Euler Buckling Solution Method

This document describes the numerical method used to compute elastic Euler-type buckling loads and mode shapes for a prismatic or stepped column with user-defined boundary conditions and nodal springs. The implementation is written in JavaScript and uses numeric.js for linear algebra.

1. Overview of the Model

The column is modeled as an assembly of Euler–Bernoulli beam elements. The user specifies a small number of primary segments (up to five) defined by their lengths L, elastic moduli E, and second moments of area I. Each primary segment may have different properties, allowing a stepped column. Nodes at the ends of these segments can be assigned boundary conditions (free, simply supported/pinned, or fixed) and optional nodal springs:

- Lateral translational springs kLat (K_l).

- Rotational (torsional) springs kTor (K^T).

The goal of the analysis is to determine the lowest elastic buckling load P_{lr} and the associated lateral deflected shape (mode shape) of the column, accounting for end conditions, stepped stiffness, and any nodal springs.

2. Primary and Refined Mesh

The user interacts with a coarse, "primary" model consisting of N_s primary segments and N_s+1 primary nodes. For numerical accuracy, the solver internally constructs a refined finite element mesh:

- Each user-defined primary segment is subdivided into 6 equal-length sub-elements.

- This creates 5 internal refined nodes per primary span, plus the primary nodes at the ends.

- The total number of refined nodes is $N_{at} = (N_s - 1)\cdot splits + 1$ with splits = 6.

Each refined node carries two degrees of freedom (DOFs):

- v: transverse (lateral) displacement.

- θ: cross-section rotation.

The refined model therefore has $2N_{at}$ global DOFs. The refined mesh improves the shape representation of the buckling modes, particularly for higher curvature regions or when springs and non-classical boundary conditions are present.

3. Element Stiffness Matrices

For each refined sub-element of length L and properties E and I inherited from its parent segment, the code builds two 4×4 element matrices in standard local DOF order [v_i, θ_i, v_r, θ_r]:

3.1 Elastic Bending Stiffness Matrix K_e

The elastic (bending) stiffness matrix for an Euler–Bernoulli beam element is:

$$K_e = (E\,I / L^3) \cdot \begin{bmatrix} 12 & 6L & -12 & 6L \\ 6L & 4L^2 & -6L & 2L^2 \\ -12 & -6L & 12 & -6L \\ 6L & 2L^2 & -6L & 4L^2 \end{bmatrix}$$

This matrix is implemented directly in the JavaScript function elemKe(E, I, L). It enforces compatibility of displacements and rotations at the element ends under pure bending.

3.2 Geometric Stiffness Matrix K_a for Unit Axial Load

Buckling is treated via geometric stiffness. For a beam-column subjected to an axial compressive load P, the geometric stiffness matrix can be written as:

$$K_a(P) = P \cdot K_{arrat}$$

where K_{arrat} is the geometric stiffness matrix corresponding to a unit axial load (P = 1). The code precomputes this unit-load matrix, then scales by P implicitly through the eigenvalue formulation. For a unit compressive load, the element matrix K_{arrat} is:

$$K_{arrat} = (1 / (30\,L)) \cdot \begin{bmatrix} 36 & 3L & -36 & 3L \\ 3L & 4L^2 & -3L & -L^2 \\ -36 & -3L & 36 & -3L \\ 3L & -L^2 & -3L & 4L^2 \end{bmatrix}$$

This matrix is implemented in the function elemKg_unit(L). It captures the stiffness contribution associated with axial load-induced bending (the classic P-Δ effect).

4. Global Assembly

Two global matrices are assembled on the refined mesh:

- K: global elastic bending stiffness matrix.

- K_{arrat}: global geometric stiffness matrix for P = 1.

For each refined sub-element, the corresponding 4×4 element matrices K_e and K_{arrat} are added into the global matrices using a standard assembly procedure based on the element DOF map [2r, 2r+1, 2(r+1), 2(r+1)+1].

After all elements are assembled, nodal springs are incorporated at the primary nodes. Each primary node p is mapped to its refined node index r = p · splits. Then:

- If kLat > 0 at node p, K[2r, 2r] is incremented by kLat (a lateral translational spring).

- If kTor > 0 at node p, K[2r+1, 2r+1] is incremented by kTor (a rotational spring).

5. Boundary Conditions and DOF Reduction

Boundary conditions are set only at primary nodes (those the user edits), then mapped to the refined mesh:

- For a primary node p with index r = p · splits in the refined mesh:

  - Fixed: both v and θ are constrained (removed from the free DOF set).

  - Simply supported / pinned: the lateral displacement v is constrained; the rotation θ is free.

  - Free: both v and θ are free.

- Interior refined nodes (those not coincident with primary nodes) are treated as free in both DOFs.

The solver constructs an index list of free DOFs and extracts reduced matrices:

- K_r: the submatrix of K restricted to free DOFs.

- K_{ar}: the submatrix of K_{arrat} restricted to free DOFs.

This reduction enforces the boundary conditions exactly and avoids singularities due to constrained DOFs.

6. Eigenvalue Buckling Formulation

For small-deflection elastic buckling of an axially loaded beam-column, the governing eigenvalue problem is:

$$(K - P\,K_a)\,\varphi = 0$$

where K is the elastic stiffness, $K_a = K_{ar}$ is the geometric stiffness matrix for unit axial load, P is the unknown critical load, and φ is the corresponding mode shape. On the reduced (free-DOF) system, this becomes:

$$(K_r - P\,K_{ar})\,\varphi_r = 0$$

Rewriting, the problem is equivalent to the standard linear eigenproblem:

$$K_r^{-1}\,K_{ar}\,\varphi_r = (1 / P)\,\varphi_r$$

The code constructs the matrix $M = K_r^{-1}\,K_{ar}$ by solving a sequence of linear systems $K_r\,X = K_{ar}$ for each column of K_{ar} using LU factorization (numeric.LU and numeric.LUsolve). The resulting matrix M is then passed to numeric.eig to obtain eigenvalues λ_i and eigenvectors φ_i:

$M \varphi_i = \lambda_i \varphi_i$

By comparison, $\lambda_i = 1 / P_i$. Thus, positive eigenvalues are converted to buckling loads via:

$P_i = 1 / \lambda_i$

The candidate eigenpairs are sorted by ascending P_i, and the lowest positive buckling load is selected as the fundamental critical load P_{lr}. The corresponding eigenvector is taken as the buckling mode shape.

7. Expansion Back to Full and Primary DOFs

The eigenvectors returned by numeric.eig correspond to the reduced (free-DOF) system. To obtain the full refined-mode shape, the solver expands these vectors back into the complete $2N_{at}$-DOF space, inserting zeros at constrained DOF positions. This step is performed by the expandVector function.

Because the user is primarily interested in displacements and rotations at the original primary nodes, the code extracts a primary-node mode vector by sampling the refined solution at $r = p \cdot splits$ for each primary node index p. This produces a $2(N_s+1)$-length vector $[v_0, \theta_0, v_1, \theta_1, \ldots, v_s, \theta_s]$.

Both the refined mode vector and the primary-node mode vector are stored. The refined vector is used to draw a smooth mode shape curve; the primary vector is sampled at the main nodes and plotted as red dots along the column.

8. Mode Shape Visualization

The drawing routine computes the vertical coordinates of all primary nodes based on the current segment lengths and a visual scale factor. The undeformed column is drawn as a stack of rectangular segments, with width proportional to $\sqrt{(EI)}$ so that stiffer segments appear wider.

When a buckling solution is available, the lateral DOFs v from the refined mode shape are normalized and then multiplied by a visual gain factor so that the mode shape is clearly visible on the canvas. The code clamps the gain to prevent the shape from leaving the drawing area.

The mode shape is plotted as a smooth red curve in (x, y) space, where y follows the column's vertical coordinate and $x = x_l + gain \cdot v$. Red dots are drawn at the primary node positions for clarity.

A small information panel is also rendered in the canvas corner. It displays:

- The computed critical load P_{lr} from the eigenvalue analysis.

- When applicable, an analytical Euler critical load P_{tssok} for a uniform column with classical pinned or fixed end conditions.

9. Analytical Euler Load for Uniform Columns

When all segments share the same E and I, and the boundary conditions correspond to classical cases (pinned–pinned, fixed–pinned, fixed–fixed), the code computes an analytical Euler buckling load for comparison:

$P_{tssok} = \pi^2 E I / (K L_{tssok})^2$

where L_{tssok} is the total column length (sum of all segment lengths) and K is the effective length factor:

- K = 1.0 for pinned–pinned (both ends simply supported).

- K ≈ 0.699 for fixed–pinned (one end fixed, one pinned).

- K = 0.5 for fixed–fixed (both ends fixed).

These analytical values are computed in the analyticPcForCurrent function and displayed alongside the numerical P_{lr} for quick verification.

10. Validation and Smoke Tests

The script includes a runSmokeTests function that executes a small set of automated checks using known configurations:

- Test that drawing and click-target construction complete without errors.

- Pinned–pinned uniform column: verify that the numerical critical load is within about 7% of the analytical Euler value.

- Fixed–fixed uniform column: verify that the end rotations (top and bottom θ) are essentially zero in the computed buckling mode.

- Fixed–pinned uniform column: verify that the numerical critical load matches the K ≈ 0.699 Euler prediction within about 7%.

These smoke tests give confidence that the finite element assembly, boundary conditions, and eigenvalue solution are working as intended.

11. Assumptions and Limitations

The current ColumnX Euler buckling solver is based on the following assumptions:

- Linear elastic material behavior (no yielding or plasticity).

- Small deflections; geometric nonlinearity is represented only via the geometric stiffness K_a.

- No initial imperfections or residual stresses are modeled explicitly.

- Axial load is uniform along the column and applied through the centroidal axis.

- Shear deformation and axial shortening are neglected (Euler–Bernoulli theory).

Within these assumptions, the method provides accurate estimates of elastic buckling loads and mode shapes, including the effects of stepped EI, mixed boundary conditions, and nodal translational or rotational springs.

The overall approach can be extended in future versions to incorporate geometric or material nonlinearities, initial imperfections, or dynamic buckling effects, but the present implementation is focused on classic static Euler-type buckling.