Homework 2

Louis Lin

February 18, 2021

University of California, San Diego

Prof. Joel Conte

SE 201B Nonlinear Analysis

Contents

List of Figure

List of Tables

Introduction

In this report, a nonlinear geometric analysis was performed on the Lee Frame as shown in Figure 1. The frame is composed of a 120 inches tall column and a 120 long beam attached at a 90° rigid joint with pins at each support. The material properties of the frame were assumed to be linear elastic with an elastic modulus of 72,000 ksi. There were 10 elements for the column and the beam, each being 12 inches long. The cross section was 30 inches wide and 20 inches tall for an area of 600 in^2 and a moment of area of 20,000 in^4. A 10 kip downward force was applied at node 13; at times this was increased in order to explore the force-displacement behavior. Nodes 1 and 20 are restrained in the x and y directions. The model was ran with 100 steps increments of a 18 kip forces.

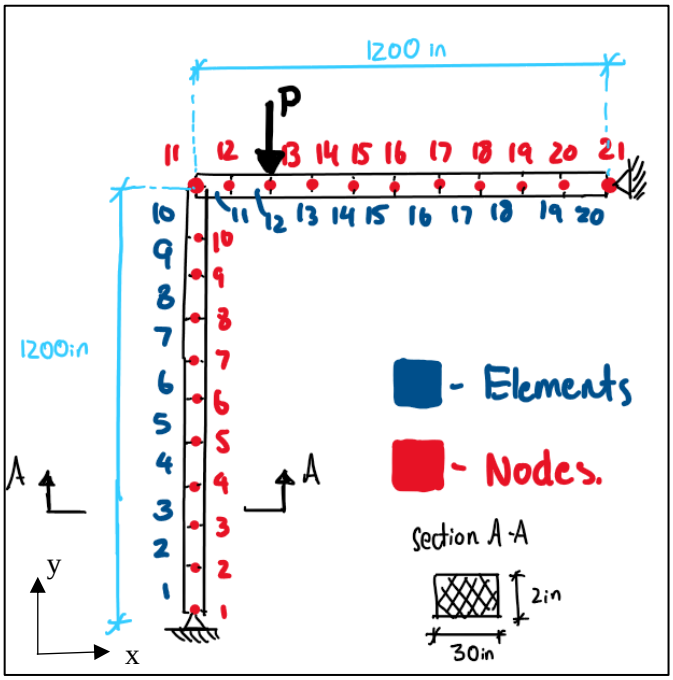

Figure 1 Lee Frame for Analysis

Problem 1– MATLAB Analysis

Part I – Nonlinear Geometric Analysis Using Corotational Assumption

Question 1. What is the load-displacement curve of a few key degrees of freedom?

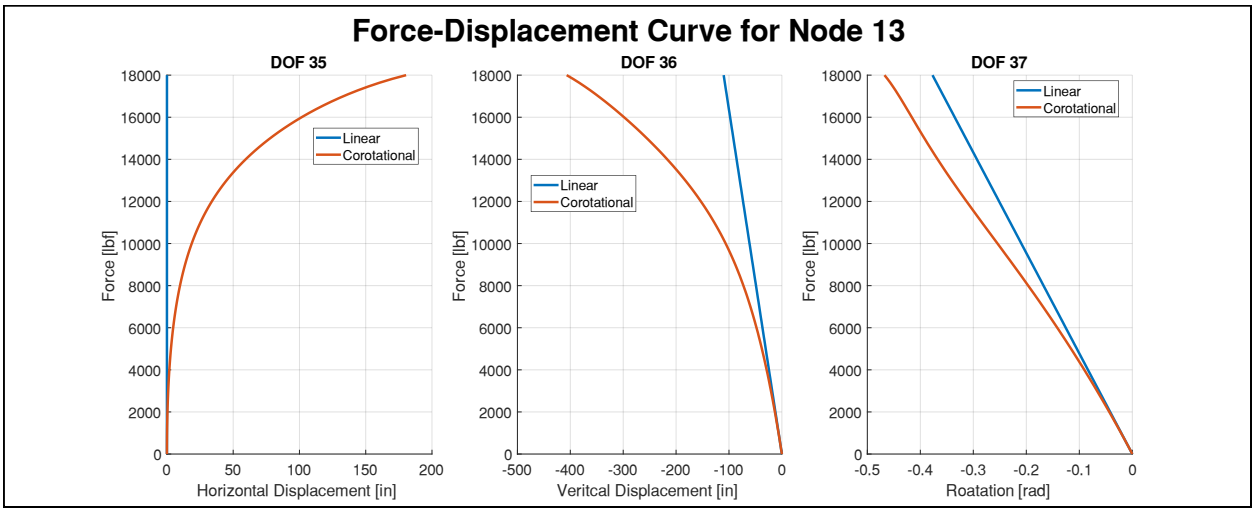

Figure 2 Load Displacement Curve of Node 13

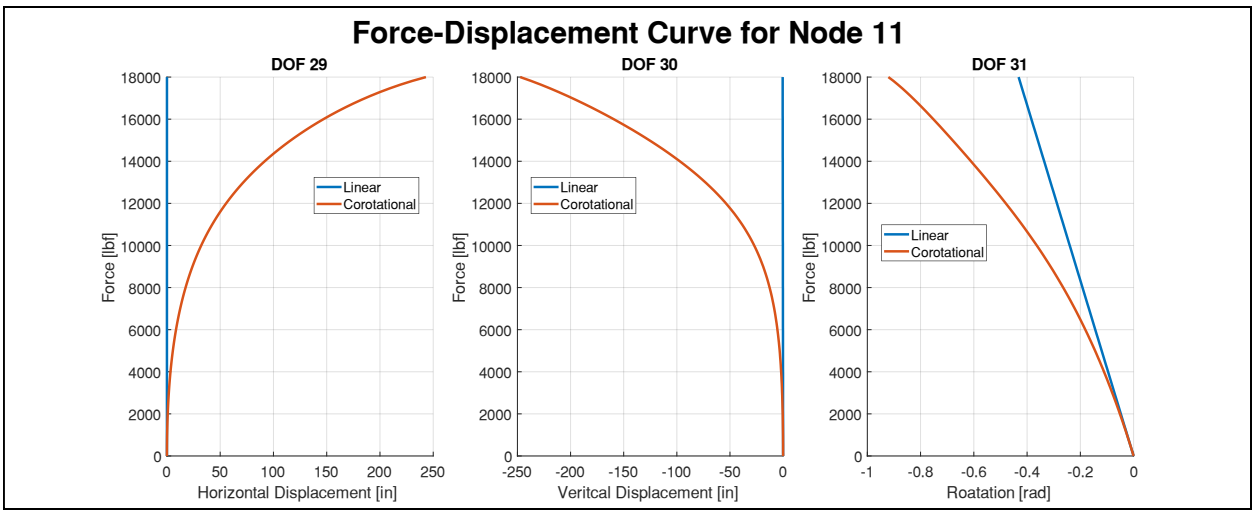

Figure 3 Load Displacement Curve of Node 11

The two nodes that are of particular interest are the node right beneath the load (node 13) and the node at the corner of the Lee frame (node 11). In addition, the rotation at the two supports is also of interest. In Figure 2 and Figure 3 the horizontal displacement, vertical displacement, and rotation of the nodes are plotted against the force applied for the corotational and linear analyses. Similarly, in Figure 5, the rotation of the two support's rotational degrees of freedom is plotted against the applied load. The values in Figure 2, Figure 3, and Figure 4 are plotted up to an applied load of 18 kips downwards. It is shown that that the force-displacement curve of the linear analysis is tangent to the corotational force-displacement curve for small forces. For small forces (relative to the stiffness of the structures), the deformed geometry of the structure will not affect the linear forces as calculated by assuming an undeformed geometric configuration. When the deformations of the structure reach a certain point, the forces develop by the material in the deformed configuration diverges from the forces develop by the

linear assumption. The internal resisting force is calculated based upon the geometry of the deformed shape and thus will appear different; the structure will need to deform more in order satisfy the equilibrium. This is seen as the structure displaces and translates more for the same level of load.

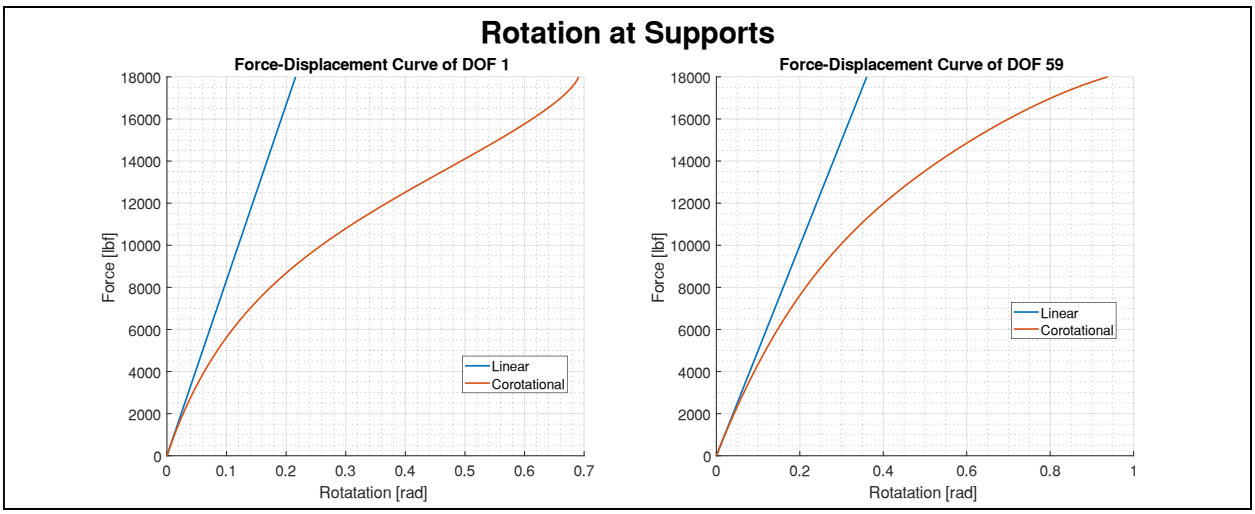

Figure 4 Rotation at the End

The linear analysis predicts a lower rotation at the ends of the column and beam which could be significant when considering the moment that is developed by a member. The rotation is related to the moment at the ends of the element through the first moment-area theorem. While the rotation of a real structure will never reach these levels, it is important to distinguish the divergent behavior of these systems at higher load levels.

Question 2. What is the Difference Between Linear and Nonlinear Geometric Analysis?

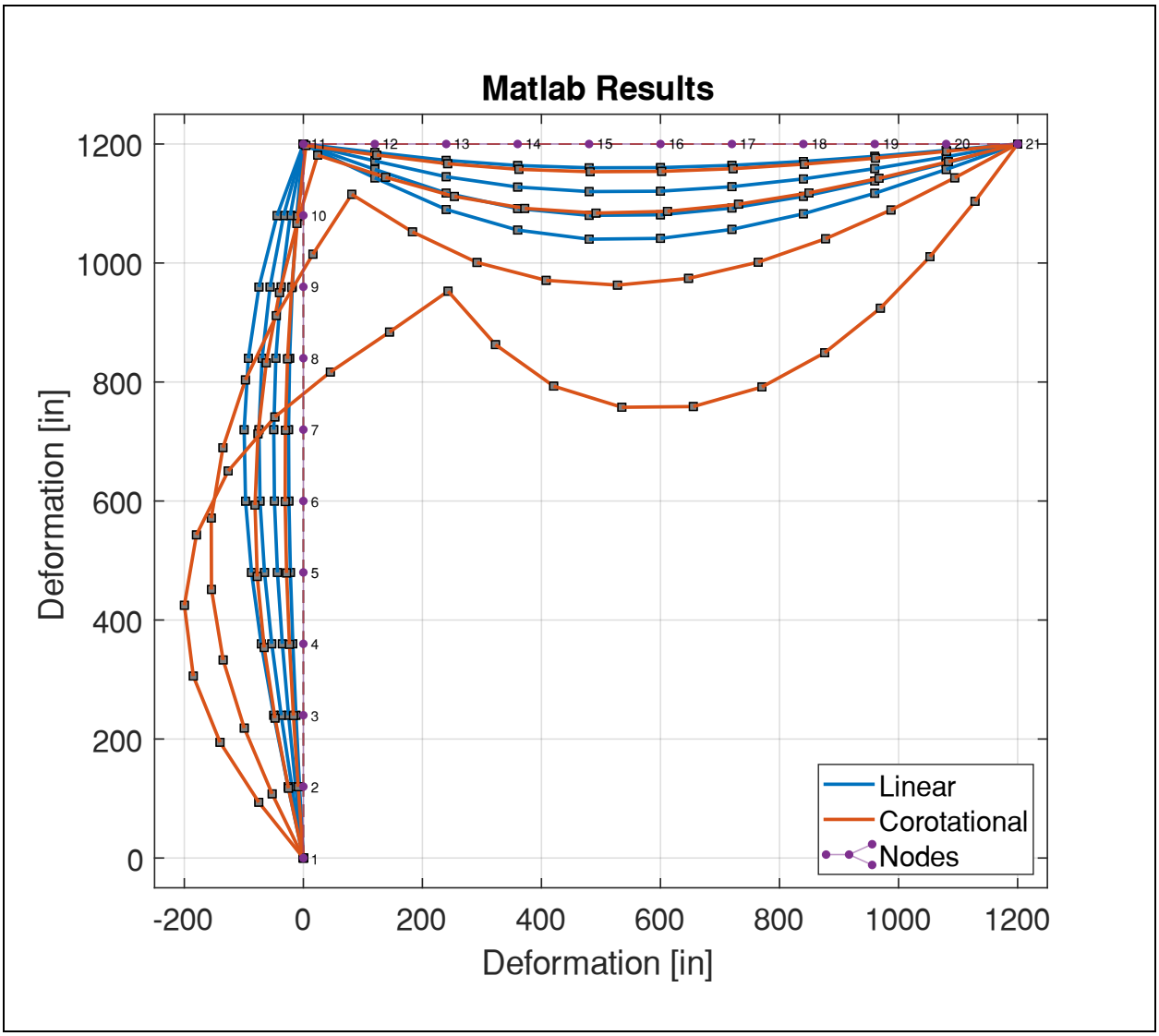

Figure 5 Deformed Shape Comparison of Nonlinear Corotational and Linear Geometric Analysis

Nonlinear geometric analysis considers the forces of each element in the structure based upon the deformed configuration of the shape while the linear geometric configuration does not. As the shape deforms, this new configuration can be separated into two forms of deformation, one attributed to the movement of the element as a rigid body and the other attributed to the basic deformations of the element. The superposition of these two deformations comprises the overall deformation of the element from which the internal forces are calculated. This method is called the corotational approach.

The rigid body deformation of an element is the translation of the element to a new position based upon the local coordinate system of the previous configuration. The translation of the element establishes a new local coordinate system at the deformed configuration which allows us to simplify the axial and transverse displacement of the element.

The basic system of an element is a way of describing the configuration of the element using only 3 degrees of freedom, or its basic deformations. These deformations are assumed to be the rotation at each

end of the element and the change in the length of the element. For the deformed configuration of the element, the change in the basic system is the change in axial length of the member and the difference in rotation of the element with respect to the rigid body deformed configuration. Using the basic system in describing the deformed configuration of the system, it can be transformed back into the local coordinate system using a linear transformation.

Using the basic element stiffness and the basic deformations, the internal forces of the basic system can calculate and then transformed into the local and then global coordinate systems which allows solving of the global equilibrium. This would allow for the incremental-iteration of the geometric configuration of the system until it reaches a new stable configuration.

For a linear analysis, the internal forces do not consider the deformed configuration of the element. When a load is applied, the system will deform elastically to the applied load and then the next deformed configuration is based upon the current deformed configuration.

In Table 1, the vertical displacement of the node under the point load is shown for the corotational geometry approach and the linear geometry approach. In the initial load steps the percent difference are small but increase to a substantial amount as the load steps increase. This shows the divergence of the two solutions as the structure deforms; the additional internal forces produced by a deformed geometry is clear when comparing these results.

Table 1 Comparison of DOF 36 Displacement for Nonlinear Corotational Linear Analysis

Load Step	Corotational	Linear	Percent Diff.	Load Step	Corotational	Linear	Percent Diff.
1	-1.2237e-01	-1.2228e-01	0%	60	-3.8005e+01	-3.0693e+01	24%
20	-6.4713e+00	-6.2363e+00	4%	70	-4.8160e+01	-3.6807e+01	31%
30	-1.3324e+01	-1.2350e+01	8%	80	-5.9693e+01	-4.2921e+01	39%
40	-2.0776e+01	-1.8464e+01	13%	90	-7.2967e+01	-4.9035e+01	49%
50	-2.8949e+01	-2.4579e+01	18%	100	-8.8440e+01	-5.5149e+01	60%

Question 3. What is the Euclidean Norm of the Unbalanced Force per Iteration Number?

Table 2 Euclidean Norm per Maximum Iteration Number per Load Step

Iteration Number	Load Step 95	Load Step 96	Load Step 97	Load Step 98
1	1.8000E+02	1.8000E+02	1.8000E+02	1.8000E+02
2	9.7215E+03	1.0389E+04	1.1244E+04	1.2370E+04
3	3.8016E+02	4.1537E+02	4.5914E+02	5.1529E+02
4	1.0785E+00	1.7446E+00	2.9655E+00	5.3555E+00
5	4.2894E-02	7.0396E-02	1.2124E-01	2.2174E-01

For three load steps in the nonlinear portions of the force-displacement curve, the unbalanced force is shown per iteration at each load step. The convergence criteria for each load step were 1.0e-5 which is evident in that every last iteration has an unbalanced force lower than the criteria. The rate of convergence seems to be similar or faster than a quadratic growth but not proven. The initial unbalanced force is the total applied load divided amount the number of load steps; 10000 lbf divided among 100 load steps is 100 lbf which is evident.

Question 4. Is there Equilibrium in the Deformed Configuration?

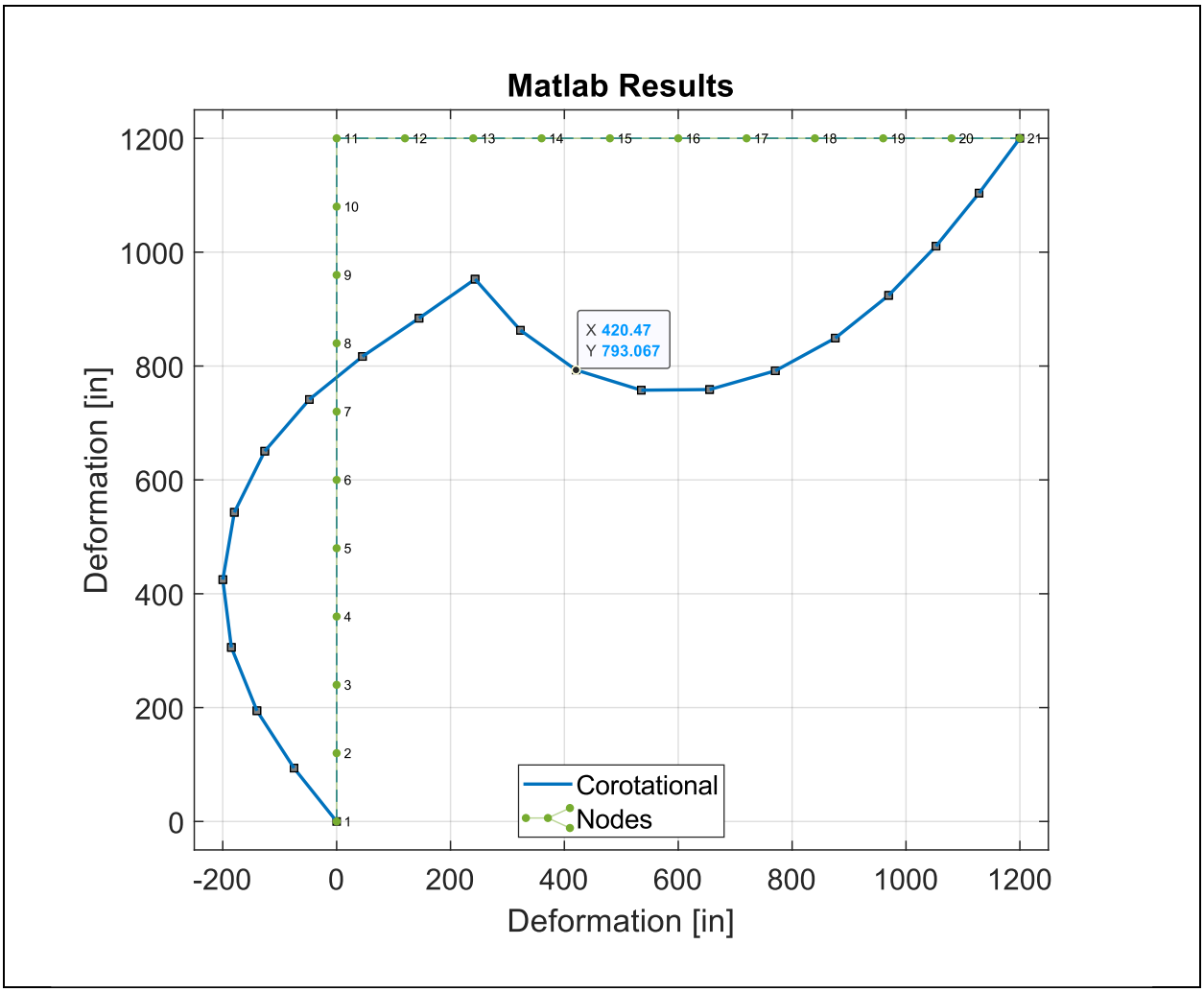

Figure 6 Final Deformed Configuration of Lee Frame with 18 kips applied

The reactions at the ends of node 1 and node 21 are given in the table below. The global equilibrium is also given below for the last load step in the analysis with 18 kips applied at node 13. The moment arm of the applied load is the total displacement of the node plus the original location of the node which is also shown in Figure 6. The summation of forces and moments show that the deformed configuration is in global equilibrium.

Table 3 Reactions at Supports

Equilibrium	Node 1	Node 21	
R_x	2051.1917	-2051.1917	[lbf]
R_y	13744.1441	4255.8554	[lbf]
M_o	-2.5611e-09	0.0000	[lbf-in]

Equilibrium		
$\rightarrow + F_x = 0$	$2051.1917 + (-2051.1917) = 0$	$= 0$
$\uparrow + F_y = 0$	$13744.144 + 4255.855 - 18000 = -0.001$	≈ 0
$\circlearrowleft + M_1 = 0$	$18000 \cdot (240 + 180.46981) + (2051.2 + 4255.9) \cdot 1200 = 0.0566$	≈ 0

For the member under the applied load, element equilibrium is also checked. In the following table the global forces of members 12 and 13 are given. Equilibrium is satisfied for a free body of a cut at node 13.

	U_1 [in]	U_2 [in]	U_3 [lbf-in]	U_4 [in]	U_5 [in]	U_6 [lbf-in]
Element 12	2.0512E+03	1.3744E+04	-2.6667E+06	-2.0512E+03	-1.3744E+04	4.1523E+06
Element 14	2.0512E+03	-4.2559E+03	-4.1523E+06	-2.0512E+03	4.2559E+03	3.7373E+06

Equilibrium		
$\rightarrow + F_x = 0$	$U_4^{12} + U_1^{13} = -2.0512\text{E}+03 + 2.0512\text{E}+03 = 0$	$= 0$
$\uparrow + F_y = 0$	$U_5^{12} + U_2^{13} - 18000 = 1.3744\text{E}+04 + 4.2559\text{E}+03 - 18000 = 0$	$= 0$
$\circlearrowleft + M_{13} = 0$	$U_6^{12} + U_3^{13} = 4.1523\text{E}+06 + -4.1523\text{E}+06 = 0$	$= 0$

Question 5. What is the Maximum Strain for a Critical Section?

Assuming that plane sections remain plane, the equation to compute the strain for any cross section is given by

$$\epsilon = \frac{1}{E} \cdot \left(\frac{N}{A} + \frac{My}{I}\right)$$

For the report, the values of $E = 72{,}000\ ksi$, $A = 600\text{in}^2$, $I = 20{,}000\text{in}^4$, $y = 10in$ were used in this calculation. N represents the axial force at a particular cross section and M represents the moment. The cross sections were assumed to be at every node. Only nodes 1-10 are in the column and node 11-20 are in the beam (node 21 is not shown as there was small values of moment and strain). From the analysis below, the axial force is shown to be constant for the linear analysis and the moment is a linear function. The maximum strain is also 1.45% at node 13 for the linear analysis which is half of the maximum strain found using the corotational analysis of 2.58% at node 13. In the corotational analysis, the axial force in the column and beam is not constant nor is the moment. This is more realistic in real structures where the geometry of a deformed configuration will lead to non-linear moment diagrams.

Table 4 Axial Force, Moment, and Strain at each Node for Linear Analysis

	Column				Beam		
Node	Axial Force	Moment	Strain	Node	Axial Force	Moment	Strain
1	-14823	0	-0.03%	11	-1223	1468131	1.02%
2	-14823	146813	0.07%	12	-1223	-310682	-0.22%
3	-14823	293626	0.17%	13	-1223	-2089495	-1.45%
4	-14823	440439	0.27%	14	-1223	-1828308	-1.27%
5	-14823	587253	0.37%	15	-1223	-1567121	-1.09%
6	-14823	734066	0.48%	16	-1223	-1305934	-0.91%
7	-14823	880879	0.58%	17	-1223	-1044747	-0.73%
8	-14823	1027692	0.68%	18	-1223	-783561	-0.55%
9	-14823	1174505	0.78%	19	-1223	-522374	-0.37%
10	-14823	1321318	0.88%	20	-1223	-261187	-0.18%

Table 5 Axial Force, Moment, and Strain at each Node for Corotational Analysis

	Column				Beam		
Node	Axial Force	Moment	Strain	Node	Axial Force	Moment	Strain
1	-9627	0	-0.02%	11	8107	-1049203	-0.71%
2	-10383	1078923	0.73%	12	5957	-2295537	-1.58%
3	-11592	2051619	1.40%	13	-2448	-3705732	-2.58%
4	-12710	2802502	1.92%	14	-1452	-3305283	-2.30%
5	-13174	3218346	2.20%	15	-504	-2840186	-1.97%
6	-12734	3224405	2.21%	16	304	-2347225	-1.63%
7	-11628	2819501	1.93%	17	936	-1851901	-1.28%
8	-10414	2076655	1.42%	18	1389	-1368084	-0.95%
9	-9638	1108596	0.75%	19	1678	-900216	-0.62%
10	-9616	31163	0.00%	20	1818	-446196	-0.31%

Question 6. Observed the Structural Behavior.

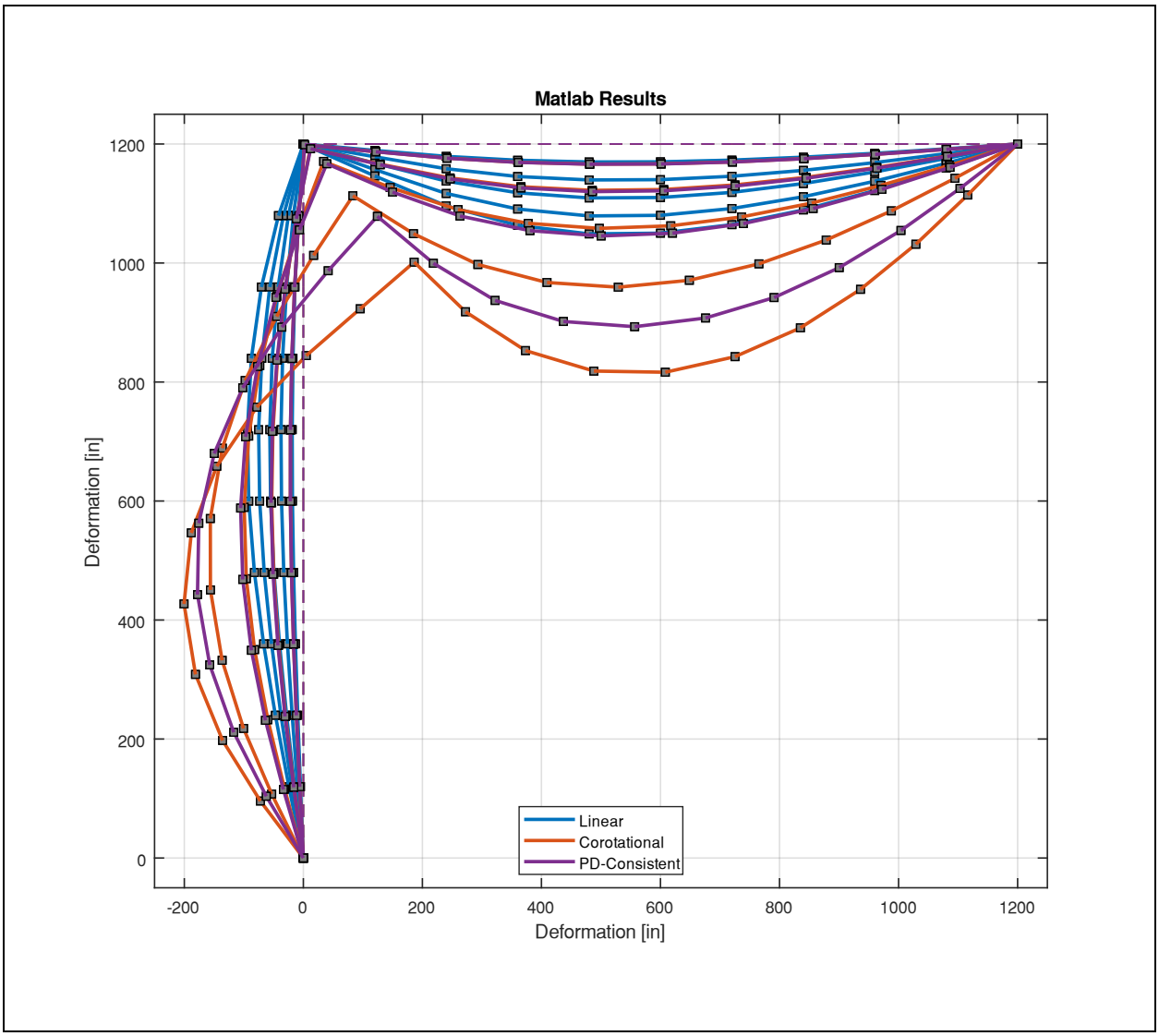

Figure 7 Observed Structural Deformations

While the P-Δ approximation has not be described it is shown along with the corotational and linear geometric deformed shape. The deformed shape of the Lee frame as prescribed by the corotational analysis shows a more accurate representation of the behavior of the column and beam of the frame. With the linear analysis, the column barely bowed out even though there was a large axial and moment couple at the top of the column. In the linear analysis, the corner node in the final configuration, displaced to the right 0.03398 inch and 0.4117 inch in the vertical direction downward. While the corotational formulation shows that the node actually displaces 186.1483 inch to the right and 198.5138 inches downward. While the observed structural behavior is not realistic for design, proper analysis of a structure at high loads should consider these additional displacements as caused by the deformed geometry of the structure.

It is noted that because of the decomposition of the deformation into the basic deformation and the rigid body deformations, the basic deformation still needs to be small or else it is necessary to include the Green-Lagrange strain tensor or the Von Karman strain term. For elements which are large, these terms are needed to properly account for the second order effects of the strains in different directions. However,

if the structure is subdivided into finite enough elements, the higher order terms can be simplified, and an almost linear local deformation can be written for the element. For the corotational formulation, the axial basic deformation over the original length of the element, D_1/L_o should be $\leq 10^{-2}$ and rotational basic deformations of the element, D_2 and D_3, should be 0.1 radians or $\cong$ 6 degrees.

In the Lee Frame it is consistent with literature that there is a snapback in the force displacement curve of the structure. A snapback is a described as a secondary path after a limit point characterized by two turning points and another limit state before it continues on its equilibrium path. In the Lee frame, after the structure exits the snapback, the structure stiffens immensely. The position before and after the snapback is show in Figure 8; both configurations of the structure is geometrically stable.

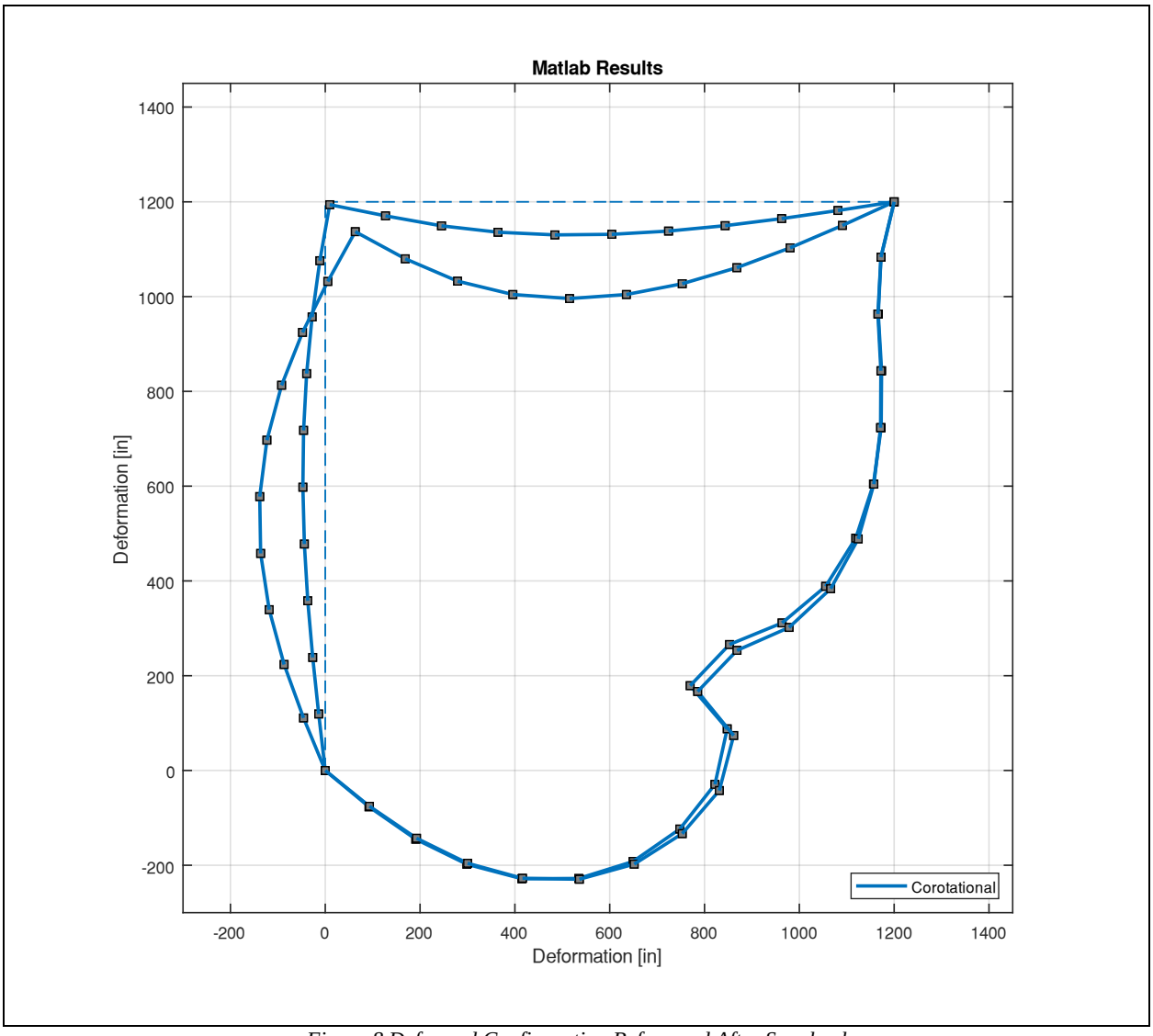

Figure 8 Deformed Configuration Before and After Snapback

Currently the corotational formulation is unable to solve the behavior of the system through the snapback phenomenon. It is shown in Figure 9 that there is a portion of the curve which the algorithm is unable to solve. This huge jump is because the algorithm was able to find the equilibrium path on the opposite side of the snapback curve. For the horizontal displacement, the curve was found to enter the snapback behavior around the 18,333 kip force step, and around 206 inches and for the vertical displacement

around 434 inches. If this is compared with the snapback curve found in Figure 10, the points where the curve enters the snapback is similar. The units are not properly accounted for but if the graph was considered unitless, the pre-snapback behavior and post-snapback behavior are considered similar! The problem with finding the peak of the force displacement curve is finding a load step small enough to find the peak but large enough to jump around the snapback phenomenon onto the equilibrium path on the other side.

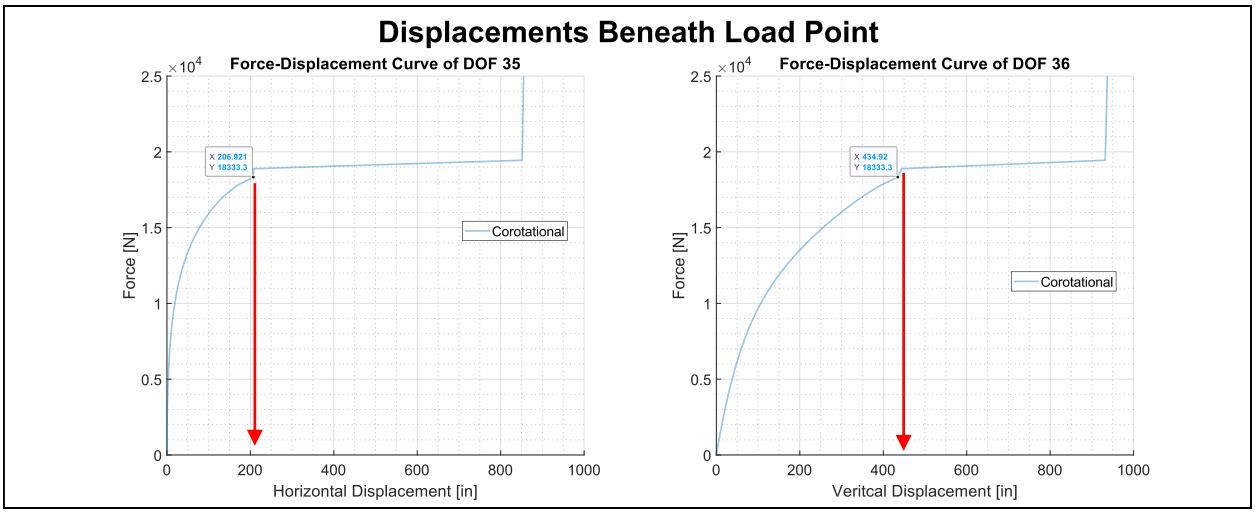

Figure 9 Force Displacement Curve of DOF Beneath Point Load

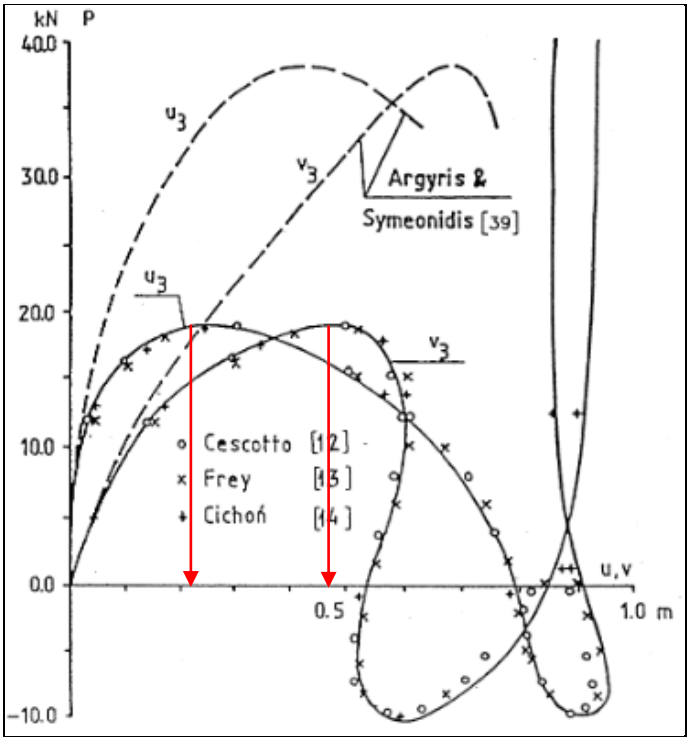

Figure 10 Force Displacement Curve from Literature

Part II – Nonlinear Geometric Analysis Using P-Δ Assumptions

Question 1. What is the PΔ Formulation?

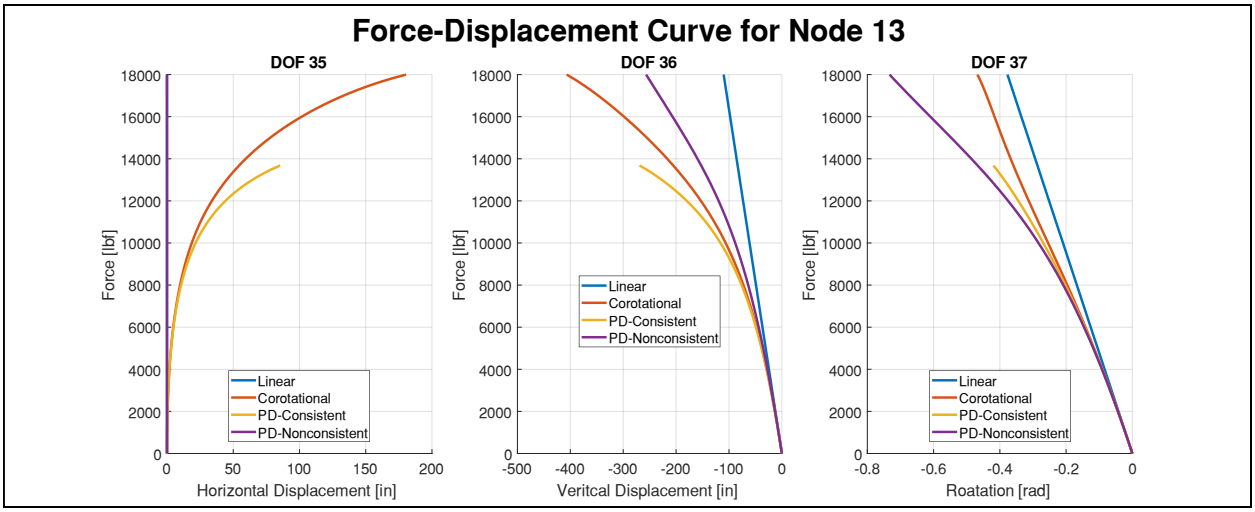

Figure 11 Load Displacement Curve of Node 13

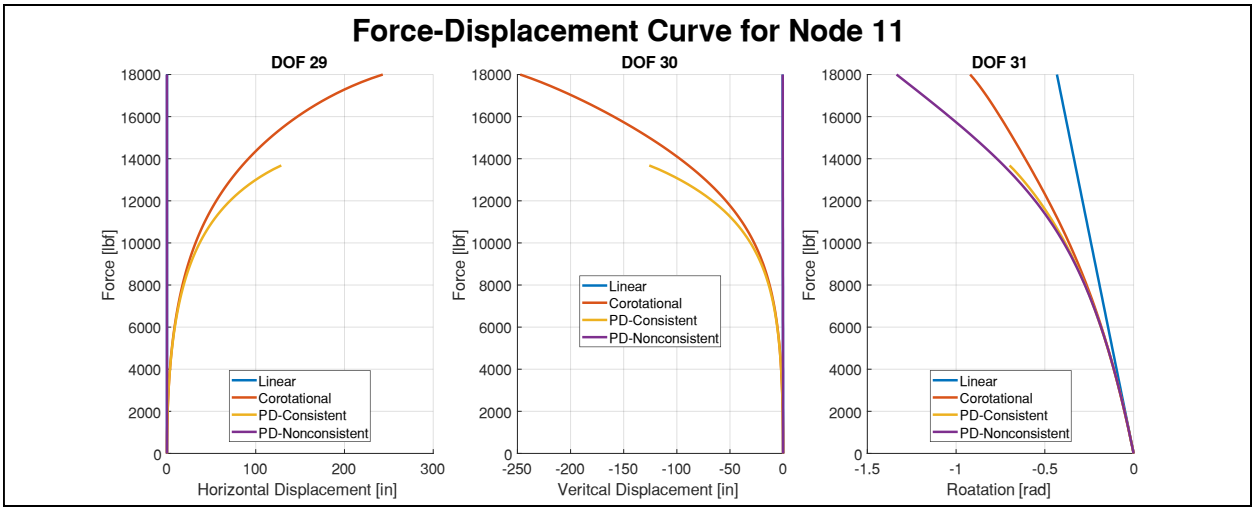

Figure 12 Load Displacement Curve of Node 11

For the different DOFs, it is shown that even though the P-Δ approximation is not as precise as the corotational approach, it proves to be more accurate than the linear formulation. The P-Δ formulation assumes that the transverse displacement field is small such that the $\sin(\beta) \approx 0$ and the shear produced by the basic deformation moments exist of the rigid body deformation geometry as opposed to the basic deformation. In this way, it still accounts for the axial stiffness of the element whereas the linear formulation does not. The main difference between the consistent and non-consistent P-Δ formulation is that the tangent stiffness matrix takes on more of a linear approximation, where the basic deformation is assumed to be <<.

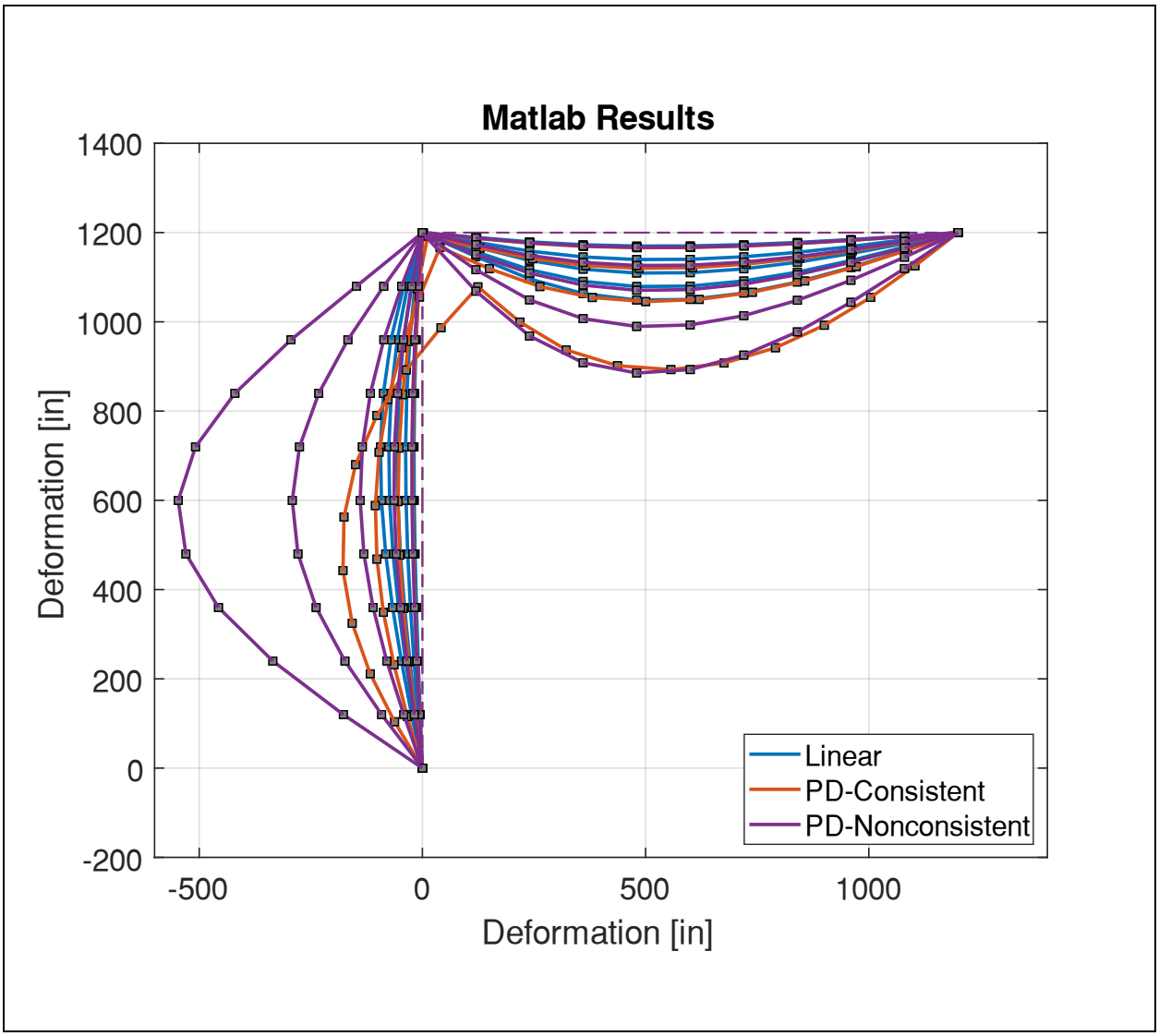

Figure 13 Deformed Shape Comparison of PΔ Approximation and Linear Geometric Analysis

In Figure 13, the linear results are compared with the PΔ consistent and non-consistent formulations. It is shown that the non-consistent formulation deforms the most. When compared with the results of the Opensees, the non-consistent results are nearly identical with the Opensees PΔ geometric analysis. The horizontal displacement of node 6 is -636.3132 inches while the horizontal displacement from Opensees was -636.4852 inches. The large horizontal displacement of this node comes from the weak horizontal resistance of the PΔ formulation in the non-consistent formulation. While the consistent PΔ formulation still considers the transverse stiffness in the rigid body mode transformation, the non-consistent does not, so as the structure deforms and takes on more lateral forces, the shape with the non-consistent formulation will bow out even more.

Part II – Opensees Analysis

Question 1. What are the Results of the Nonlinear Geometric Analysis Using Opensees?

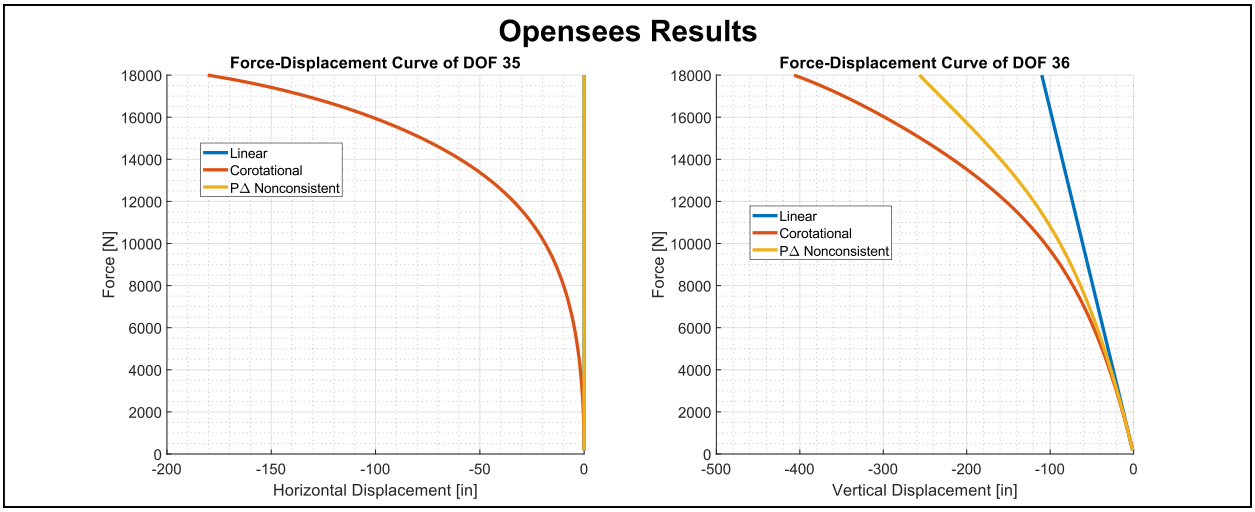

Figure 14 Opensees Load Displacement Curve of Node 13

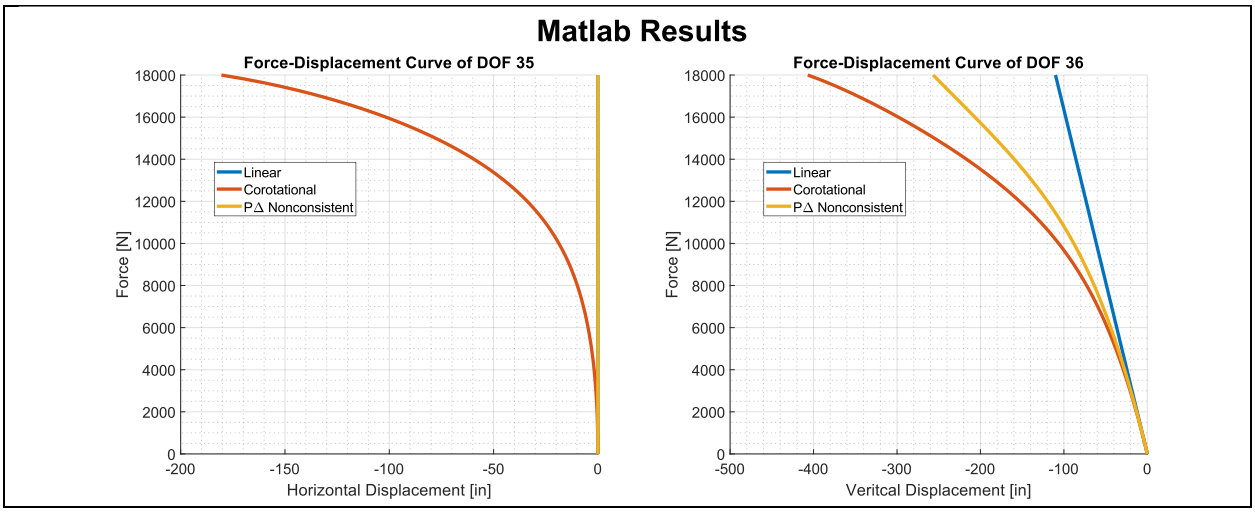

Figure 15 MATLAB Load Displacement Curve of Node 13

The force displacement curve for the MATLAB code and Opensees analysis are similar by inspection. A closer look in Table 6, Table 7, and Table 8 show that these results are less than 0.01% different overall. The formulation of the Opensees nonlinear geometric analysis is similar to the one that was programmed in MATLAB. Divergence of the results may arise from convergence criteria or precision difference in both programs. It was also necessary to change the convergence algorithm to the Modified Newton with initial elastic stiffness for convergence of the PΔ. The force displacement curve shows similar trends with the linear curve tangent to the two formulations for small displacement and forces. This shows again that while the PΔ formulation may not be accurate for higher loads, it is still computationally faster than the corotational formulation and more accurate than the linear formulation. Accounting for the deformed geometry can be done by the PΔ approximation.

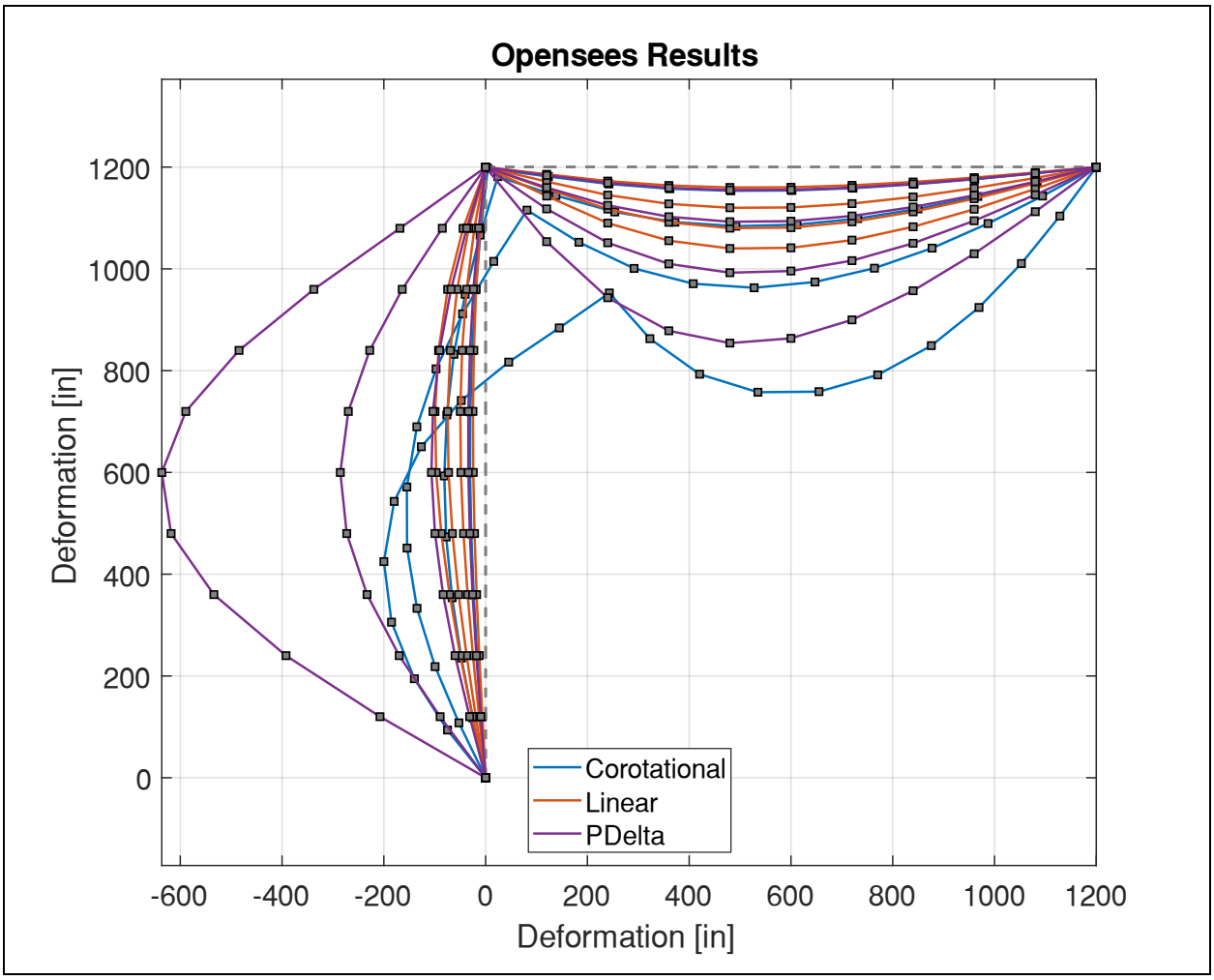

Figure 16 Deformed Shape Comparison of Nonlinear Geometric Analysis from Opensees

The deformed shapes for the Corotational, Linear and PΔ formulations from Opensees is shown in Figure 16. The results are tabulated in Table 6, Table 7, and Table 8 and show sufficient accuracy in the program as it relates to a more developed software like Opensees.

Table 6 Corotation Formulation Comparison Between MATLAB and Opensees

X-displacement			Y-displacement		
Opensees [in]	MATLAB [in]	% Difference	Opensees [in]	MATLAB [in]	% Difference
0.3237	0.3237	0.001%	-11.7692	-11.7693	0.000%
1.4372	1.4371	0.004%	-25.4193	-25.4198	-0.002%
3.6475	3.6472	0.007%	-41.7220	-41.7237	-0.004%
7.4498	7.4490	0.011%	-61.9428	-61.9471	-0.007%
13.6883	13.6861	0.016%	-88.1079	-88.1174	-0.011%
23.8182	23.8132	0.021%	-123.1423	-123.1610	-0.015%
40.1992	40.1894	0.024%	-170.2999	-170.3300	-0.018%
66.2442	66.2282	0.024%	-231.5967	-231.6380	-0.018%
107.2060	107.1854	0.019%	-307.6025	-307.6480	-0.015%
180.4670	180.4698	-0.002%	-406.9326	-406.9510	-0.005%

Table 7 PΔ Formulation Comparison Between MATLAB and Opensees

X-displacement			Y-displacement		
Opensees [in]	MATLAB [in]	% Difference	Opensees [in]	MATLAB [in]	% Difference
0.0027	0.0027	-0.001%	-11.5110	-11.5106	0.003%
0.0051	0.0051	0.000%	-24.2380	-24.2379	0.000%
0.0069	0.0069	0.000%	-38.6050	-38.6046	0.001%
0.0078	0.0078	0.001%	-55.2570	-55.2568	0.000%
0.0075	0.0075	-0.001%	-75.1790	-75.1791	0.000%
0.0050	0.0050	0.001%	-99.7790	-99.7789	0.000%
-0.0007	-0.0007	0.000%	-130.6500	-130.6455	0.003%
-0.0107	-0.0107	0.001%	-168.4700	-168.4699	0.000%
0.0027	0.0027	-0.001%	-11.5110	-11.5106	0.003%
0.0051	0.0051	0.000%	-24.2380	-24.2379	0.000%

Table 8 Linear Formulation Comparison Between MATLAB and Opensees

X-displacement			Y-displacement		
Opensees [in]	MATLAB [in]	% Difference	Opensees [in]	MATLAB [in]	% Difference
0.0029	0.0029	0.000%	-11.0053	-11.0053	0.000%
0.0058	0.0058	0.000%	-22.0106	-22.0106	0.000%
0.0086	0.0086	0.000%	-33.0160	-33.0160	0.000%
0.0115	0.0115	0.000%	-44.0213	-44.0213	0.000%
0.0144	0.0144	0.000%	-55.0266	-55.0266	0.000%
0.0173	0.0173	0.000%	-66.0319	-66.0319	0.000%
0.0202	0.0202	0.000%	-77.0372	-77.0372	0.000%
0.0230	0.0230	0.000%	-88.0425	-88.0425	0.000%
0.0259	0.0259	0.000%	-99.0479	-99.0479	0.000%
0.0288	0.0288	0.000%	-110.0532	-110.0530	0.000%

Appendix A. MATLAB Code

```
% Lee Frame
clear; clc; close;
format shortg
%% Pre-Processing
% Set up system geometry
[Struct.nodal_coord(:,1), Struct.nodal_coord(:,2)] = readvars('.\matlab\coordinates.txt');
Struct.num_nodes = length(Struct.nodal_coord);
% Set up connectivity
[Struct.element_connectivity(:,1), Struct.element_connectivity(:,2)] = readvars('.\matlab\connectivity.txt');
Struct.num_elems = length(Struct.element_connectivity);
% Set up structural DOF
Struct.node_dof = ones(Struct.num_nodes,3);
Struct.node_dof(1,1:2) = 0;
Struct.node_dof(21,1:2) = 0;
Struct = define_DOF_number(Struct);
% Set up ID array, gamma_rot, and element coordinates
Struct = define_orientaion(Struct);
% Set up material properties
Struct = define_section(Struct, 30, 20, 7.2e4);
% Set up Forces
dof_load = 36;
Struct = define_point_load(Struct, dof_load , -18000, 100);

%% Run analysis
clc;
tic;
% Linear, Corotational, PD-Consistent, PD-Nonconsistent
Analysis_1 = geomtric_analysis(Struct, 'Linear');
Analysis_2 = geomtric_analysis(Struct, 'Corotational');
Analysis_3 = geomtric_analysis(Struct, 'PD-Consistent');
Analysis_4 = geomtric_analysis(Struct, 'PD-Nonconsistent');
% In order to find the snap back- Increase load, larger load step
Struct = define_point_load(Struct, dof_load , -25000, 45);
Analysis_5 = geomtric_analysis(Struct, 'Corotational');
toc;
%% Post-Processing
% Deformed Shape
plot_figure_1([Analysis_1, Analysis_2], dof_load, 1);
plot_figure_2([Analysis_1, Analysis_2], dof_load, 2);
plot_figure_3([Analysis_1, Analysis_2], dof_load, 3);
plot_figure_4([Analysis_1, Analysis_2], 4, 25);
plot_figure_5(Analysis_2,5);
plot_figure_6([Analysis_1, Analysis_2, Analysis_3],5);
plot_figure_7(Analysis_5,7,dof_load);
plot_figure_8(Analysis_5,dof_load,8);
plot_figure_9([Analysis_1,Analysis_2, Analysis_3,Analysis_4], dof_load, 2);
plot_figure_10([Analysis_1,Analysis_2, Analysis_3,Analysis_4], dof_load, 2);
plot_figure_11([Analysis_1, Analysis_3,Analysis_4], dof_load, 20);
plot_figure_12([Analysis_1, Analysis_2,Analysis_4], dof_load);
% close all;

function plot_figure_1(list_of_analysis, dof_load, figNum)
    close all; clc
    plot_load_displacement_of_node(list_of_analysis, 13, dof_load, figNum);
    print_figure("01 RU Node 13- Corotational");
end
function plot_figure_2(list_of_analysis, dof_load, figNum)
```

```
    close all; clc
    plot_load_displacement_of_node(list_of_analysis, 11, dof_load, figNum);
    print_figure("02 RU Node 11- Corotational");
end
function plot_figure_3(list_of_analysis, dof_load, figNum)
    close all; clc
    sgtitle('Rotation at Supports','FontWeight','bold', 'Fontsize',36)
    subplot(1,2,1);
    plot_load_rotation(list_of_analysis, 1, dof_load,1);
    subplot(1,2,2);
    plot_load_rotation(list_of_analysis, 59, dof_load,1);
    print_figure("03 Rotation at Supports- Corotational");
end
function plot_figure_4(list_of_analysis, figNum, num_step)
    close all; clc;
    plot_deformed_shape(list_of_analysis, figNum, num_step);
    axis([-250,1250,-50,1250])
    print_figure("04 Matlab Deformed Shape Results- Corotational");
end
function plot_figure_5(list_of_analysis, figNum)
    close all; clc;
    num_step = 'end';
    for Analysis = list_of_analysis
       plot_deformed_shape(Analysis, figNum, num_step);
    end
    axis([-250,1250,-50,1250])
    print_figure("05 Finding Equlibrium- Corotational");
end
function plot_figure_6(list_of_analysis, figNum)
    close all; clc;
    plot_deformed_shape(list_of_analysis, figNum,20,0)
    axis([-250,1250,-50,1250])
    print_figure("06 Observe Structural Deformation- Corotational");
end
function plot_figure_7(list_of_analysis, figNum,dof_load)
    close all; clc;
    plot_deformed_shape(list_of_analysis, figNum,10,0);
    axis([-300,1450,-300,1450]);
    print_figure("07 Snapback Structural Deformation- Corotational");
end
function plot_figure_8(list_of_analysis, dof_load, figNum)
    close all; clc
    sgtitle('Displacements Beneath Load Point','FontWeight','bold', 'Fontsize',36)
    subplot(1,2,1);
    plot_load_displacement(list_of_analysis, 35, dof_load,1); xlabel("Horizontal Displacement [in]");
    subplot(1,2,2);
    plot_load_displacement(list_of_analysis, 36, dof_load,1); xlabel("Veritcal Displacement [in]");
    print_figure("08 Displacement of Snapback");
end
function plot_figure_9(list_of_analysis, dof_load, figNum)
    close all; clc
    plot_load_displacement_of_node(list_of_analysis, 13, dof_load, figNum);
    set(gca,'DefaultLineLineWidth',4)
    print_figure("09 RU Node 13- Pdelta");
end
function plot_figure_10(list_of_analysis, dof_load, figNum)
    close all; clc
```

```
115     plot_load_displacement_of_node(list_of_analysis, 11, dof_load, figNum);
116     set(gca,'DefaultLineLineWidth',4)
117     print_figure("10 RU Node 11- Pdelta");
118 end
119 function plot_figure_11(list_of_analysis, figNum, num_step)
120     close all; clc;
121     plot_deformed_shape(list_of_analysis, figNum, num_step,0);
122     axis([-600,1400,-200,1400])
123     print_figure("11 Matlab Deformed Shape Results- Pdelta");
124 end
125 function plot_figure_12(list_of_analysis, dof_load)
126     close all; clc
127     sgtitle('Matlab Results','FontWeight','bold', 'Fontsize',36)
128     subplot(1,2,1);
129     set(gca,'DefaultLineLineWidth',4)
130     plot_load_displacement(list_of_analysis, 35, dof_load,1); xlabel("Horizontal Displacement [in]");
131
132     legend('Linear','Corotational','P\Delta Nonconsistent');
133     subplot(1,2,2);
134     set(gca,'DefaultLineLineWidth',4)
135     plot_load_displacement(list_of_analysis, 36, dof_load,1); xlabel("Veritcal Displacement [in]");
136
137     legend('Linear','Corotational','P\Delta Nonconsistent');
138     print_figure("12 Matlab Dispalcement Node 13");
139 end
140
141
142 %% Main Functions
143 function Struct = geomtric_analysis(Struct, AnalysisType)
144     Struct.name = AnalysisType;
145     Struct.U_global =    zeros(Struct.num_dofs, Struct.num_loadsteps);
146     Struct.dU_global =  zeros(Struct.num_dofs, Struct.num_loadsteps);
147     Struct.F_internal = zeros(Struct.num_dofs, Struct.num_loadsteps);
148     Struct.Beta = zeros(Struct.num_elems, Struct.num_loadsteps);
149     Struct.L = zeros(Struct.num_elems, Struct.num_loadsteps);
150     max_iteration = 1000;
151     end_loadstep = Struct.num_loadsteps;
152     fprintf('\nStarting %s analysis',Struct.name)
153     % Main loop
154     for nload = 1:Struct.num_loadsteps
155         itn = 0;
156         error = 1;
157         while error > 1.e-5 && itn < max_iteration
158             % Structural State determination
159                 [K_struct, Struct] = get_struct_state(Struct, nload, AnalysisType);
160             % Calculate the unbalanced force
161                 P_unb = Struct.F_external(:,nload) - Struct.F_internal(:,nload);
162             % Update the displacements
163                 Struct.dU_global(:,nload) = K_struct\P_unb; % Calculate the delta in displacement
164                 Struct.U_global(:,nload) = Struct.U_global(:,nload) + Struct.dU_global(:,nload);  %Update total/current displ.
165                 itn = itn + 1; % Iteration counter
166                 error = norm(P_unb,2);
167                 Struct.unbalanced_force(nload, itn) = error;
168         end
169         if itn == max_iteration; end_loadstep = nload;fprintf('\nThe %s analysis did not fully converged. \nThe last converged step was %d', ↙
Struct.name, end_loadstep); break; end;
170         Struct.U_global(:,nload+1) = Struct.U_global(:,nload); % Seed the next global displacement
```

```
171         Struct.Beta(:,nload+1) =  Struct.Beta(:,nload); % Seed the next beta
172     end
173         fprintf('\nThe %s analysis was completed.\n',Struct.name)
174         Struct.end_loadstep = end_loadstep; % Grab the last load step
175         % Only save up to the last converged load steps
176         Struct.U_global = Struct.U_global(:,1:end_loadstep);
177         Struct.F_external = Struct.F_external(:,1:end_loadstep);
178         Struct.F_internal = Struct.F_internal(:,1:end_loadstep);
179         % Caculate the strain of the structure
180         Struct = calculate_strains(Struct,Struct.end_loadstep);
181 end
182 function [K_struct, Struct] = get_struct_state(Struct, nload, AnalysisType)
183     %% Extract Material Properties
184     E_ = Struct.Mat.E;
185     A_ = Struct.Mat.A;
186     I_ = Struct.Mat.I;
187     L0_ = Struct.Mat.L0;
188     ID_ = Struct.ID;
189     g_rot_ = Struct.g_rot;
190     Beta = Struct.Beta(:,nload);
191     U_global = Struct.U_global(:,nload);
192     dU_global = Struct.dU_global(:,nload);
193     % Initialize structural matrix and internal resisting force
194     K_struct = zeros(Struct.num_dofs, Struct.num_dofs); % Initializes the global strucural stiffness
195     F_internal = zeros(Struct.num_dofs, 1); % Initializes the internal resisting forces matrix
196     % Necessary functions
197     g_rbm_func = @(c,s,L) [-c, -s, 0, c,  s, 0; -s/L,  c/L, 1, s/L, -c/L, 0;-s/L,  c/L, 0, s/L, -c/L, 1];
198     g_rbm_pd_func = @(d, L) [-1, -d/L, 0, 1,  d/L, 0; 0,  1/L, 1, 0, -1/L, 0; 0,  1/L, 0, 0, -1/L, 1];
199     k_basic_func = @(E,A,I,L) [E*A/L, 0, 0; 0, 4*E*I/L, 2*E*I/L; 0, 2*E*I/L, 4*E*I/L];
200     axial_func = @(s,c) [ s^2, -s*c, 0,-s^2,s*c,0; -s*c,c^2, 0, s*c -c^2, 0; 0,0,0,0,0,0; -s^2,s*c,0,s^2,-s*c,0;s*c,-c^2,0,-s*c,c^2,0; 0,0,0,0,0,0];
201     tran_func = @(s,c) [-2*s*c, c^2-s^2,0,  2*s*c,-c^2+s^2,0; c^2-s^2,2*s*c,0,-c^2+s^2,-2*s*c,0; 0,0,0,0,0,0; 2*s*c,-c^2+s^2,0,-2*s*c,c^2-s^2, ↙
0;-c^2+s^2,-2*s*c,0,c^2-s^2,2*s*c,0; 0,0,0,0,0,0];
202     k_tan_g_func = @(Q,L,s,c) Q(1)/L*axial_func(s,c) + (Q(2)+Q(3))/L^2*tran_func(s,c);
203     %% Loop over each Element
204     for ele_no = 1:Struct.num_elems
205         % Get the properties of the current element
206             E = E_(ele_no);
207             I = I_(ele_no);
208             A = A_(ele_no);
209             L0 = L0_(ele_no);
210             g_rot = g_rot_(:,:,ele_no);
211             ID = ID_(ele_no,:);
212             beta = Beta(ele_no);
213         % Get the element displacement
214             u_global = get_U(U_global,ID); % Extract element displacement from global matrix
215             du_global = get_U(dU_global, ID); % Extract element displacement from global matrix
216             u_local = g_rot*u_global'; % Transform to local displacement with the gamma_rot
217             du_local = g_rot*du_global'; % Transform to local displacement with the gamma_rot
218         % Define the orientation of the element
219             dUx1 = u_local(4) - u_local(1);
220             dUx2 = u_local(5) - u_local(2);
221             L = ((L0 + dUx1)^2 +(dUx2)^2)^0.5; % Sqrt(Change)
222             Struct.L(ele_no,nload) = L;
223             c = (L0 + dUx1)/L; % Definition of cos
224             s = dUx2/L; % Definition of cos
225         % Find rotation of element based on previous rotation
226             dbeta_dU = [dUx2/L^2; -(L0+dUx1)/L^2; 0; -dUx2/L^2; (L0+dUx1)/L^2; 0];
```

```
227         dbeta = sum(dbeta_dU.*du_local);
228         beta = beta + dbeta;
229         Beta(ele_no) = beta;
230 %           beta = atan(dUx2/(L0+dUx1)); % Can also be used but will not
231 %           be able to handle >90 deg rotations
232       % Analyze the structure
233         switch AnalysisType
234         case 'Corotational'
235             g_rbm = g_rbm_func(c,s,L); % Calculate the rigid body mode transformation
236             u_basic = [L - L0; u_local(3) - beta; u_local(6) - beta]; % Calculate basic deformation
237             k_basic = k_basic_func(E, A, I, L); % Calculate the basic stiffness matrix
238             f_basic = k_basic * u_basic; % Calculate the basic forces
239             f_global = g_rot' * g_rbm' *f_basic ; % Transform to global forces
240             k_tan_g = k_tan_g_func(f_basic, L,s,c); % Calcualte the geometric tangent stiffness
241             k_tan_m = g_rbm'*k_basic*g_rbm; % Calculate the material tangent stiffness
242             k_local = k_tan_g + k_tan_m; % Calculate the tangent local stiffness
243             k_global = g_rot' * k_local * g_rot; % Transform to global stiffness
244         case 'PD-Consistent'
245             g_rbm = g_rbm_pd_func(dUx2, L0); % Calculate the rigid body mode transformation for PD
246             u_basic = [dUx1 + dUx2^2/2/L0; u_local(3) - dUx2/L0; u_local(6) - (dUx2)/L0]; % Calculate basic deformation
247             k_basic = k_basic_func(E, A, I, L0); % Calculate the basic stiffness matrix
248             f_basic = k_basic * u_basic;
249             f_global = g_rot' *g_rbm' *f_basic  ; % Transform to global forces
250             k_tan_g = f_basic(1)/L0*axial_func(s,c);
251             k_local = k_tan_g + g_rbm' * k_basic * g_rbm;
252             k_global = g_rot' * k_local * g_rot; % Transform to global stiffness
253         case 'PD-Nonconsistent'
254             g_rbm_lin = g_rbm_func(1,0,L0); % Calculate the rigid body mode transformation for PD
255             g_rbm = g_rbm_pd_func(dUx2, L0);
256             u_basic = [dUx1; u_local(3) - dUx2/L0; u_local(6) - (dUx2)/L0]; % Calculate basic deformation
257             k_basic = k_basic_func(E, A, I, L0); % Calculate the basic stiffness matrix
258             f_basic = k_basic * u_basic;
259             f_global = g_rot' * g_rbm' * f_basic; % Transform to global forces
260             k_tan_g = f_basic(1)/L0*axial_func(s,c);
261             k_local = k_tan_g + g_rbm_lin' * k_basic * g_rbm_lin; %
262 %               Another PD- assumption that is equivalent
263 %               k_local = k_tan_g + g_rbm' * k_basic * g_rbm_lin;
264             k_global = g_rot' * k_local * g_rot; % Transform to global stiffness
265         case 'Linear'
266             g_rbm = g_rbm_func(1,0,L0); % Calculate the rigid body mode transformation Linear Approx
267             u_basic = g_rbm * u_local; % Calculate basic deformation from global
268             k_basic = k_basic_func(E, A, I, L0); % Caculate basic stiffness matrix
269             f_basic =  k_basic * u_basic;
270             f_global = g_rot' * g_rbm' * f_basic; % Transform to global forces
271             k_global = g_rot' * g_rbm' * k_basic * g_rbm * g_rot; % Transform to global stiffness
272         end
273         Struct.F_global(ele_no,:,nload) = f_global; % Store global force vectors for each element
274         Struct.F_basic(ele_no,:,nload) = f_basic; % Store basic force vectors for each element
275         % Assemble into larger matrix
276         F_internal = add_to_F_internal(F_internal, f_global, ID); % Assemble into global internal resisting force
277         K_struct = add_to_K_struct(K_struct, k_global, ID); % Assemble into structural stiffness
278     end
279     Struct.F_internal(:,nload) = F_internal; % Save the internal resisting force for the load step
280     Struct.Beta(:,nload) = Beta; % Saves the beta of the current load step
281 end
282 %% Preprocessing Functions
283 function Struct = define_DOF_number(Struct)
```

```
284     dof_num = 1;
285     for n = 1:Struct.num_nodes
286         if (Struct.node_dof(n,1)~=0); Struct.node_dof(n,1) = dof_num; dof_num = dof_num +1; end
287         if (Struct.node_dof(n,2)~=0); Struct.node_dof(n,2) = dof_num; dof_num = dof_num +1; end
288         if (Struct.node_dof(n,3)~=0); Struct.node_dof(n,3) = dof_num; dof_num = dof_num +1; end
289     end
290     Struct.num_dofs = dof_num-1;
291 end
292
293 function Struct = define_orientaion(Struct)
294     g_rot_func =@(c,s) [c s 0 0 0 0; -s c 0 0 0 0; 0 0 1 0 0 0; 0 0 0 c s 0; 0 0 0 -s c 0; 0 0 0 0 0 1];
295     for n = 1:Struct.num_elems
296     % Setting ID array
297         node_i = Struct.element_connectivity(n,1); % Get 1st node of element n
298         node_j = Struct.element_connectivity(n,2); % Get 2nd node of element n
299         Struct.ID(n,:) = [Struct.node_dof(node_i,:), Struct.node_dof(node_j,:)]; % Set the DOF number of element n
300     % Setting XY
301         P1 = Struct.nodal_coord(node_i,:); % Get coordinates for node i
302         P2 = Struct.nodal_coord(node_j,:); % Get coordinates for node j
303         Struct.element_coord(n,:) = [P1, P2]; % Set element n coordinates to [X1 Y1 X2 Y2]
304     % Setting Orientation
305         Struct.Mat.L0(n) = ((P2(2)-P1(2))^2 +(P2(1)-P1(1))^2)^0.5; % Original Length of each member
306         c0 = (P2(1)-P1(1))/Struct.Mat.L0(n); % cosine of each memeber
307         s0 = (P2(2)-P1(2))/Struct.Mat.L0(n); % sine of each member
308         Struct.g_rot(:,:,n) = g_rot_func(c0, s0); % Gamma_Rotation
309     end
310 end
311
312 function Struct = define_section(Struct, b, h, E)
313     Struct.Mat.b = b*ones(1,Struct.num_elems); % in; width
314     Struct.Mat.h = h*ones(1,Struct.num_elems); % in; height
315     Struct.Mat.E = E*ones(1,Struct.num_elems); % [ksi] Young's modulus
316     Struct.Mat.A = b*h*ones(1,Struct.num_elems); % [in^2]
317     Struct.Mat.I = b*h^3/12*ones(1,Struct.num_elems); % [in^4] moment of inertia
318 end
319
320 function Struct = define_point_load(Struct, node, load, steps)
321     Struct.num_loadsteps = steps;
322     Struct.F_external = zeros(Struct.num_dofs, steps);
323     Struct.F_external(node,1:end)= load/steps .* (1:steps);
324 end
325 %% Analysis Functions
326 function Struct = calculate_strains(Struct, load_step)
327     E = [Struct.Mat.E, Struct.Mat.E(end)];
328     I = [Struct.Mat.I, Struct.Mat.I(end)]';
329     A = [Struct.Mat.A, Struct.Mat.A(end)]';
330     y = [Struct.Mat.h/2,Struct.Mat.h(end)/2]';
331     N = [Struct.F_basic(:,1,load_step);Struct.F_basic(end,1,load_step)]; % Assumes last element =
332     M = [Struct.F_basic(:,2,load_step);Struct.F_basic(end,3,load_step)];% Grabs last element moment
333     Struct.N = N; Struct.M = M;
334     Struct.Strain = 1/E'.*(N./A+M.*y./I);
335     Struct.Strain = Struct.Strain(:,1);
336 end
337
338 % Extraction of Element Displacement
339 function u_global = get_U(U_global, ID)
340     u_global = zeros(1,6);
```

```
341     for i=1:6
342         if (ID(i)~=0)
343             u_global(i) = U_global(ID(i));
344         end
345     end
346 end
347 % Assmeble Structural Stiffness Matrix
348 function K_struct = add_to_K_struct(K_struct, k_global, ID)
349     for i=1:6
350         for j=1:6
351             if ((ID(i)~=0) && (ID(j)~=0))
352                 K_struct(ID(i),ID(j)) = K_struct(ID(i),ID(j)) + k_global(i,j);
353             end
354         end
355     end
356 end
357 % Assemble Internal Resisting Force
358 function F_internal = add_to_F_internal(F_internal, f_internal, ID)
359     for i=1:6
360         if (ID(i)~=0)
361             F_internal(ID(i)) = F_internal(ID(i)) + f_internal(i);
362         end
363     end
364 end
365
366 %% Plotting Functions
367 %% Plot Load Displacement Function
368 function plot_load_displacement(list_of_analysis, dof_num, dof_load,figNum)
369     figure(figNum);  hold on; title(["Force-Displacement Curve of DOF " + string(dof_num)]);
370     for Analysis = list_of_analysis
371         displacement_history = [0, Analysis.U_global(dof_num,:)];
372         force_history = -[0,Analysis.F_external(dof_load,:)];
373         p = plot(displacement_history,force_history,...
374                 'DisplayName',Analysis.name);
375 %         p.Color(4) = 0.4;
376     end
377     set(gcf,'Position',[0,0,2000,800]);set(gca,'DefaultLineLineWidth',4)
378     xlabel('Displacement [in]'); ylabel('Force [N]');
379     grid on; grid minor; legend('Location','Best');
380 end
381 %
382 function plot_load_rotation(list_of_analysis, dof_num, dof_load,figNum)
383     figure(figNum);  hold on;
384     title(["Force-Displacement Curve of DOF " + string(dof_num)]);
385     for Analysis = list_of_analysis
386         displacement_history = abs([0, Analysis.U_global(dof_num,:)]);
387         force_history = -[0,Analysis.F_external(dof_load,:)];
388         p = plot(displacement_history,force_history,...
389                 'DisplayName',Analysis.name,...
390                 'LineWidth',2);
391 %         p.Color(4) = 0.4;
392     end
393     set(gcf,'Position',[0,0,2000,800]);
394     xlabel('Rotatation [rad]'); ylabel('Force [lbf]');
395     grid on; grid minor; legend('Location','Best');
396 end
397 %
```

```
function plot_load_displacement_of_node(list_of_analysis, node_num, dof_load, figNum)
    figure(figNum); hold on;
    sgtitle(["Force-Displacement Curve for Node " + string(node_num)],...
        'FontWeight','bold', 'Fontsize',36);
    for Analysis = list_of_analysis
        i = 1;
        for dof = Analysis.node_dof(node_num,:)
            subplot(1,3,i); hold on;
            displacement_history = [0, Analysis.U_global(dof,:)];
            force_history = -[0,Analysis.F_external(dof_load,:)];
            p = plot(displacement_history,force_history,...
                'DisplayName',Analysis.name,...
                'LineWidth',3);
%             p.Color(4) = 0.4;
            title("DOF " + string(dof));
            if i == 3; xlabel('Roatation [rad]'); elseif i==2; xlabel('Veritcal Displacement [in]'); else; xlabel('Horizontal Displacement [in]'); end;
            grid on; grid minor; legend('Location','best');ylabel('Force [lbf]');
            i = i +1;
        end
    end
    set(gcf,'Position',[0,0,2000,800]);
end
%% Plot Deformed Shape Function
function plt = plot_deformed_shape(list_of_analysis, figNum, num_step, addnode)
    arguments list_of_analysis; figNum; num_step; addnode= true; end
    scale = 1;
    color = ["#0072BD", "#D95319", "#7E2F8E","#4DBEEE"]; i = 1;
    for Analysis = list_of_analysis
        if i == 1; linewidth = 2; else; linewidth = 2; end;
        if num_step == 'end'; lsPlot = Analysis.end_loadstep; else lsPlot = num_step:num_step:Analysis.end_loadstep;end
        [~, ~, ~, plt ] = Plot_ANY_Deflected_Shape(Analysis.U_global, scale, Analysis.ID', Analysis.element_coord, lsPlot, color(i), 1, color(i), ↙
linewidth, figNum, Analysis.name); i = i+1;
    end
    title("Matlab Results"); set(gcf,'Position',[0,0,900,800]);
    legend('Location','Southeast'); xlabel("Deformation [in]"); ylabel("Deformation [in]");
    if addnode ==true; addnodes(list_of_analysis(1),figNum); end
end

function addnodes(Analysis, figNum)
    figure(figNum);
    G = graph(Analysis.element_connectivity(:,1), Analysis.element_connectivity(:,2));
    P = plot(G,'xdata', Analysis.nodal_coord(:,1),'ydata',Analysis.nodal_coord(:,2));
    P.DisplayName = 'Nodes';
end
%% Print Current Figure Funciton
function print_figure(file_name)
    % Saves the figures in a consistent manner
    orient(gcf,'landscape');
    folder = '.\figures\';
    name = 'Figure ' +string(file_name);
%     print(folder+name,'-dpdf','-fillpage','-PMicrosoft Print to PDF','-r600','-painters')
    print(folder+name,'-dsvg','-PMicrosoft Print to PDF','-r600','-painters')
end
```

```
function [ fig, undefPlot, defPlot ] = Plot_ANY_Deflected_Shape(U_glob, scale, ID, XY, lsPlot, undefColor, undefLineWidth, defColor, defLineWidth, figNum, name)
%% Plot Deflected Shape for ANY 2-D Frame Structure
% Angshuman Deb
% Please read the document 'Plotting_Deflected_Shapes_README.pdf' before
% using this function.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

nlsPlot = length(lsPlot);
ndf = size(ID,1);
n_ele = size(ID,2);
U_ele = zeros(ndf,nlsPlot,n_ele);

for ls = 1:nlsPlot
    for i = 1:n_ele
        for m = 1:ndf
            if ID(m,i) ~= 0
                U_ele(m,ls,i) = scale*U_glob(ID(m,i),lsPlot(ls));
            end
        end
    end
end

fig = figure(figNum);

hold on;
grid on;
box on;
% Undeformed Shape
for i = 1:n_ele
    undefPlot = ...
        plot([XY(i,1) XY(i,3)], [XY(i,2) XY(i,4)], 'LineStyle' ,'--',...
        'LineWidth', undefLineWidth, 'Color', undefColor,'HandleVisibility','off');
end
% Deformed Shape
for ls = 1:nlsPlot
    figure(figNum)
    for i = 1:n_ele
        plot([XY(i,1)+U_ele(1,ls,i) XY(i,3)+U_ele(4,ls,i)],...
            [XY(i,2)+U_ele(2,ls,i) XY(i,4)+U_ele(5,ls,i)],...
            'ks','LineWidth',1,'MarkerFaceColor',[0.5 0.5 0.5],'HandleVisibility','off');
        defPlot = ...
            plot([XY(i,1)+U_ele(1,ls,i) XY(i,3)+U_ele(4,ls,i)],...
            [XY(i,2)+U_ele(2,ls,i) XY(i,4)+U_ele(5,ls,i)],...
            'LineStyle', '-', 'LineWidth', defLineWidth, 'Color', defColor,'HandleVisibility','off');
%           defPlot.Color(4) = 0.4;
    end
end
axis equal
defPlot.HandleVisibility ='on';
defPlot.DisplayName = name;
legend();
end
```

1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9
9	10
10	11
11	12
12	13
13	14
14	15
15	16
16	17
17	18
18	19
19	20
20	21

0	0
0	120
0	240
0	360
0	480
0	600
0	720
0	840
0	960
0	1080
0	1200
120	1200
240	1200
360	1200
480	1200
600	1200
720	1200
840	1200
960	1200
1080	1200
1200	1200

```
clc;
filename ="C:\Users\Louis Lin\Workspace\Academic\UCSD\SE 201B\HW\HW2\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);

%%
close all; clc;
list = ["Corotational","Linear","PDelta"];
i =1;
color = ["#0072BD", "#D95319", "#7E2F8E","#4DBEEE"];
set(groot,'DefaultAxesFontSize', 18)
for analysistype = list
    modelDataDirPath = './opensees/'+analysistype+'/Model';
    deflectedShapeDirPath = './opensees/'+analysistype+'/DeflectedShape';
    nDim = 2;
    loadStep = 25:25:100;
    defScale = 1;
    crdTransfMatrix = eye(2);
    defColor = color(i);
    figNum = 1; i = i+1;
    undefColor = [0.5 0.5 0.5];
    undefLineWidth = 1.5;
    defLineWidth = 1.5;
%     plot_lineModel(modelDataDirPath, nDim, crdTransfMatrix, undefLineWidth, figNum);
    [~, ~, plots] = plot_lineDeflectedShape(modelDataDirPath, deflectedShapeDirPath, nDim, loadStep, defScale, crdTransfMatrix, undefColor, ↙
undefLineWidth, defColor, defLineWidth, figNum);
    plots.HandleVisibility = 'on';
    plots.DisplayName = analysistype;
    title("Opensees Results");
    xlabel("Deformation [in]"); ylabel("Deformation [in]");
    legend('Location','south');
     set(gcf, 'position', [1400 0 1000 800])
end
%%
[DOF35_Linear, DOF36_Linear] = readvars("./opensees/Linear/DeflectedShape/dispNode_13.txt");
[DOF35_Cor, DOF36_Cor] = readvars("./opensees/Corotational/DeflectedShape/dispNode_13.txt");
[DOF35_PDelta,  DOF36_PDelta] = readvars("./opensees/PDelta/DeflectedShape/dispNode_13.txt");
force = [-180,-360,-540,-720,-900,-1080,-1260,-1440,-1620,-1800,-1980,-2160,-2340,-2520,-2700,-2880,-3060,-3240,-3420,-3600,-3780,-3960, ↙
-4140,-4320,-4500,-4680,-4860,-5040,-5220,-5400,-5580,-5760,-5940,-6120,-6300,-6480,-6660,-6840,-7020,-7200,-7380,-7560,-7740,-7920, ↙
-8100,-8280,-8460,-8640,-8820,-9000,-9180,-9360,-9540,-9720,-9900,-10080,-10260,-10440,-10620,-10800,-10980,-11160,-11340,-11520,-11700, ↙
-11880,-12060,-12240,-12420,-12600,-12780,-12960,-13140,-13320,-13500,-13680,-13860,-14040,-14220,-14400,-14580,-14760,-14940,-15120, ↙
-15300,-15480,-15660,-15840,-16020,-16200,-16380,-16560,-16740,-16920,-17100,-17280,-17460,-17640,-17820,-18000];

close all;
figure(1);
sgtitle('Opensees Results','FontWeight','bold', 'Fontsize',36)
subplot(1,2,1);hold on;

title("Force-Displacement Curve of DOF 35");
set(gca,'DefaultLineLineWidth',4)
p = plot(-abs(DOF35_Linear), -force(1:numel(DOF35_Linear)),'DisplayName','Linear');
```

```
53 p = plot(-abs(DOF35_Cor), -force(1:numel(DOF35_Cor)),'DisplayName','Corotational');
54 p = plot(-abs(DOF35_PDelta), -force(1:numel(DOF35_PDelta)),'DisplayName','P\Delta Nonconsistent');
55 set(gca,'DefaultLineLineWidth',3)
56 xlabel('Horizontal Displacement [in]'); ylabel('Force [N]');
57 grid on; grid minor; legend('Location','Best');
58
59 subplot(1,2,2); hold on;
60 title("Force-Displacement Curve of DOF 36");
61 set(gca,'DefaultLineLineWidth',4)
62 p = plot(-abs(DOF36_Linear), -force(1:numel(DOF35_Linear)),'DisplayName','Linear');
63 p = plot(-abs(DOF36_Cor), -force(1:numel(DOF35_Cor)),'DisplayName','Corotational');
64 p = plot(-abs(DOF36_PDelta), -force(1:numel(DOF35_PDelta)),'DisplayName','P\Delta Nonconsistent');
65 set(gca,'DefaultLineLineWidth',3)
66 xlabel('Vertical Displacement [in]'); ylabel('Force [N]');
67 grid on; grid minor; legend('Location','Best');
68
69 set(gcf,'Position',[0,0,2000,800]);
```

Appendix B. Opensees

```tcl
# ##########################################################################################
# SE 201B: NONLINEAR STRUCTURAL ANALYSIS (WI 2021)
# Lee Frame Analysis
# ##########################################################################################

# ------------------------------------------------------------------------------------------
# DEFINE NODES
# ------------------------------------------------------------------------------------------
set beamLength [expr 1200.*$in];
set numEleBeam 10.;
set eleLengthBeam [expr $beamLength/$numEleBeam];

set colLength [expr 1200.*$in];
set numEleCol 10.;
set eleLengthCol [expr $colLength/$numEleCol];

set nodeList {};
set nodeCtr 0;

for {set i 1} {$i <= [expr $numEleCol + 1]} {incr i 1} {
    incr nodeCtr 1
    node $nodeCtr 0.0 [expr ($i-1)*$eleLengthCol];
    puts $modelExportFileID "node $nodeCtr 0.0 [expr ($i-1)*$eleLengthCol];"
    lappend nodeList $nodeCtr
}
for {set i 1} {$i <= $numEleBeam} {incr i 1} {
    incr nodeCtr 1
    node $nodeCtr [expr $i*$eleLengthBeam] $colLength;
    puts $modelExportFileID "node $nodeCtr [expr $i*$eleLengthBeam] $colLength;"
    lappend nodeList $nodeCtr
}

# ------------------------------------------------------------------------------------------
# DEFINE CONSTRAINTS
# ------------------------------------------------------------------------------------------
fix [lindex $nodeList 0]   1 1 0;
fix [lindex $nodeList end] 1 1 0;

# ------------------------------------------------------------------------------------------
# DEFINE MATERIAL & SECTION PARAMETERS
# ------------------------------------------------------------------------------------------
set b       [expr 30.*$in];
set d       [expr 20.*$in];

set E       [expr 72000.*$ksi];
set ABeam   [expr $b*$d];
set ACol    [expr $b*$d];
set IzBeam  [expr $b*$d**3/12.];
set IzCol   [expr $b*$d**3/12.];

# ------------------------------------------------------------------------------------------
# DEFINE GEOMETERIC TRANSFORMATION FOR ELEMENTS
# ------------------------------------------------------------------------------------------
set transfTypeBeam "Corotational"; # Corotational, Linear, PDelta
set transfTypeCol "Corotational";
set transfTagBeam 3;
set transfTagCol 4;
# For a two-dimensional problem:
# geomTransf $transfType $transfTag <-jntOffset $dXi $dYi $dXj $dYj>
geomTransf $transfTypeBeam $transfTagBeam;
geomTransf $transfTypeCol $transfTagCol;

# ------------------------------------------------------------------------------------------
# DEFINE ELEMENT
# ------------------------------------------------------------------------------------------
# For a two-dimensional problem:
# element elasticBeamColumn $eleTag $iNode $jNode $A $E $Iz $transfTag
set nodeCtr 0;
set eleCtr 0;
```

```tcl
for {set i 1} {$i <= [expr $numEleCol]} {incr i 1} {
    incr nodeCtr 1
    incr eleCtr 1
    element elasticBeamColumn $eleCtr $nodeCtr [expr $nodeCtr + 1] $ACol $E $IzCol $transfTagCol;
    puts $modelExportFileID "element elasticBeamColumn $eleCtr $nodeCtr [expr $nodeCtr + 1] $ACol $E $IzCol $transfTagCol;"
}
for {set i 1} {$i <= [expr $numEleBeam]} {incr i 1} {
    incr nodeCtr 1
    incr eleCtr 1
    element elasticBeamColumn $eleCtr $nodeCtr [expr $nodeCtr + 1] $ABeam $E $IzBeam $transfTagBeam;
    puts $modelExportFileID "element elasticBeamColumn $eleCtr $nodeCtr [expr $nodeCtr + 1] $ABeam $E $IzBeam $transfTagBeam;"
}
close $modelExportFileID

# ------------------------------------------------------------------------------
# DEFINE TIME SERIES AND LOAD PATTERN
# ------------------------------------------------------------------------------
set tsTag 2; # Tag for the time series
timeSeries Linear $tsTag

set patternTag 1; # Tag for the load pattern
pattern Plain $patternTag $tsTag {
    load 13 0. $loading 0.;
}

```

```
# ##########################################################################################
# SE 201B: NONLINEAR STRUCTURAL ANALYSIS (WI 2021)
# OpenSees Tutorial # 3
# 2D FRAME ANALYSIS (LINEAR MATERIAL + NONLINEAR GEOMETRY) EXAMPLE
# ##########################################################################################


for {set i 1} {$i <= [llength $nodeList]} {incr i 1} {
    set dispNodeTag [lindex $nodeList [expr $i-1]];
    set dispNodeFile "dispNode_$dispNodeTag.txt"
    recorder Node -file $deflectedShapeDirectory/$dispNodeFile -node $dispNodeTag -dof
    1 2 disp
}
```

```
58   # For a two-dimensional problem:
59   # geomTransf $transfType $transfTag <-jntOffset $dXi $dYi $dXj $dYj>
```

```tcl
# ##############################################################################################
# SE 201B: NONLINEAR STRUCTURAL ANALYSIS (WI 2021)
# Lee Frame Analysis
# ##############################################################################################

# ----------------------------------------------------------------------------------------------
# DEFINE NODES
# ----------------------------------------------------------------------------------------------
set beamLength [expr 1200.*$in];
set numEleBeam 10.;
set eleLengthBeam [expr $beamLength/$numEleBeam];

set colLength [expr 1200.*$in];
set numEleCol 10.;
set eleLengthCol [expr $colLength/$numEleCol];

set nodeList {};
set nodeCtr 0;

for {set i 1} {$i <= [expr $numEleCol + 1]} {incr i 1} {
    incr nodeCtr 1
    node $nodeCtr 0.0 [expr ($i-1)*$eleLengthCol];
    puts $modelExportFileID "node $nodeCtr 0.0 [expr ($i-1)*$eleLengthCol];"
    lappend nodeList $nodeCtr
}
for {set i 1} {$i <= $numEleBeam} {incr i 1} {
    incr nodeCtr 1
    node $nodeCtr [expr $i*$eleLengthBeam] $colLength;
    puts $modelExportFileID "node $nodeCtr [expr $i*$eleLengthBeam] $colLength;"
    lappend nodeList $nodeCtr
}

# ----------------------------------------------------------------------------------------------
# DEFINE CONSTRAINTS
# ----------------------------------------------------------------------------------------------
fix [lindex $nodeList 0]   1 1 0;
fix [lindex $nodeList end] 1 1 0;

# ----------------------------------------------------------------------------------------------
# DEFINE MATERIAL & SECTION PARAMETERS
# ----------------------------------------------------------------------------------------------
set b       [expr 30.*$in];
set d       [expr 20.*$in];

set E       [expr 72000.*$ksi];
set ABeam   [expr $b*$d];
set ACol    [expr $b*$d];
set IzBeam  [expr $b*$d**3/12.];
set IzCol   [expr $b*$d**3/12.];

# ----------------------------------------------------------------------------------------------
# DEFINE GEOMETERIC TRANSFORMATION FOR ELEMENTS
# ----------------------------------------------------------------------------------------------
set transfTypeBeam "Linear"; # Corotational, Linear, PDelta
set transfTypeCol "Linear";
set transfTagBeam 5;
set transfTagCol 6;
# For a two-dimensional problem:
# geomTransf $transfType $transfTag <-jntOffset $dXi $dYi $dXj $dYj>
geomTransf $transfTypeBeam $transfTagBeam;
geomTransf $transfTypeCol $transfTagCol;

# ----------------------------------------------------------------------------------------------
# DEFINE ELEMENT
# ----------------------------------------------------------------------------------------------
# For a two-dimensional problem:
# element elasticBeamColumn $eleTag $iNode $jNode $A $E $Iz $transfTag
set nodeCtr 0;
set eleCtr 0;
```

```
70  for {set i 1} {$i <= [expr $numEleCol]} {incr i 1} {
71      incr nodeCtr 1
72      incr eleCtr 1
73      element elasticBeamColumn $eleCtr $nodeCtr [expr $nodeCtr + 1] $ACol $E $IzCol
        $transfTagCol;
74      puts $modelExportFileID "element elasticBeamColumn $eleCtr $nodeCtr [expr $nodeCtr
        + 1] $ACol $E $IzCol $transfTagCol;"
75  }
76  for {set i 1} {$i <= [expr $numEleBeam]} {incr i 1} {
77      incr nodeCtr 1
78      incr eleCtr 1
79      element elasticBeamColumn $eleCtr $nodeCtr [expr $nodeCtr + 1] $ABeam $E $IzBeam
        $transfTagBeam;
80      puts $modelExportFileID "element elasticBeamColumn $eleCtr $nodeCtr [expr $nodeCtr
        + 1] $ABeam $E $IzBeam $transfTagBeam;"
81  }
82  close $modelExportFileID
83
84  # ------------------------------------------------------------------------------
85  # DEFINE TIME SERIES AND LOAD PATTERN
86  # ------------------------------------------------------------------------------
87  set tsTag 3; # Tag for the time series
88  timeSeries Linear $tsTag
89
90  set patternTag 1; # Tag for the load pattern
91  pattern Plain $patternTag $tsTag {
92      load 13 0. $loading 0.;
93  }
94
```

```
# ##########################################################################################
# SE 201B: NONLINEAR STRUCTURAL ANALYSIS (WI 2021)
# OpenSees Tutorial # 3
# 2D FRAME ANALYSIS (LINEAR MATERIAL + NONLINEAR GEOMETRY) EXAMPLE
# ##########################################################################################

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

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

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

# ------------------------------------------------------------------------------------------
# CREATE THE INTEGRATION SCHEME
# ------------------------------------------------------------------------------------------
set lambda 0.01; # Set the load factor increment.
integrator LoadControl $lambda; # The LoadControl scheme

# ------------------------------------------------------------------------------------------
# CREATE THE CONVERGENCE TEST
# ------------------------------------------------------------------------------------------
test NormDispIncr 1.0e-5 1000000;

# ------------------------------------------------------------------------------------------
# CREATE SOLUTION ALGORITHM
# ------------------------------------------------------------------------------------------
algorithm Newton

# ------------------------------------------------------------------------------------------
# CREATE THE ANALYSIS OBJECT
# ------------------------------------------------------------------------------------------
analysis Static;

# ------------------------------------------------------------------------------------------
# RECORD AND SAVE OUTPUT
# ------------------------------------------------------------------------------------------
source generateRecorders.tcl;

# ------------------------------------------------------------------------------------------
# ANALYZE, i.e., INVOKE RUN ON ANALYSIS OBJECT
# ------------------------------------------------------------------------------------------
set nSteps [expr int(1./$lambda)];
set ok [analyze $nSteps];

```

```
# ##########################################################################################
# SE 201B: NONLINEAR STRUCTURAL ANALYSIS (WI 2021)
# OpenSees Tutorial # 3
# Lee Frame
# ##########################################################################################

#Always start with
wipe; # Clear memory of all past model definitions
model BasicBuilder -ndm 2 -ndf 3; # Define the model builder, ndm=#dimension, ndf=#dofs

# ------------------------------------------------------------------------------------------
# DEFINE UNITS
# ------------------------------------------------------------------------------------------
source units.tcl;

set loading -18000.;
# ------------------------------------------------------------------------------------------
# ------------------------------------------------------------------------------------------
# ------------------------------------------------------------------------------------------
set analysis2 "PDelta";
# ------------------------------------------------------------------------------------------
# SETUP DATA DIRECTORY FOR SAVING OUTPUTS
# ------------------------------------------------------------------------------------------
set analysisResultsDirectory "$analysis2/AnalysisResults";  # Set up name of data
directory
file mkdir $analysisResultsDirectory; # Create data directory

set modelDirectory "$analysis2/Model";
file mkdir $modelDirectory;
set modelExportFileID [open "$modelDirectory/modelData.txt" "w"];

set deflectedShapeDirectory "$analysis2/DeflectedShape"
file mkdir $deflectedShapeDirectory;

# ------------------------------------------------------------------------------------------
# DEFINE MODEL WITH LOADS
# ------------------------------------------------------------------------------------------
source Lee_frame_PDelta.tcl;
# ------------------------------------------------------------------------------------------
# ANALYSIS
# ------------------------------------------------------------------------------------------
source analysisPushover.tcl;


# Don't forget to
remove recorders
# and
wipe;
reset;

set analysis1 "Corotational";
# ------------------------------------------------------------------------------------------
# SETUP DATA DIRECTORY FOR SAVING OUTPUTS
# ------------------------------------------------------------------------------------------
set analysisResultsDirectory "$analysis1/AnalysisResults";  # Set up name of data
directory
file mkdir $analysisResultsDirectory; # Create data directory

set modelDirectory "$analysis1/Model";
file mkdir $modelDirectory;
set modelExportFileID [open "$modelDirectory/modelData.txt" "w"];

set deflectedShapeDirectory "$analysis1/DeflectedShape"
file mkdir $deflectedShapeDirectory;

# ------------------------------------------------------------------------------------------
# DEFINE MODEL WITH LOADS
# ------------------------------------------------------------------------------------------
source Lee_frame_Corotational.tcl;
```

```tcl
# ------------------------------------------------------------------------------
# ANALYSIS
# ------------------------------------------------------------------------------
source analysisPushover.tcl;


# Don't forget to
remove recorders
# and
wipe;
reset;


# ------------------------------------------------------------------------------
# ------------------------------------------------------------------------------
# ------------------------------------------------------------------------------
set analysis3 "Linear";
# ------------------------------------------------------------------------------
# SETUP DATA DIRECTORY FOR SAVING OUTPUTS
# ------------------------------------------------------------------------------
set analysisResultsDirectory "$analysis3/AnalysisResults";  # Set up name of data
directory
file mkdir $analysisResultsDirectory; # Create data directory

set modelDirectory "$analysis3/Model";
file mkdir $modelDirectory;
set modelExportFileID [open "$modelDirectory/modelData.txt" "w"];

set deflectedShapeDirectory "$analysis3/DeflectedShape"
file mkdir $deflectedShapeDirectory;

# ------------------------------------------------------------------------------
# DEFINE MODEL WITH LOADS
# ------------------------------------------------------------------------------
source Lee_frame_Linear.tcl;
# ------------------------------------------------------------------------------
# ANALYSIS
# ------------------------------------------------------------------------------
source analysisPushover.tcl;


# Don't forget to
remove recorders
# and
wipe;
reset;
```