Homework 1

Louis Lin

January 28, 2021

University of California, San Diego

Prof. Joel Conte

SE 211 Advance Structural Concrete

Contents

List of Table

List of Figure

Introduction

In this report, two different nonlinear analyses are performed on a single degree of freedom structure. A quasi-static, cyclic push-over analysis is first performed using the incremental-iterative Newton-Raphson method to solve the nonlinear equation of static equilibrium. Secondly, a nonlinear time history analysis is performed using the Newmark-Beta method as the time stepping scheme along with the Newton-Raphson method to solve the nonlinear system. The use of the modified N-R method is also explored for each analysis. These results were then verified using Opensees.

Methodology

In this report, the two nonlinear structural analysis programs were developed based upon the code provided by the instructor, Professor Conte. The pseudocode for both the static analysis and transient analysis is provided in Appendix A.

In this report, one main file `SE201B_HW1_P2_Run.m` was used to run all of the trials. Two functions `Static.m` and `Transient.m` were used to solve the nonlinear systems. These functions took as input, the material data of the system, the initial material state, the algorithm used for solving the system, and the max iterations for the solver loop. The solutions of the analysis were then loaded into a *record* data structure that also stored all of the inputs. The *record* structure stored anything that needed to be accessed later such as the load steps, applied accelerations, relative displacement, iterations per load step, material data, unbalanced forces, and so forth. The *record* data structures were used in the analysis and plotting as seen in this report.

A `Static_no_record.m` is also provided for the timed assessment of the algorithm which eliminates the storage of variables in order to better assess the speed of the function itself.

Several other MATLAB files were also made in order to compartmentalize most of the code; and allows for other uses in the future. The files were mainly to initialize the material data and material state: `Initialize_MatData.m`, `Initialize_Material_State.m`, and `equivalent_truss.m`.

The `Menegotto_Pinto.m` function was used to calculate the Menegotto-Pinto Uniaxial model force-displacement curve given a set of converged displacements which relied heavily on `Mate25n.m` file provided by the instructor. The `Mate25n.m` implemented the Menegotto-Pinto uniaxial constitutive model.

The file `Plot.m` was used to plot all of the results as scalable vector graphic file. It is assumed that the records from running the main file is in the current workspace as the file will be calling them by record specific names.

The following files were provided by Angshuman Deb in order to facilitate the Opensees portion of this report. The `run_opensees.m` runs the Opensees executable sourcing the `run.tcl` file and stores the results in the Results folder. The `postProcess.m` plots the results as well as runs `get_materialHysteresis.m` to get the Menegotto-Pinto uniaxial model results from Opensees for the given load. Depending on the analysis type in the `run.tcl`, either the `analysisPushover.tcl` or `analysisTimeHistory.tcl` were called to run a pushover analysis or time history analysis. The `run.tcl` also specify the type of algorithm to be used in the analysis.

The `trussModel.tcl` was written with the properties given in the problem statement. This was not provided.

Problem 1 – Deriving the Material Tangent Modulus

The Bauschinger curve is the stress-strain relationship used in the Menegotto-Pinto uni-axial material constitutive model described by (1). R is a parameter that describes the curvature of the curve.

$$(1)\ \sigma^* = b \cdot \epsilon^* + \frac{(1-b)\cdot\epsilon^*}{(1+|\epsilon^*|^R)^{1/R}}$$

Where σ^*, ϵ^* is defined by (2) and (3). σ_r, ϵ_r is the stress-strain at the last load reversal state. σ_y, ϵ_y is the yield stress-strain of the current curve.

$$(2)\ \ \sigma^* = \frac{\sigma-\sigma_r}{\sigma_y-\sigma_r}$$

$$(3)\ \epsilon^* = \frac{\epsilon-\epsilon_r}{\epsilon_y-\epsilon_r}$$

The tangent of the stress strain curve is necessary for the incremental changes used in the Newton-Raphson method. For the stress-strain curve, the tangent is called the tangential modulus, given by (4).

$$(4)\ E_T = \frac{d\sigma}{d\epsilon} = \frac{d\sigma}{d\sigma^*}\cdot\frac{d\sigma^*}{d\epsilon^*}\cdot\frac{d\epsilon^*}{d\epsilon}$$

Rewriting the equation (3),

$$(5)\ \sigma = \sigma^* \cdot \left(\sigma_y - \sigma_r\right) - \sigma_r$$

and solving for each portion of the differential,

$$(6)\ \frac{d\sigma}{d\sigma^*} = \frac{d}{d\sigma^*}\left(\sigma^*\cdot\left(\sigma_y-\sigma_r\right)-\sigma_r\right) = \left(\sigma_y-\sigma_r\right)$$

$$\begin{aligned}
(7)\ \frac{d\sigma^*}{d\epsilon^*} &= \frac{d}{d\epsilon^*}\left(b\cdot\epsilon^* + (1-b)\frac{\epsilon^*}{(1+|\epsilon^*|^R)^{\frac{1}{R}}}\right)\\
&= b + (1-b)\cdot\frac{\left(1+|\epsilon^*|^{\frac{1}{R}}\right)-\epsilon^*\cdot\frac{1}{R}(1+|\epsilon^*|^R)^{\frac{1}{R}-1}\cdot R|\epsilon^*|^{R-1}}{(1+|\epsilon^*|^R)^{\frac{2}{R}}}\quad ;\ \textit{assumed}\ \frac{d}{d\epsilon^*}|\epsilon^*|\ \textit{didn't have} \pm \textit{cases}\\
&= b + \frac{(1-b)}{(1+|\epsilon^*|^R)^{\frac{1}{R}}}\left(1-|\epsilon^*|^R(1+|\epsilon^*|^R)^{-1}\right)\ \ ;\ \textit{factored out}\ (1+|\epsilon^*|^R)^{\frac{1}{R}}\\
&= b + \frac{(1-b)}{(1+|\epsilon^*|^R)^{\frac{1}{R}+1}}\ ;\ \textit{simplify}
\end{aligned}$$

$$(8)\ \frac{d\epsilon^*}{d\epsilon} = \frac{d}{d\epsilon}\left(\frac{\epsilon-\epsilon_r}{\epsilon_y-\epsilon_r}\right) = \frac{1}{(\epsilon_y-\epsilon_r)}$$

Therefore, the tangent modulus can be given as

$$(9)\ E_T = \frac{(\sigma_y-\sigma_r)}{(\epsilon_y-\epsilon_r)}\cdot\left(b + \frac{(1-b)}{(1+|\epsilon^*|^R)^{\frac{1}{R}+1}}\right)$$

In the provided code, the tangent curve is given as equivalently as

$$(10)\ E_T = \frac{(\sigma_y-\sigma_r)}{(\epsilon_y-\epsilon_r)}\cdot\left(b + \frac{(1-b)\cdot\left(1-\frac{|\epsilon^*|^R}{1+|\epsilon^*|^R}\right)}{(1+|\epsilon^*|^R)^{\frac{1}{R}}}\right)$$

Problem 2 – Modeling A Nonlinear Single Degree of Freedom System

Quasi-Static

In this problem, a quasi-static cyclic push over analysis and a nonlinear time history analysis is performed for a single-story building structure. The building has a roof weight of $mg = 2000$ kips, an initial natural period of vibration of $T_0 = 0.20$s, and a yield strength of $R_{y0} = 0.15mg$.

The structure is modeled as a simply-supported equivalent single truss element with the following fictitious material properties for the Menegotto-Pinto uni-axial constitutive law.

$$E = 30{,}000\text{ksi},\ b = 0.02,\ R_0 = 5,\ a_1 = 3.0,\ a_2 = 0.15,\ a_3 = a_4 = 0$$

The relationship between the axial strain, ϵ, and axial stress, σ, of the equivalent truss element model to the roof displacement relative to the ground U, and internal resisting force R, of the structure is given by

$$U = L \cdot \epsilon \ \text{ and } R = A \cdot \sigma$$

In order to simplify the analysis of the system, the SDOF structure can to be equated to an equivalent simply-supported truss element. This truss element should have properties that allow it to develop the same internal resisting force as the real system and have the same yield stress. It was decided that the truss would have an arbitrary length of 60 inches. Using this and the properties given by the instructor,

(1) $K_0 = \frac{4\pi^2 W}{g \cdot T_0^2} = \frac{4\pi^2(2000\text{ kip})}{(386.4\frac{\text{in}}{\text{s}^2})(0.2\text{s})^2} = 5110.0\frac{\text{kip}}{\text{in}}$; stiffness of the truss element

(2) $A = \frac{K_0 L}{E} = \frac{\left(5113.8\frac{\text{kip}}{\text{in}}\right)(60\text{in})}{(30000\text{ksi})} = 10.2\text{in}^2$; cross-sectional area of the truss element

(3) $\sigma_y = \frac{R_{y0}}{A} = \frac{0.15(2000\text{kip})}{10.227\text{in}^2} = 29.3$ ksi; initial yield stress of the truss element

The load levels for this analysis are given as a vector P and shown in Figure 1. Note that there is a smaller load step when there's a load reversal which aids the program in converging when switching branches in the Bauschinger curve. $P = [0.,\ 200,\ 300,\ 400,\ 399.9,\ 300,\ 200,\ 0,\ -200,\ -300,\ -400,\ -399.9,\ -300,\ -200,\ 0, 150,\ 250,\ 300,\ 350,\ 349.9,\ 250,\ 100]$ kips.

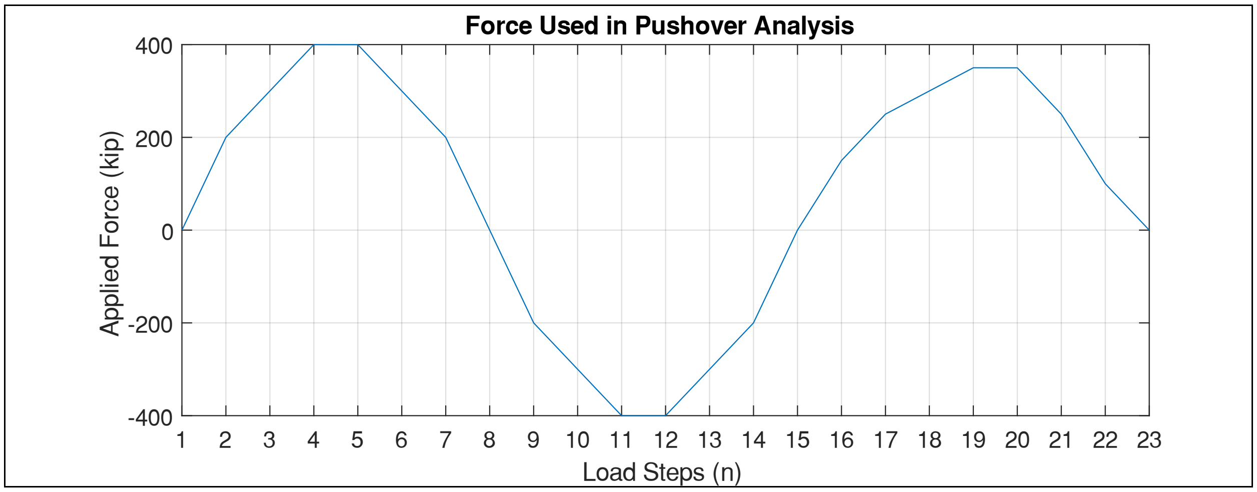

Figure 1 Push-Over Curve Applied Loads

Part (a) Quasi-Static Cyclic Push-Over Analysis using the Newton- Raphson Method

The quasi-static, cyclic loading, push-over curve solved with the incremental-iterative Newton-Raphson method is presented in Figure 2. On top of the converged path, the force-displacement curve of the Menegotto-Pinto Uni-axial Model is also plotted. This was accomplished by transforming the converged displacement into a strain time history that was ran through the Menegotto-Pinto model with 500 linearly interpolated strains between each converged displacement. The Menegotto-Pinto model provides a better displacement time history than the N-R method since the path is already converged the model is just solving the system procedurally.

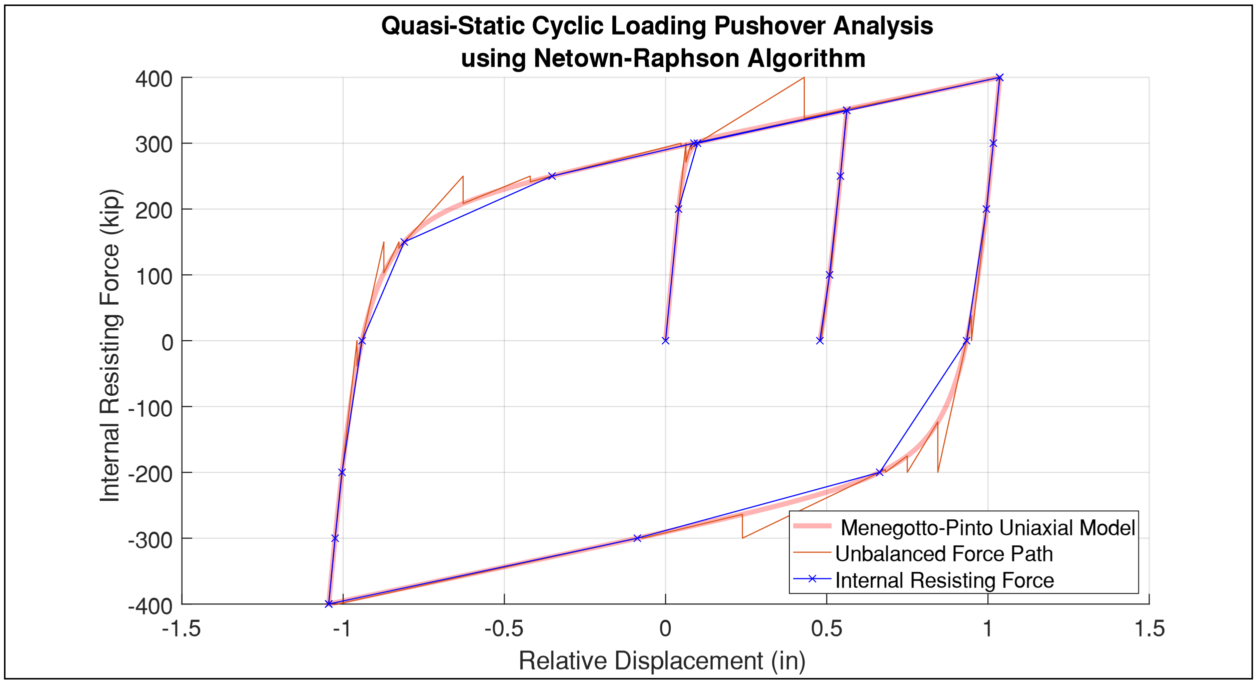

Figure 2 Quasi-Static Cyclic Loading Using Newtown Raphson Method

Part (b) Quasi-Static Cyclic Push-Over Analysis using the Modified Newton-Raphson Method I

Using the modified Newton-Raphson method, the push-over curve is presented in Figure 3. The modified N-R method sets the tangent stiffness at every iteration to the initial stiffness at the beginning of each load step, which is the tangent stiffness of the last converged point. This becomes an issue for load reversal points, where the tangent at the previous converged point would overshoot the next load step and has a problem converging since the next load step is in the opposite direction. Thus, when the method did not converge, the normal N-R method was used to solve these load steps. This can be seen in the unbalancing force path overshooting the edge of the pushover curve at the load reversal points.

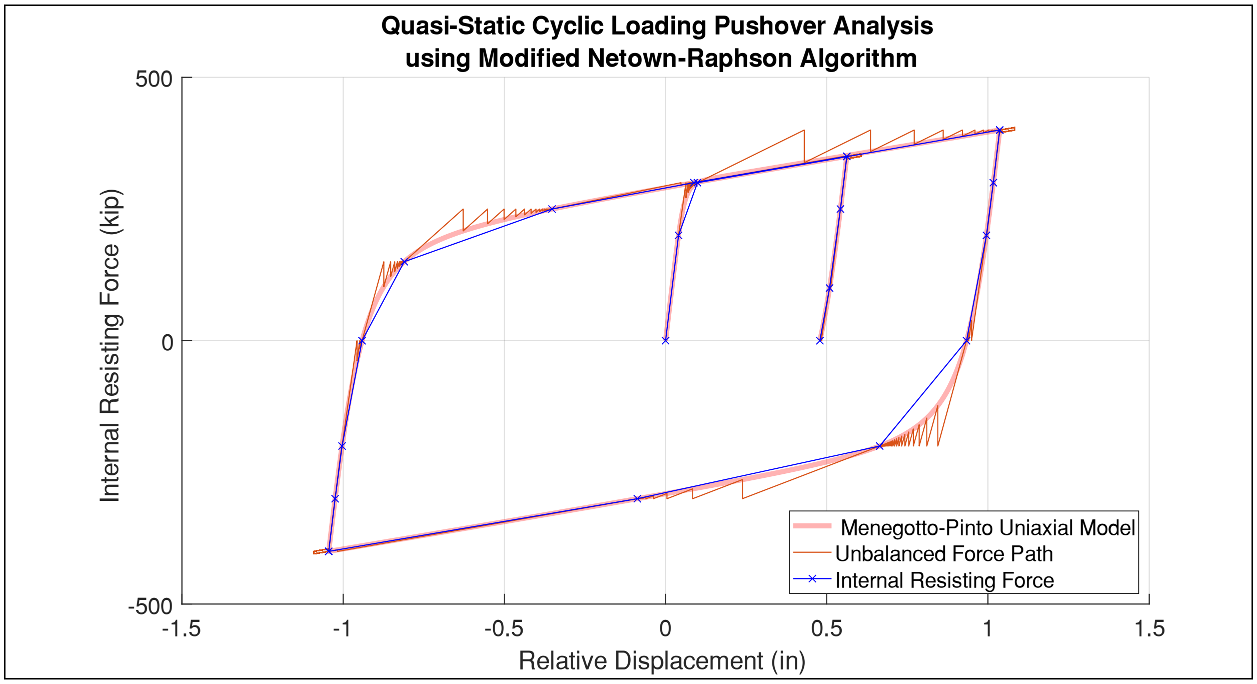

Figure 3 Quasi-Static Cyclic Loading Using Modified Newtown Raphson Method I

Part (c) Quasi-Static Cyclic Push-Over Analysis using the Modified Newton-Raphson Method II

The same problem was solved with another modified Newton-Raphson method where the tangent at every iteration equal to the initial elastic stiffness. This is evident in that all the unbalance force paths have the same slope as the initial elastic branch as shown in Figure 4.

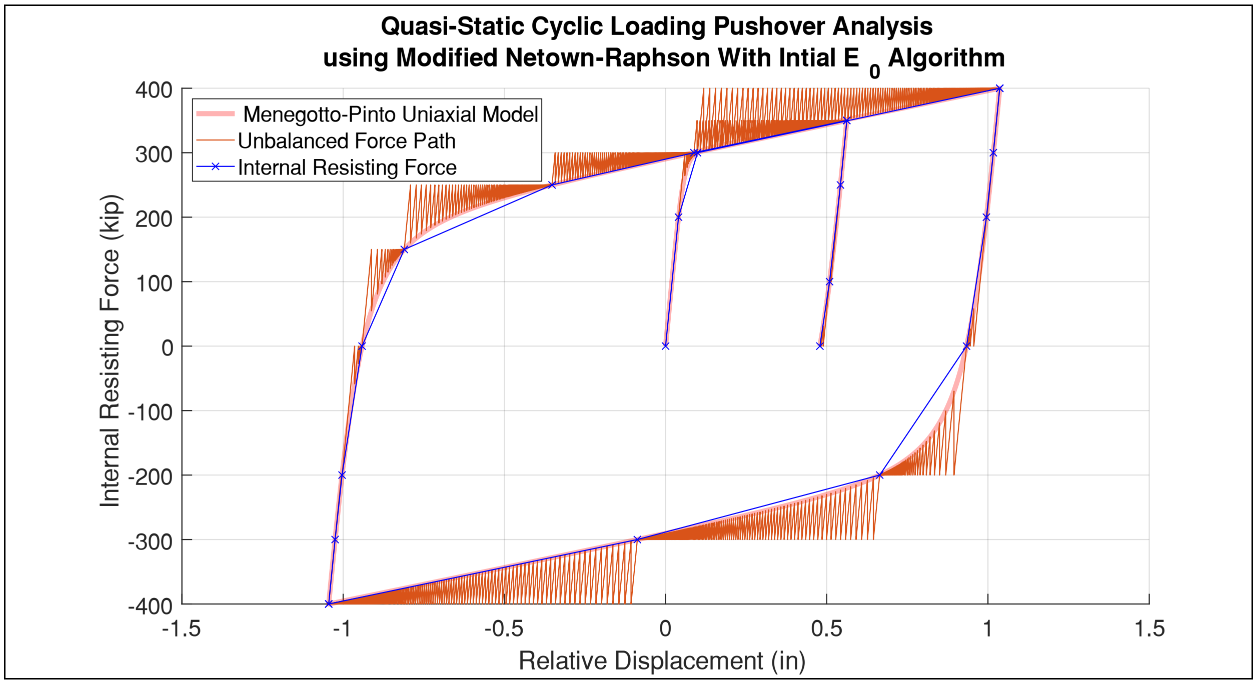

Figure 4 Quasi-Static Cyclic Loading Using Modified New town Raphson Method II

Part (a-c) Comparison of N-R and Modified N-R methods

In Figure 5, the absolute value of the unbalance force was plotted on a log scale against the load step number. This figure shows that the N-R method converged to a norm-convergence criterion of 10^{-5} faster than the modified N-R method. While the N-R method took a maximum of 7 iterations across all load steps to converge, some iterations of the Modified N-R method I and II took ≥ 700. Note load step 4, 11, and 19 where the modified algorithm did not converge.

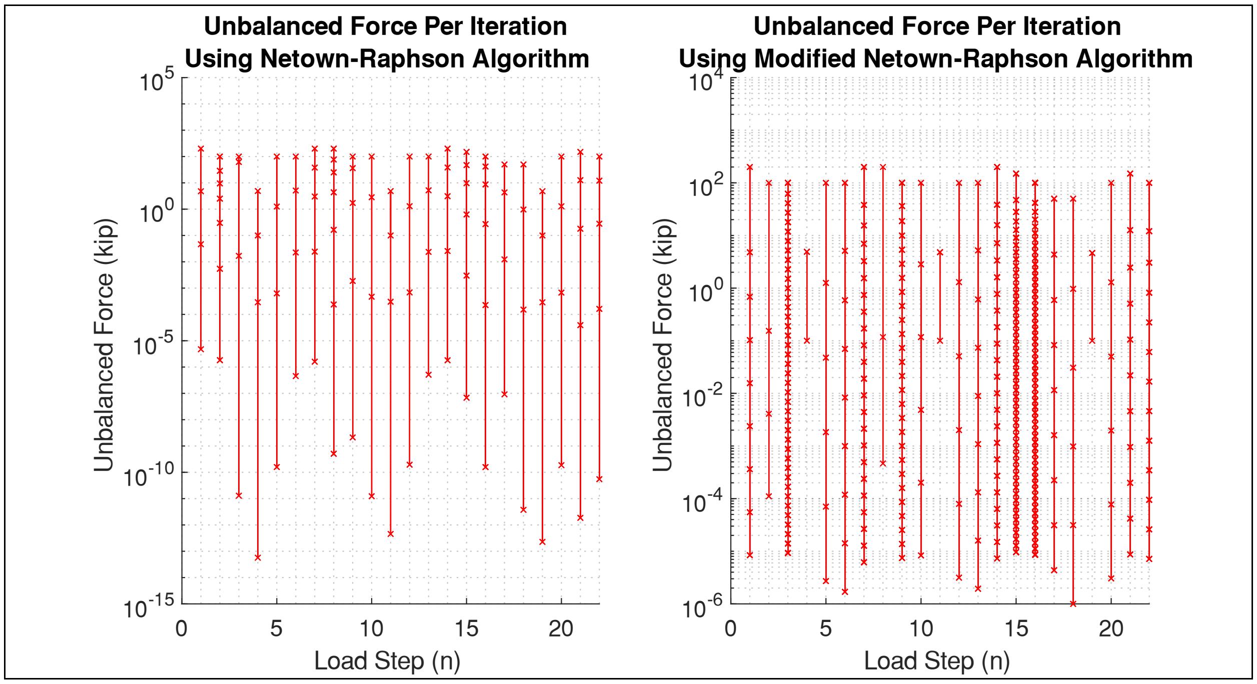

Figure 5 Plot of the Norm of Unbalanced Force per Load Step

The norms for load step 10 and load step 14 for the N-R method and Modified N-R method are presented in Table 1. For the N-R method, the unbalance force after the initial residual, decreased in magnitude from $10^0, 10^{-4}$,to 10^{-11} and $10^1, 10^0, 10^{-2}$, to 10^{-6}, whereas the modified N-R method had smaller changes in magnitude. The rate of the magnitude change in the N-R method is faster or equal to the quadratic rate of convergence given by $10^0, 10^{-2}, 10^{-4}, 10^{-6}, \dots$.

Table 1. Norm of the Unbalanced Force for Two Load Steps in the Pushover Analysis

	N-R Method		Modified N-R Method I	
Iteration Step	Load Step 10	Load Step 14	Load Step 10	Load Step 14
1	1.000 e+02	2.000e+02	1.00E+02	2.00E+02
2	2.835 e+00	3.833e+01	2.84E+00	3.83E+01
3	4.701 e-04	3.099e+00	1.17E-01	1.57E+01
4	1.222 e-11	2.570e-02	4.85E-03	7.13E+00
5	-	1.809e-06	2.01E-04	3.35E+00
6	-	-	8.33E-06	1.60E+00
…	-	-	-	…

In Figure 6, the number of iterations per load step is shown. The load pattern and location of load reversal in Figure 1, coincides with where the modified N-R method had an increased number of iterations for convergence. One explanation for the increase in iterations is the change in slope of the constitutive model (for this report the Menegotto-Pinto model) which would occur when the fictitious material yields.

If there are sharp changes in the stress-strain curve, the modified algorithms perform worse since the slope to calculate the next converged point is non-optimal. Load steps with multiple or steep changes in the stress-strain curves take longer time for the modified algorithms to converge as seen in Figure 6.

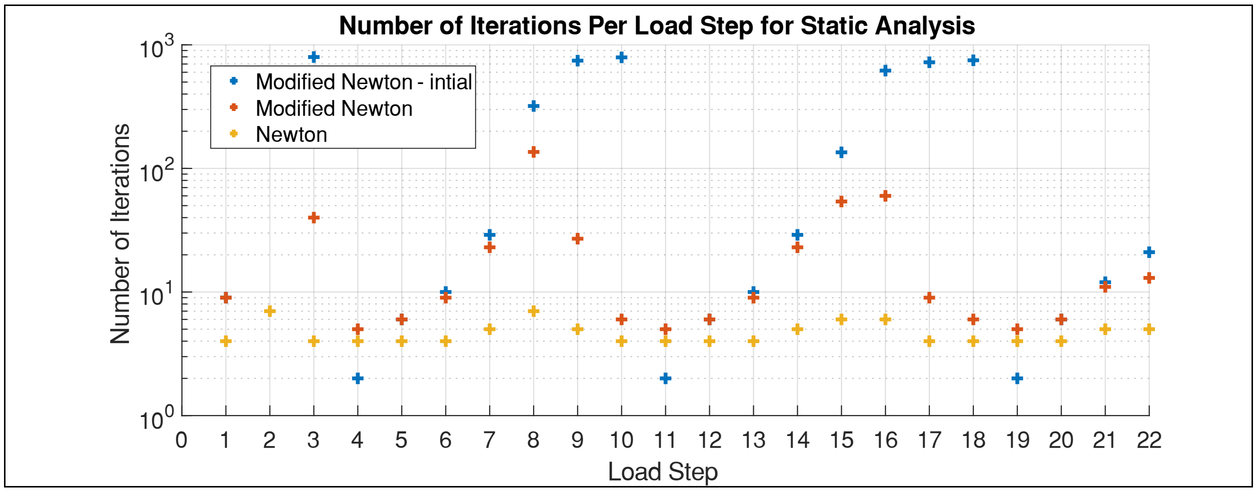

Figure 6 Number of Iterations per Load Step in the Quasi-Static Analysis

In terms of performance, each algorithm performed differently due to their rate of convergence which was dependent on the slope each chose to use. The measure of run time per algorithm was measured as the average of 500 runs of the loop solving the system of equation. The initialization of material properties and other parameters were not examined and excluded from the time measured. The storing of data was thought to have contributed to the slowdown of the algorithm and so only the storage of the displacement and force was left in the main function, to retain some functionality. For the N-R method, the mean of the runs was 1.22 milliseconds. For other modified N-R methods, this was 13.6 milliseconds and 56.7 milliseconds. This shows that the N-R method is an order of magnitude better than the modified N-R method and the modified N-R method with initial elastic stiffness. The slowdown is a result of using non-optimal slopes which increased the number of iterations necessary to converge for the modified algorithms. The modified algorithm did save computationally by not computing the tangent modulus at every iteration, but this was offset by the increase number of iterations performed overall. The usefulness of the modified algorithms is to solve the nonlinear system if the next converge point might be overshot by the normal N-R method.

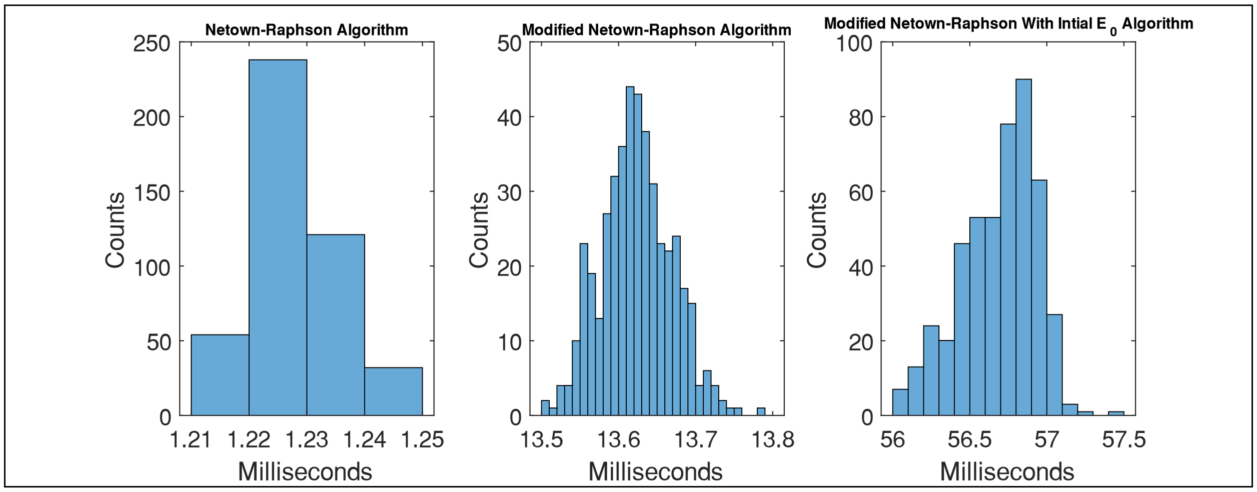

Figure 7 Histogram of Measured Time for 500 Runs of the Nonlinear Algorithms

Nonlinear Dynamic Analysis

In this portion of the report, a nonlinear time history analysis was performed for the structure for the first 20 seconds of the 1994 Northridge Earthquake (M6.7) recorded at the Sylmar Hospital station as shown in Figure 8. The Newmark-beta method was used as the time stepping scheme (using the constant average acceleration method) and the iterative N-R method was used to solve the nonlinear system of static equilibrium. The integration time step was set to 0.02 seconds which is the same as earthquake record time step. The building was assumed to have a damping ratio of 2 percent.

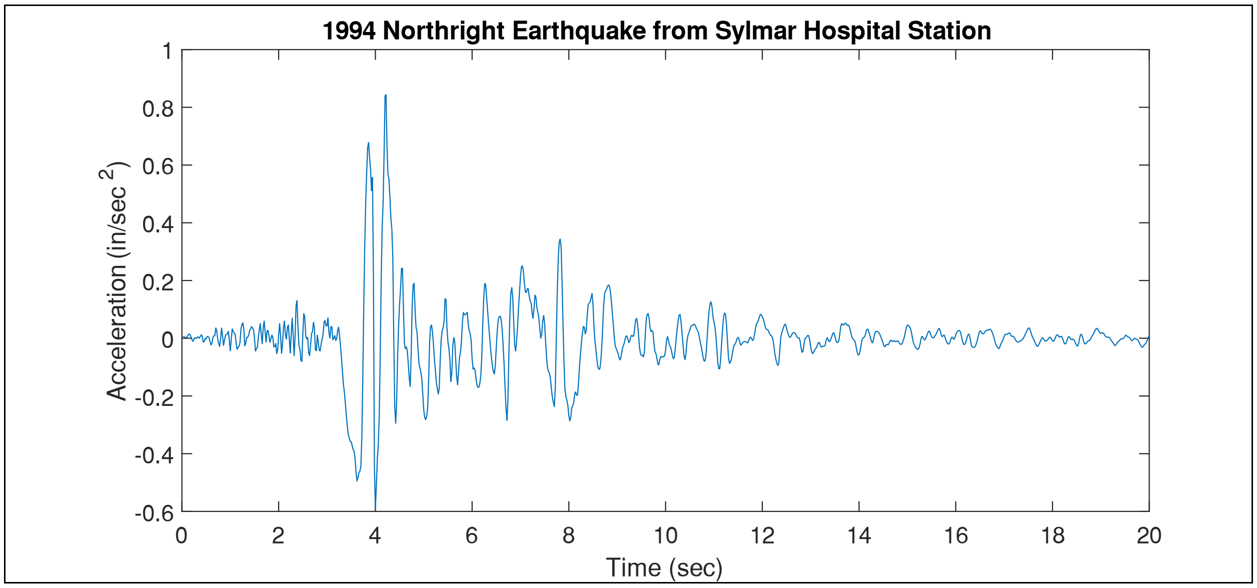

Figure 8 Number of Iterations per Load Step in the Quasi-Static Analysis

Part (f) Time History Response- Base Case

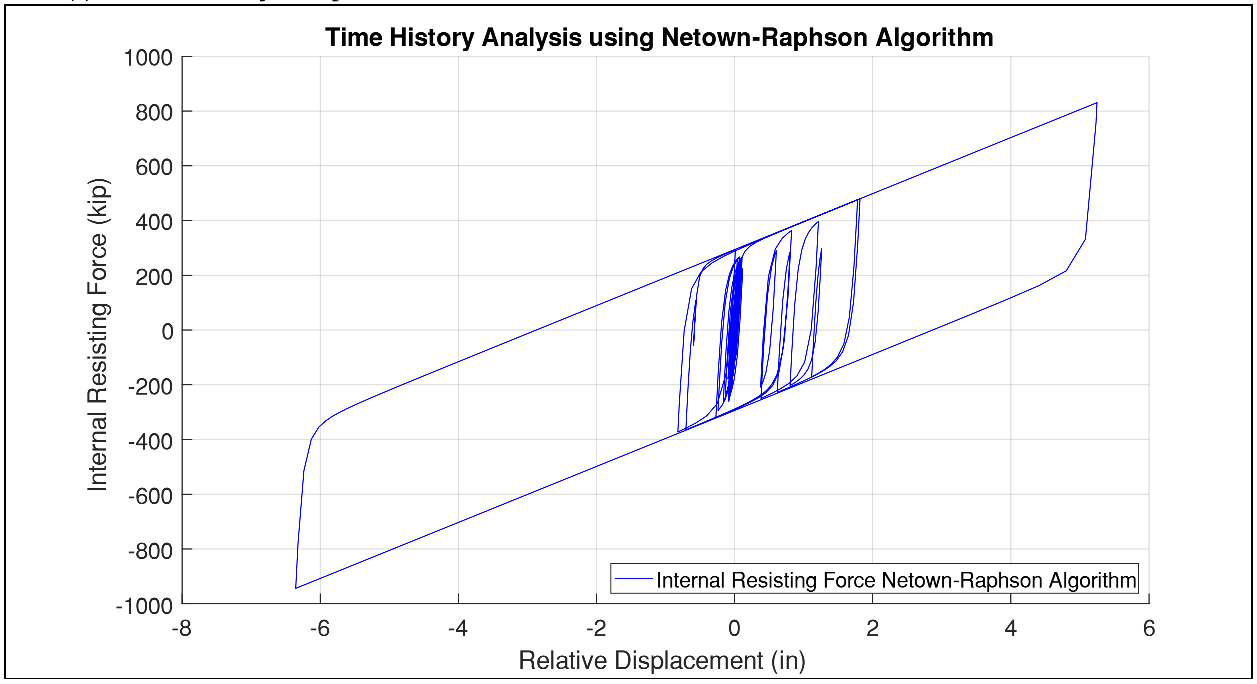

Figure 9 Time History Analysis Using N-R Method and $\Delta = 0.2s$

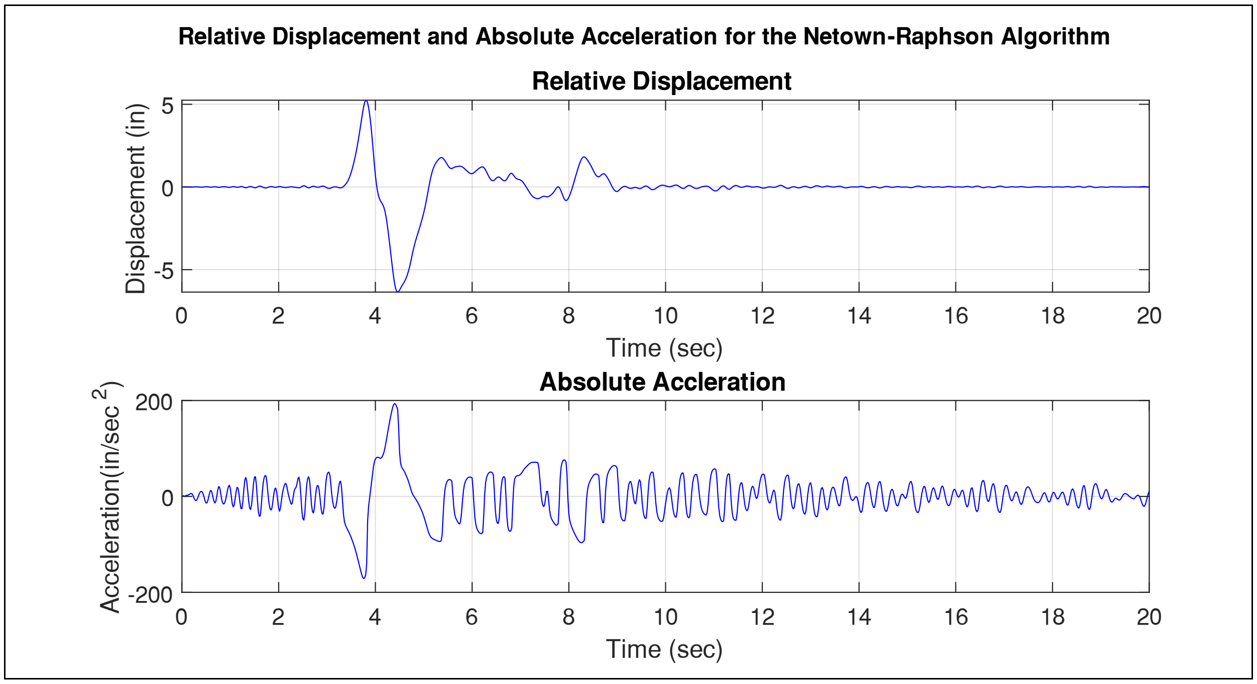

Figure 10 Relative Displacement and Absolute Acceleration for the Time History Analysis

In Figure 9 the force- relative displacement curve is plotted for the time history analysis. The maximum positive displacement was 5.25 inches at 3.80s. At that moment, the structure was experiencing 830.61 kip of internal resisting force. The maximum negative displacement was -6.36 inches at 4.46s. At that moment, the structure was experiencing -943.93 kip of internal resisting force. The maximum absolute acceleration felt by the structure was 0.50g at 3.82s.

Part (e) Analyzing the Usage of Different Algorithms on Time History Response

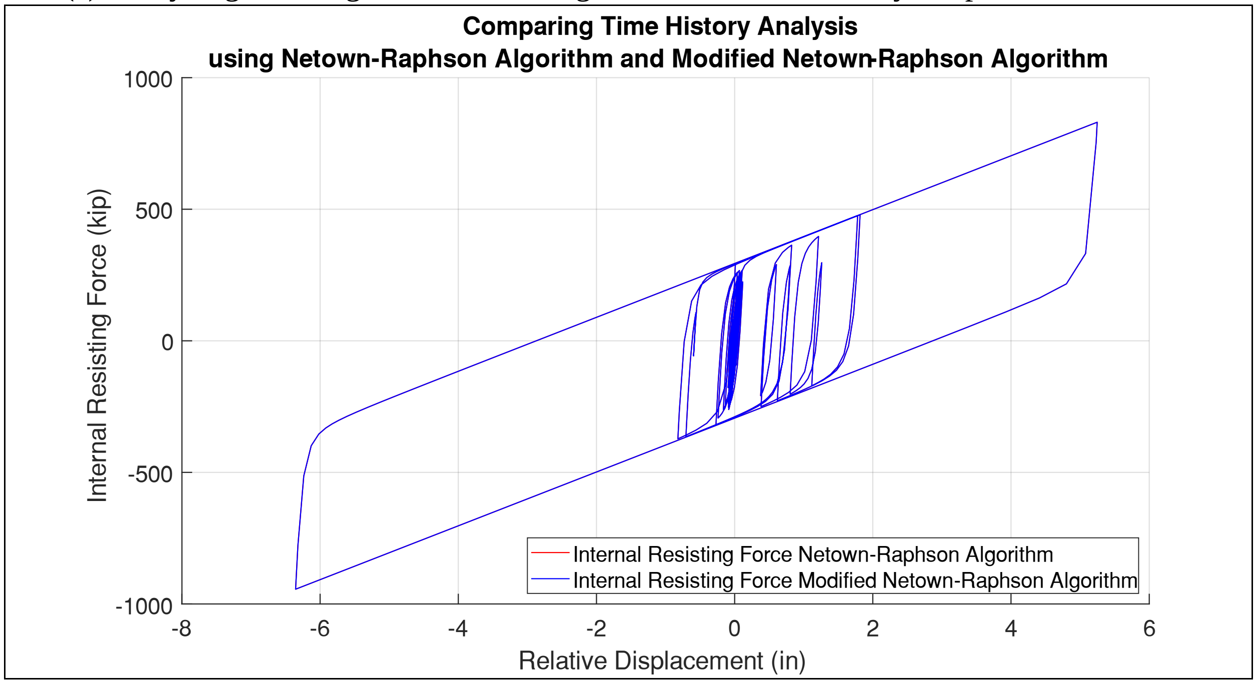

Figure 11 Comparing Time History Analysis Using N-R Method and Modified N-R Method

The force-relative displacement curve of the N-R and modified N-R method is plotted in Figure 11. It is shown that they converged to the same path. In figure 12, the relative displacement and absolute acceleration is plotted. The maximum positive displacement was 5.25 inches at 3.80 seconds. At that moment, the structure was experiencing 830.61 kip of internal resisting force. The maximum negative displacement was -6.36 inches at 4.46 seconds. At that moment, the structure was experiencing -943.93 kip of internal resisting force. The maximum absolute acceleration felt by the structure was 0.50g at 3.82 seconds.

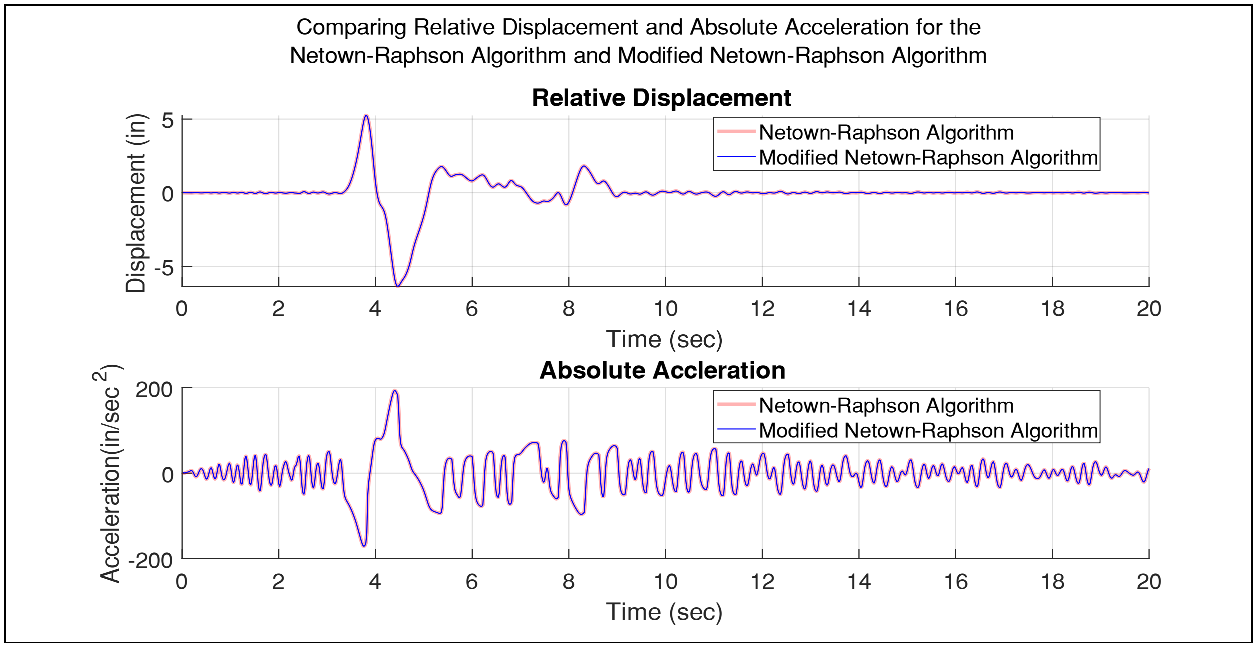

Figure 12 Comparing Relative Displacement and Absolute Acceleration in N-R and Modified N-R Methods

Table 2 shows the norm of the unbalance force for two load steps in the strong phase of the earthquake ground motion. The N-R method shows a faster than quadratic rate of convergence while the modified N-R method does not show a consistent rate of convergence. Since the time steps were equally spaced and close together, the algorithm did not take too many iterations to converge.

Table 2 Norm of the Unbalanced Force for Two Load Steps in the Dynamic Loading Analysis

	N-R Method		Modified N-R Method I	
Iteration Step	Load Step 199	Load Step 233	Load Step 199	Load Step 233
1	3.228e+04	1.941e+03	3.228e+04	1.941e+03
2	1.561e-02	1.766e+02	1.561e-02	1.766e+02
3	7.276e-12	1.506e-01	1.376e-08	1.479e+01
4	-	1.055e-07	-	1.252e+00
5	-	-	-	1.060e-01
6	-	-	-	8.964e-03
…	-	-	-	…

In Figure 13, the number of iterations needed for convergence for each load step is presented for both algorithms. Visually the N-R method takes on average less iterations to converge than the modified algorithm. The average iteration for the N-R method was 3.45 while the modified N-R method had an average of 4.86 iterations. The addition of inertial and damping terms made the system more linear and since the load steps are closely spaced, the number of iterations for both algorithms are relatively low. If Figure 13 was plotted against the acceleration time history, the peak accelerations would closely coincide with increase number of iterations.

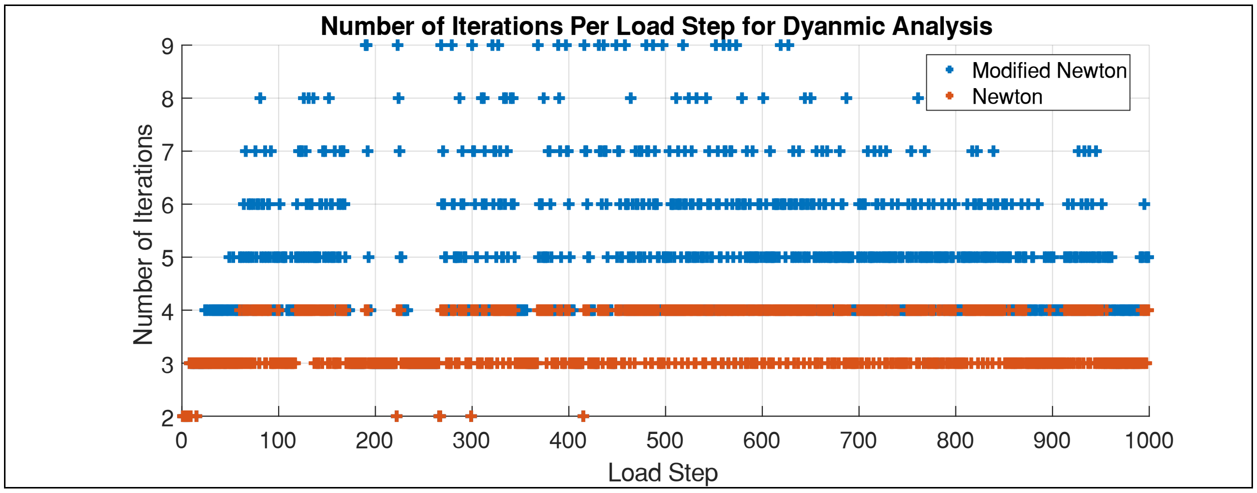

Figure 13 Number of Iterations per Load Step in the Quasi-Static Analysis

Part (f) Analyzing the Effect of Decreasing Time Step on Time History Response

The time step for the Newmark-Beta method was decreased to $\Delta t = 0.1$sec and the acceleration was linearly interpolated. The effects of decreasing the time step are minimal as the spacing of the acceleration time history was adequate to converge and did not lead to spurious branches. With increased points of integration, the force response curve is smoother without changing the overall curve, as shown in Figure 14. For $\Delta t = 0.1$sec, the average iteration was 3.05 which is lower than the $\Delta t = 0.2$sec, as the distance between two points is more linear, the algorithm can solve the system with less iterations.

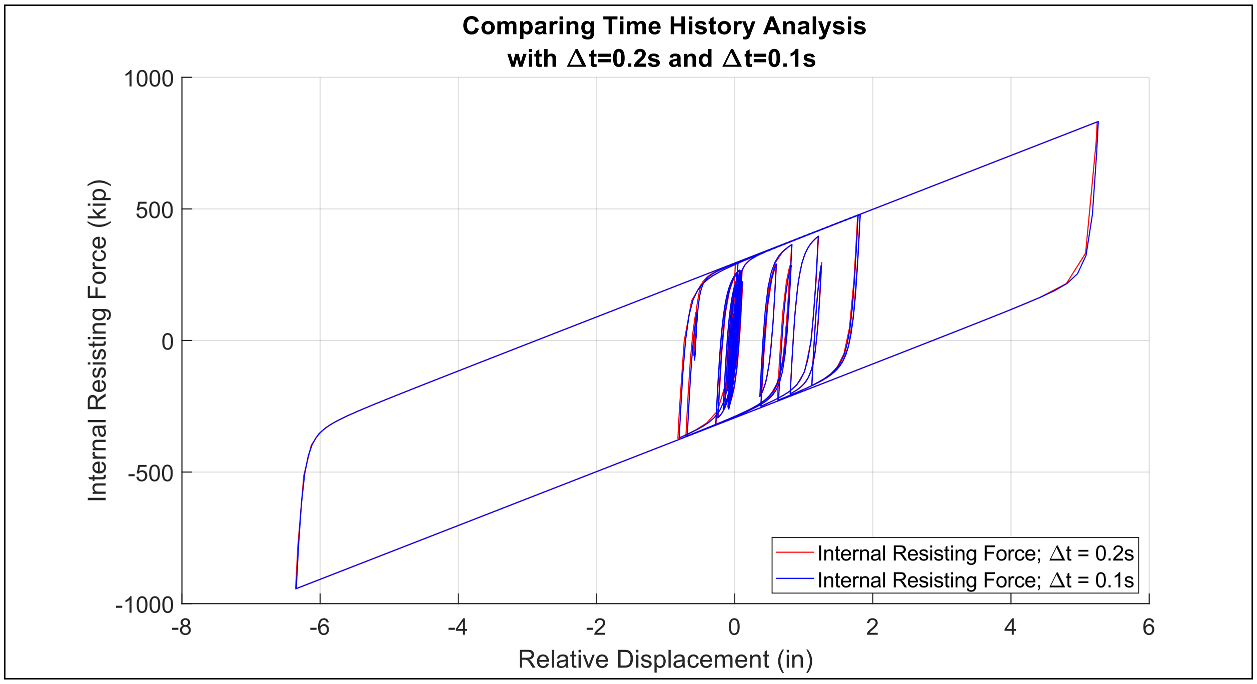

Figure 14 Number of Iterations per Load Step in the Quasi-Static Analysis

The increase precision didn't change the values of the response. The maximum positive displacement was 5.25 inches at 3.80 seconds. At that moment, the structure was experiencing 830.61 kip of internal resisting force. The maximum negative displacement was -6.36 inches at 4.46 seconds. At that moment, the structure was experiencing -943.93 kip of internal resisting force. The maximum absolute acceleration felt by the structure was 0.50g at 3.82 seconds. The average iteration for this record is 4.86. This is the same results as with $\Delta t = 0.2$sec.

Linearly interpolating between points is one method that can fill in missing time history when the samples is not clean and can help with issues of convergence. For this structure, the stability condition $\frac{\Delta t}{T_n} \leq 0.551$ is satisfied since $\frac{\Delta t}{T_n} = \frac{0.02}{0.20} = 0.1$.

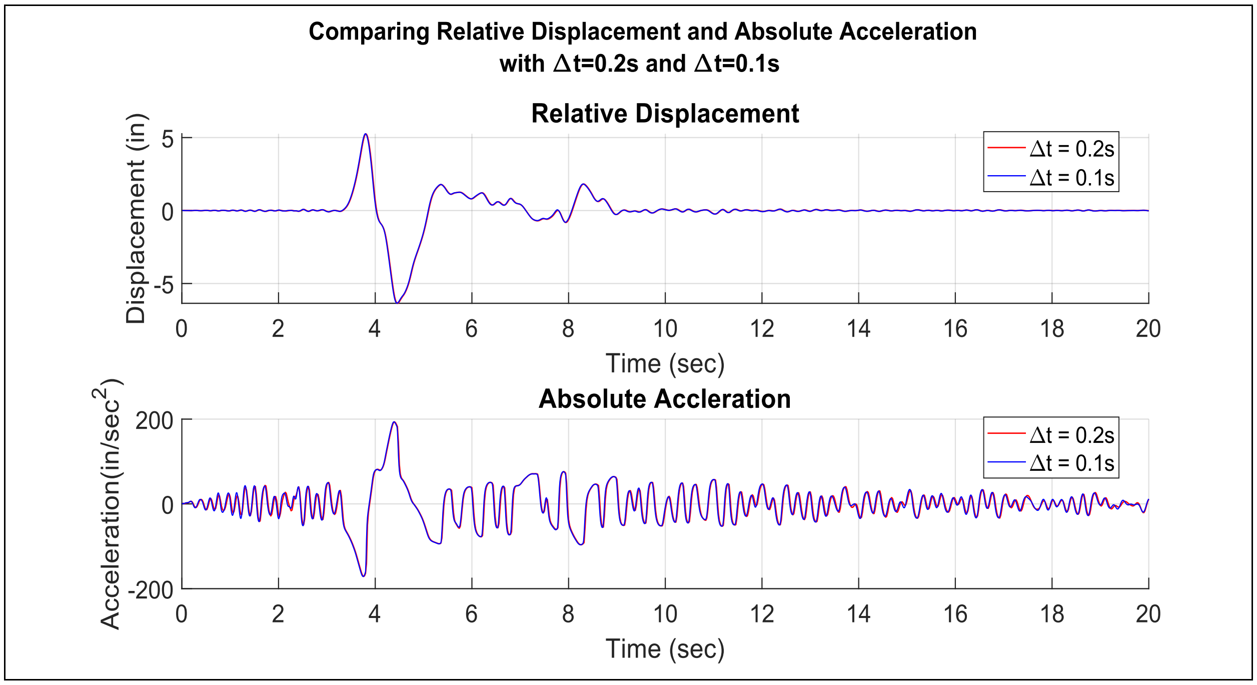

Figure 15 Comparing Relative Displacement and Absolute Acceleration using $\Delta t = [0.1\,sec, 0.2\text{ sec}]$

Part (g) Analyzing the Effects of Scaling Accelerations on Time History Response

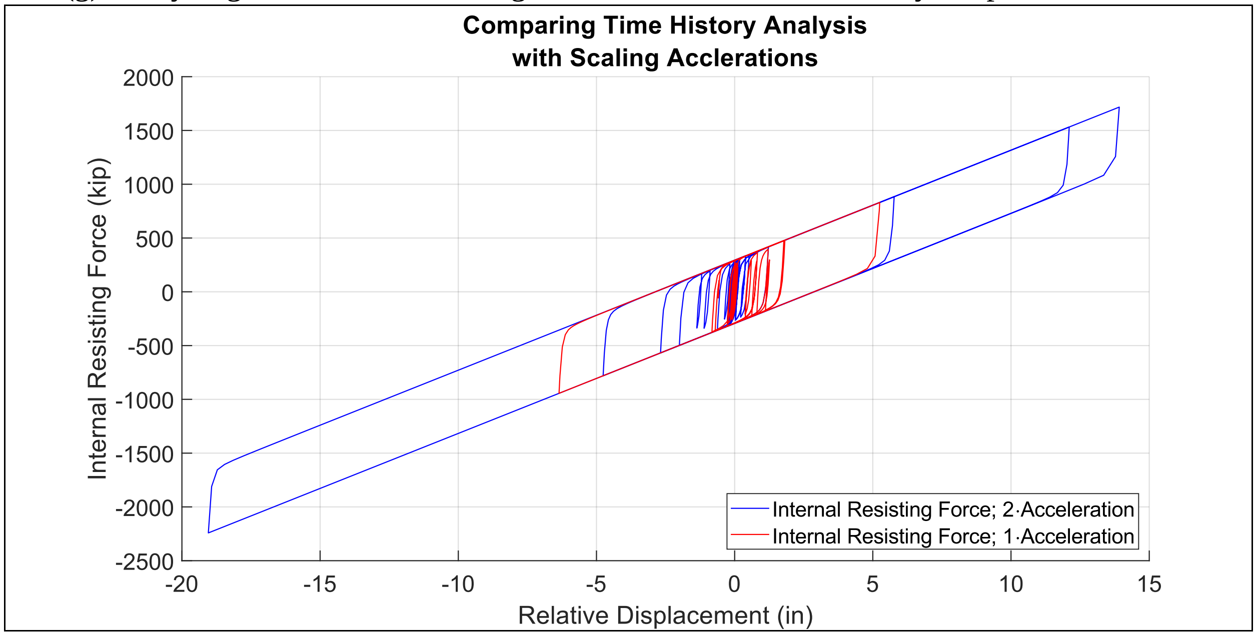

Figure 16 Comparison of Scaled Acceleration Force-Relative Displacement Response Curve

In this portion of the report, the ground motion was scaled by 2. The force-displacement hysteric response is shown in Figure 16. In Figure 17, the relative displacement and absolute acceleration is shown.

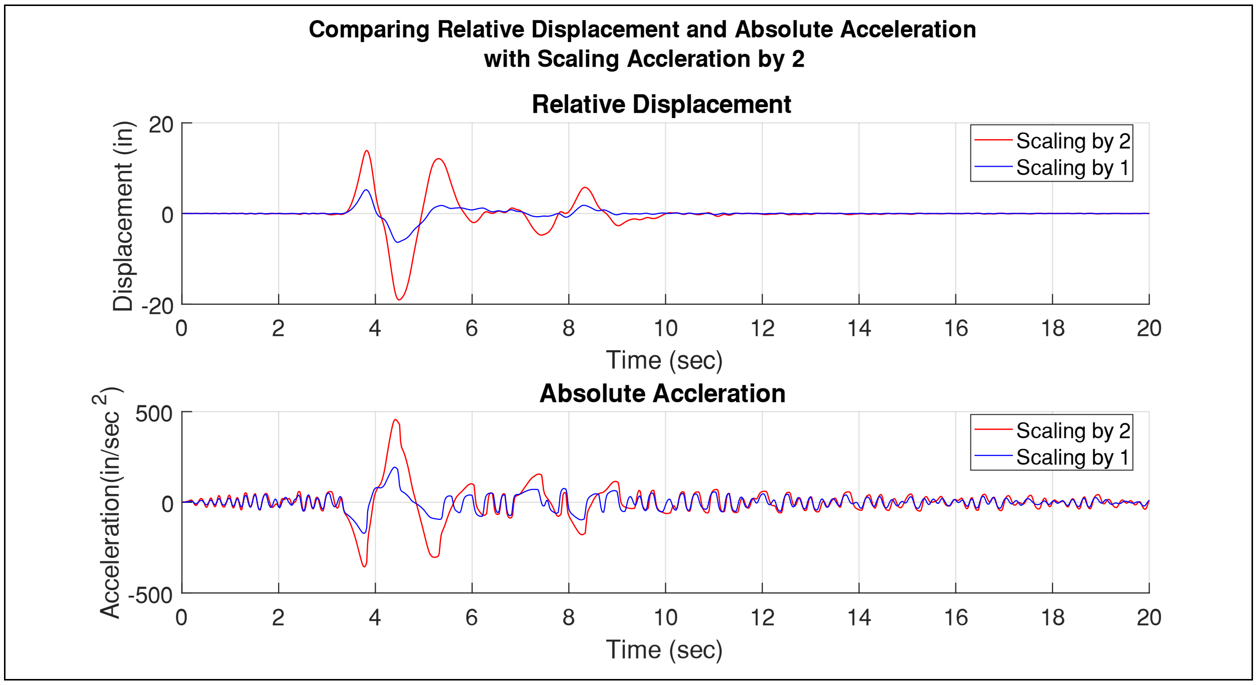

Figure 17 Comparison of Relative Displacement and Absolute Acceleration for Scaled Acceleration

The maximum positive displacement was 13.92 inches at 3.82 seconds. At that moment, the structure was experiencing 1717.42 kip of internal resisting force. The maximum negative displacement was -19.04 inches at 4.50 seconds. At that moment, the structure was experiencing -2241.78 kip of internal resisting force. The maximum absolute acceleration felt by the structure was 2.00g at 3.82 second.

By scaling the acceleration by two, the internal resisting forces and relative acceleration of the structure were also scaled up by two times, however the displacement was scaled up by three. This shows for nonlinear systems, the scaling of the acceleration is not a 1:1 to the structural response. Let record 1 be the response of the original acceleration time history and record 2 be the scaled acceleration time history. Record 1 had a maximum displacement of 6.36 inches at 4.46 seconds. Record 2 had a maximum displacement of 19.04 inches at 4.50 seconds. The ratio between records 2 to 1 is 3.00. The structure in Record 1 resisted a maximum force of -943.93 kips. The structure in Record 2 resisted a maximum force of -2241.78 kips. The ratio between records 2 to 1 is 2.37. Record 1 had a maximum acceleration of 0.50g Record 2 had a maximum acceleration of 1.18g. The ratio between records 2 to 1 is 2.36. A similar analysis can be performed for the linear elastic system. This is tabulated in Table 3.

Table 3 Ratio of Response Variables for Nonlinear and Linear Elastic Systems

	Nonlinear			Linear Elastic		
Acceleration Factors	Factor=1	Factor=2	Ratio	Factor=1	Factor=2	Ratio
Max. Relative Displacement (in)	6.36	19.04	**3.00**	0.51	1.03	**2.00**
Max. Internal Force (kip)	-943.93	-2241.78	**2.37**	-2616.99	-5233.98	**2.00**
Max. Relative Acceleration (g)	0.50	1.18	**2.36**	1.31	2.62	**2.00**

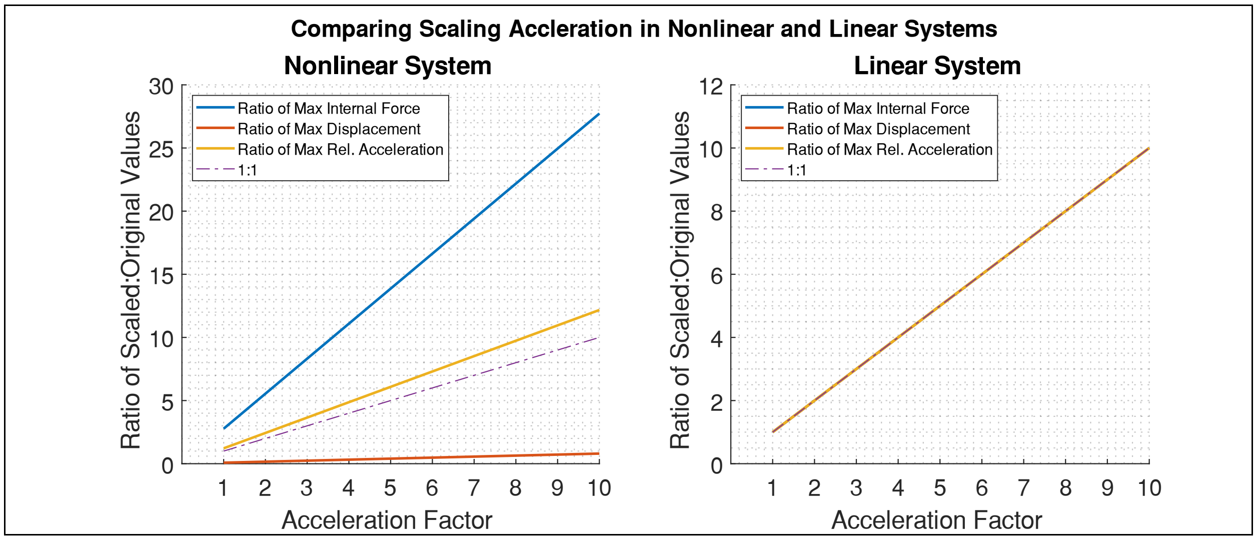

Figure 18 Ratio of Various Parameter as Acceleration is scaled in a Nonlinear and Linear Elastic System

In Figure 18, the ratio for the maximum relative displacement, maximum internal force, and maximum relative acceleration for various acceleration factors is shown. If the structure was perfectly elastic, the scaling of the acceleration in relation to these parameters are linear. Designing an elastic building will increase the amount of force the building has to resist dramatically as the other components (inertia and damping) resist less of the force since the displacement and velocity are lower. As opposed to a system that is able to yield, the maximum internal force increases around 1:1 time the scale of the acceleration.

Part (h) Analyzing the Effects of Linearizing the System on the Time History Response

If the structure is idealized as linear-elastic, the force-relative displacement curve is given in Figure 19. The curve is a straight line because the material fiber is assumed to never yield and always remain elastic. Note the linear elastic system is parallel with the initial elastic region of the nonlinear system.

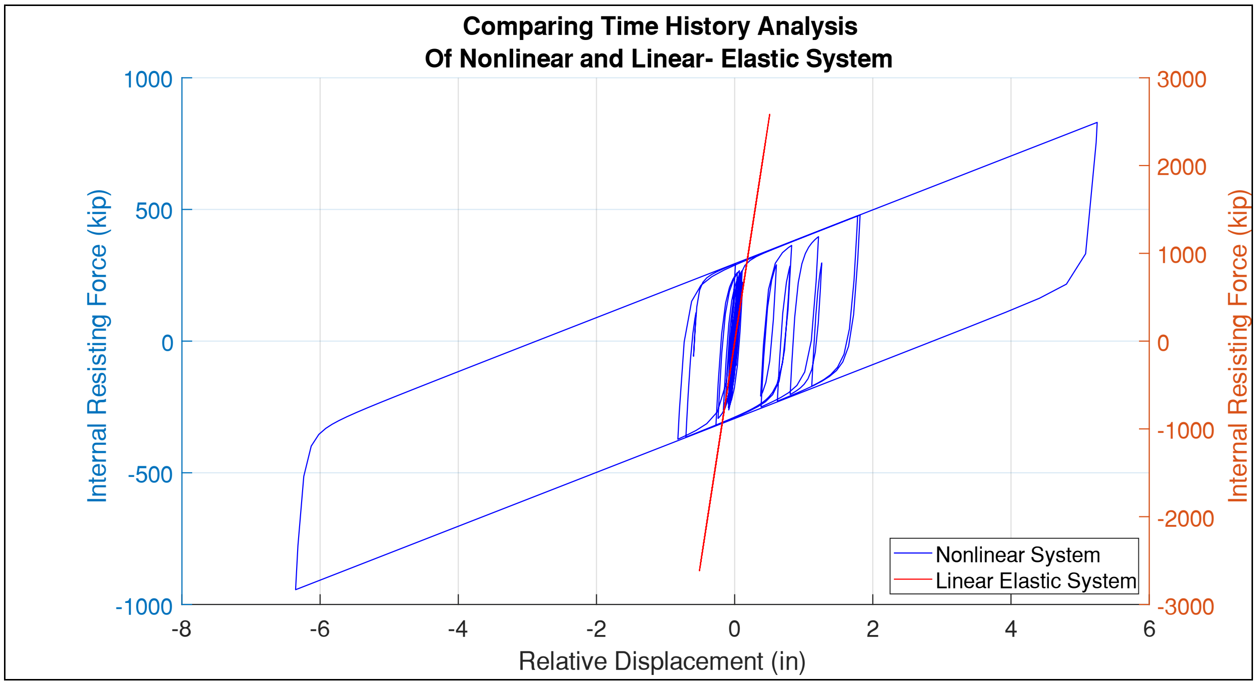

Figure 19 Comparison of Nonlinear and Elastic System Force-Relative Displacement Response Curve

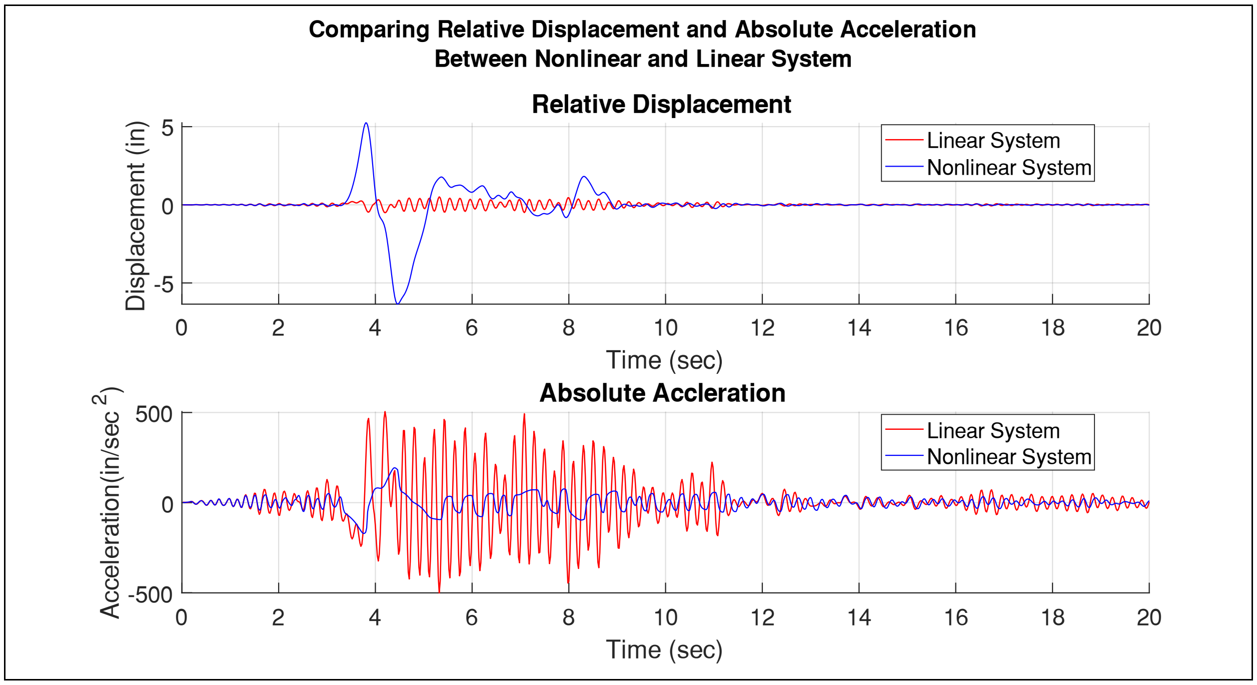

Figure 20 Comparison of Relative Displacement and Absolute Acceleration for Linear Elastic System

The linear-elastic structure has a significant decreased in relative displacement. The linear elastic system had a maximum positive displacement of 0.50 inches at 5.32 seconds. At that moment, the structure was experiencing 2537.35 kip of internal resisting force. The maximum negative displacement was -0.51 inches at 4.20 seconds. At that moment, the structure was experiencing -2613.18 kip of internal resisting force. The maximum absolute acceleration felt by the structure was 1.11g at 5.42 seconds.

The ratio of the peak elastic strength of the elastic system over the original yield strength is 8.71. This is calculated as the maximum absolute internal resisting force of 2613.2 kips over the yield strength 300 kips. By inspection, the acceleration seems to last longer than the ones from before, Figure 20. It seems that there is little damping or inertia contribution to the internal resistance, so the acceleration seems to perpetuate longer.

Problem 3 – Results from Opensees

In order to compare the results of the report with more developed software, Opensees was used. In Figure 21, the result of the pushover analysis is shown. The average error between the displacement and force from the Opensees analysis and the displacement from the MATLAB code was 0.1%. The true force displacement curve is different than the Menegotto-Pinto curve that was found in Figure 1. This might be that Opensees uses a more actual Uniaxial model than the one being implemented. The curve is within the linearized pushover curve as where the Menegotto-Pinto curve is outside the linear curve in the MATLAB analysis.

The initial values as found in part 1 was kept the same. The yield stress was 29.332 ksi and the cross sectional area was $10.217\ \text{in}^2$.

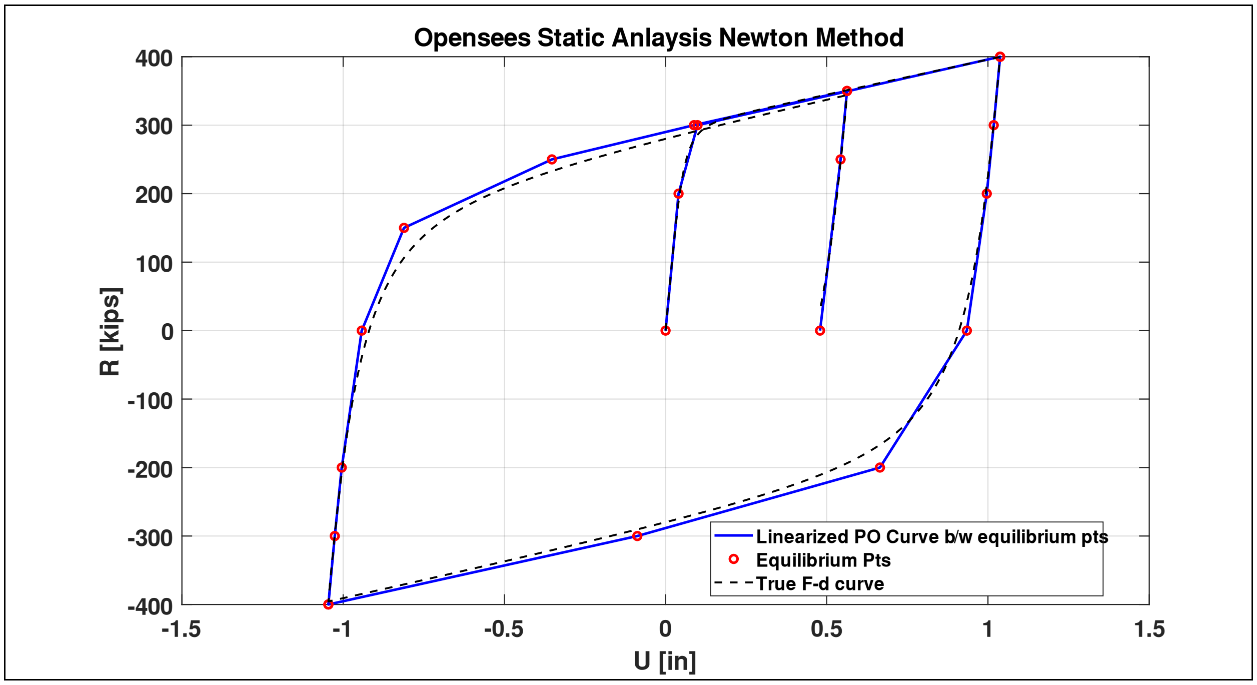

Figure 21 Opensees Result of the Pushover Analysis

In Figure 22 and 23, the results of the time history analysis using Opensees of the Northridge Earthquake are shown. As shown in Figure 23, the maximum relative displacement is 6.262 in, the maximum relative velocity is $35.791\frac{in}{s}$, and maximum acceleration was $193.753\frac{\text{in}}{\text{s}^2}$. For the MATLAB analysis, the run resulted had a maximum relative displacement is 6.355 in, the maximum relative velocity is $35.754\frac{in}{s}$, and maximum acceleration was $193.552\frac{\text{in}}{\text{s}^2}$. These results varied only in the thousandths place meaning that the precision of the MATLAB code and the algorithm are comparable in solving SDOF nonlinear systems under dynamic loading.

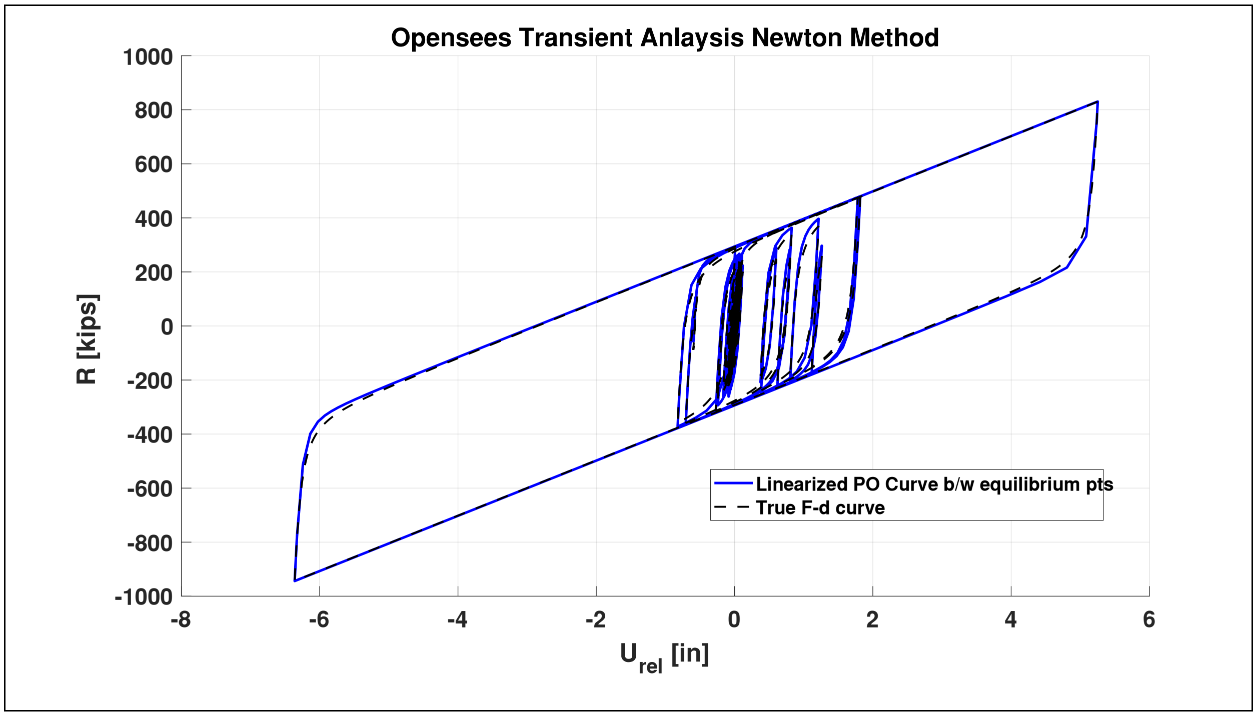

Figure 22 Opensees Result of Transient Analysis

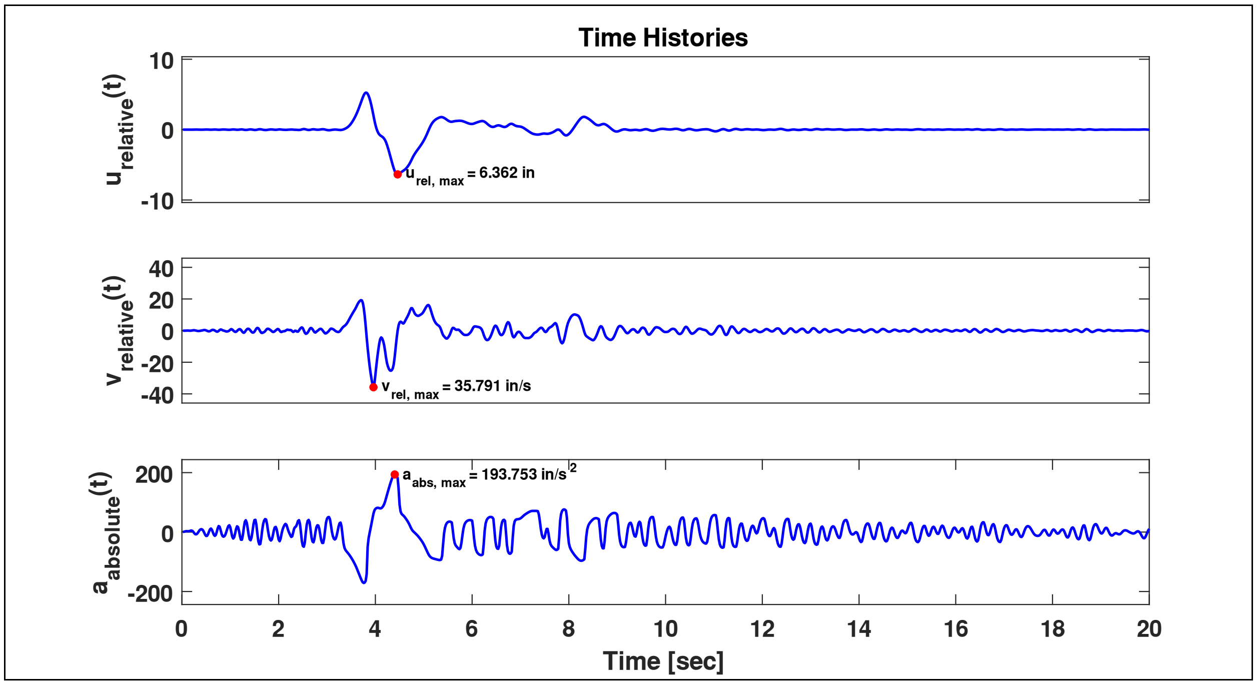

Figure 23 Opensees Results of Absolute Acceleration Relative Velocity, and Relative Displacement

Appendix A. Pseudo Code

Nonlinear Quasi-Static Analysis of a SDOF structure- Using Newton-Raphson Method

```
Step 1.  Initialize Material Data
Step 2.  Initialize Material State
Step 3.  Initialize Force vector $P$
Step 4.  For all load steps
    a)  Do while not converged and iteration < max iterations
        i)   Material state determination
        ii)  Calculate internal force: $R\left(U_{n+1}^i\right) = \sigma_{n+1}^i \cdot A$
        iii) Calculate unbalance force: $\Psi_{n+1}^i = P_{n+1} - R\left(U_{n+1}^i\right)$
        iv)  If unbalanced force met convergence criterion $|\Psi_{n+1}^i| \leq 10^{-5}$
            (1) Commit $U_{n+1}^i$ as $U_{n+1}$
            (2) Set $\Delta U_{n+1}^i = 0$ and material state $\Delta\epsilon_{n+1}^i = 0$
            (3) continue
        v)   Else
            (1) Calculate the tangent stiffness $K_T\left(U_{n+1}^i\right) = \frac{E_T A}{L}$
            (2) Calculate change in displacement as $\delta U_{n+1}^i = \Psi(U_{n+1}^i)/K_T$
            (3) Update the change in displacement from last converge displacement$\Delta U_{n+1}^i = \Delta U_{n+1}^i + \delta U_{n+1}^i$
            (4) Update the total displacement $U_{n+1}^{i+1} = U_{n+1} + \delta U_{n+1}^i$
```

Nonlinear Dynamic Analysis of a SDOF structure Using Newmark-Beta Time-Stepping Method and Newton-Raphson Method

```
Step 1.  Initialize Material Data
Step 2.  Initialize Material State
Step 3.  Initialize acceleration; Convert Acceleration to Force
Step 4.  For all load steps
    a)  Calculate the known internal resisting force
        $\tilde{P}_{n+1} = P_{n+1} + m\left[\frac{1}{\beta(\Delta t)^2}U_n + \frac{1}{\beta(\Delta t)}\dot{U}_n - \left(1-\frac{1}{2\beta}\right)\ddot{U}_n\right] + c\left[\frac{\alpha}{\beta(\Delta t)}U_n - \left(1-\frac{\alpha}{\beta}\right)\dot{U}_n - \Delta t\left(1-\frac{\alpha}{2\beta}\right)\ddot{U}_n\right]$
    b)  Do while not converged and iteration < max iteration
        i)   Material state determination
        ii)  Calculate the internal force: $R\left(U_{n+1}^i\right) = \sigma_{n+1}^i \cdot A$
        iii) Calculate the unbalance force: $\Psi(U_{n+1}^i) = \tilde{P}_{n+1} + \left[\frac{1}{\beta(\Delta t)^2}mU_{n+1}^i + \frac{\alpha}{\beta(\Delta t)}cU_{n+1}^i + R\left(U_{n+1}^i\right)\right]$
        iv)  If unbalanced force met convergence criterion $|\Psi_{n+1}^i| \leq 10^{-5}$
            (1) Commit $U_{n+1}^i$ as $U_{n+1}$;
            (2) Commit $\dot{U}_{n+1}$= $\Delta t\left(1-\frac{\alpha}{2\beta}\right)\ddot{U}_n + \left(1-\frac{\alpha}{\beta}\right)\dot{U}_n + \frac{\alpha}{\beta(\Delta t)}(U_{n+1} - U_n)$
            (3) Commit $\ddot{U}_{n+1} = \left(1-\frac{1}{2\beta}\right)\ddot{U}_n - \frac{1}{\beta(\Delta t)}\dot{U}_n + \frac{1}{\beta(\Delta t)^2}(U_{n+1} - U_n)$
            (4) Set $\Delta U_{n+1}^i = 0$ and material state $\Delta\epsilon_{n+1}^i = 0$
            (5) continue
        v)   Else
            (1) Calculate the tangent stiffness $K_T\left(U_{n+1}^i\right) = \frac{E_T A}{L}$
            (2) Calculate the dynamic stiffness $K_T^{dyn} = \frac{1}{\beta(\Delta t)^2}m + \frac{\alpha}{\beta(\Delta t)}c + K_T\left(U_{n+1}^i\right)$
            (3) Calculate change in displacement $\delta U_{n+1}^i = \Psi(U_{n+1}^i)/K_T^{dyn}$
            (4) Update the change in displacement from last converge$\Delta U_{n+1}^i = \Delta U_{n+1}^i + \delta U_{n+1}^i$
            (5) Update total displacement $U_{n+1}^{i+1} = U_{n+1} + \delta U_{n+1}^i$
```

Appendix B. MATLAB Files

```
%% Initialize the system
% UNITS:  kips, inches
addpath('.\functions') % Loads in all of the functions
clear; clc;

% Define Structure Parameters as given
mg = 2000; % kips; weight of the sytem
T0 = 0.2;  % seconds; natural period of the system
Ry0 = 0.15*mg; % kip; yeild force of the system
E0 = 30000; % ksi; Elastic modulus
g = 386; % in/s^2; gravitational constant
L = 60; % in; arbitrary length

[K0, A, s_y0] = eqivalent_truss(mg, T0, Ry0, L, E0, g); % Finds the equivalent parameter of a equivalent truss for the SDOF
% MatData = Initialize_MatData(E0, s_y0, b, R0, cR1, cR2, a1, a2, L, A)
MatData = Initialize_MatData(E0, s_y0, 0.02, 5, 3, 0.15, 0, 0, L, A); % Initialize the material data
% Initialize_Material_State(sig, Et, epmin, epmax,epex, ep0, s0, epr, sr, kon, [initial_strains])
MatState = Initialize_Material_State(0, MatData.E, MatData.fy, -MatData.ey, MatData.ey, 0, MatData.ey, 0, 0, 0, [0 ,0, 0 ,0]); % Initialize the ↙
material State

%% Nonlinear SDOF System Subjected to Quasi-Static Cyclic Loading
P = load('P.txt');% Load Forces
record_static1 = Static(P, MatData,MatState,"Newton", 10); % Run static analysis with Newton Method
record_static2 = Static(P, MatData,MatState,"ModifiedNewton", 200); % Run static analysis with modified Newton Method
record_static3 = Static(P, MatData,MatState,"ModifiedNewton -initial", 800); % Run static analysis with modified Newton Method

%% Nonlinear SDOF System Subjected to Dynamic Loading
% Newmark Beta
time_step_method.alpha = 0.5;
time_step_method.beta = 0.25;
time_step_method.time_step = 0.02; % sec
% Define Structural Properties for Dynamic Analysis
MatData.mass = mg/g; % mass of the system
xi = 0.02; % Damping coefficient
MatData.damping =  2*xi*sqrt(K0*mg/g); % damping of the system
% Accelerations
[~, acc] = readvars(".\SYL360.txt");  % Loads in time and acceleration
acc= acc(2:end)*g; % Removes the first zero
record_Trans1 = Transient(acc, MatData, MatState, time_step_method,"Newton" , 10);
record_Trans2 = Transient(acc, MatData, MatState, time_step_method,"ModifiedNewton" , 10);
record_Trans4 = Transient(2*acc, MatData, MatState, time_step_method,"Newton" , 10); % Scaling the acceleration by 2

%% Nonlinear SDOF System Subjected to Dynamic Loading
% Decreasing the Time Step
time_step_method.time_step = 0.01; % sec
acc = interp(acc,2); % Linearly interpolate the data
record_Trans3 = Transient(acc, MatData, MatState, time_step_method,"Newton" , 10);

%% Nonlinear SDOF System Subjected to Dynamic Loading
% Linear System
time_step_method.time_step = 0.02; % sec
[~, acc] = readvars(".\SYL360.txt");  % Loads in time and acceleration
acc= acc(2:end)*g; % Resets the accelerations
MatData.fy = 10000*MatData.fy; % Sets the yield stress such that it remains elastic
record_Trans5 = Transient(acc, MatData, MatState, time_step_method,"Newton" , 10); % Linear Elastic System
record_Trans6 = Transient(2*acc, MatData, MatState, time_step_method,"Newton" , 10); % Linear Elastic System
```

```
%% Plot Everything
% Short Hand Notations
close all; clc;

force = "Applied Force (kip)";
U = "Relative Displacement (in)";
iter = "Number of Iterations";
LS = "Load Steps (n)";
T = "Time (sec)";
A = "Acceleration (in/sec^2)";

analy1 = "Quasi-Static Cyclic Loading Pushover Analysis";
analy2 = "Time History Analysis";

algo1 = "Netown-Raphson Algorithm";
algo2 = "Modified Netown-Raphson Algorithm";
algo3 = "Modified Netown-Raphson With Intial E_0 Algorithm";

R = "Internal Resisting Force";
UnBaFo = "Unbalanced Force";
MP = "Megenetto-Pinto Uniaxial Model";
plot_size = [1200, 0, 1300,700];

%% Figure 1 - Force Pushover Curve
close all;
P = load('P.txt');
name = "Force Used in Pushover Analysis";
plot(P);
title(name); xlabel(LS); ylabel(force); grid on;
xticks(1:23); xlim([1,23]);
set(gcf, 'Position',  [1200, 0, 1300,500])
print_file(1, name)
%% Figure 2 -Quasi Static Cyclic Loading Using the Newton Algorithm
close;
name = [analy1, "using "+algo1];
fig = 2;
grid on; hold on;
plot_MenegottoPinto(record_static1);
plot_iterations(record_static1);
plot_RU_curve(record_static1, name);
set(gcf, 'Position',  plot_size)
set(gca,"FontSize",18)
print_file(fig, name)
%% Figure 3 -Quasi Static Cyclic Loading Using the Modified Newton Algorithm
close;
fig = 3;
name = [analy1, "using "+algo2];
grid on; hold on;
plot_MenegottoPinto(record_static2);
plot_iterations(record_static2);
plot_RU_curve(record_static2, name);
set(gcf, 'Position',  plot_size)
print_file(fig, name)
%% Figure 4 -Quasi Static Cyclic Loading Using the Modified Newton Algorithm -Inital
close;
name = [analy1, "using "+algo3];
fig = 4;
```

```
grid on; hold on;
plot_MenegottoPinto(record_static3);
plot_iterations(record_static3);
plot_RU_curve(record_static3, name);
set(gcf, 'Position',  plot_size)
print_file(fig, name)
%% Figure 5 -Unbalanecd Force Per Iterations
close all;
fig = 5;
subplot(1,2,1); hold on;
name = [UnBaFo+" Per Iteration","Using " + algo1];
for i = 1:size(record_static1.Unb,1)
    Unb = record_static1.Unb(i,:);
    Unb = Unb(Unb ~=0);
    plot(i*ones(numel(Unb)), Unb,'r-x');
end
title(name); set(gca, 'YScale', 'log','xminorgrid','on','yminorgrid','on'); ylabel("Unbalanced Force (kip)"); xlabel("Load Step (n)"); xlim([0,22]);
subplot(1,2,2); hold on;
name = [UnBaFo+" Per Iteration","Using " + algo2];
for i = 1:size(record_static2.Unb,1)
    Unb = record_static2.Unb(i,:);
    Unb = Unb(Unb ~=0);
    if length(Unb) >= 100; step = 50; else; step = 1; end;
    Unb = Unb(1:step:end);
    plot(i*ones(numel(Unb)), Unb,'r-x');
end
set(gca, 'YScale', 'log','xminorgrid','on','yminorgrid','on'); ylabel("Unbalanced Force (kip)"); xlabel("Load Step (n)");xlim([0,22]);
title(name);
set(gcf, 'Position',  plot_size)
%
print_file(fig, "Unbalanced Force Per Iteration")
%% Figure 6 - Number of Iterations Per Load Step for Static Analysis
close all;
fig = 6;
name = "Number of Iterations Per Load Step for Static Analysis";
title(name); hold on; grid on;
scatter(1:length(record_static3.iter),record_static3.iter,70,'+',"LineWidth",3,'DisplayName',"Modified Newton - intial");
scatter(1:length(record_static2.iter),record_static2.iter,70,'+',"LineWidth",3,'DisplayName',"Modified Newton");
scatter(1:length(record_static1.iter),record_static1.iter,70,'+',"LineWidth",3,'DisplayName',"Newton");
set(gca, 'YScale', 'log'); legend("location","best");
ylabel("Number of Iterations"); xlabel("Load Step"); xlim([0,22]); xticks(0:22)
set(gcf, 'Position',  [1200, 0, 1300,500])
print_file(fig, name)
%% Figure 7 - Milliseconds to Run Each Algorithm
n = 500;
[T1, T2, T3] = deal(zeros(1,n));
for i = 1:n
    tic
    ans = Static_no_record(P, MatData,MatState,"Newton", 10);
    T1(i) = toc;
end
%
for i = 1:n
    disp("Running Modified Newton. On Iteration " + i)
    tic
    ans = Static_no_record(P, MatData,MatState,"ModifiedNewton", 200);
    T2(i) = toc;
```

```matlab
end
%
for i = 1:n
    disp("Running Modified Newton - initial. On Iteration " + i)
    tic
    ans = Static_no_record(P, MatData,MatState,"ModifiedNewton -initial", 800);
    T3(i) = toc;
end
close all; figure;
fig = 7;
name = "Milliseconds to run each Algorithm";

subplot(1,3,1);
histogram(T1*10^3,"BinWidth",0.01,"BinLimit",[1.21,1.25]);
xlabel("Milliseconds"); ylabel("Counts");title(algo1,'FontSize',12);
subplot(1,3,2);
histogram(T2*10^3,"BinWidth",0.01,"BinLimit",[13.5,13.8]);
xlabel("Milliseconds"); ylabel("Counts");title(algo2,'FontSize',12);
subplot(1,3,3);
histogram(T3*10^3,"BinWidth",0.1,"BinLimit",[56,57.5]);
xlabel("Milliseconds"); ylabel("Counts"); title(algo3,'FontSize',12);
set(gcf, 'Position',  [1200, 0, 1300,500])
print_file(fig, name)
%% Figure 8 -1994 Northright Earthquake from Sylmar Hospital Station
close;
name = "1994 Northright Earthquake from Sylmar Hospital Station";
fig = 8;
[time, acc] = readvars(".\SYL360.txt");  % Loads in time and acceleration
acc= acc(2:end); time = time(2:end);
plot(time,acc);
title(name); xlabel(T);ylabel(A);
set(gcf, 'Position',  [1200, 0, 1300,600]);
print_file(fig, name)
%% Figure 9 -Dyanmic Loading Using the Newton Algorithm
close;
name = ["Comparing " + analy2, "using "+algo1+" and "+algo2];
fig = 9;
title(name); grid on; hold on;
p = plot_RU_curve(record_Trans1, R+" "+ algo1); p.Color = 'r';
plot_RU_curve(record_Trans2, R+" "+ algo2);
set(gcf, 'Position',  plot_size)
print_file(fig, name)
%% Figure 10 -Dyanmic Loading Relative Displacement and Absolute Acceleration
close all; clc
name = ["Comparing Relative Displacement and Absolute Acceleration for the", strjoin([algo1,algo2]," and ")];
sgtitle((name),"FontSize",18);
fig = 10;
plot_disp_acc(record_Trans1,record_Trans2,[algo1, algo2]);
set(gcf, 'Position',  plot_size)
print_file(fig,strjoin(name))
%% Figure 11- Number of Iterations Per Load Step For Dynamic Anlaysis
close all;
fig = 11;
name = "Number of Iterations Per Load Step for Dyanmic Analysis";
title(name); hold on; grid on;
scatter(1:length(record_Trans2.iter),record_Trans2.iter,70,'+',"LineWidth",3,'DisplayName',"Modified Newton");
scatter(1:length(record_Trans1.iter),record_Trans1.iter,70,'+',"LineWidth",3,'DisplayName',"Newton");
```

```
legend("location","best");
ylabel("Number of Iterations"); xlabel("Load Step");
set(gcf, 'Position',  [1200, 0, 1300,500])
print_file(fig, name)
%% Figure 12 -Comparing Force History Hisory Using  Delta = 0.1
close all; clc
name = ["Comparing " + analy2, "with \Deltat=0.2s and \Deltat=0.1s "];
fig = 12;
title(name); hold on; grid on;
p = plot_RU_curve(record_Trans1, R+"; \Deltat = 0.2s"); set(p,'Color','r');
plot_RU_curve(record_Trans3, R+"; \Deltat = 0.1s");
set(gcf, 'Position',  plot_size)
% print_file(fig, "Comparing Changes in Time Step")
%% Figure 13 -Dyanmic Loading Relative Displacement and Absolute Acceleration
close all; clc; hold on;
name = ["Comparing Relative Displacement and Absolute Acceleration" ,"with \Deltat=0.2s and \Deltat=0.1s "];
t = sgtitle((name),"FontSize",18); t.FontWeight =' bold';
fig = 13;
plot_disp_acc(record_Trans1, record_Trans3,["\Deltat = 0.2s", "\Deltat = 0.1s"]);
set(gcf, 'Position',  [1200, 0, 1300,650])
print_file(fig,"Comparing Displacment Delta ")
%% Figure 14 -Comparing Force Hisory with Acceleration*2
name = ["Comparing " + analy2, "with Scaling Accleration by 2"];
close all; clc; grid on; hold on; title(name)
fig = 14;
plot_RU_curve(record_Trans4,  R+"; 2\cdotAcceleration");
p = plot_RU_curve(record_Trans1, R+"; 1\cdotAcceleration"); set(p,'Color','r');
set(gcf, 'Position',  [1200, 0, 1300,650])
% print_file(fig, name)
%% Figure 15 -Comparing Relative Displacement and Absolute Acceleration with Scaling Acceleration by 2
close all; clc; hold on;
name = ["Comparing Relative Displacement and Absolute Acceleration" ,"with Scaling Accleration by 2"];
t = sgtitle((name),"FontSize",18); t.FontWeight =' bold';
fig = 15;
plot_disp_acc(record_Trans4, record_Trans1,["Scaling by 2", "Scaling by 1"]);
set(gcf, 'Position',  plot_size)
print_file(fig,strjoin(name))
%% Figure 16- Ratio of Various Paramter as Acceleration is scaled- Nonlinear
close all; clc;
fig = 16;
name = "Comparing Scaling Accleration in Nonlinear and Linear Systems";
t = sgtitle((name),"FontSize",18); t.FontWeight =' bold';
maxU = max(abs(record_Trans1.U));
maxR = max(abs(record_Trans1.R));
maxA = max(abs(record_Trans1.A));
n = 10;
[ratio_U, ratio_R, ratio_G] = deal(zeros(1,n));
for i = 1:n
    record = Transient(i*acc, MatData, MatState, time_step_method,"Newton" , 10);
    ratio_U(i) = max(abs(record.U))/maxU;
    ratio_R(i) = max(abs(record.R))/maxR;
    ratio_G(i) = max(abs(record.A))/maxA;
end
subplot(1,2,1);hold on; grid minor;
title("Nonlinear System")
plot(ratio_R,"DisplayName","Ratio of Max Internal Force","LineWidth",2);
plot(ratio_U,"DisplayName","Ratio of Max Displacement","LineWidth",2);
```

```matlab
plot(ratio_G,"DisplayName","Ratio of Max Rel. Acceleration","LineWidth",2);
plot(1:n,'-.',"DisplayName","1:1"); xticks(1:n);
xlabel("Acceleration Factor"); ylabel("Ratio of Scaled:Original Values"); legend("Location","Northwest","FontSize",12);


maxU = max(abs(record_Trans5.U));
maxR = max(abs(record_Trans5.R));
maxA = max(abs(record_Trans5.A));
[ratio_U, ratio_R, ratio_G] = deal(zeros(1,n));
for i = 1:n
    record = Transient(i*acc, MatData, MatState, time_step_method,"Newton", 10);
    ratio_U(i) = max(abs(record.U))/maxU;
    ratio_R(i) = max(abs(record.R))/maxR;
    ratio_G(i) = max(abs(record.A))/maxA;
end
subplot(1,2,2); hold on; grid minor;
title("Linear System")
plot(ratio_R,"DisplayName","Ratio of Max Internal Force","LineWidth",2);
plot(ratio_U,"DisplayName","Ratio of Max Displacement","LineWidth",2);
plot(ratio_G,"DisplayName","Ratio of Max Rel. Acceleration","LineWidth",2);
plot(1:n,'-.',"DisplayName","1:1");xticks(1:n);
xlabel("Acceleration Factor"); ylabel("Ratio of Scaled:Original Values"); legend("Location","Northwest","FontSize",12);

set(gcf, 'Position',  [1200, 0, 1300,550])
print_file(fig,name)
%% Figure 17 -Comparing Force History of Linear and Nonlinear System
close all; clc
name = ["Comparing " + analy2, "Of Nonlinear and Linear- Elastic System"];
fig = 17;
title(name); hold on; grid on;
yyaxis right; p = plot_RU_curve(record_Trans5, "Linear Elastic System"); set(p,'Color','r');
yyaxis left ;plot_RU_curve(record_Trans1, "Nonlinear System");
set(gcf, 'Position',  plot_size)
print_file(fig, name)
%% Figure 18 -Comparing Relative Displacement and Absolute Acceleration with Linear System
close all; clc; hold on;
name = ["Comparing Relative Displacement and Absolute Acceleration" ,"Between Nonlinear and Linear System"];
t = sgtitle((name),"FontSize",18); t.FontWeight =' bold';
fig = 18;
plot_disp_acc(record_Trans5, record_Trans1,["Linear System", "Nonlinear System"]);
set(gcf, 'Position',  plot_size)
print_file(fig,strjoin(name))
%% Figure 19 -Dyanmic Loading Using the Newton Algorithm Base Case
close all; clc;
name = analy2+ " using "+algo1;
fig = 19;
title(name); grid on; hold on;
p = plot_RU_curve(record_Trans1, R+" "+ algo1);
set(gcf, 'Position',  plot_size)
print_file(fig, name)
%% Figure 10 -Relative Displacement and Absolute Acceleration
close all; clc
name = "Relative Displacement and Absolute Acceleration for the " + algo1;
t = sgtitle((name),"FontSize",18); t.FontWeight =' bold';
fig = 20;
plot_disp_acc(record_Trans1,[],[]);
set(gcf, 'Position',  plot_size)
```

```
print_file(fig,strjoin(name))
%%
clc;
print_info(record_Trans1)
compare_records(record_Trans1, record_Trans4)
compare_records(record_Trans5, record_Trans6)

%%
function print_info(record)
disp(newline + "Printing Information On the Record")
maxU = max(record.U);
t_maxU = record.time(record.U ==max(record.U));
maxR = record.R(record.U ==max(record.U));
minU = min(record.U);
t_minU = record.time(record.U ==min(record.U));
minR = record.R(record.U ==min(record.U));
maxV = max(abs(record.V));
t_maxV = record.time(max(abs(record.V))  == abs(record.V));
maxG = max(abs(record.A+record.acc));
t_maxG = record.time(max(abs(record.A))  == abs(record.A));
str = [sprintf("The maximum positive displacement was %.3f inches at %.2f seconds.",maxU,t_maxU) ;...
       sprintf("At that moment, the structure was experiencing %.3f kip of internal resisting force.", maxR); ...
       sprintf("The maximum negative displacement was %.3f inches at %.2f seconds.",minU,t_minU);...
       sprintf("At that moment, the structure was experiencing %.3f kip of internal resisting force.", minR);...
       sprintf("The maximum absolute velocity of the structure was %.3f in/s at %.3f seconds.",maxV,t_maxV);...
       sprintf("The maximum absolute acceleration felt by the structure was %.3f in/sec^2 at %.3f seconds.",maxG,t_maxG);...
       sprintf("The average iteration for this record is %.2f.",mean(record.iter))];
    disp(strjoin(str));
disp(str);
end

%%
function compare_records(record1, record2)
disp(newline+ "Comparing Two Records")
maxU1 = max(abs(record1.U));
t_maxU1 = record1.time(abs(record1.U) ==max(abs(record1.U)));
maxU2 = max(abs(record2.U));
t_maxU2 = record2.time(abs(record2.U) ==max(abs(record2.U)));
maxR1 = record1.R(abs(record1.U) ==max(abs(record1.U)));
maxR2 = record2.R(abs(record2.U) ==max(abs(record2.U)));
maxG1 = max(abs(record1.A+record1.acc))/386;
maxG2 = max(abs(record2.A+record2.acc))/386;

str = [sprintf("Record 1 had a maximum displacement of %.2f inches at %.2f seconds.",maxU1,t_maxU1) ;...
       sprintf("Record 2 had a maximum displacement of %.2f inches at %.2f seconds.",maxU2,t_maxU2) ;...
       sprintf("The ratio between records 2 to 1 is %.2f.",maxU2/maxU1);...

       sprintf("The structure in Record 1 resisted a maximum force of %.2f kips.",maxR1) ;...
       sprintf("The structure in Record 2 resisted a maximum force of %.2f kips.",maxR2) ;...
       sprintf("The ratio between records 2 to 1 is %.2f.",maxR2/maxR1);...

       sprintf("Record 1 had a maximum acceleration of %.2fg",maxG1) ;...
       sprintf("Record 2 had a maximum acceleration of %.2fg.",maxG2) ;...
       sprintf("The ratio between records 2 to 1 is %.2f.",maxG2/maxG1)];
disp(strjoin(str));
disp(str);
end
```

```

%% Functions
function plot_disp_acc(record1, record2, name)
    if ~isempty(record2)
    subplot(2,1,1); hold on;
    plot(record1.time,record1.U,'Color',[1,0,0,1],"DisplayName",name(1),"LineWidth",1);
    plot(record2.time,record2.U,'Color',[0,0,1,1],"DisplayName",name(2));
    xlabel("Time (sec)"); ylabel("Displacement (in)"); title("Relative Displacement");grid on; legend("Location","best");

    subplot(2,1,2); hold on;
    plot(record1.time,record1.A+record1.acc,'Color',[1,0,0,1],"DisplayName",name(1),"LineWidth",1);
    plot(record2.time,record2.A+record2.acc,'Color',[0,0,1,1],"DisplayName",name(2));
    xlabel("Time (sec)"); ylabel("Acceleration(in/sec^2)");title("Absolute Accleration");grid on; legend("Location","best");
    else
    subplot(2,1,1);
    plot(record1.time,record1.U,'b');
    xlabel("Time (sec)"); ylabel("Displacement (in)"); title("Relative Displacement");grid on;
    subplot(2,1,2);
    plot(record1.time,record1.A+record1.acc,'b');
    xlabel("Time (sec)"); ylabel("Acceleration(in/sec^2)");title("Absolute Accleration");grid on;
    end
    % subplot(3,1,2); plot(time,record1.V,'b',"DisplayName","Velocity"); xlabel("Time (sec)"); ylabel("Velocity (in/sec)");
end
function plot_iterations(record)
    plot(record.U_iter,record.P_iter,"DisplayName","Unbalanced Force Path");
end
function plot_MenegottoPinto(record)
    [MP_U, MP_Force] = Menegotto_Pinto(record.U,record.MatData, record.MatState);
    plt = plot(MP_U , MP_Force,'r','LineWidth',4, "DisplayName"," Menegotto-Pinto Uniaxial Model" );
    plt.Color(4) = 0.3;
end
function pl = plot_RU_curve(record, name)
    pl = plot(record.U,record.R,'b',"DisplayName",name,'MarkerEdgeColor','r',"MarkerSize",4);
    xlabel("Relative Displacement (in)"); ylabel("Internal Resisting Force (kip)"); legend("Location","Southeast");
end
function print_file(no, name)
    print("figures\"+string(no)+" "+ strjoin(name),'-dsvg','-PMicrosoft Print to PDF','-r600','-painters');
end
```

```matlab
function record = Static(P, MatData, MatState, algorithm_type , max_iter)
  record.P = P; % Saves the applied forces
  record.R = []; % Init internal force
  record.P_iter = []; % Record of R at every iteration
  record.U_iter = [];% Record of U at every iteration
  record.iter = []; % Record iterations per run
  record.MatData = MatData; % Saves the material data
  U_conv = 0; % Initialized last converged displacement as 0
  Delta_U = 0; % Initialized distance from last converged displacement was 0

   switch algorithm_type %  Checks which algorithm is used
      case "Newton" % N-R method
         tag = 1;
      case "ModifiedNewton" % Modified N-RMethod
         tag = 2;
      case "ModifiedNewton -initial" % Modified N-R Method with initial elastic tangent
         tag = 3;
   end

   for n = 1:numel(P)-1% Loop over load steps
      conv = 0; % Not converged at the beggining of the load
      j = 1; % Iteration counter reset to 1

      switch tag % Checks for modified N-R Method
         case 2
            algorithm_type = "ModifiedNewton";
            Ktan = MatData.A*MatState.Pres.Et/MatData.L;  % Tangent stiffness
         case 3
            algorithm_type = "ModifiedNewton -initial";
            Ktan = MatData.A*MatData.E/MatData.L;  % Initial stiffness
      end

      while ( j <= max_iter && conv == 0) % Loop over Newton-Raphson iterations while not converged
         MatState = Mate25n(MatData,MatState); % Update the material state
         R = MatData.A*MatState.Pres.sig; % Calculate teh internal resisting force
         Unb = P(n+1)-R; % Calculate the unbalance force
         record.Unb(n,j) = abs(Unb); % Record the unbalance force

         if (j > 1)
            record.U_iter = [record.U_iter,U_conv]; % Record the unbalance force path
            record.P_iter = [record.P_iter,R];% Record the unbalance force path
         end
        % Converged branch
         if (abs(Unb) < 1.e-5) % norm convergence criteria
            record.U(n+1) = U_conv; % Record the converged displacement
            record.R(n+1) = R;% Record the internal resisting force
            Delta_U = 0; % Reset total dispalcement from last displacement
            MatState.eps(1,2) = 0; % Reset total dispalcement from last displacement
            MatState.eps(1,3) = 0; % Reset incremental dispalcement from last displacement
            MatState.Past = MatState.Pres; % Commitees the present to the past. Total Strain remains
            conv = 1; % Convergence is true
            record.iter(n) =  j; % Record the iterations
         else
            if algorithm_type == "Newton" % Checks for Newton Algorithm to be used
               Ktan = MatData.A*MatState.Pres.Et/MatData.L;  % Tangent stiffness
            end
            if j == max_iter
```

```
                disp("Could not converged using " + algorithm_type + newline + "Switching to Newton-Raphson Method");
                j = 1; algorithm_type = "Newton"; % Switch to Newton if not converged
            end
            delta_U = Unb/Ktan; % Calculate the horizontal movement
            Delta_U = Delta_U + delta_U; % Calculate total dispalcement from last displacement
            U_conv = U_conv + delta_U; % Calculate converged dispalcement
            MatState.eps(1,1) = U_conv/MatData.L;  % Total strain
            MatState.eps(1,2) = Delta_U/MatData.L;  % Total incremental strain from last converged state
            MatState.eps(1,3) = delta_U/MatData.L;  % Last incremental strain
            j = j + 1; % Iteration counter
            record.U_iter = [record.U_iter,U_conv]; % Record the unbalance force path
            record.P_iter = [record.P_iter,P(n+1)];% Record the unbalance force path
        end
    end
  end
end
```

```matlab
function record = Static_no_record(P, MatData, MatState, algorithm_type , max_iter)
  record.P = P; % Saves the applied forces
  record.R = []; % Init internal force
  U_conv = 0; % Initialized last converged displacement as 0
  Delta_U = 0; % Initialized distance from last converged displacement was 0

  switch algorithm_type %  Checks which algorithm is used
     case "Newton" % N-R method
        tag = 1;
      case "ModifiedNewton" % Modified N-RMethod
         tag = 2;
      case "ModifiedNewton -initial" % Modified N-R Method with initial elastic tangent
         tag = 3;
   end
   for n = 1:numel(P)-1% Loop over load steps
      conv = 0; % Not converged at the beggining of the load
      j = 1; % Iteration counter reset to 1
      switch tag % Checks for modified N-R Method
         case 2
            algorithm_type = "ModifiedNewton";
            Ktan = MatData.A*MatState.Pres.Et/MatData.L;  % Tangent stiffness
         case 3
            algorithm_type = "ModifiedNewton -initial";
            Ktan = MatData.A*MatData.E/MatData.L;  % Initial stiffness
      end
      while ( j <= max_iter && conv == 0) % Loop over Newton-Raphson iterations while not converged
         MatState = Mate25n(MatData,MatState); % Update the material state
         R = MatData.A*MatState.Pres.sig; % Calculate teh internal resisting force
         Unb = P(n+1)-R; % Calculate the unbalance force
         if (abs(Unb) < 1.e-5) % norm convergence criteria is met
            record.U(n+1) = U_conv; % Record the converged displacement
            record.R(n+1) = R;% Record the internal resisting force
            Delta_U = 0; % Reset total dispalcement from last displacement
            MatState.eps(1,2) = 0; % Reset total dispalcement from last displacement
            MatState.eps(1,3) = 0; % Reset incremental dispalcement from last displacement
            MatState.Past = MatState.Pres; % Commitees the present to the past. Total Strain remains
            conv = 1; % Convergence is true
         else
            if algorithm_type == "Newton" % Checks for Newton Algorithm to be used
               Ktan = MatData.A*MatState.Pres.Et/MatData.L;  % Tangent stiffness
            end
            if j == max_iter
               disp("Could not converged using " + algorithm_type + newline + "Switching to Newton-Raphson Method");
               j = 1; algorithm_type = "Newton"; % Switch to Newton if not converged
            end
            delta_U = Unb/Ktan; % Calculate the horizontal movement
            Delta_U = Delta_U + delta_U; % Calculate total dispalcement from last displacement
            U_conv = U_conv + delta_U; % Calculate converged dispalcement
            MatState.eps(1,1) = U_conv/MatData.L;  % Total strain
            MatState.eps(1,2) = Delta_U/MatData.L;  % Total incremental strain from last converged state
            MatState.eps(1,3) = delta_U/MatData.L;  % Last incremental strain
            j = j + 1; % Iteration counter
         end
      end
   end
end
```

```
function[record] = Transient(acc, MatData, MatState, time_step_method, algorithm_type, max_iter)
 %  Time Steping
 beta = time_step_method.beta; % Time stepping Newmark-Beta Method parameter
 alpha = time_step_method.alpha; % Time stepping Newmark-Beta Method parameter
 dt = time_step_method.time_step;% Time stepping Newmark-Beta Method time step

 % initial all recorded variables
 [record.R_total, record.R, record.C, record.M, record.num_steps, record.U, record.V, record.A] = deal(zeros(numel(acc),1));
 record.time = dt:dt:dt*numel(acc); % Record down the time
  record.acc = acc; % Record acceleration
  record.MatData = MatData; % Record material data

  % Structural Variables
  m = MatData.mass; % mass
  c = MatData.damping; % dampin coefficient
  P = -m * acc; % Convert to force
  A = MatData.A; % area
  L =MatData.L; % length
  Delta_u = 0; % Total dispalcemet from last converged displacement
  U_conv = 0; % Last converged displacement

  % Check for algorithm to use
  switch algorithm_type
     case "Newton"
        tag = 1;
     case "ModifiedNewton"
        tag = 2;
  end

  % Solving the system
  for n = 1:numel(P)-1% Loop over "time"-load steps
     conv = 0; % Convergence is false
     j = 1; % Iteration counter
     switch tag % Check for algorithm to use
        case 2
           algorithm_type = "ModifiedNewton";
           Ktan = MatData.A*MatState.Pres.Et/MatData.L;  % Tangent stiffness
     end
     m_= m* (1/beta/dt^2*record.U(n) + 1/beta/dt*record.V(n) - (1-1/2/beta)*record.A(n)); % Calculate a temporary term
     d_ = c * ( alpha/beta/dt*record.U(n) - (1-alpha/beta)*record.V(n) - dt*(1-0.5*alpha/beta)*record.A(n)); % Calculate a temporary term
     P_tilde = P(n+1) + m_ + d_;  % Known Resisting forces- Calcualte P_tilde
     %% Loop over Newton-Raphson iterations
     while ( j <= max_iter && conv == 0)
        %% Calculate Unbalance force
        % Unknown Resisting forces; These depend on the iterations, Ui
        m_u = m* (1/beta/dt^2*U_conv) ; % Mass term
        c_u = c * (alpha/beta/dt*U_conv); % Damping term
        MatState = Mate25n(MatData,MatState); R = A*MatState.Pres.sig;  % R(Ui) % Stiffness term, requires material state determination
        Unb = P_tilde - (m_u + c_u + R); % Unbalanced force
        record.Unb(n,j) = abs(Unb);  % Record unbalance force
        %% Check Convergence
        % Converged branch
        if (abs(Unb) < 1.e-5)  % Converged criteria; norm of the residual
           % Commit the next displacement, velocity, and acceleration
           record.U(n+1) = U_conv; % Set the next displacement as the current state displacement
           record.V(n+1) = dt*(1-alpha/2/beta)*record.A(n) + (1-alpha/beta)*record.V(n)+alpha/beta/dt*(record.U(n+1)-record.U(n)); % Next↙
time step velocity
```

```
                record.A(n+1) = (1-1/beta/2)*record.A(n) - 1/beta/dt*record.V(n) +1/beta/dt^2*(record.U(n+1)-record.U(n)); % Next time step acceleration
                record.R(n+1) = R; % Record Internal Resisting force
                record.C(n+1) = c*record.V(n+1); % Record Damping Forces
                record.M(n+1) = m*record.A(n+1);% Record Inertia Forces
                record.total(n+1) = m*record.A(n+1) + c*record.V(n+1)+ R; % Record total resisting force
                % Reset State variables
                Delta_u = 0; % Reset ΔU for the iteration
                MatState.eps(1,2) = 0; % Reset Δε for the iteration
                MatState.eps(1,3) = 0;% Reset δε for the iteration
                MatState.Past = MatState.Pres; % Saves the state
                conv = 1; % Converged
                record.iter(n) = j; % Record number of iterations
            else % Has not converged
                % Check algorithm to use
                if algorithm_type == "Newton"
                    Ktan = MatData.A*MatState.Pres.Et/MatData.L;  % Tangent stiffness
                end
                if j == max_iter
                    disp("Could not converged using " + algorithm_type + newline + "Switching to Newton-Raphson Method");
                    j = 1; algorithm_type = "Newton";
                end
                Ktan_dynamic = 1/beta/dt^2 *m + alpha/beta/dt*c + Ktan; % Dynamic tangential stiffness
                % Update displacement variables
                delta_u = Unb/Ktan_dynamic; % Calculate (δUi)_n+1
                Delta_u = Delta_u + delta_u; % ΔU = ΔU + δU for the iteration
                U_conv = U_conv + delta_u; % U = U + δU for the iteration
                % Update strain variables
                MatState.eps(1,1) = U_conv/L;  % Total strain
                MatState.eps(1,2) = Delta_u/L;  % Total incremental strain from last converged state
                MatState.eps(1,3) = delta_u/L;  % Last incremental strain
                j = j + 1; % Increaseiteration counter
            end
        end
    end
end
```

```
function [K0, A, s_y0] = eqivalent_truss(mg, T0, Ry0, L, E0, g)
  % Takes in SDOF system with properties
    % mg; Weight
    % T0; Natural period
    % Ry0;  Yeild Strength
    % L; Length
    % E0; Elastic stiffness
    % g ; Gravitational constant
  % Returns the structural parameter of equivalent Truss
     % K0; stiffness
     % A; Area
     % s_y0; Initial Yeilding stress
   K0 = 4*pi^2*mg/g/T0^2; % kip/in; stiffness
   A = K0*L/E0; % in^2
   s_y0 = Ry0/A; % ksi
end
```

```matlab
function  [ out ] = get_materialHysteresis( matDef, inputData, numIncr, localOpenSeesPath )
%% DESCRIPTION

% INPUT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% matDef              : material definition string from OpenSees
% inputData           : a vector of input deformation/strain time history
% numIncr             : num of points to include between inputData(i) and inputData(i+1)
% localOpenSeesPath   : full path to OpenSees executable

% OUTPUT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% out                 : [deformation force] or [strain stress]

%% Function
[matDef, localOpenSeesPath] = convertStringsToChars(matDef, localOpenSeesPath);
inputData = arrayfun(@(x) num2str(x), inputData, 'UniformOutput', 0);
temp = strsplit(matDef);
matTag = temp{3};

materialTesterFid = fopen('matTest.tcl','w+');
fprintf(materialTesterFid,['wipe;\n']);
fprintf(materialTesterFid,['model testUniaxial;\n']);
fprintf(materialTesterFid,['set matTag ', num2str(matTag,'%u'), ';\n']);
fprintf(materialTesterFid,['set strainHistory {', strjoin(inputData), '};\n']);
fprintf(materialTesterFid,['set fileOut "hysteresis_matTag_$matTag.txt";\n']);
fprintf(materialTesterFid,['set out [open $fileOut w];\n']);
fprintf(materialTesterFid,[matDef '\n']);
fprintf(materialTesterFid,...
    ['uniaxialTest $matTag;\n',...
    'set strain 0.0;\n',...
    'set count 1;\n',...
    'set iTime 0;\n',...
    'set strain [expr $strain];\n',...
    'strainUniaxialTest $strain;\n',...
    'set stress [stressUniaxialTest];\n',...
    'set tangent [tangUniaxialTest];\n',...
    'set iTime [expr $iTime+1];\n',...
    'puts $out "$strain $stress";\n',...
    'foreach {strainExtremeVal} $strainHistory {\n',...
    '       set numIncr ' num2str(numIncr,'%u') ';\n',...
    '       set strainIncr [expr ($strainExtremeVal - $strain)/$numIncr];\n',...
    '       for {set i 0} {$i < $numIncr} {incr i 1} {\n',...
    '           set strain [expr $strain+$strainIncr];\n',...
    '           strainUniaxialTest $strain;\n',...
    '           set stress [stressUniaxialTest];\n',...
    '           set tangent [tangUniaxialTest];\n',...
    '           set iTime [expr $iTime+1];\n',...
    '           puts $out "$strain $stress";\n',...
    '       }\n',...
    '}\n',...
    'close $out;\n',...
    'puts "MATERIAL TESTER RAN SUCCESSFULLY!";\n',...
    'wipe;\n'...
    ]);
fclose(materialTesterFid);

```

```
[~, ~] = system(['"',localOpenSeesPath,'" "matTest.tcl"']);

fid = fopen(['hysteresis_matTag_' num2str(matTag,'%u') '.txt'],'r');
dataRead = textscan(fid, repmat('%f ',1,2), 'CollectOutput',true);
out = dataRead{1};
fclose(fid);
delete(['hysteresis_matTag_' num2str(matTag,'%u') '.txt']);
delete('matTest.tcl');

end
```

```matlab
function MatData = Initialize_MatData(E0, s_y0, b, R0, cR1, cR2, a1, a2, L, A)
    % Inputs
        % E0,; Modulus of Elasticity
        % s_y0; Yeild Stress
        % b, R0, cR1, cR2, a1, a2, ;Menegotto-Pinto Model parameters
        % L; length
        % A; Area
    % Returns the structural parameter of equivalent Truss
        % MatData; Structure with all of the above information
    MatData.E = E0;
    MatData.fy = s_y0;
    MatData.b = b;
    MatData.R0 = R0;
    MatData.cR1 = cR1;
    MatData.cR2 = cR2;
    MatData.a1 = a1;
    MatData.a2 = a2;
    MatData.L = L;
    MatData.A = A;
    MatData.ey = MatData.fy/MatData.E;
end
```

```
function MatState = Initialize_Material_State(sig, Et, fy, epmin, epmax, epex, ep0, epr, sr, kon, initial_strains)
  % Inputs
     % sig,; Modulus of Elasticity
     % Et; Yeild Stress
     % fy;Menegotto-Pinto Model parameters
     % epmin; minimum yield strain
     % epmax; maxmimum yield strain
     % ep0; Initial strain
     % epr; Max reversal starin
      % sr; Max reversal stress
      % kon; Initial branch
      % initial_strains; initial strains of the system stored as MatState.eps
   % Returns the structural parameter of equivalent Truss
      % MatState; Structure with all of the above information
   MatState.Pres.sig   = sig;
   MatState.Pres.Et    = Et;
   MatState.Pres.s0    = fy;
   MatState.Pres.epmin = epmin;
   MatState.Pres.epmax = epmax;
   MatState.Pres.epex  = epex;
   MatState.Pres.ep0   = ep0;

   MatState.Pres.epr   = epr;
   MatState.Pres.sr    = sr;
   MatState.Pres.kon   = kon;

   MatState.sig  = 0;
   MatState.Et   = Et;
   MatState.Past = MatState.Pres;
   for i = 1:length(initial_strains)
      MatState.eps(1,i) = initial_strains(i);
   end

end

```

```
function State = Mate25n (MatData,State)
%MATE25 hysteretic stress-strain relation after Menegotto-Pinto
%      with isotropic and kinematic hardening after Filippou et al. (EERC83-19)
% varargout = Mate25 (action,Mat_no,MatData,State)
%
% varargout : variable return argument list
%  varargout = MatData      for action 'chec'
%  varargout = State        for action 'init' with        fields sig, Et and Pres
%  varargout = State        for action 'stif' with updated fields sig, Et and Pres
%  varargout = State        for action 'forc' with updated field  sig     and Pres
%  varargout = [sig Post]   for action 'post'
%           where Et   = material tangent modulus
%                 sig  = current stress
%                 Pres = data structure with current values of material history variables
%                 Post = material post-processing information
% action   : switch with following possible values
%                'chec' material checks data for omissions
%                'data' material prints properties
%                'init' material initializes and reports history variables
%                'stif' material returns current stiffness and stress
%                'forc' material returns current stress only
%                'post' material stores information for post-processing
% Mat_no   : material number
% MatData  : data structure of material properties
% State    : current material state; data structure with updated fields eps, Past and Pres
%      .eps(:,1): total strains
%      .eps(:,2): strain increments from last convergence
%      .eps(:,3): strain increments from last iteration
%      .eps(:,4): strain rates
%      .Past    : history variables at last convergence
%      .Pres    : history variables at last iteration

% =========================================================================
% FEDEAS Lab - Release 2.3, March 2001
% Matlab Finite Elements for Design, Evaluation and Analysis of Structures
%
% Copyright (c) 1998, Professor Filip C. Filippou, filippou@ce.berkeley.edu
% Department of Civil and Environmental Engineering, UC Berkeley
% =========================================================================
% function contributed by Paolo Franchin & Alessio Lupoi, (c) January 2001

% Material Properties
% MatData.E   : initial modulus
%        .fy  : yield strength
%        .b   : strain hardening ratio
%        .R0  : exp transition elastic-plastic (default value 20)
%        .cR1 : coefficient for variation of R0 (default value 18.5)
%        .cR2 : coefficient for variation of R0 (default value 0.15)
%        .a1  : coefficient for isotropic hardening (default value 0)
%        .a2  : coefficient for isotropic hardening (default value 0)
%
% Material History Variables
% epmin : max strain in compression
% epmax : max strain in tension
% epex  : plastic excursion
% ep0   : strain at asymptotes intersection
% s0    : stress at asymptotes intersection
```

```matlab
% epr   : strain at last inversion point
% sr    : stress at last inversion point
% kon   : index for loading/unloading

% extract material properties
b   = MatData.b;
R0  = MatData.R0;
cR1 = MatData.cR1;
cR2 = MatData.cR2;
a1  = MatData.a1;
a2  = MatData.a2;
E   = MatData.E;
fy  = MatData.fy;
% compute some material parameters
Es2 = b*E;        % hardening modulus
ey  = fy/E;       % yield strain

% material state determination
% ==========================================================================
% Retrieve history variables from Past
sig   = State.Past.sig;   % stress at last converged state
Et    = State.Past.Et;
epmin = State.Past.epmin;
epmax = State.Past.epmax;
epex  = State.Past.epex;
ep0   = State.Past.ep0;
s0    = State.Past.s0;
epr   = State.Past.epr;
sr    = State.Past.sr;
kon   = State.Past.kon;   % kon = 0 for virgin state, kon = 1 for loading,  kon = 2 for unloading
sigp  = sig;  % saved version of stress at last converged state

epm   = max(abs(epmin),abs(epmax));
epss  = State.eps(1,1); % total strain  (total strain at current iteration)
depss = State.eps(1,2); % total strain increment from last convergence

if kon==0 % the material is virgin
   if depss == 0
       sig  = 0;
       Et   = E;
   end
   if (depss>0)
       kon  = 1;
       ep0  = epm;
       s0   = fy;
       epex = epm;
       [sig,Et] = Bauschinger(epex,ep0,ey,R0,cR1,cR2,epss,epr,b,s0,sr);
    end
    if (depss<0)
       kon  = 2;
       ep0  = -epm;
       s0   = -fy;
       epex = -epm;
       [sig,Et] = Bauschinger(epex,ep0,ey,R0,cR1,cR2,epss,epr,b,s0,sr);
    end

else % material is damaged
```

```
  if (kon==1 & depss>0)|(kon==2 & depss<0) % keep loading in the previous step direction
     [sig,Et] = Bauschinger(epex,ep0,ey,R0,cR1,cR2,epss,epr,b,s0,sr);
  elseif (kon==1 & depss<0) % inversion from tensile to compressive
     kon  = 2;
     epr  = epss-depss;
     sr   = sigp;
     if epr>epmax  epmax = epr; end
     epm  = max(abs(epmin),abs(epmax));
     sst  = fy*a1*(epm/ey-a2);
     sst  = max(sst,0);
     ep0  = (sr+fy+sst-(E*epr+Es2*ey))/(Es2-E);
     s0   = Es2*(ep0+ey)-fy-sst;
     epex = epmin;
     [sig,Et] = Bauschinger(epex,ep0,ey,R0,cR1,cR2,epss,epr,b,s0,sr);
  elseif (kon==2 & depss>0) % inversion from compressive to tensile
     kon  = 1;
     epr  = epss-depss;
     sr   = sigp;
     if epr<epmin  epmin = epr; end
     epm  = max(abs(epmin),abs(epmax));
     sst  = fy*a1*(epm/ey-a2);
     sst  = max(sst,0);
     ep0  = (sr+Es2*ey-(E*epr+fy+sst))/(Es2-E);
     s0   = fy+sst+Es2*(ep0-ey);
     epex = epmax;
     [sig,Et] = Bauschinger(epex,ep0,ey,R0,cR1,cR2,epss,epr,b,s0,sr);
  end
end

% save history variables
State.Pres.sig   = sig;
State.Pres.Et    = Et;
State.Pres.epmin = epmin;
State.Pres.epmax = epmax;
State.Pres.epex  = epex;
State.Pres.ep0   = ep0;
State.Pres.s0    = s0;
State.Pres.epr   = epr;
State.Pres.sr    = sr;
State.Pres.kon   = kon;

State.sig = sig;
State.Et  = Et;

% ===========================================================================

% +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
function [sig,Et] = Bauschinger(epex,ep0,epy,R0,cR1,cR2,epss,epr,b,s0,sr)
% calculate stress and moduls

xi    = abs((epex-ep0)/epy);
R     = R0-(cR1*xi)/(cR2+xi); %
e_str = (epss-epr)/(ep0-epr); %
s_str = b*e_str+(1-b)*e_str/(1+abs(e_str)^R)^(1/R);
sig   = s_str*(s0-sr)+sr; %

dSdE  = b + (1-b) * (1-abs(e_str)^R/(1+abs(e_str)^R)) / (1+abs(e_str)^R)^(1/R);
```

```
Et    = dSdE*(s0-sr)/(ep0-epr);

% ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

function [Displacement, Force] = Menegotto_Pinto(U_conv, MatData, MatState)
% Runs the Menegotto Pinto Model with given converged displacements
spacing = 100; % Linearly interpolates 100 points between each converged displacement
length_of_U = numel(U_conv); % Amount of displacements
Displacement = zeros(1,length_of_U*spacing-spacing); % zero vector for newly interpolated points
Force = Displacement; % zero vector

for i = 1:length(U_conv)-1
    Displacement(i*spacing - spacing +1 : i*spacing) = linspace(U_conv(i),U_conv(i+1),spacing); % Interpolateion with linspace
end
eps = Displacement/MatData.L; % Converts to strain history

for n = 1:numel(eps)
    MatState.eps(1,1) = eps(n);
    if n == 1
        MatState.eps(1,2) = 0.;
    else
        MatState.eps(1,2) = eps(n)-eps(n-1); % Change in strain is change in strain
    end
    MatState = Mate25n(MatData,MatState); % Runs the Menegotto
    Force(n) = MatState.Pres.sig * MatData.A; % Calculates the force for the iteration
    MatState.Past = MatState.Pres; %Updates the state
end
end
```

Appendix C. Opensees Analysis Files

```
# SE 201B: NONLINEAR STRUCTURAL ANALYSIS (WI 2021)
# HOMEWORK # 1
# NONLINEAR QUASI-STATIC & TIME-HISTORY ANALYSIS OF A SDOF SYSTEM
# ##################################################################################
# Louis Lin
# UNITS: kip, in, sec

# DEFINE EQUIVALENT TRUSS MODEL PROPERTIES ------------------------------------------
set g 386.4; # Acceleration due to gravity
set pi [expr 2*asin(1.0)]; # pi
set wt 2000.; # Weight of the structure
set m [expr $wt/$g]; # Mass of the structure
set T0 0.20; # Initial time period of the structure
set K0 [expr 4.*pow($pi,2.)*$m/(pow($T0,2))]; # Initial stiffness of the structure
set xi 0.02; # Damping ratio
set c [expr 2.*$xi*sqrt($K0*$m)]; # Damping coefficient
set E0 30000.; # Modulus of elasticity (steel)
set Ry0 [expr 0.15*$wt]; # Yield strength
set L 60.; # Length of the equivalent truss model
set A [expr $K0*$L/$E0]; # Area of cross-section of the equivalent truss model

# DEFINE NODES ----------------------------------------------------------------------
set nodeTag1 1;
set nodeTag2 2;

#node $nodeTag  (ndm $coords)
node $nodeTag1 0
node $nodeTag2 60

# DEFINE MASS -----------------------------------------------------------------------
# Needed for transient (dynamic) analysis only
#mass $nodeTag (ndf $massValues)
mass $nodeTag2 $m

# APPLY CONSTRAINTS-----------------------------------------------------------------
#fix $nodeTag (ndf $constrValues)
fix 1 1
fix 2 0

# DEFINE MATERIAL PARAMETERS --------------------------------------------------------
# ...
set matTag_1 1
set Fy [expr $Ry0/$A]
set E $E0
set b 0.02
set R0 5
set cR1 3
set cR2 0.15

# DEFINE MATERIAL -------------------------------------------------------------------
#uniaxialMaterial Steel02 $matTag             $Fy         $E  $b    $R0 $cR1 $cR2 <$a1
$a2 $a3 $a4 $sigInit>
uniaxialMaterial Steel02 $matTag_1 $Fy $E $b $R0 0.6 0.15 0 1 0 1

# DEFINE ELEMENT --------------------------------------------------------------------
#element truss $eleTag $iNode    $jNode    $A $matTag
element truss 1 $nodeTag1 $nodeTag2 $A $matTag_1
```

```
filename = "C:\Users\Louis Lin\Workspace\Academic\UCSD\SE 201B\HW\HW1\opensees\run.tcl";
openseesPath = "C:\Users\Louis Lin\Workspace\Academic\UCSD\SE 201B\Opensees\bin\OpenSees.exe";

[filepath,name,ext] = fileparts(filename);
[filepath,name,ext,openseesPath] = convertStringsToChars(filepath,name,ext,openseesPath);
currpath = pwd;
cd(filepath)
system(['"',openseesPath,'" "' name, ext, '"']);
cd(currpath);
```

```
%% POST PROCESSOR FOR HW 1 (SE 201B: NONLINEAR STRUCTURAL ANALYSIS)

% Run this file in the same folder as run.tcl

% Run this file after running the code run.tcl in OpenSees. The text
% files generated in the process will be loaded and used for plotting.

close all
clc

%% INPUT
matDef = 'uniaxialMaterial Steel02 1 29.332482664456787 30000. 0.02 3. 0.5 0.15 0. 1. 0. 1. 0.';
L = 60;
A = 10.216981781666002;
localOpenSeesPath = "C:\Users\Louis Lin\Workspace\Academic\UCSD\SE 201B\Opensees\bin\OpenSees.exe"; % full path to OpenSees ↙
executable
analysisType = 'Transient'; % Choose between 'Static' & 'Transient'
algorithm = 'Newton'; % Choose between 'Newton', 'ModifiedNewton', and 'ModifiedNewton -intial'
%% PLOT
switch analysisType
    case 'Transient'
        U_TH = load(['Results/disp_' analysisType '_' algorithm '.txt']);
        U = U_TH(:,2);
        t = U_TH(:,1);
        R_TH = load(['Results/res_' analysisType '_' algorithm '.txt']);
        R = -R_TH(:,2);
        V_TH = load(['Results/vel_' analysisType '_' algorithm '.txt']);
        V = V_TH(:,2);
        A_TH = load(['Results/acc_' analysisType '_' algorithm '.txt']);
        Ac_total = A_TH(:,2);

        % Time Histories
        figure();
        subplot(3,1,1);
        plot(t,U,'b','LineWIdth',2);
        indexmax = find(max(abs(U)) == abs(U));
        Umax = U(indexmax);
        tmax = t(indexmax);
        hold on
        plot(tmax,Umax,'ro','MarkerFaceColor','r')
        text(tmax,Umax,sprintf('  u_{rel, max} = %1.3f in',abs(Umax)),'VerticalAlignment','middle','HorizontalAlignment','left','FontSize',↙
12,'FontWeight','Bold');
        ylabel('u_{relative}(t)');
        set(gca,'XTick',[]);
        set(gca,'YLim',[-abs(Umax)-4,abs(Umax)+4])
        set(gca,'XLim',[0 t(end)])
        title('Time Histories');
        set(gca,'FontSize',18,'FontWeight','Bold')

        subplot(3,1,2);
        plot(t,V,'b','LineWIdth',2);
        indexmax = find(max(abs(V)) == abs(V));
        Vmax = V(indexmax);
        tmax = t(indexmax);
        hold on
        plot(tmax,Vmax,'ro','MarkerFaceColor','r')
        text(tmax,Vmax,sprintf('  v_{rel, max} = %1.3f in/s',abs(Vmax)),'VerticalAlignment','middle','HorizontalAlignment','left','FontSize',↙
```

```matlab
12,'FontWeight','Bold');
        ylabel('v_{relative}(t)');
        set(gca,'XTick',[]);
        set(gca,'YLim',[-abs(Vmax)-10,abs(Vmax)+10])
        set(gca,'XLim',[0 t(end)])
        set(gca,'FontSize',18,'FontWeight','Bold')

        subplot(3,1,3);
        plot(t,Ac_total,'b','LineWIdth',2);
        indexmax = find(max(abs(Ac_total)) == abs(Ac_total));
        Amax = Ac_total(indexmax);
        tmax = t(indexmax);
        hold on
        plot(tmax,Amax,'ro','MarkerFaceColor','r')
        text(tmax,Amax,sprintf('  a_{abs, max} = %1.3f in/s^2',abs(Amax)),'VerticalAlignment','middle','HorizontalAlignment','left','FontSize',12,'FontWeight','Bold');
        ylabel('a_{absolute}(t)');
        xlabel('Time [sec]')
        set(gca,'YLim',[-abs(Amax)-50,abs(Amax)+50])
        set(gca,'XLim',[0 t(end)])
        set(gca,'FontSize',18,'FontWeight','Bold')
        set(gcf, 'Position',  [1200,0,1300,700])
        print("..\matlab\P2\submittal\figures\"+string(21)+" Opensees A-V-U",'-dsvg','-PMicrosoft Print to PDF','-r600','-painters');

        % Resistance - Displacement Curve
        figure(); hold on;
        plot(U,R,'b','LineWIdth',2);
        matHyst = get_materialHysteresis(matDef, U./L, 10, localOpenSeesPath);
        plot(matHyst(:,1).*L, matHyst(:,2).*A,'k--','LineWidth',1.5)
        xlabel('U_{rel} [in]')
        ylabel('R [kips]')
        set(gca,'FontSize',18,'FontWeight','Bold'); grid on;
        legend('Linearized PO Curve b/w equilibrium pts','True F-d curve','location','best',"FontSize",14)
        title("Opensees " + analysisType + " Anlaysis " + algorithm + " Method");
        set(gcf, 'Position',  [1200,0,1300,700])
        print("..\matlab\P2\submittal\figures\"+string(21)+" Opensees "+analysisType,'-dsvg','-PMicrosoft Print to PDF','-r600','-painters');

    case 'Static'
        U = load(['Results/disp_' analysisType '_' algorithm '.txt']);
        R = -load(['Results/res_' analysisType '_' algorithm '.txt']);

        % Resistance - Displacement Curve
        figure();
        plot([0;U],[0;R],'b','LineWIdth',2);
        hold on
        plot([0;U],[0;R],'ro','LineWIdth',2);
        matHyst = get_materialHysteresis(matDef, U./L, 100, localOpenSeesPath);
        plot(matHyst(:,1).*L, matHyst(:,2).*A,'k--','LineWidth',1.5)
        xlabel('U [in]')
        ylabel('R [kips]')
        legend('Linearized PO Curve b/w equilibrium pts','Equilibrium Pts','True F-d curve','location','best',"FontSize",14)
        set(gca,'FontSize',18,'FontWeight','Bold'); grid on;
        title("Opensees " + analysisType + " Anlaysis " + algorithm + " Method");
        set(gcf, 'Position',  [1200,0,1300,700])
        print("..\matlab\P2\submittal\figures\"+string(21)+" Opensees "+analysisType,'-dsvg','-PMicrosoft Print to PDF','-r600','-painters');
end
```

```
# SE 201B: NONLINEAR STRUCTURAL ANALYSIS (WI 2021)
# HOMEWORK # 1
# NONLINEAR QUASI-STATIC & TIME-HISTORY ANALYSIS OF A SDOF SYSTEM
# ##########################################################################
# Angshuman Deb

# PUSHOVER ANALYSIS (LOAD CONTROLLED) -------------------------------------------

# SET LOAD LEVELS VECTOR ------------------------------------------------------
set loadFileName "P.txt";           # load file name with one col (force values) (Need
to set it manually)

# EXTRACT LOAD DATA -----------------------------------------------------------
set data_fid [open $loadFileName "r"]
set data [read $data_fid]
close $data_fid
set data_new [split $data "\n"]
set P {}
for {set k 0} {$k <= [expr [llength $data_new] - 2]} {incr k 1} {
    set data_t [lindex $data_new $k]
    lappend P [lindex $data_t 0]
}
set nls [expr [llength $P] - 1]; # Number of load steps

# DEFINE TIME SERIES ----------------------------------------------------------
set tsTag 1; # Tag for the time series
set dt 1.; # Time step of 1.0 for load application. Doesn't matter for a static analysis.

#timeSeries Path $tag   -dt $dt -values {list_of_values}
timeSeries Path $tsTag -dt 1.  -values  $P;

# DEFINE LOAD PATTERN ---------------------------------------------------------
set loadTag 1; # Tag for the load pattern
pattern Plain $loadTag $tsTag {
    #load $nodeTag      (ndf $LoadValues)
    load  $nodeTag2     1.0;         # Reference load = 1.0 will be multiplied by the
    values in the list P.
}

# CREATE THE SYSTEM OF EQUATIONS ----------------------------------------------
system BandGeneral;

# CREATE THE CONSTRAINT HANDLER -----------------------------------------------
constraints Plain;

# CREATE THE DOF NUMBERER -----------------------------------------------------
numberer Plain;

# CREATE THE CONVERGENCE TEST -------------------------------------------------
test NormUnbalance 1.0e-5 1000; # The norm of the displacement increment with a
tolerance of 1e-5 and a max number of iterations of 1000. The "1" or "0" at the end
shows/doesn't show all iterations.

# CREATE SOLUTION ALGORITHM ---------------------------------------------------
set algorithmBasic [split $algorithmString]
algorithm {*}$algorithmBasic

# CREATE THE INTEGRATION SCHEME -----------------------------------------------
set lambda 1.; # Set the load factor increment. A value of 1 indicates no further
divison of load levels into steps. A value of 0.1, for example, would mean subdivision
of each load step into 10 further steps.
integrator LoadControl $lambda; # The LoadControl scheme

# CREATE THE ANALYSIS OBJECT --------------------------------------------------
analysis $analysisType;

# RECORD AND SAVE OUTPUT ------------------------------------------------------
source generateRecorders.tcl; # Call file Recorders.tcl to record desired structural
responses and save as an output file
```

```

# ANALYZE ----------------------------------------------------------------------------
set nSteps 1; # Set the number of steps in which the structure is to be analyzed for each load increment. A value of 1 is used to analyze the structure for each load step at a time and not in one go.
for {set i 1} {$i <= $nls} {incr i 1} {
    set ok [analyze $nSteps]; # Analyze the structure for each load increment (0 - 200, 200 - 300, etc.). Sets ok to 0 if successful.
    # If the solution algorithm fails, as expected at reversals, change analysis option to Newton
    if {$ok != 0} {
        puts "Solution algorithm failed! Might be a load reversal! Changing algorithm to Newton"
        algorithm Newton;
        set ok [analyze $nSteps];
        # If successful, revert to original algorithm
        if {$ok == 0} {
            puts "Changing to Newton helped. Going back to original algorithm"
            algorithm {*}$algorithmBasic
        }
    }
}
```

```
# SE 201B: NONLINEAR STRUCTURAL ANALYSIS (WI 2021)
# HOMEWORK # 1
# NONLINEAR QUASI-STATIC & TIME-HISTORY ANALYSIS OF A SDOF SYSTEM
# ####################################################################################
# Angshuman Deb

# TIME-HISTORY ANALYSIS -----------------------------------------------------------

# SET UP GROUND-MOTION-ANALYSIS PARAMETERS ----------------------------------------
set gmDirection 1;                                          # ground-motion direction (Need to set it manually)
set gmFact 1.0;                                             # ground-motion scaling factor (Need to set it manually)
set gmFileName "SYL360.txt";                                # ground motion file name with two cols (time and u_g_ddot) (Need to set it manually)
set ratio 1.0;                                              # Ratio of DtAnalysis/dt (Need to set it manually)

# EXTRACT GROUND MOTION DATA ------------------------------------------------------
set data_fid [open $gmFileName "r"]
set data [read $data_fid]
close $data_fid
set data_new [split $data "\n"]
set timeData {}
set gmData {}
for {set k 0} {$k <= [expr [llength $data_new] - 2]} {incr k 1} {
    set data_t [lindex $data_new $k]
    lappend timeData [lindex $data_t 0]
    lappend gmData   [lindex $data_t 1]
}

set tMaxAnalysis [lindex $timeData end];                    # Maximum duration of GM analysis
set dt [expr [lindex $timeData 1] - [lindex $timeData 0]];  # ground motion sampling time
set dtAnalysis  [expr $dt*$ratio];                          # time-step for analysis

# INCLUDE DAMPING -----------------------------------------------------------------
# Only alphaM is needed since for SDOF,
set alphaM [expr $c/$m];
set betaK 0.;
set betaKinit 0.;
set betaKcomm 0.;
rayleigh $alphaM $betaK $betaKinit $betaKcomm; # RAYLEIGH damping

# DEFINE TIME SERIES AND LOAD PATTERN ----------------------------------------------
set loadTag 1;  # LoadTag for uniform ground motion excitation
set gmFact [expr $g*$gmFact]; # Since data in input file is in units of g.
set tsTag 1;
timeSeries Path $tsTag -dt $dt -values $gmData -factor $gmFact;
pattern UniformExcitation $loadTag $gmDirection -accel $tsTag;      # create Uniform excitation

# PERFORM DYNAMIC GROUND-MOTION ANALYSIS ------------------------------------------
set NewmarkGamma 0.50;  # Newmark-integrator gamma parameter (also HHT)
set NewmarkBeta  0.25;  # Newmark-integrator beta parameter

# CREATE THE SYSTEM OF EQUATIONS --------------------------------------------------
system BandGeneral;

# CREATE THE CONSTRAINT HANDLER ---------------------------------------------------
constraints Plain;

# CREATE THE DOF NUMBERER ---------------------------------------------------------
numberer Plain;

# CREATE THE CONVERGENCE TEST -----------------------------------------------------
test NormUnbalance 1.0e-5 1000 0; # The norm of the displacement increment with a tolerance of 1e-5 and a max number of iterations of 1000. The 1/0 at the end
```

```
shows/doesn't show all iterations.

# CREATE SOLUTION ALGORITHM ----------------------------------------------------------
set algorithmBasic [split $algorithmString]
algorithm {*}$algorithmBasic

# CREATE THE INTEGRATION SCHEME ------------------------------------------------------
integrator Newmark $NewmarkGamma $NewmarkBeta;

# CREATE THE ANALYSIS OBJECT ---------------------------------------------------------
analysis $analysisType;

# RECORD AND SAVE OUTPUT -------------------------------------------------------------
source generateRecorders.tcl; # Call file Recorders.tcl to record desired structural responses and save as an output file

# ANALYZE ----------------------------------------------------------------------------
set nSteps 1; # Set the number of steps in which the structure is to be analyzed. Here we go one step at a time
set tCurrent [getTime];
set ok 0;
while {$ok == 0 && $tCurrent <= $tMaxAnalysis} {
    set ok [analyze $nSteps $dtAnalysis];  # Analyze the structure for each time step. Sets ok to 0 if successful.
    if {$ok == 0} { puts " TIME: $tCurrent >> CONVERGED" }
    set tCurrent [getTime];
    # If the solution algorithm fails, as expected at reversals, change analysis option to Newton
    if {$ok != 0} {
        puts "Solution algorithm failed! Might be a load reversal! Changing algorithm to Newton"
        algorithm Newton;
        set ok [analyze $nSteps $dtAnalysis];
        # If successful, revert to original algorithm
        if {$ok == 0} { puts " TIME: $tCurrent >> CONVERGED" } {
            set tCurrent [getTime];
            puts "Changing to Newton helped. Going back to original algorithm"
            algorithm {*}$algorithmBasic
        }
    }
}
```

```
# SE 201B: NONLINEAR STRUCTURAL ANALYSIS (WI 2021)
# HOMEWORK # 1
# NONLINEAR QUASI-STATIC & TIME-HISTORY ANALYSIS OF A SDOF SYSTEM
# ######################################################################################
# Angshuman Deb

if {$analysisType == "Static"} {
    set dispfile "disp_$analysisType\_$algorithmString.txt";
    recorder Node -file $dataDir/$dispfile -node $nodeTag2 -dof 1 disp; # Record nodal displacements

    set resfile "res_$analysisType\_$algorithmString.txt";
    # ############# Since it is the reaction, note that in order to get the F - d plot, you need to take the negative of each value of the reaction. ###############
    recorder Node -file $dataDir/$resfile -node $nodeTag1 -dof 1 reaction; # Record reaction

} elseif {$analysisType == "Transient"} {
    set dispfile "disp_$analysisType\_$algorithmString.txt";
    recorder Node -file $dataDir/$dispfile -time -node $nodeTag2 -dof 1 disp; # Record nodal displacements (relative)

    set resfile "res_$analysisType\_$algorithmString.txt";
    # ############# Since it is the reaction, note that in order to get the F - d plot, you need to take the negative of each value of the reaction. ###############
    recorder Node -file $dataDir/$resfile -time -node $nodeTag1 -dof 1 reaction; # Record reaction

    set velfile "vel_$analysisType\_$algorithmString.txt";
    recorder Node -file $dataDir/$velfile -time -node $nodeTag2 -dof 1 vel; # Record nodal velocities (relative)

    set accfile "acc_$analysisType\_$algorithmString.txt";
    recorder Node -file $dataDir/$accfile -timeSeries $tsTag -time -node $nodeTag2 -dof 1 accel; # Record nodal accelerations (for absolute accel, need to provide timeSeries tag)
}
```

```
# SE 201B: NONLINEAR STRUCTURAL ANALYSIS (WI 2021)
# HOMEWORK # 1
# NONLINEAR QUASI-STATIC & TIME-HISTORY ANALYSIS OF A SDOF SYSTEM
# ##################################################################################
# Angshuman Deb

# UNITS: kip, in, sec (OpenSees doesn't have units. So be consistent!)

# INITIALIZATION --------------------------------------------------------------------
wipe; # Clear memory of all past model definitions
model BasicBuilder -ndm 1 -ndf 1; # Define the model builder, ndm=#dimension, ndf=#dofs

# SETUP DATA DIRECTORY FOR SAVING OUTPUTS -------------------------------------------
set dataDir "Results";  # Set up name of data directory
file mkdir $dataDir; # Create data directory

# SET ANALYSIS TYPE -----------------------------------------------------------------
set analysisType "Transient"; # Change between Static & Transient
set algorithmString "Newton"; # Change between Newton, ModifiedNewton and
ModifiedNewton -initial

# SOURCE MODEL ----------------------------------------------------------------------
source "trussModel.tcl"

# ANALYSIS --------------------------------------------------------------------------
if {$analysisType == "Static"} {
    source analysisPushover.tcl;
} elseif {$analysisType == "Transient"} {
    source analysisTimeHistory.tcl
}

if {$ok == 0} {
    puts "ANALYSIS DONE!"; # Spit out a success message
} else {
    puts "ANALYSIS FAILED!"; # Spit out a failure message
}

# DON'T FORGET TO -------------------------------------------------------------------
remove recorders;
# AND/OR
wipe;
```