CEE 512: Nonlinear Analysis of Structures

Project - II

The purpose of this report is to document our work on part two of our final project for the CEE 512: Nonlinear Analysis of Structures class. The objective of part two was to modify the 2-D Matlab Program to conduct geometric and material nonlinear analysis of truss or frame structures.

We did this by incorporating the principles of object-oriented programming in Matlab. We created structural objects with properties and methods that could be used in the simulation, viz. position, degrees of freedom and material properties.

We performed geometric and material nonlinear analysis of the structure using incrementive-iterative methods, viz. Load Control (Newton-Raphson), Work Control and Riks Arc Control Methods.

We verified the results of our simulation with analytical solutions and found our results closely approximated the analytical solution.

We also undertook a parametric analysis of the test cases by changing the simulation method and the step size of each increment.

The results are presented in the text and the relevant Matlab Code can be found online at https://github.com/jacob-umich/nonlin-struct-analysis

Jacob Pavelka
Sravan Potturi

Introduction:

A linear (static) analysis is an analysis where a linear relation holds between applied forces and displacements. Here, the model's stiffness matrix is constant, and the solving process is short.

Contrarily, a nonlinear analysis is an analysis where a nonlinear relation holds between applied forces and displacements. Nonlinear effects can originate from varied geometrical nonlinearities (i.e. large deformations), material nonlinearities (i.e. elasto-plastic material), non-linear loading and constraints. These effects result in a stiffness matrix which is not constant during the load application and necessitates a different solver to solve such problems.

(A) Geometric Nonlinearity:

When there are changes in the geometry of the structure during the analyses, we observe the effects of geometric nonlinearity in the response of the structure.

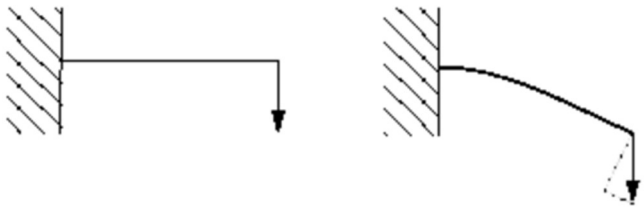


Figure 1: Large deflection of a cantilever beam.

Consider a cantilever beam loaded vertically at the tip. If the tip deflection is small, the analysis can be considered as being approximately linear. However, if the tip deflections are large, the shape of the structure and, hence, its stiffness changes. In addition, if the load does not remain perpendicular to the beam, the action of the load on the structure changes significantly.

As the cantilever beam deflects, the load can be resolved into a component perpendicular to the beam and a component acting along the length of the beam. Both effects contribute to the nonlinear response of the cantilever beam (i.e., the changing of the beam's stiffness as the load it carries increases).

E.g. **Snap-through behavior of a large shallow panel.**

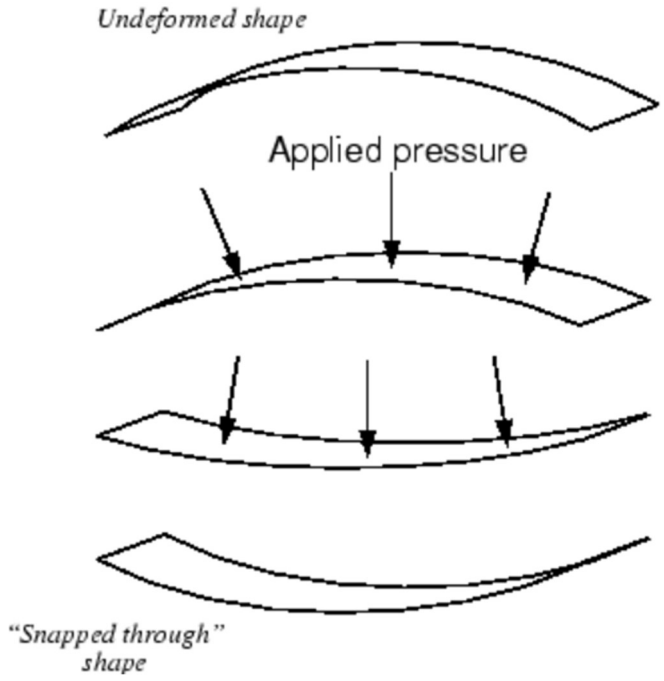


Figure 2: Snap-through behavior

Here, there is a dramatic change in the stiffness of the panel as it deforms. As the panel "snaps through," the stiffness becomes negative. Thus, although the magnitude of the displacements, relative to the panel's dimensions, is quite small, there is significant geometric nonlinearity in the simulation, which must be taken into consideration.

(B) Material Nonlinearity:

Material non-linearities occur in solid mechanics when the relationship between stress and strain, otherwise known as the constitutive relationship of the material, is no longer linear. The variation of the constitutive relationship also causes the stiffness of the structure or component consisting of the non-linear material to vary also. Thus, the stiffness of the structure or component may vary as a function of the combined or individual load level and load history.

Non-linear material models describe the macroscopic behaviour of the material; hence they are approximations to the real behaviour of the material as the real behaviour is also related to micro-mechanical effects within the material.

For example, the plastic behaviour of metals is related to dislocations and slip planes within the crystal lattice. These defects are assumed to be randomly distributed throughout the material such that a degree of homogeneity can be assumed by the model at a macroscopic level. This

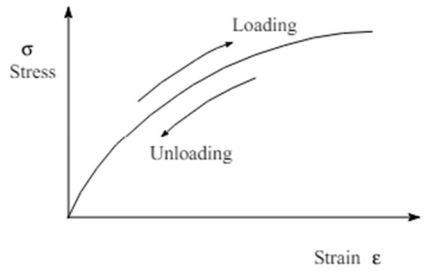
allows a uniform macroscopic approximation of the discrete microscopic behaviour of the material over a suitably large volume.

Nonlinear material behaviour in solid mechanics can be broadly divided into 2 buckets: rate-independent and rate-dependent.

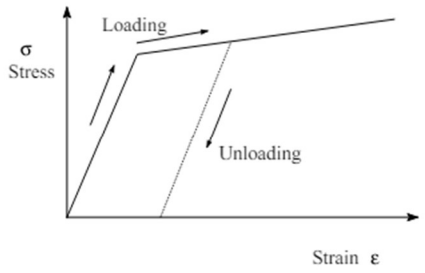
Rate-Independent Material Nonlinearity:

The cases of material non-linearity described under this category are assumed to be independent of time. This is an immediate approximation as all materials are dependent to some degree upon the rate at which the load is applied. The rate dependence for some materials under specific loading conditions is such that it can be neglected, without reasonable loss of accuracy.

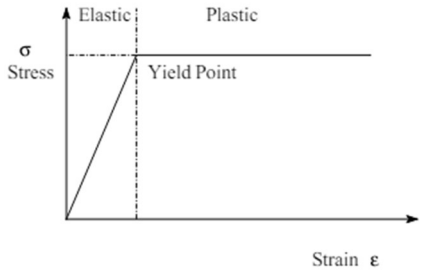
Important cases of Rate-Independent Nonlinear Elasticity include nonlinear elasticity and elasto-plasticity.



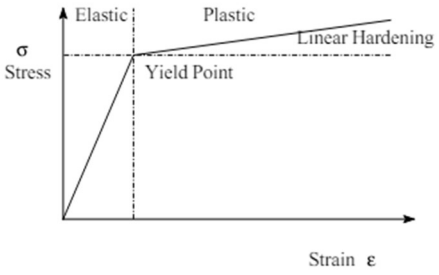
Nonlinear Elasticity



Elasto-Plasticity



Elastic, perfectly Plastic



Elastic, linear Work-Hardening.

Rate-Dependent Material Non-linearity:

Non-linearity described in this category is time dependent. This is true for many materials under specific conditions, where the rate dependency of the material can no longer be neglected.

Examples of such nonlinearity are found in materials displaying visco-plasticity, creep and stress relaxation.

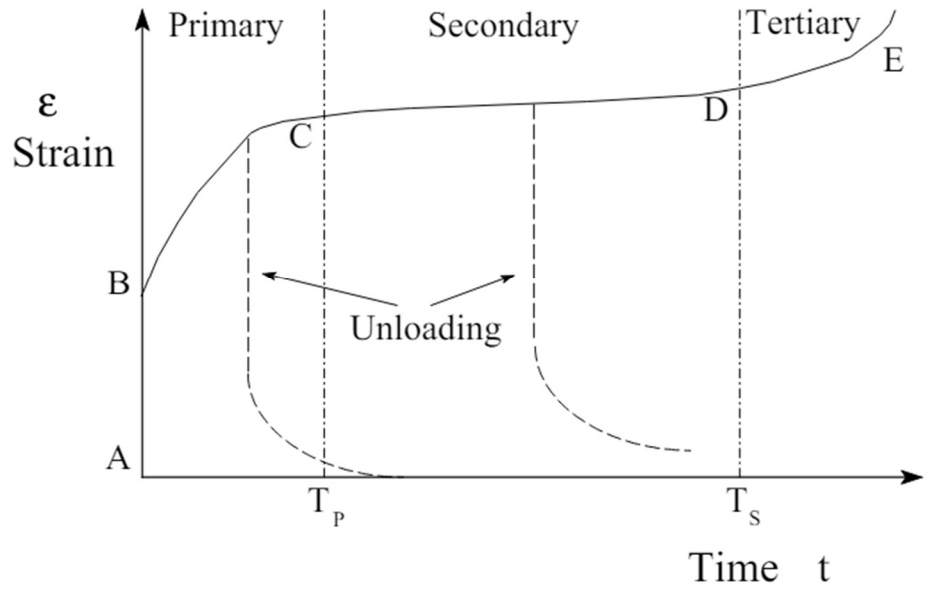


Figure 3: Uniaxial Strain-Time Curve

Solutions of Nonlinear Equilibrium Equations:

Nonlinear equilibrium equations formulated on the deformed (and yielded) geometry are to be solved to obtain the displacement and stresses in the structure. This can be achieved using direct methods or iterative methods. The former are effective when solving linear elastic structures as they readily yield exact solutions while the latter are used to solve nonlinear structures.

The overarching philosophy behind solving nonlinear equilibrium equations involves discretizing the structure into infinitesimally smaller parts and performing linear analysis on each of the smaller parts and incrementing and aggregating them over the entire structure.

The total load $\{\mathbf{P}\}$, or its equivalent, $\{\mathbf{P}\} = \lambda\{\mathbf{P_{ref}}\}$ is applied through a series of infinitesimal load increments $\{\mathbf{dP_i}\}$. Mathematically,

$$\{\mathbf{P}\} = \lambda\{\mathbf{P_{ref}}\} = \Sigma\{\mathbf{dP_i}\}$$

(Here λ is a scaling factor)

Corresponding infinitesimal displacement $\mathbf{\{d\Delta_i\}}$ response due to the infinitesimal load increment is aggregated over the entire length to give the total displacement response of the structure. Mathematically,

$$\{\mathbf{\Delta}\} = \Sigma\{\mathbf{d\Delta_i}\}$$

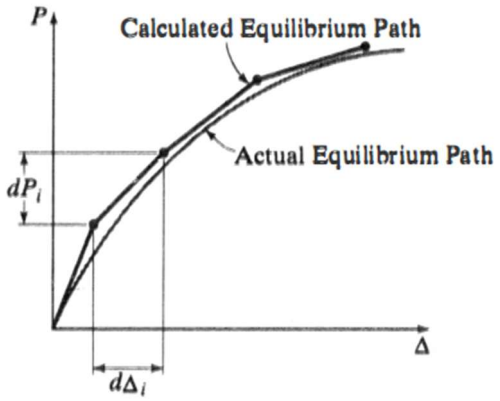


Figure 4: Piecewise infinitesimal linear fit

The degree to which a piecewise infinitesimal linear fit approximates the actual equilibrium solution is a function of how accurately the nonlinear relationship between $\{dP_i\}$ and $\{d\Delta_i\}$ is represented in each load increment.

Many kinds of solutions are suggested to solve the nonlinear equilibrium equations and all of them differ mainly on the relationship they have between $\{dP_i\}$ and$\{d\Delta_i\}$.
They can be broadly classified into two buckets:
(A) Incremental Single Step Methods and
(B) Incremental – Iterative Methods.

We have used incremental – iterative methods to solve our system of equations. Here, we used the Load Control (Newton-Raphson), Arc Control and Work Control methods to estimate the deformations of the structure.

A brief overview of the methods follows.

(A) Load Control (Newton-Raphson) Method:

Here, each increment is subdivided into a number of steps and deformations at each of the steps are found out and aggregated over the entire step, which is aggregated over the entire increment.

In each step j, the unknown displacements $\{d\Delta_i^j\}$ are found by solving the following equation:

$$\mathbf{[K_i^{j-1}]\{d\Delta_i^j\} = \{dP_i^j\} + \{R_i^{j-1}\}}$$

Here,

- $[K_i^{j-1}]$ is the stiffness matrix evaluated on the deformed geometry and corresponds to the element forces up to the previous iteration.
- $\{dP_i^j\}$ is the load applied at each step.
    - $\{dP_i^j\} = d\lambda_i^j\{P_{ref}\}$
- $\{R_i^{j-1}\}$ represents the force imbalance between the existing external and internal forces. The force unbalance can be calculated according to:
  $\{R_i^{j-1}\} = \{P_i^{j-1}\} - \{F_i^{j-1}\}$
    - Here, $\{P_i^{j-1}\}$ is the total external force applied and $\{F_i^{j-1}\}$ is the net internal forces generated at each degree of freedom.

The deformation at each iteration can be found by:

$$\{\mathbf{d\Delta_i^j}\} = \{\overline{\mathbf{d\Delta_t^J}}\} + \{\overline{\overline{\mathbf{d\Delta_i^J}}}\}$$

Here,

1. $\overline{\mathbf{d\Delta_t^J}} = \mathbf{[K_i^{j-1}]^{-1}}\, d\lambda_i^j \mathbf{\{P_{ref}\}}$
2. $\overline{\overline{\mathbf{d\Delta_i^J}}} = \mathbf{[K_i^{j-1}]^{-1}\{R_i^{j-1}\}}$

Workflow for the Load Control method is as follows:

1. We apply a force $\{dP_i^j\} = d\lambda_i^j \{P_{ref}\}$ on the structure at the j^{th} step of the i^{th} load increment. Corresponding stiffness matrix $[K_i^{j-1}]$ is evaluated and used to find the displacement due to the applied load using $\{\overline{d\Delta_i^J}\} = [K_i^{j-1}]^{-1} d\lambda_i^j \{P_{ref}\}$.
   Here, $d\lambda_i^j = 1$ for j=1 and $d\lambda_i^j = 0$ for j>=2.
2. We find the force unbalance $\{R_i^{j-1}\}$ by deducting the total applied force $\{P_i^{j-1}\}$ from the internal forces $\{F_i^{j-1}\}$ generated in the structure. Corresponding displacement of the structure at the j^{th} step of the i^{th} load increment is evaluated using $\{\overline{\overline{d\Delta_1^J}}\} = [K_i^{j-1}]^{-1}\{R_i^{j-1}\}$.
3. We find the total deformation of the structure at the j^{th} step of the i^{th} load increment by summing the displacements due to the applied load $\{\overline{d\Delta_i^J}\}$ and force unbalance $\{\overline{\overline{d\Delta_1^J}}\}$.
   $\{d\Delta_i^j\} = \{\overline{d\Delta_i^J}\} + \{\overline{\overline{d\Delta_1^J}}\}$

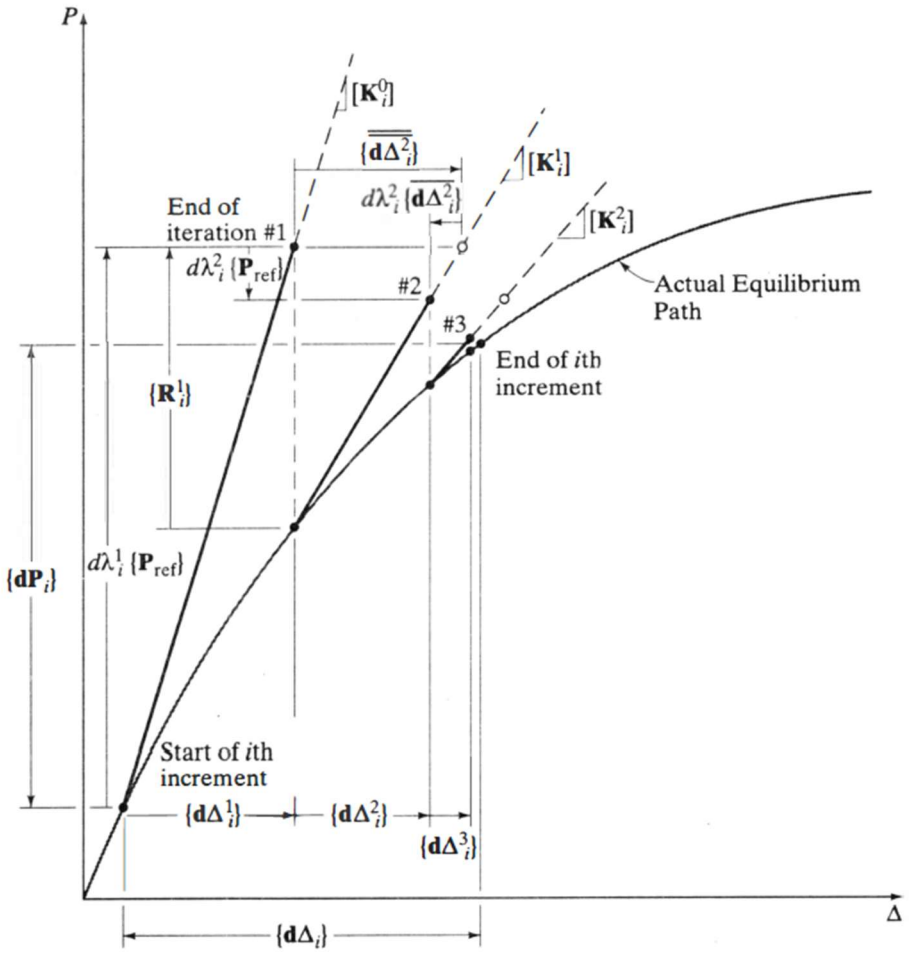


Figure 5: Schematic of the Load Control Method

(B) Work Control Method:

Here, we use both loads and displacements to establish equilibrium at the deformed geometry of the structure. We require for the work done by the applied force to be zero at the j^{th} step of the i^{th} load increment (j>=2).

$$dW_i^j = \{dP_i^j\}^T \{d\Delta_i^j\} = 0 \text{ (for j>=2)}$$

This leads us to a load increment ratio of:

$$d\lambda_i^j = -\{P_{ref}\}^T \{\overline{\overline{d\Delta_1^J}}\} / \{P_{ref}\}^T \{\overline{d\Delta_t^J}\} \quad \text{for j>=2}$$

Workflow for the Work Control method is as follows:

1. Apply an initial external force of $d\lambda_i^1 \{P_{ref}\}$ on the structure. It is a good practice to have $d\lambda_i^1$ of [0.10- 0.20] for better convergence. Corresponding stiffness matrix $[K_i^{j-1}]$ is evaluated and used to find the displacement due to the applied load using $\{\overline{d\Delta_t^1}\} = [K_i^1]^{-1} d\lambda_i^1 \{P_{ref}\}$.
2. Evaluate the infinitesimal strain (dϵ) from the displacement $\{\overline{d\Delta_t^1}\}$, and using Hooke's law, calculate the net internal forces in the structure. .
3. Evaluate the force unbalance $\{R_i^j\}$ by deducting the total applied force $d\lambda_i^1 \{P_{ref}\}$ from the internal forces $\{F_i^j\}$generated in the structure.
3. Compute the stiffness $[K_i^j]$ at the deformed configuration and use it to evaluate deformations ($\{\overline{d\Delta_t^J}\}$ and $\{\overline{\overline{d\Delta_1^J}}\}$) due to the total external force $\{P_{ref}\}$ and force unbalance $\{R_i^1\}$. $\mathbf{\overline{d\Delta_t^J} = [K_i^{j-1}]^{-1}\{P_{ref}\}}$ $\mathbf{\overline{\overline{d\Delta_1^J}} = [K_i^{j-1}]^{-1}\{R_i^{j-1}\}}$
4. Update $d\lambda_i^j = -\{P_{ref}\}^T \{\overline{\overline{d\Delta_1^J}}\} / \{P_{ref}\}^T \{\overline{d\Delta_t^J}\}$ for j>=2
5. We find the total deformation of the structure at the j^{th} step (for j>2) of the i^{th} load increment by summing the displacements due to the applied load $\{\overline{d\Delta_t^J}\}$ and force unbalance $\overline{\overline{\{d\Delta_1^J\}}}$. $\{d\Delta_i^j\} = d\lambda_i^j \{\overline{d\Delta_t^J}\} + \{\overline{\overline{d\Delta_1^J}}\}$

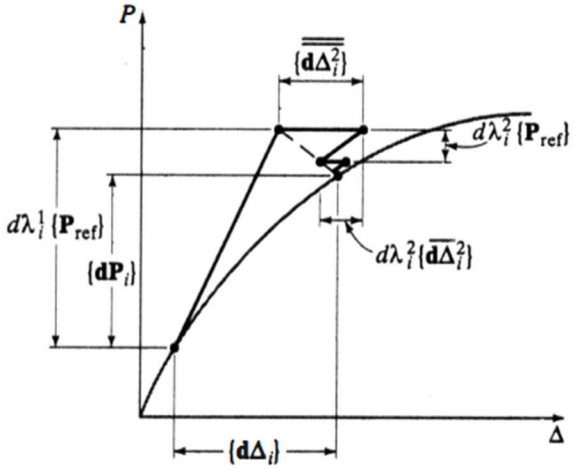


Figure 6: Schematic of the Work Control Method

(C) Riks Arc Control Method:

Unlike the Load Control or Work Control Methods, in the Riks Arc Control Method, we use an arbitrary arc length ds in each equilibrium iteration according to,

$ds^2 = \{\overline{d\Delta_i^1}\}^T \{d\Delta_i^j\} + d\lambda_i^1 \, d\lambda_i^j = 0$ for j>=2

Workflow for Arc Control Method:

1. Apply an initial external force of $d\lambda_i^1 \{P_{ref}\}$ on the structure. It is a good practice to have $d\lambda_i^1$ of [0.10- 0.20] for better convergence. Corresponding stiffness matrix $[K_i^{j-1}]$ is evaluated and used to find the displacement due to the applied load using $\{\overline{d\Delta_i^1}\} = [K_i^1]^{-1} \, d\lambda_i^1 \{P_{ref}\}$.
2. Evaluate the infinitesimal strain (dϵ) from the displacement $\{\overline{d\Delta_i^1}\}$, and using Hooke's law, calculate the net internal forces in the structure. .
3. Evaluate the force unbalance $\{R_i^j\}$ by deducting the total applied force $d\lambda_i^1 \{P_{ref}\}$ from the internal forces $\{F_i^j\}$ generated in the structure.
4. Compute the stiffness $[K_i^j]$ at the deformed configuration and use it to evaluate deformations ($\{\overline{d\Delta_i^J}\}$ and $\{\overline{\overline{d\Delta_1^J}}\}$) due to the total external force $\{P_{ref}\}$ and force unbalance $\{R_i^1\}$.
   $\mathbf{\overline{d\Delta_i^J} = [K_i^{j-1}]^{-1}\{P_{ref}\}} \quad \mathbf{\overline{\overline{d\Delta_1^J}} = [K_i^{j-1}]^{-1}\{R_i^{j-1}\}}$
5. Update $d\lambda_i^j = -\{\overline{d\Delta_i^1}\}^T \{\overline{\overline{d\Delta_1^J}}\} / (\{\overline{d\Delta_i^1}\}^T \{\overline{d\Delta_i^J}\} + d\lambda_i^1)$ for j>=2
6. We find the total deformation of the structure at the j^{th} step (for j>2) of the i^{th} load increment by summing the displacements due to the applied load $\{\overline{d\Delta_i^J}\}$ and force unbalance $\overline{\overline{\{d\Delta_1^J\}}}$.
   $\{d\Delta_i^j\} = d\lambda_i^j \{\overline{d\Delta_i^J}\} + \{\overline{\overline{d\Delta_1^J}}\}$

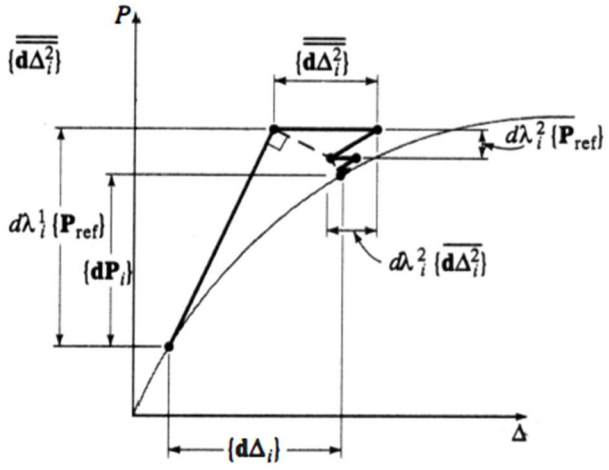


Figure 7: Schematic of Arc Control Method

Nonlinear Simulation Library:

We developed an object-oriented library to simulate the nonlinear response of structures. This library is mainly composed of four classes: The structure, element, node, and material class. The structure class is composed of all other classes and is the main interface of the program. The nonlinear solution methods (work-control, arc-control, etc.) were written as separate functions that take a structure object as an input. Apart from this, an additional set of scripts were developed to provide an interactive command-line simulation environment. These scripts were derived from the original interactive scripts. An additional script was added to plot the load-deformation response history.

This class structure facilitates the nonlinear finite element analysis of structures using an object-oriented approach and allows users to perform operations on the structure as a whole. The separation of concerns among different classes ensures modularity and maintainability of the code. Each class inherits the handle class, which is essential because it allows the changes in an object's state in one place to be seen in another.

The structure object encapsulates all the necessary components and interactions to simulate the structural behavior in response to applied loads in a linear fashion or in an infinitesimal linear fashion.

The following are its properties and methods:

Properties:

- elements: An array of Element objects representing the truss elements of the structure.
- nodes: An array of Node objects representing the nodes in the structure.
- n_nodes: The total number of nodes.
- n_free: Number of free degrees of freedom (DOF) without imposed constraints.
- n_fix: Number of fixed degrees of freedom after accounting for constraints.
- n_dof: Total number of degrees of freedom in the structure.
- orig_pos: Original positions of the nodes. This property is useful for plotting the undeformed shape.
- delta_hist: A history of displacement vectors over time or iterations. This property will store the history of a nonlinear analysis
- lambda_hist: A history of displacement vectors over time or iterations. This property will store the history of a nonlinear analysis

Methods:

- Constructor Structure(nodes, elements): Initializes the Structure object, assigns DOFs to nodes, computes the number of free and fixed DOFs, and sets up the original positions. Nodes and Elements are created before the Structure, but global properties, like degrees of freedom, aren't assigned to these objects until they are all accounted for in the structure object. The code uses dynamic allocation and incrementation of DOFs as it iterates through nodes, handling various conditions like fixed, free, and mixed DOF

nodes. All free nodes are given DOF's first, then fixed nodes are assigned. This way, there is a clear distinction between free and fixed degrees of freedom.

- get_loads(): Compiles the external force vector P and an element force vector PF for the entire structure by checking for forces on each node and element associated with the structure.
- get_stiffness(): Computes and returns the global elastic stiffness matrix K of the structure.
- get_tan_stiffness(): Computes and returns the global tangent stiffness matrix, considering both material and geometric stiffness.
- update_disp(pos): Updates the nodal positions based on the input displacement vector pos. This method will affect the magnitude of the internal forces generated from each element.
- get_internal_force(): Calculates and returns the vector F of internal forces for each DOF. The forces computed from this method are computed based on the difference in the current position of each node and the original position of each node.
- reset_pos(): Resets the nodal positions back to their original positions. This method is used if the simulation computes a nonrealistic state at the end of a nonlinear simulation. After this method is called, a previous set of displacements can then be reapplied using the update_disp method.
- store_load_disp(delta, lambda): Updates the displacement history delta_hist and load factor history lambda_hist with the current values of delta and lambda. This can be called after every increment of a nonlinear solution method.

The Element class in this library is designed to encapsulate all the functionalities and properties of a truss element in nonlinear finite element analysis. This class represents the individual, discrete components that, when combined, make up the structure being analyzed. Elements have their own method calls to fetch transformation matrices, nodal loads, internal forces, and stiffness matrices.

Properties:

- nodes: An array of two Node objects which represent the endpoints of the element. The first node represents the ith node, and the second is the jth, depending on how they were inputted into the constructor.
- material: A Material object containing the material properties relevant to the element.
- loads: An axial tangential load that can optionally be specified .
- dofs: An array representing the degrees of freedom associated with the nodes of the element. This is a useful property that helps avoid recollecting the terms form the node.
- original_length: The initial, undeformed length of the element. This property is used to compute strains and fixed end forces
- orig_pos: The initial position coordinates of the nodes of the element. This property is useful for computing strains
- id: An identifier for the element.

Methods:

- Constructor Element(node_i,node_j,material): Initializes an Element with two nodes and a material. It calculates the original length and original positional coordinates so they can be used later when the nodal positions get updated.
- get_elem_len(): Computes the length of the element based on the updated position of the nodes.
- set_loads(loads): Sets the loads applied to the element. This action be done by setting the load property directly too.
- get_etran(): Creates a transformation matrix that relates global coordinates to local element coordinates based on the element's orientation. This function uses the updated node positions.
- get_stiffness(): Calculates the element's elastic stiffness matrix in local coordinates.
- get_g_e_stiffness(): Transforms the local stiffness matrix into the global coordinate system using the get_etran method.
- get_nodal_loads(): Calculates the nodal load vector due to element loads using the equivalent fixed end force method and linear shape functions.
- get_dofs(): Retrieves the degrees of freedom for the element's nodes. This can also be done by directly accessing the dofs property.
- get_coords(): Obtains a vector of the coordinates of the element's nodes. This is used to quickly collect positions from node objects.
- get_strain(): Computes the strain in the element based on current and original node positions. This strain computation is based on the first derivative of the shape function, which for a 2-node truss element corresponds to the engineering strain equation.
- get_internal(): Computes the internal forces in the element based on its material properties and strain. The material properties can also depend on strain.
- get_geometric_stiffness(): Computes the element's geometric stiffness matrix given a current internal force. This matrix is used in computing the tangent stiffness matrix.
- get_material_stiffness(): computes the material nonlinearity stiffness matrix to orient plastic flow in the correct direction. This matrix is used in computing the tangent stiffness matrix.

The Node class is much simpler than the previous two. This class is mainly used to store positions, fixities, and nodal forces in one convenient location. The Node class represents the points or vertices where elements connect and where loads can be applied or displacements can occur in the structure. Here is an explanation of its properties and methods:

Properties:

- dof: An array representing the degrees of freedom (DOF) for the node. The first degree of freedom corresponds to the x direction of the node, and the second degree of freedom corresponds to the y direction of the node. The dof property is the global degree of

freedom in the context of a system for the simulation, thus providing a mapping during the assembly process of the global stiffness matrix.

- pos: The current position coordinates of the node in inches (e.g., [x, y]).
- orig_pos: The original, undeformed position coordinates of the node, used for comparing with the deformed state.
- loads: An array of external loads applied to the node, in the same order as the DOFs (e.g., [Fx, Fy]). This property must be set after the object's initialization.
- fixity: An array indicating the fixity of the node's DOFs, where a 0 indicates a free DOF, and a value of 1 indicates a fixed or restrained DOF. This property must be set after the object's initialization.
- id: An identifier for the node, which can be used for bookkeeping and referencing within the structure.

Methods:

- Constructor Node(coord): Creates a new Node instance, initializing its position and original position to the given coordinates.
- set_fixity(fixity): Sets the fixity status of the node's DOFs, dictating which DOFs are fixed (with a value of 1) or free (with a value of 0).
- set_load(loads): Assigns external loads to the node's DOFs.
- set_dof(dof_in): Assigns the node's DOFs to the global structure's DOFs, linking the node's local DOF indexes to the global indexes.

It is worth noting that all setter methods are not necessary to set the values of the properties because the properties are public. All nodes start in an undeformed state (orig_pos equals pos at construction) and are subject to change (deformation) throughout the simulation process. By default, a new node is created with no external loads (loads = [0,0]) and with free DOFs (fixity = [0,0]), indicating that it can move freely unless otherwise specified by using the setter methods.

The material class is similar to the node method in that it mainly serves as a way to aggregate data, but this class differs in that one of its properties is a function that should represent the stress-strain relationship. While the class is just called material, it contains properties for the member's section too. Here is an explanation of its properties and methods:

Properties:

- id: An identifier for the material, which can help differentiate between different materials in a simulation.
- e_base: The base value of Young's modulus for the material, representing its initial stiffness. The change in stiffness can be described in terms of this material.
- area: The cross-sectional area of the truss element associated with this material.
- material_func: A function handle that defines the material's behavior, particularly its stress-strain relationship. This function takes strain and e_base as inputs and returns the effective modulus of elasticity (stiffness) that may vary with strain.

Methods:

- Constructor Material(id, e_base, area, moi, material_func): Initializes a new instance of Material with the specified properties. The id parameter uniquely identifies the material; e_base is the starting modulus of elasticity; area is the cross-sectional area pertinent to the material; moi is the moment of inertia; and material_func is the function handle defining how the Young's modulus varies with strain.
- get_moe(strain): Uses the material's material_func to calculate the modulus of elasticity (moe) for a given strain value. This method allows the material to exhibit monotonic nonlinear material behavior. However, if load reversal is present, this model will not retain the plastic strain.

Structural Analysis Process:

A typical structural analysis can be conducted interactively by running the main.m script. This script will prompt the user to define the parameters of the analysis, then return several results from the simulation. First, the script asks the user to define the structure.

Structures can be defined in two ways in order to be analyzed by this library: by using an interactive command-line too, or by creating a generating function. The interactive command line tool is a modified version of the pre.m script given to us. However, this script assigns input variables to objects instead of variables in the global scope. Once all objects are defined, they are assigned to variables in the global scope. The function approach is much more appealing for complex structures because the structural definition does not need to be re-written and can easily be modified. For this approach, the user needs to define a function that returns a structure object. Then, all material, node, element, and structure objects need to be defined in the body of the function. This function can then be called from the iteractive command line tool to save the structure object as a variable in the global scope.

Once the structure is defined, it is used as an input to one of the solution methods. As mentioned earlier, 4 solution methods were provided. These methods differ by how applied loads and displacements are calculated during each increment but mostly have the same structure, excluding linear-elastic analysis. To begin the analysis, the nodal and element loads are calculated and organized based on the structural definition. This load is taken as the reference load in the above derivations. The accumulated λ value, the accumulated Δ value, and the original stiffness matrix are initialized be before the simulation begins.

Then a incremental, iterative process where in each increment of a while loop and calculates the incremental displacements (delta_free) and corresponding load factor (lambda) by iteratively updating them until they converge to satisfy equilibrium. The maximum iterations allowed were 50, however, to account for situations where force-controlled methods cannot converge.

Each increment generally begins with the initialization of certain values, followed by the iterative convergence towards equilibrium, and finalized by checking stopping conditions and updating values for the next iteration. For the first increment, the starting λ_i is set to 0.2.

```
% get loads. PF is loads from fixed end forces
[P,PF]=structure.get_loads();
P_total = P+PF;

% total load applied at step 0
lambda=0;

% initial lambda for first increment
lambda_i=0;
d_lambda=0.2;

% establising originial stiffness matrix to measure nonlinearity
k_orig=structure.get_tan_stiffness()(1:structure.n_free,1:structure.n_free);

% setting initial stiffness matrix
k_free=k_orig;

% initialize displacements to be added to
delta_free = zeros(structure.n_free,1);
count = 1;
```

Then, the iterative procedure begins by computing the displacement associate with a tangential step of the incremental load.

```
            % compute delta j
            total = ((P_total(1:structure.n_free,1)*d_lambda)+R);
            delta_free_j=k_free\total(1:structure.n_free);
```

Next, these displacements are applied to the structure by using the update_disp method. Now, affected nodes have new positions but also have their original position information. Using the new and old positions, the next step is to calculate the internal force for each member in the structure using the get_internal_force method.

```
            % get total displacement at step j
            delta_free=delta_free+delta_free_j;
            structure.update_disp(delta_free_j);

            % get internal force for step j
            F = structure.get_internal_force();
```

```
F = F(1:structure.n_free,1);
```

This internal force can be used to determine the nonlinearity of the analyzed structure in terms of the difference in the applied load and residual load. It is important to note that when computing the internal force, the updated locations are used to compute the transformation matrix because it is derived from the current node locations. The transformation matrix is used to equilibrate the internal forces in the local coordinate system to the global applied forces. If the residual is large enough, then another iteration is necessary.

```
% compute residual for step j
P = (P_total*(lambda+lambda_i));
R=P(1:structure.n_free,1)-F;

% stop iteration if residual is small
if max(abs(R))<1e-3
    break
end
```

In this case, the tangent stiffness matrix must be updated for the new geometry and the λ_i must be updated for the next iteration according to the method being used. At this point, the structural tangent stiffness matrix can be updated as well according to the new geometry.

```
% compute stiffness for step j+1
k_free= structure.get_stiffness();
k_free = k_free(1:structure.n_free,1:structure.n_free);


% compute d_lambda for step j+1
dDelta_double_bar = (k_free\R(1:structure.n_free));
dDelta_bar = (k_free\P_total(1:structure.n_free));
numerator = -delta_free_j_1'*dDelta_double_bar;
denominator = delta_free_j_1'*dDelta_bar+d_lambda_j_1;
d_lambda=numerator/denominator;
```

Once equilibrium is achieved, S is computed based on the level of nonlinearity, S is used to compute λ_i^1, and the current increments loads and displacements are recorded for analysis later. How much d_lambda is incremented can affect the resolution of the simulation.

```
% update accumulated lambda
lambda = lambda+lambda_i;
% compute dlambda for next increment
d_delta_bar_1 = k_orig\P_total(1:structure.n_free);
d_delta_bar_i = k_free\P_total(1:structure.n_free);
```

```
        S =
(P_total(1:structure.n_free)'*d_delta_bar_1)/(P_total(1:structure.n_free)'*d_delt
a_bar_i);
        d_lambda=sign *0.01*abs(S)^(1/2);
```

If the full reference load is achieved or there is a negative eigenvalue, indicating post-peak response, the simulation is finished.

```
        if lambda>=1 || e<0
            if e<0
                warning("Critical point passed. displayed reactions will not show
full capacity.")
                break
            end
            break
        end
```

Once the simulation finishes, the final displacements and loadings are returned from the function.

Numerical Verification and Examples:

This section will present the simulation of 4 separate structures. Two structures will be used to verify our simulations, and the remaining will be used to a show a large scale simulation. The first of these structures is the 3 node arch, which is a classic example of geometric nonlinearity.

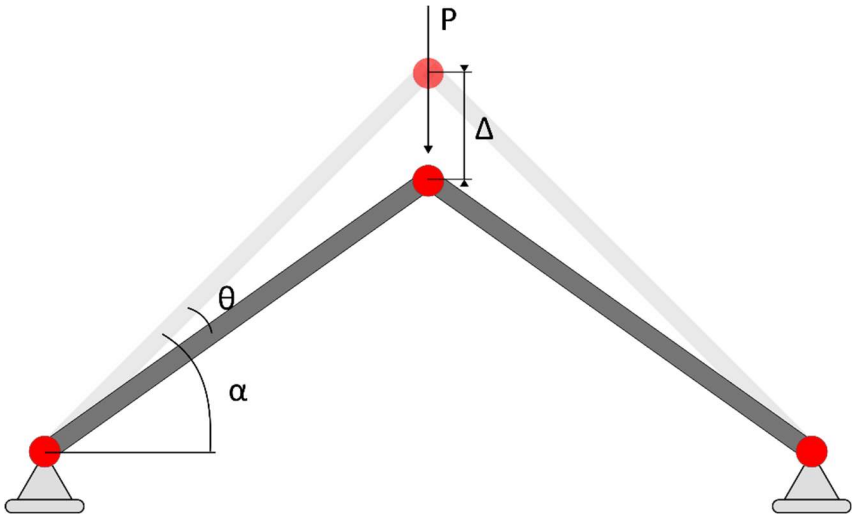


Figure 8: Schematic for three-node arch structure

The analytical solution for the load-deformation response can be derived easily. Δ can be defined in terms of θ and α.

$$\Delta = L \sin\theta \sec(\alpha - \theta)$$

This term can then be used to find the change in length of either member

$$\delta = L - L\cos\alpha \sec(\alpha - \theta)$$

δ can then be used to derive the internal force of either member

$$F = \frac{\delta}{L} * EA$$

Then, using equilibrium at the top node, an equation relating the applied load to the deflection angle can be derived

$$P = 2EA(\sin(\alpha - \theta) - \cos(\alpha)\tan(\alpha - \theta))$$

Setting the first derivative of this equation to zero finds the deformation angle which requires the maximum load.

Our 3 node arch had a total width of 10 inches, a height of 5 inches, a cross-sectional area of 10 square inches, and a modulus of elasticity of 29000 kips per square inch. With this configuration, the maximum load of 52,000 kips is achieved at a displacement of 2.085 inches. Our simulation predicts the following load-displacement relationship as shown in Figure 9. A λ value of 1 cooresponds to a load of 56000 kips, indicating our simulation captures the capacity of the

structure relatively well. However, our simulation overestimates the displacement at the maximum load.

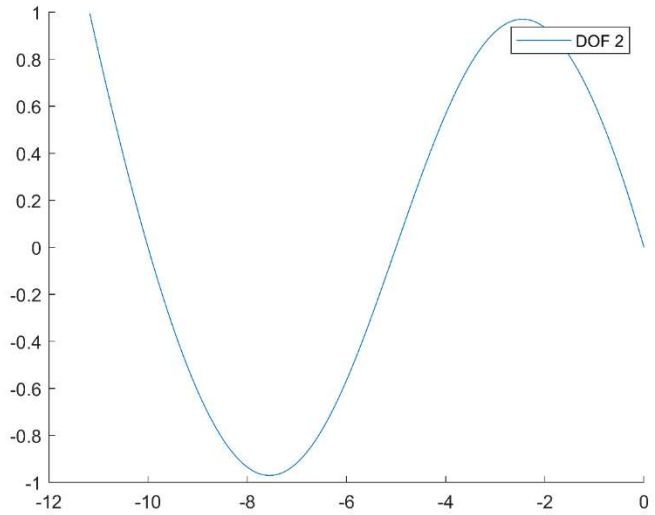


Figure 9: Load-Deformation response for arch structure

The next structure used for verification is a simple axial member with a strain hardening material. This axial member has a cross-sectional area of 10 square inches and is 100 inches long. It has a yield strength of 36 ksi and a 2% strain hardening. When a 400 kip axial load applied, the expected displacement is 0.81 inches. Figure 10 shows the response history when the same column is simulated using our library.

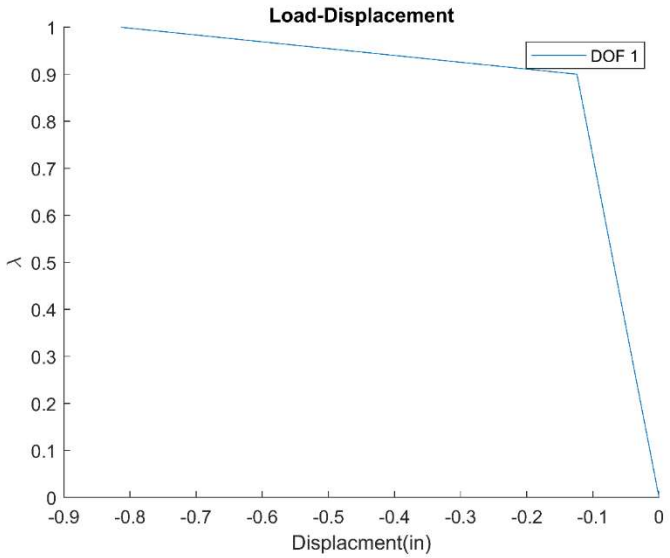


Figure 10: Load-Deformation Response for Column Structure

Figure 11 shows a schematic of the bridge structure we simulated. Mat_1 corresponds to a 40 square inch member that yields at 36 ksi and has 2% strain hardening. The horizontal loads applied to the structure are 75 kips, and the vertical loads applied are 1000 kips.

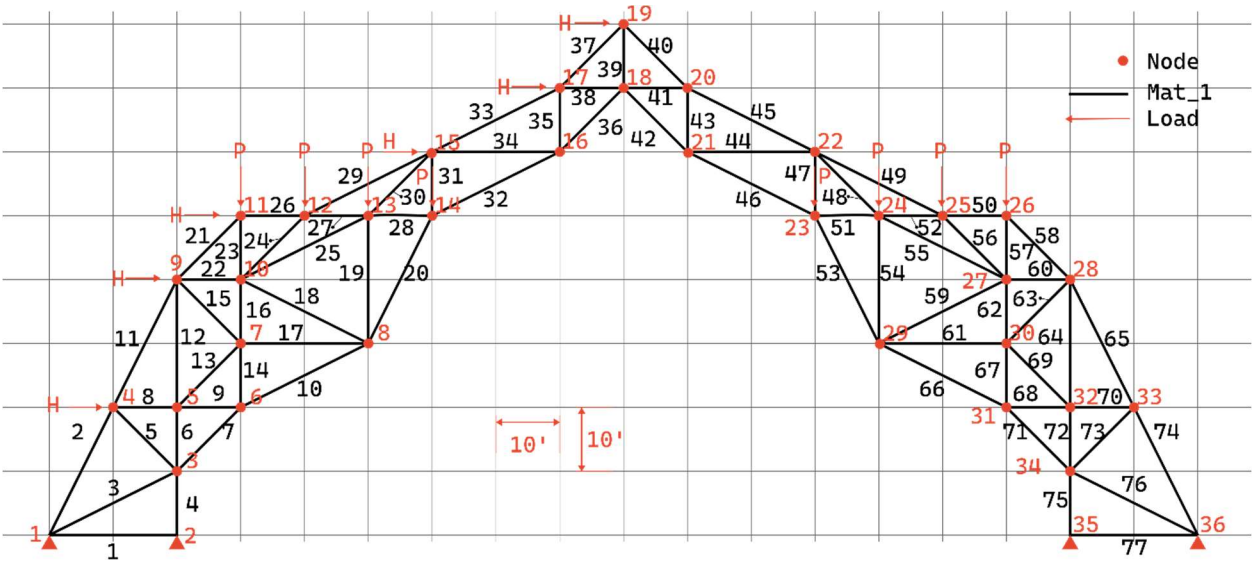


Figure 11: Schematic for bridge structure

Figure 12 shows the vertical displacements of nodes 11, 19, and 26 simulated from our library.

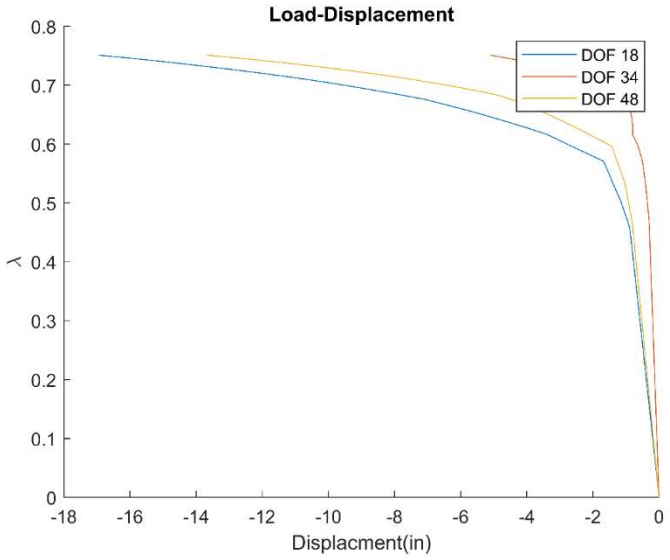


Figure 12: Load-Deformation response for bridge structure

The deformed configuration is as shown in Figure 13.

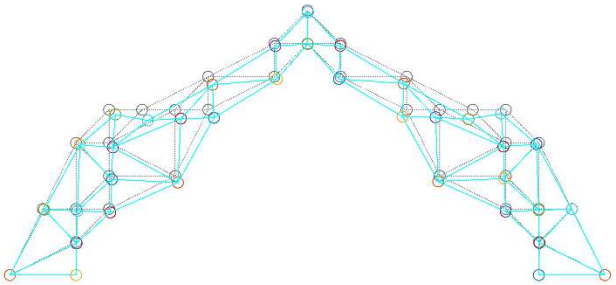


Figure 13: Displaced configuration for bridge structure

An second version of the bridge structure was simulated with different material properties, as shown in Figure 14. Mat-1 has an area of 20 square inches, Mat-2 has an area of 30 square inches, and Mat-3 has an area of 40 square inches. The same loading and stress strain properties are applied from the previous analysis.

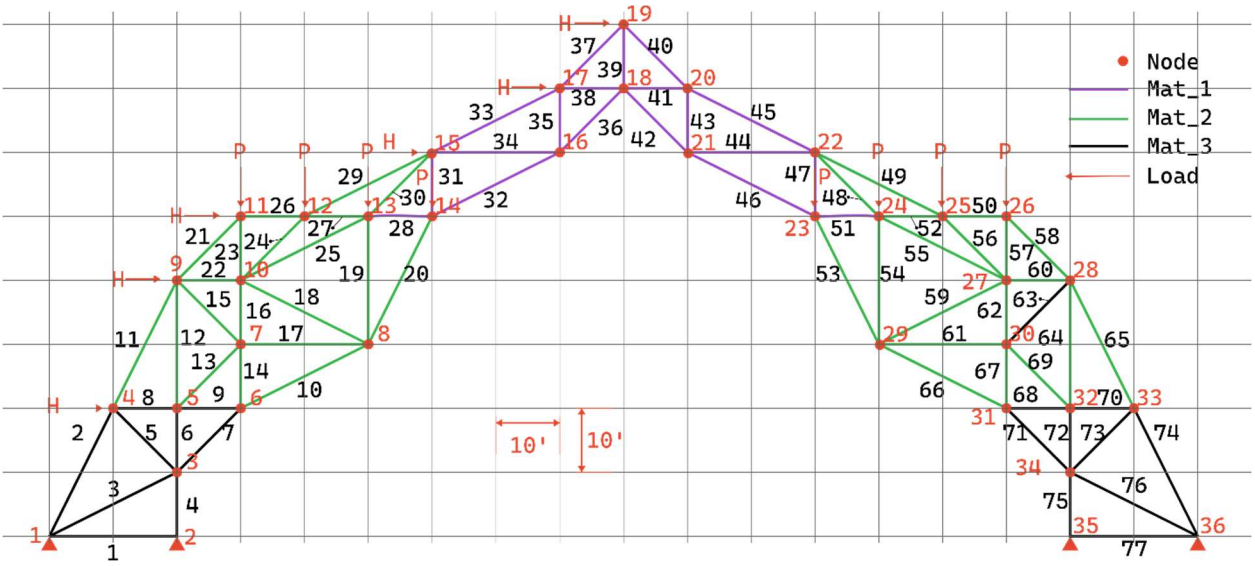


Figure 14: Schematic for second bridge structure

Figure 15 shows the vertical displacements of nodes 11, 19, and 26 simulated from our library.

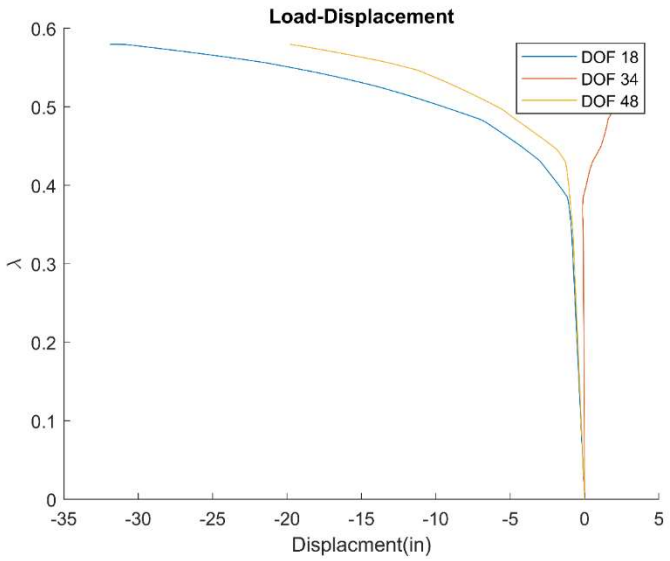


Figure 15: Load-Deformation Response for second Bridge Structure

The deformed configuration is as shown in Figure 16.

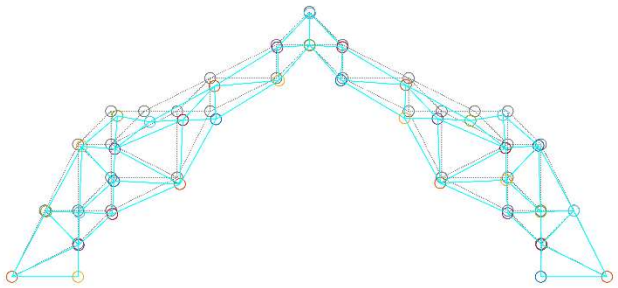


Figure 16: displaced configuration of second bridge strucutre

Parametric study:

The various simulation methods we programmed can vary in performance. For example, the newton-raphson method cannot capture post-peak response. This last section will analyze the different effects of simulation methods and lambda increments. For this parametric study, we will analyze the bridge structure. The λ_i that is used at the beginning of each increment can be determined as follows:

$$\lambda_i = \pm f|S|^{\gamma}$$

Where f is some number less than 1. The smaller the value for f, the more increments that will be included in the analysis before a value of $\lambda = 1$ is achieved. For our analysis, we used values of 0.1, 0.01, and 0.001, limiting the number of increments to 20, 200, and 2000 respectively.

Figures 17 through 26 show the results from the parametric study.

Linear Elastic Method

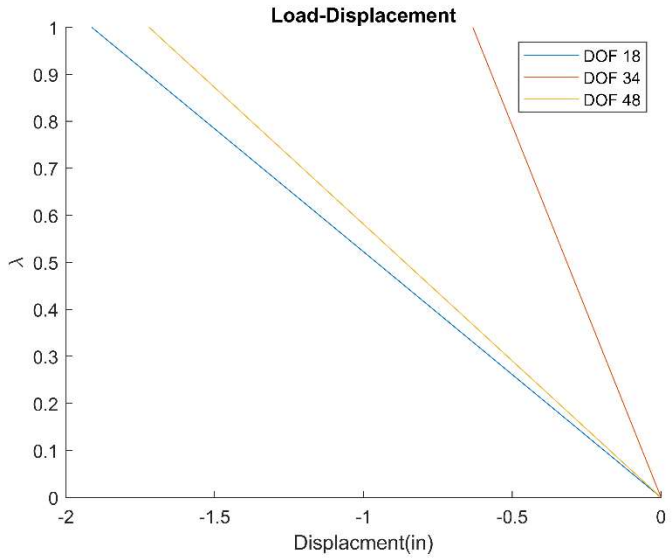


Figure 17: Linear Elastic

Load Control (Newton-Raphson) Method

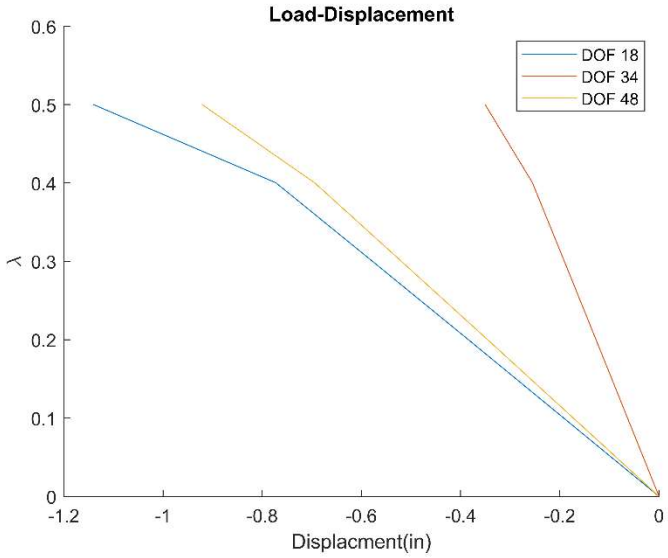


Figure 18: f=0.1

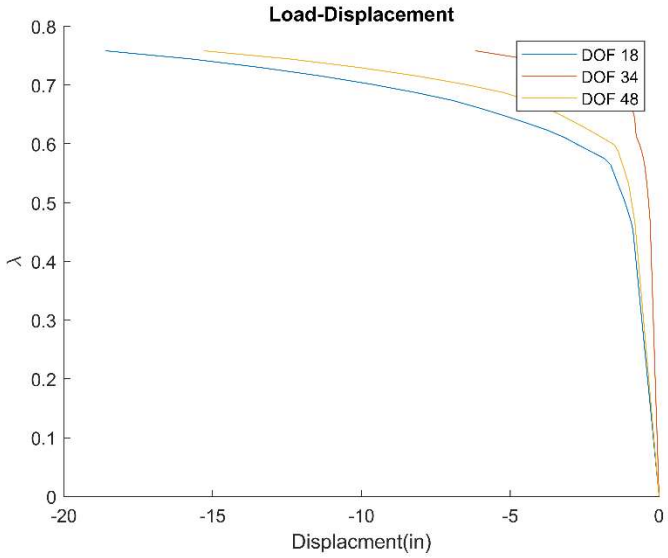


Figure 19: f=0.01

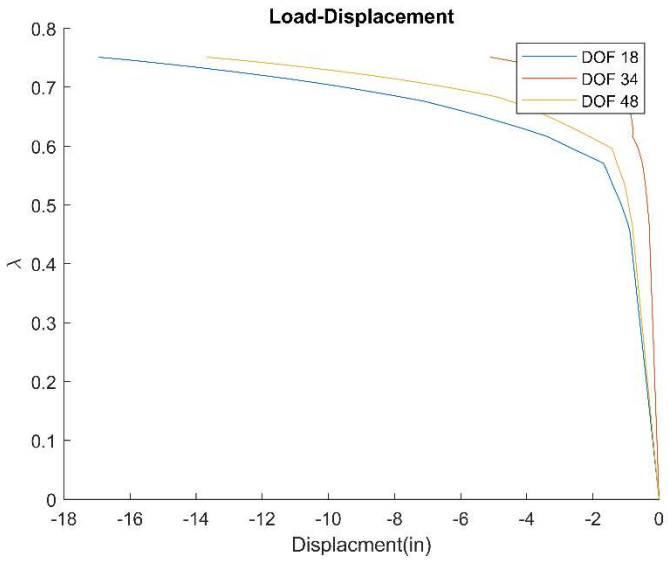


Figure 20:f=0.001

Work Control Method

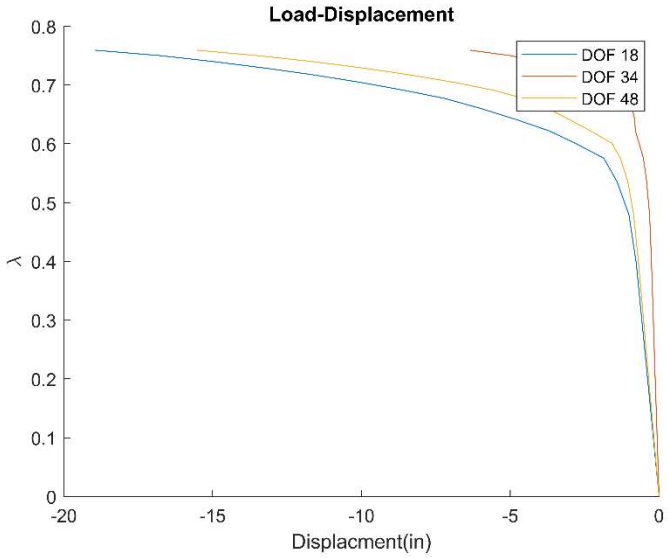


Figure 21: f=0.1

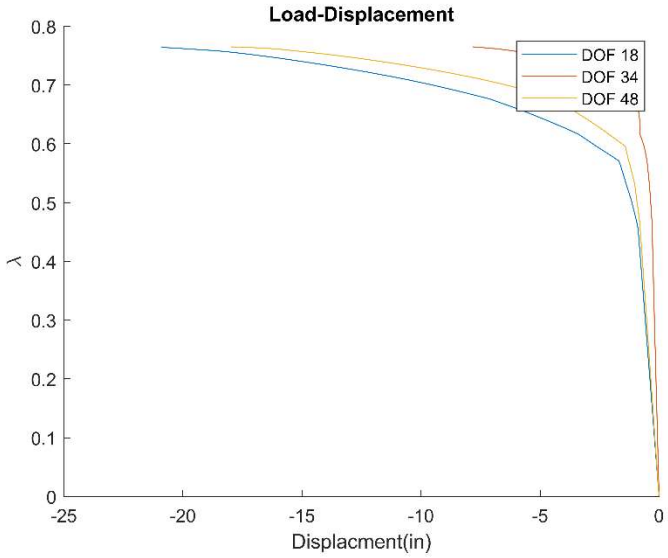


Figure 22:f=0.01

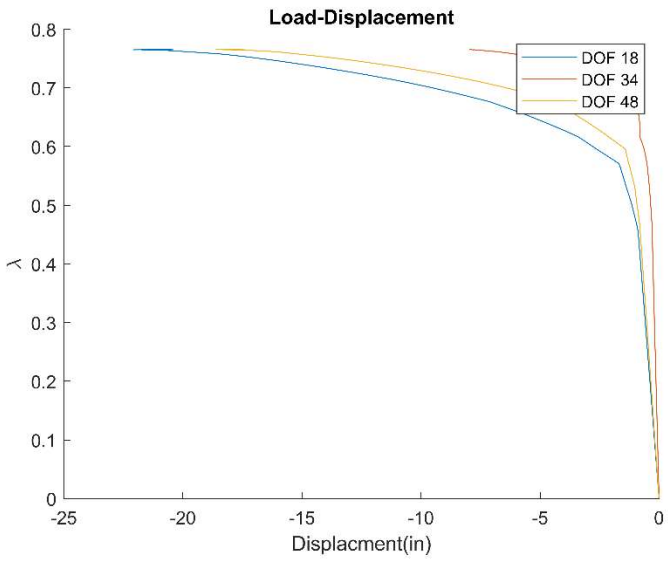


Figure 23:f=0.001

Arc Control Method

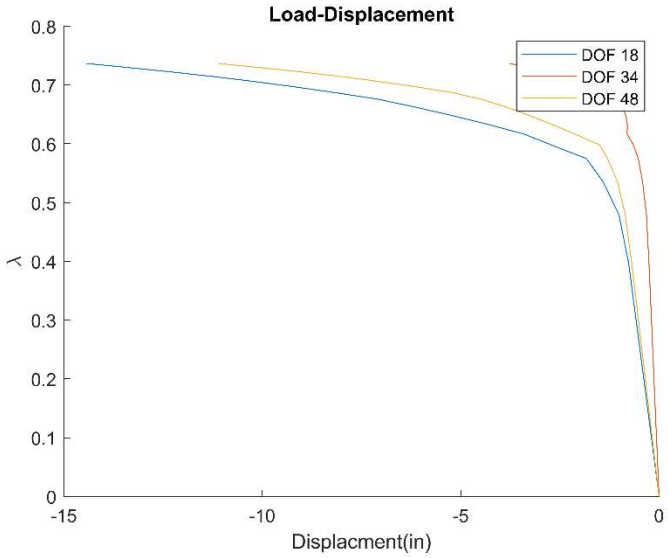


Figure 24:f=0.1

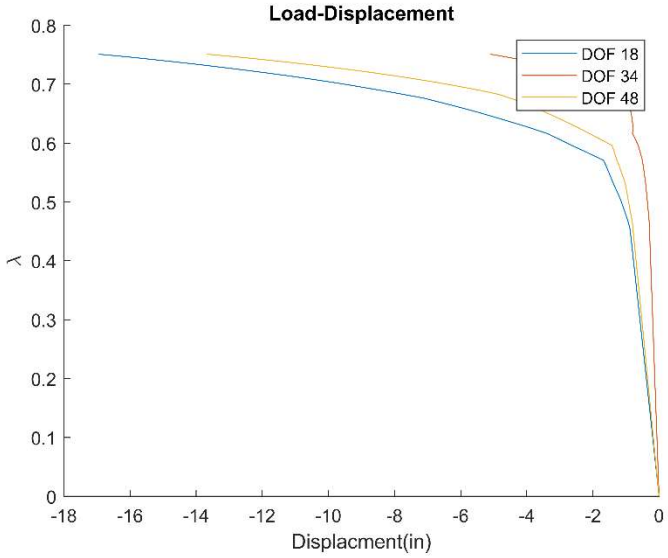


Figure 25:f=0.01

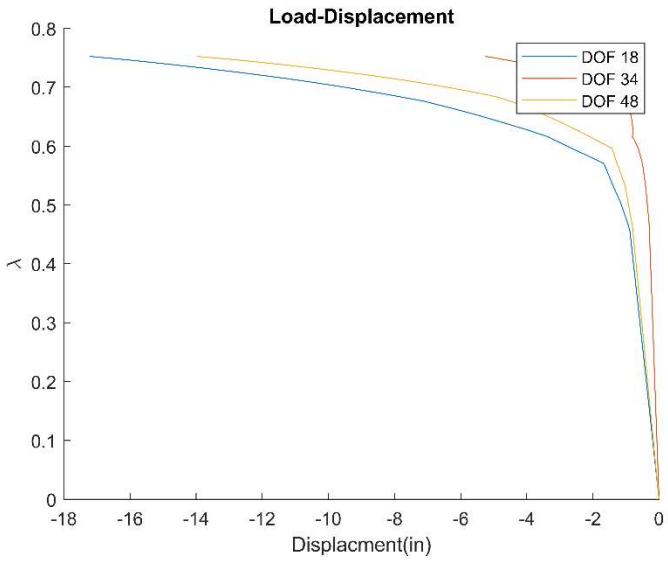


Figure 26:f=0.001

Results and Conclusion:

From these results, we can see the simulation methods mostly have the same result. If the simulation would have reached post-peak response, the arc-length and work-control method would have varied from the Newton-Raphson method more. The arc-length and work-control methods have a fair resolution at all step sizes, but the Newton-Raphson has a low resolution response at large step sizes. The arc-length control method generated smaller displacements overall compared to the work-control method.