# Beam Modal Solver

# User & Method Documentation

*Generated: January 26, 2026*

## 1. Overview

The Beam Modal Solver is a browser-based finite element (FE) tool for computing the first three bending natural frequencies and corresponding mode shapes of an Euler–Bernoulli beam assembled from user-defined segments. Each segment can have its own elastic modulus (E), second moment of area (I), length (L), density (ρ), and cross-sectional area (A). Boundary conditions and optional nodal attachments (lumped weight, translational spring, rotational spring) are applied at the primary nodes. The solver constructs global stiffness and mass matrices on a refined internal mesh and solves an eigenvalue problem to estimate modal frequencies and shapes.

## 2. What the Tool Computes

- Global mass matrix M and stiffness matrix K for bending (w, θ DOFs per refined node).
- Eigenvalues λ of the reduced system $A = M^{-1}K$ (after applying constraints).
- Natural frequencies $f = \sqrt{\lambda} / (2\pi)$ in Hz for the lowest three positive eigenvalues.
- Mode shapes (normalized so max $|w| = 1$) and plots/animation of deflected shape versus x.

## 3. User Interface

### 3.1 Toolbar Controls

| Control | Purpose | Notes / Typical Use |
|---|---|---|
| Segments (max 10) | Sets number of beam segments. | Click “Build Beam” to create the segment and node arrays. |
| Gravity (G) | Sets gravitational constant used to convert weight to mass. | Mass added as W/G. Use 386 in/s² (US) or 9.806 m/s² (SI). |

| | | |
|---|---|---|
| Build Beam | Initializes segment and node properties. | Resets plots and modal results; sets default total length of 96 split evenly. |
| Compute Modes | Assembles refined FE model and solves for first 3 modes. | Populates frequency list, toggles, and combined mode plot. |
| Export JSON | Exports current configuration and computed mode data. | Includes segments, nodes, gravity, frequencies, and mode ordinates. |
| Import JSON | Imports a previously exported configuration. | Restores segments/nodes/gravity; recompute modes after import. |
| Export CSV | Exports mode ordinates versus x as a CSV table. | Columns include x and each mode shape ordinate. |

## 3.2 Beam Canvas Editing

The upper canvas displays the beam as a sequence of gray bars (segments) with blue dots (primary nodes). Click interactions open SweetAlert2 forms to edit properties.

- Segment editing:
  - Click inside a segment bar to edit that segment's properties (E, I, L, ρ, A).
  - After editing, the beam redraws and segment labels update.
- Node editing:
  - Click on a blue node dot to edit boundary condition and attachments.
  - Node form fields: BC (free/pinned/fixed), W, Kv, Kr.
- On-canvas glyphs:
  - Pinned support: triangular support symbol.
  - Fixed support: vertical clamp symbol.
  - Kv > 0: vertical spring glyph.
  - Kr > 0: rotational spring glyph.
  - W > 0: node dot drawn larger; labels show W/Kv/Kr above the node.

### 3.3 Results & Visualization Panels

- Frequency list:
  - Displays f1–f3 in Hz and a note that modes are normalized to max |w| = 1.
- Mode toggles:
  - Checkboxes allow hiding/showing each computed mode in the combined plot.
- Combined mode plot:
  - Plots selected mode shapes as normalized deflection versus x (three guide lines at +1, 0, −1).
- Animation controls:
  - Play/Pause animate a single selected mode as $w(x,t)=w(x)\cdot\sin(2\pi f t\cdot\text{speed})$.
  - Mode selector chooses which mode to animate.
  - Speed slider scales animation frequency; Amp slider scales displayed amplitude.

## 4. Typical Workflow

1. Set the desired number of segments (1–10) and gravity constant G.
2. Click Build Beam to initialize the model.
3. Click segments to edit E, I, L, ρ, and A as needed.
4. Click nodes to set boundary conditions (free/pinned/fixed) and optional attachments (W, Kv, Kr).
5. Click Compute Modes to compute the first three bending natural frequencies and plot the mode shapes.
6. Optionally hide/show modes, animate a selected mode, or export the results to JSON/CSV.

## 5. Solution Method

### 5.1 Structural Model and DOFs

The solver uses Euler–Bernoulli beam bending with two degrees of freedom per node: transverse deflection w and rotation θ. Primary nodes are defined at segment boundaries (S+1 nodes). For modal accuracy, each segment is internally refined into a fixed number of sub-elements: INTERNAL_NODES_PER_SEG = 10, giving (m+1) sub-elements per segment and a refined node count of $Nn\_ref = S\cdot(m+1) + 1$.

### 5.2 Element Stiffness Matrix

For each refined sub-element of length $L_e$ and bending rigidity EI, the standard 4×4 Euler–Bernoulli stiffness matrix is used:

$$k_e = (EI / L_e^3) \cdot \begin{bmatrix} 12 & 6L_e & -12 & 6L_e \\ 6L_e & 4L_e^2 & -6L_e & 2L_e^2 \\ -12 & -6L_e & 12 & -6L_e \\ 6L_e & 2L_e^2 & -6L_e & 4L_e^2 \end{bmatrix}$$

## 5.3 Consistent Element Mass Matrix

The element mass is assembled using the consistent Euler–Bernoulli beam mass matrix. The mass per unit length is:

$$\mu = (\rho / G) \cdot A$$

where ρ is weight density (force/volume) and G is the gravitational constant (length/time²). For each sub-element:

$$m_e = (\mu L_e / 420) \cdot \begin{bmatrix} 156 & 22L_e & 54 & -13L_e \\ 22L_e & 4L_e^2 & 13L_e & -3L_e^2 \\ 54 & 13L_e & 156 & -22L_e \\ -13L_e & -3L_e^2 & -22L_e & 4L_e^2 \end{bmatrix}$$

## 5.4 Nodal Attachments: Lumped Weight and Springs

At each primary node p, the tool optionally adds:

- Lumped mass from an applied weight W: $M_{ww} \mathrel{+}= W/G$ (added to the transverse DOF only).
- Translational spring Kv: $K_{ww} \mathrel{+}= Kv$.
- Rotational spring Kr: $K_{\theta\theta} \mathrel{+}= Kr$.

## 5.5 Boundary Conditions and Constraint Handling

Constraints are applied at primary nodes and mapped onto the refined grid (primary node p corresponds to refined node index $p \cdot (m+1)$). Boundary conditions are enforced by removing constrained DOFs from the eigenproblem:

- Free: no constrained DOFs.
- Pinned: $w = 0$ (constrains transverse DOF only; rotation remains free).
- Fixed: $w = 0$ and $\theta = 0$ (constrains both transverse and rotational DOFs).

After forming reduced matrices $K_{red}$ and $M_{red}$, the solver constructs $A = M_{red}^{-1} K_{red}$. Eigenpairs of A yield $\lambda = \omega^2$, with $\omega = \sqrt{\lambda}$ and $f = \omega/(2\pi)$.

## 5.6 Eigenvalue Solution and Mode Normalization

The code uses numeric.js to compute eigenvalues and eigenvectors of A. It filters to positive finite eigenvalues, sorts them, and retains the lowest three. Each corresponding eigenvector is expanded back to the full DOF set (constrained DOFs set to zero) and normalized such that max

|w| across all refined nodes equals 1. This normalization makes the plotted mode shapes directly comparable.

### 5.7 Plotting and Animation

Mode shapes are plotted as normalized deflection w(x) versus x along the refined node positions. For animation, a single selected mode is displayed as:

$$w(x,t) = w(x) \cdot \sin(2\pi f t \cdot \text{speed}) \cdot \text{amp}$$

where speed and amp are user-controlled scalars that affect the visualization only (not the computed frequencies).

## 6. Units, Assumptions, and Limitations

### 6.1 Units

The tool does not enforce a unit system; the user must supply consistent units. A common US-consistent choice is:

- Length: in; Force: lbf; Mass: lbf·s²/in; Gravity: G = 386 in/s².
- E: lbf/in²; I: in⁴; A: in²; ρ: lbf/in³; W: lbf; Kv: lbf/in; Kr: in·lbf/rad.

For SI, use meters, newtons, kilograms, and G = 9.806 m/s², with ρ as N/m³ if you keep the same $\mu=(\rho/G)A$ interpretation.

### 6.2 Modeling Assumptions

- Euler–Bernoulli beam theory (plane sections remain plane; shear deformation neglected).
- Linear elasticity, small deflections, and constant properties within each segment (piecewise constant along the beam).
- Undamped, free vibration (no damping matrix, no forcing).
- Out-of-plane modes only (transverse bending in one plane).

### 6.3 Practical Limitations and Failure Modes

- If M_red is singular (e.g., zero density/area everywhere and no lumped masses), the eigenproblem cannot be formed; the tool reports an error.
- Highly ill-conditioned models (extreme E·I contrasts or very small sub-elements) can reduce numerical accuracy.
- The approach $A=M^{-1}K$ solves a standard eigenproblem but does not exploit symmetric generalized eigen-solvers; for large problems a dedicated solver is preferred.
- Rotational inertia is included implicitly through the consistent mass matrix; if you intend a lumped-mass-only model, results will differ.

## 7. Export / Import Formats

### 7.1 JSON Export

Export JSON writes a single file containing: gravity, segment array, node array, frequencies (f and λ), and a table of mode ordinates versus x. The mode ordinate stored is the normalized transverse displacement w at each refined node.

### 7.2 JSON Import

Import JSON restores the configuration (segments, nodes, gravity) and resets results. After import, click Compute Modes to regenerate frequencies and mode shapes.

### 7.3 CSV Export

Export CSV produces a table with one row per refined x position and one column per computed mode. This is suitable for plotting in Excel or for post-processing in Python/Matlab.

## 8. References


7. S. S. Rao, Mechanical Vibrations, 6th ed., Pearson, 2017 — modal analysis fundamentals and beam vibration examples.
8. L. Meirovitch, Fundamentals of Vibrations, McGraw-Hill, 2001 — eigenvalue problems and mode normalization.
9. R. D. Cook, D. S. Malkus, M. E. Plesha, and R. J. Witt, Concepts and Applications of Finite Element Analysis, 4th ed., Wiley, 2001 — Euler–Bernoulli beam element stiffness and consistent mass matrices.
10. K.-J. Bathe, Finite Element Procedures, Prentice Hall, 1996 — matrix assembly and eigenvalue solution practices.


## Appendix A. Implementation Notes (Code-Level Mapping)

Refinement: each user segment is subdivided into (m+1) equal sub-elements, where m = INTERNAL_NODES_PER_SEG. Primary node p maps to refined node index p·(m+1). Constraints and nodal attachments are applied at these mapped refined DOFs.

Matrices: the code assembles dense M and K arrays. For larger models, sparse storage and a symmetric generalized eigensolver ($K\,\phi = \omega^2\,M\,\phi$) would be more efficient and numerically robust.