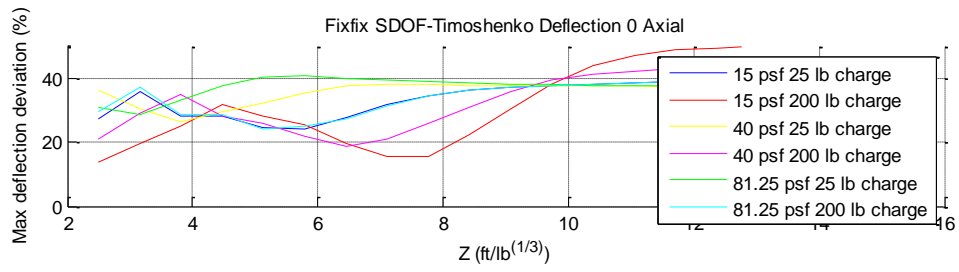
Fixed-Pinned Bernoulli Timoshenko



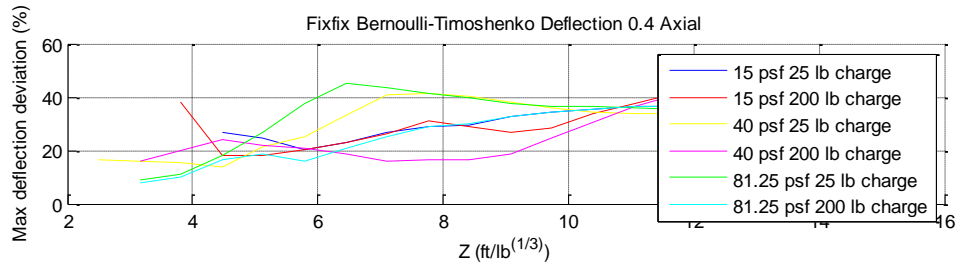
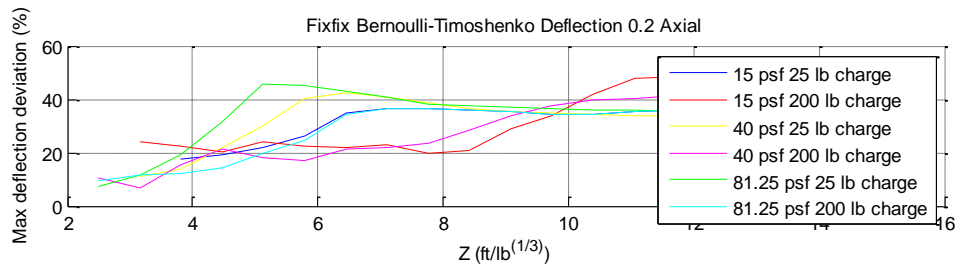
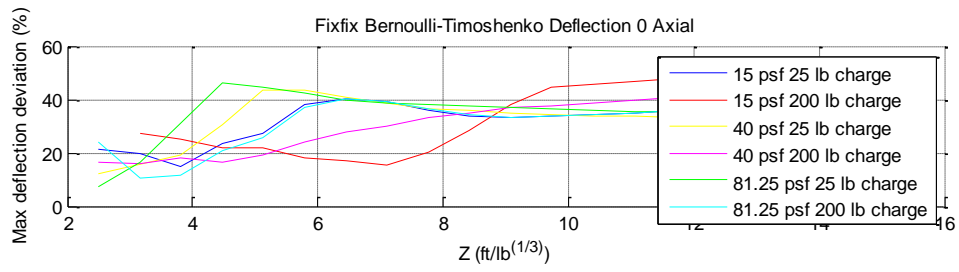
Fixed-Fixed SDOF Bernoulli



Fixed-Fixed SDOF Timoshenko



Fixed-Fixed Bernoulli Timoshenko



Vita

Evan Mullen was born in on February 15, 1992 in Philadelphia, Pennsylvania. He is the son of Mark and Lauren Mullen. Evan graduated from Lehigh University in Bethlehem, Pennsylvania in May of 2014 with a Bachelor of Science in Civil Engineering. In January 2016, Evan will receive his Master of Science in Structural Engineering from Lehigh University. Evan works as a structural engineer for Thornton Tomasetti in Austin, Texas.