Project Manual and Solver Documentation

Auto-generated Documentation

December 7, 2025

Contents

List of Figures

List of Tables

Preface

PoDESL—short for "Portable Domain-Specific Language"—began as a compact script for prototyping finite element solvers and has grown into a full-stack ecosystem: a declarative DSL, a suite of structural/thermal/multiphysics solvers, a browser-based IDE, and a packaging pipeline that ships cross-platform executables. This manual is the official reference for that ecosystem. It is designed to scale with the project and will ultimately span roughly 200 pages, covering everything from first principles to implementation details.

0.1 Audience and Scope

Contributors need architectural insight. They should find guidance on extending the parser, adding new solver modules, and updating the IDE.

Analysts/Power users want reliable workflows. They can jump to the DSL chapters for syntax, then to solver chapters for domain-specific advice and benchmark data.

Stakeholders/Students require a high-level narrative plus references to reproducible examples. The appendices and example directory play that role.

0.2 Document Structure

- **Part I** (Ch. 1–3) focuses on the language and runtime.
- **Part II** (Ch. 5–4) catalogues every solver family, including equations, assumptions, and validation workflows.
- **Part III** (Ch. 2) documents the IDE, notebook mode, and automation hooks.
- **Part IV** (Ch. 7) covers testing, packaging, and documentation pipelines.
- **Appendices** (B–A) aggregate formula sheets and an annotated example directory.

0.3 How to Contribute to the Manual

To transform this outline into a full technical handbook:

1. **Expand with narrative**: each section should include a mix of prose, figures, tables, and code snippets. When adding a solver, document its assumptions, inputs, and outputs here.

2. **Cross-reference code**: cite modules (e.g., `podesl/solvers/frame2d_static.py`) so that readers can inspect the implementation.
3. **Include reproducible examples**: tie every major concept to a DSL file under `examples/`, and show how to run it via CLI or IDE.
4. **Update figures regularly**: the IDE evolves quickly; keep screenshots and workflows current.
5. **Automate builds**: use `pdflatex` or `latexmk` to render the manual as part of release candidates.

0.4 Versioning and Change Log

Each major release of PoDESL should bump the manual's version and include a "What's New" summary (future work). Keep a change log in this chapter so readers can trace evolution across releases.

0.5 Suggested Reading Path

For newcomers, we recommend the following order:

1. Read the DSL overview to grasp syntax and style.
2. Jump to the solver chapter corresponding to your domain of interest (structural, thermal, multiphysics).
3. Explore the IDE chapter to understand how workflows are executed in the browser.
4. Review the appendices for formulas and detailed example descriptions.

Chapter 1

DSL Overview

The PoDESL language is the entry point for every simulation workflow in this project. This chapter attempts to be both a tutorial and a reference: it introduces the language philosophy, provides a formal grammar, and walks through nontrivial examples. Future revisions should continue expanding each subsection with figures, flowcharts, and extended case studies so that the complete chapter spans dozens of pages. In particular, we recommend dedicating future pages to screen captures of the IDE, annotated sample DSL files, and tables comparing different solver families.

1.1 Language Philosophy

PoDESL is a declarative domain-specific language that translates directly into Python code. Its goals are:

- **Legibility**: a DSL file reads like a report. Problem statements, inputs, solution steps, and reporting directives are grouped into explicit blocks.
- **Composability**: DSL snippets can call other DSL files (e.g., study drivers) or export intermediate data that flows into downstream tools such as Abaqus.
- **Tooling parity**: the exact same DSL file can be executed via `python -m podesl.cli`, inside the browser IDE (editor or notebook mode), or embedded in CI smoke tests.

1.2 Grammar and Syntax Reference

The grammar is intentionally compact. A complete EBNF, which we plan to include in a future revision, can be derived from `podesl/parser.py`. Until then, this section captures the essential structure in prose and tables so that new authors understand what the parser expects.

1.2.1 High-Level Grammar Sketch

Construct	Description
`problem`	header + `block_list`
`block_list`	one or more block definitions
`block`	`GIVEN`/`SOLVE`/`EQUATIONS`/`REPORT`
`statement`	assignment or command (`print`, `export`, ...)
`expression`	evaluated by `eval_expr` under the PoDESL sandbox

Comments begin with `#` and run to the end of the line. Statements within a block must be indented consistently (two spaces recommended). Multiple statements on the same line may be separated by semicolons, but this is discouraged because it complicates diff reviews.

1.2.2 Problem Header

Every file begins with a `PROBLEM` declaration:

```
PROBLEM "Domain : Category : Objective"
```

The title feeds into `podesl/problem_codes.py`, which normalizes it into a canonical code such as `SOLID.Truss2D.Static`. Table 1.1 shows a subset of the mapping.

Title Example	Description	Canonical Code
`Solid : Truss2D : Static`	2D bar structures with axial DOFs only.	`SOLID.Truss2D.Static`
`Thermal : 2D : Transient`	Heat conduction with time integration and optional convection.	`THERMAL.2D.Transient`
`Study : MonteCarlo : Run`	Statistical post-processing over repeated solves.	`STUDY.MonteCarlo.Run`

Table 1.1: Sample mapping from human-readable titles to canonical solver codes.

1.2.3 Blocks and Statements

The language accepts the following top-level blocks:

`GIVEN` Declare inputs: scalars, arrays, dictionaries, lambda expressions, or nested data used during setup.

`SOLVE` Optional block for in-DSL manipulations before dispatching to the solver (e.g., computing loads or boundary collections).

`EQUATIONS` Special block for algebraic systems solved by the linear algebra module.

`REPORT` Actions executed after the solver returns.

Statements inside a block follow Python-like assignment syntax. The snippet below annotates how variables flow into the runtime environment:

```
PROBLEM "Solid : Truss2D : Static"
GIVEN
  nodes = [[0,0],[1,0],[1,1]]
  elems = [[0,1],[1,2]]
  E = 210e9
  loads = [[2, 0.0, -1000.0]]
  fix = [[0,"both",0.0],[1,"uy",0.0]]
REPORT
  print U    # prints env['U'] (displacements)
```

The parser stores each assignment in `env`, so later expressions can refer to previously defined names. The `REPORT` commands can access both `env` and the solver result dictionary.

1.2.4 Expressions and Safe Calls

Expressions are evaluated using Python's `eval` on a whitelisted set of globals: `numpy` (aliased as `np`), `math`, and helper functions like `linspace`, `array`, `ones`, etc. For example:

```
  loads = [[2, 0.0, -1e3 * np.sin(time)] for time in linspace(0, 1, 5)]
```

Any attempt to call disallowed functions raises a runtime error, preserving the sandbox. Table 1.2 lists the default helpers; contributors can extend `podesl/runtime.py` to add more.

Function	Description
`linspace(a,b,n)`	Generates evenly spaced points using NumPy.
`array([...])`	Converts lists to NumPy arrays.
`zeros(n), ones(n)`	Convenience vector constructors.
`diag(entries)`	Builds diagonal matrices.
`help("Title")`	Prints solver documentation from the CLI.

Table 1.2: Safe helper functions available inside expressions.

1.3 Example Gallery

1.3.1 Truss Primer

Walk through the truss example introduced above:

1. Analyze the geometry (nodes/elements).
2. Derive the element stiffness matrix and annotate the DSL lines that feed into the solver.
3. Show the resulting displacements and axial forces, perhaps with a plot.

1.3.2 Thermal Conduction

Demonstrate a 2D transient problem that mixes Dirichlet and flux boundaries. Explain how arrays of boundary conditions are normalized by `podesl/solvers/input_utils.py`. Include code listings and sample plots. To help the reader, document the mapping between DSL keywords and solver calls. For example, show the dictionary passed to `solve_heat2d` and point out where `dirichlet`, `flux`, and `convection` entries land.

1.3.3 Multiphysics Study

Highlight a study workflow (e.g., Monte Carlo) where the DSL file references another DSL. Explain how `REPORT export "file.csv" result["stats"]` is used to capture aggregated data.

1.4 Style Guide and Validation Rules

1.4.1 Formatting

Use two-space indentation and group multiple assignments per line only when short. The IDE exposes a formatter (Ctrl+Shift+F) that enforces these rules.

1.4.2 Validation

The parser enforces that statements are indented under their block headers and rejects commands in unsupported blocks. When developing new language features, add regression tests under `tests/` and update the manual accordingly.

1.5 Common Pitfalls and Patterns

1.5.1 Unit Consistency

Because PoDESL does not enforce units, teams should adopt an agreed-upon base system (e.g., SI). When mixing modules, document expected units in the DSL header or comments. Future work could introduce a lightweight unit checker, outlined here with TODO markers for contributors.

1.5.2 Parameter Sweeps

Parameter sweeps can be scripted directly in DSL (arrays of loads or material sets) or delegated to study drivers. This section should eventually include a mini case study showing how the IDE notebook mode can vary a parameter (e.g., cross-sectional area) across cells and plot the resulting responses.

1.6 Next Steps for Contributors

1. Add a formal grammar appendix referencing parser functions such as `_parse_block` and `_split_inline_assignments`.

2. Create sidebars comparing DSL syntax to equivalent Python pseudocode to help new users transition.

3. Insert IDE screenshots showing the File/Tools menus, notebook cells, and chart/report panels to reinforce how DSL scripts are authored and executed.

Chapter 2

IDE and Workflow Automation

The PoDESL IDE provides a browser-based environment with notebook mode, tabs, chart/report builders, history tracking, and Abaqus export. This chapter should grow into a self-contained user guide containing screenshots, user stories, and implementation details. The current layout mirrors Jupyter: a white theme, menu-driven commands, and the ability to switch between editor and notebook mode without leaving the browser.

2.1 Architecture Overview

2.1.1 Server

- **Core module**: `podesl/ide/server.py`.
- **Endpoints**: `/api/run`, `/api/format`, `/api/examples`, `/api/file`, `/api/chart`, `/api/report`, `/api/export`, and `/api/history`.
- **Execution context**: uses `podesl.execution` to capture stdout/stderr and store the last solver result for chart/report evaluation.

2.1.2 Frontend

Implemented with plain JavaScript:

- Ace editor (light `xcode` theme) for traditional coding.
- Notebook renderer that stores cells in application state and displays per-cell output panes.
- Chart.js for interactive plotting and a Markdown report builder that downloads summaries as '.md' files.

2.1.3 VS Code Extension

The companion extension under `vscode-extension/podesl-ide` launches the IDE server ('python -m podesl.cli –ide –ide-no-browser') and opens the browser from within VS Code. Future pages should include installation screenshots.

2.2 Notebook Mode

1. Users switch modes via the File menu. Notebook cells appear below the tab bar, while the traditional editor is hidden.
2. Each cell has a textarea, a toolbar (‘Run Cell‘, ‘Delete‘), and an output block that stores solver logs.
3. ‘Run Cell‘ and ‘Run All‘ send the concatenated DSL to `/api/run`; results populate the cell outputs and run history.

Discuss how notebook mode reuses the same backend as the editor and how engineers migrating from Jupyter can paste their narratives directly into DSL cells. Provide a table of keyboard shortcuts (Ctrl+Enter to run, Ctrl+S to save, Ctrl+Shift+F to format, etc.) and note how to rearrange cells or duplicate them for parameter sweeps.

2.3 Menus and Tooling

All commands live in the File/Tools menus:

File New/Save/Save As, mode switching, job settings modal, template builder modal.

Tools Run (editor or notebook), export Abaqus, copy result, format document, generate plots/reports.

Describe the modals in detail. For example, the Template Builder prompts for problem type, span, load, and material data, then inserts a ready-to-run DSL snippet into the active tab. Document the job settings dialog (job name, target path) and mention how those values feed into Abaqus export commands.

2.4 Diagnostics and History

Run results are recorded in a history list stored server-side (bounded deque). Each entry includes:

- Status (‘ok‘ or ‘error‘).
- Timestamp and file path.
- Optional diagnostics (line numbers parsed from error messages).

The frontend converts diagnostics into Ace annotations so the user can jump to the offending lines. History entries also provide quick “Copy JSON” buttons for sharing. Future revisions should include screenshots showing a run failure with highlighted lines and the corresponding history entry.

2.5 Collaboration and Export

2.5.1 Clipboard and Reports

‘Copy Result‘ pushes console output + JSON result to the clipboard. The report builder evaluates user-specified expressions and downloads a Markdown summary containing metrics, notes, and raw JSON.

2.5.2 Abaqus Export

Triggered via the Tools menu or from notebook scripts. Describe the workflow: user opens the Job Settings modal, specifies the job name and target path, then runs “Export Abaqus”. The server generates the ‘.inp‘ file using `podesl/abaqus_export.py` and returns node/element counts.

2.5.3 Light Theme and Customization

Explain the CSS variables controlling the light palette and note where to tweak them if custom branding is required. Mention that the IDE intentionally mirrors Jupyter’s visual style to ease onboarding.

2.6 Roadmap for IDE Chapter

1. Add annotated screenshots for each panel (sidebar, editor, notebook, console).
2. Include a “day in the life” scenario describing how an analyst edits a DSL file, runs notebook cells, exports results, and files a report.
3. Document the VS Code companion workflow in depth, including how the output channel displays server logs.

Chapter 3

Parser and Runtime Internals

This chapter exposes the internal mechanics of `podesl/parser.py`, `podesl/transpile.py`, and `podesl/runtime.py`. It is aimed at contributors who need to extend or debug the language tooling.

3.1 Parsing Pipeline

3.1.1 Line Reader and Block Scanner

Parsing begins with a simple line reader: the source is split into lines, comments (`# ...`) are stripped, and blank lines are skipped. The parser expects the first non-comment line to be a `PROBLEM` declaration; failure to do so raises `DSLParseError`. Once the header is read, the parser scans for block headers (`GIVEN`, `SOLVE`, `EQUATIONS`, `REPORT`) aligned at column zero. Pseudocode:

```
for line in lines:
  if line startswith BLOCK and indentation == 0:
     current_block = new block list
  else:
     assert indentation > 0
     current_block.append(line.strip())
```

Future revisions should include diagrams that illustrate how indentation levels map to nested statements and how inline assignments (multiple statements on one line) are split.

3.1.2 AST Structure

The parser outputs a dictionary:

- `problem`: metadata (title, canonical code, domain, categories).
- `blocks`: list of block dictionaries, each containing `type` and `stmts`.
- `stmts`: statements have `kind` (`assign` or `command`), `name`, `value`, and `line` (for diagnostics).

Add an illustration showing the AST for a simple DSL file, labeling each field.

3.2 Transpilation and Execution

3.2.1 Python Generation

`podesl/transpile.py` traverses the AST and emits Python source. Important details:

1. Variables defined in `GIVEN`/`SOLVE` become assignments in the transpiled script.
2. `REPORT` commands translate to function calls (`report_print`, `report_export_csv`, `report_plot`).
3. The generated `main()` function calls `dispatch` and then `store_last_run` so that IDE tooling can inspect the result.

Include a code listing that shows a small DSL and the corresponding Python output.

3.2.2 Execution Helpers

`podesl/execution.py` provides:

- `execute_transpiled`: runs the generated code in-process while capturing stdout/stderr using `contextlib.redirect_stdout`.
- `execute_source`: convenience wrapper that parses, transpiles, and executes a DSL string.
- `store_last_run` / `evaluate_last_expression`: expose the last environment/result so that the IDE can generate charts, reports, and per-cell outputs in notebook mode.

3.3 Safety and Expression Evaluation

Expressions are evaluated via Python's `eval` on a sandboxed dictionary:

- `__builtins__` is empty.
- The globals include `numpy` (aliased `np`), `math`, boolean literals, and a curated set of safe helper functions. These are stored in `SAFE_CALLS`.
- Results can reference solver outputs using the `result` mapping.

Future work: document potential enhancements (static analysis, type hints, unit checking) and note how to add new safe functions.

3.4 Extending the Parser

When adding new block types or statements:

1. Update the parser to recognize the block header and statement syntax.
2. Extend the transpiler to emit corresponding Python code.
3. Add tests under `tests/` covering success/failure cases.

4. Document the feature here and in the README.

Consider adding a flowchart showing how a DSL change propagates through parser, transpiler, runtime, and IDE features.

Chapter 4

Multiphysics and Study Solvers

Multiphysics drivers orchestrate multiple solver passes or wrap an entire class of statistical experiments. This chapter describes how to configure them via DSL, what data structures they produce, and how to extend them.

4.1 Fusion Studies

4.1.1 Steady Fusion

The steady fusion driver alternates between two physics domains until residuals fall below a tolerance:

1. Solve the `mechanical` DSL referenced in the `GIVEN` section.
2. Pass displacements/strains into the `thermal` DSL (or vice versa).
3. Apply relaxation factors (α) and repeat until $\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\| < \epsilon$.

Document the DSL keys (`mechanical`, `thermal`, `max_iters`, `atol`) and provide a flowchart that states how JSON-like dictionaries are exchanged between runs. Suggest adding convergence plots (iteration vs. residual) to future revisions.

4.1.2 Transient Fusion

Transient fusion extends the above with time marching. The DSL accepts `dt`, `t_end`, and control options regarding whether both physics domains are advanced with the same timestep. Describe strategies for lagging one field, subcycling, or using predictor-corrector schemes.

4.2 Monte Carlo and Scenario Studies

4.2.1 Sampling Specification

DSL inputs such as

```
perturb = {
  "E": {"dist": "normal", "mean": 210e9, "std": 5e9},
```

```
  "A": {"dist": "lognormal", "mean": 1e-4, "sigma": 0.1}
}
```

define random variables. Document supported distributions (normal, lognormal, uniform) and how correlations could be introduced in future work.

4.2.2 Outputs and Reports

Monte Carlo runs populate `result["samples"]` and summary statistics (`mean`, `std`, histograms). Show how to export histograms via the IDE report builder or via `REPORT export` commands. Scenario drivers (`study_scenarios`) produce comparative tables; explain the JSON schema so that users can script dashboards or spreadsheets. Provide a worked example with two scenarios and demonstrate how to compare them using plots exported from the IDE.

4.3 Optimization

Optimization DSLs specify an objective function and optional constraints. The current implementation uses finite-difference gradients and gradient descent. Outline:

- How to define design variables in the DSL (lists/dictionaries).
- Step-size and termination criteria.
- Example: minimizing cross-sectional area subject to displacement limits.

Mention opportunities for extension (adjoint methods, constraint solvers) and note where in the codebase (`podesl/solvers/optimize.py`) such changes would live.

4.4 Future Work

Provide TODO lists for additional multiphysics couplings (FSI, thermo-electrical apps) with references to planned solver modules. Encourage contributors to add case studies covering calibration workflows, probabilistic safety factors, or coupled thermo-structural fatigue analysis.

Chapter 5

Structural Solvers

This chapter catalogues every structural solver shipped with PoDESL. Each subsection now describes the underlying mathematics, DSL inputs, and expected outputs. When fully expanded it should include derivations, convergence studies, and example plots so that a reader could reproduce the solver from scratch.

5.1 Truss Solvers

5.1.1 Linear 2D Truss

The element stiffness matrix in local coordinates is

$$\mathbf{k}^{(\text{local})} = \frac{EA}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \tag{5.1}$$

which is rotated via

$$\mathbf{k}^{(\text{global})} = \mathbf{T}^{\top}\mathbf{k}^{(\text{local})}\mathbf{T}, \qquad \mathbf{T} = \begin{bmatrix} c & s & 0 & 0 \\ 0 & 0 & c & s \end{bmatrix}, \tag{5.2}$$

where $c = (x_2 - x_1)/L$ and $s = (y_2 - y_1)/L$. The DSL inputs map directly to these symbols. Create a multi-page walkthrough of a cantilever truss, including:

1. Tabulated global stiffness matrix entries.
2. Boundary condition vectors, showing how Dirichlet conditions are merged.
3. Axial force plots generated from IDE output or Matplotlib. Describe how to export data via `REPORT export` and how to load it into plotting scripts.

Recommended Tables

- **Table:** element-by-element stiffness assembly.
- **Table:** mapping of DSL fields to solver arguments.
- **Figure:** truss geometry with DOFs labeled.

5.1.2 3D Truss and Modal Analysis

3D trusses extend the DOF count to three translations per node. Modal analyses solve the eigenproblem

$$(\mathbf{K} - \omega^2 \mathbf{M})\mathbf{u} = \mathbf{0}. \tag{5.3}$$

Document how to specify `rho` or nodal masses, how the DSL triggers a modal run (by selecting the appropriate `PROBLEM` code), and how the result dictionary exposes eigenpairs. Add benchmark data from `examples/truss3d_modal_*.dsl`.

5.2 Frame and Beam Solvers

5.2.1 Frame2D Static and Nonlinear

Frame solvers add rotational DOFs and use Hermitian shape functions. For nonlinear runs we employ a Newton-Raphson loop with optional arc-length control. Discuss:

1. How Rayleigh damping coefficients (`rayleigh_a0`, `rayleigh_a1`) are interpreted.
2. Load stepping strategies defined in the DSL (e.g., `load_steps`).
3. P-delta effects in `frame2d_pdelta`.

Provide pseudocode for the solver (assemble $\mathbf{K}$, apply BCs, iterate until residual $< \varepsilon$) and connect each step to the relevant DSL fields.

Example: Portal Frame

1. Present the DSL snippet for a simple portal frame (nodes, elements, section properties).
2. Describe each Newton iteration, including update of internal forces, tangent stiffness, and load factor.
3. Include a table comparing the nonlinear response to reference solutions (tip displacement vs. load).

Plan to add plots of the load-displacement curve and deformed shapes, ideally exported from the IDE or a Matplotlib script that reads `U` and reaction arrays.

5.2.2 Frame3D and Beam Columns

Frame3D solvers generalize the above to six DOFs per node. Document:

- Required cross-section properties (`A`, `Iy`, `Iz`, `J`).
- Load history arrays (`loads_t`) for dynamic runs.
- How supports vs. Dirichlet BCs are normalized into the legacy format.

Add a table comparing solver modules (`frame2d_static.py`, `frame3d_beamcolumn.py`, etc.) and note the expected outputs (`U`, `member_forces`, reaction arrays). Plan to attach appendices with derivations of Timoshenko beam relationships and axial/shear load coupling.

5.3 Solid and Shell Solvers

5.3.1 Solid3D Hex8/Hex20

Solid solvers rely on isoparametric elements. Outline:

1. Constitutive matrix **D** for isotropic elasticity.
2. Gaussian quadrature rules (2x2x2 for Hex8, 3x3x3 for Hex20).
3. Load vector construction, including body forces and point loads.

Include a table (future work) enumerating required inputs: modulus, Poisson's ratio, density, damping ratios, nonlinear coefficients. Reference the DSL examples and explain how boundary conditions are applied via the merged format.

Cook's Membrane Create a detailed case study of Cook's membrane using the Hex8 solver:

- Mesh description (node coordinates, element indexing).
- Boundary conditions (clamped edge vs. shear loading).
- Convergence table (displacement at top corner vs. mesh refinement).
- IDE workflow for visualizing the VTK export.

5.3.2 Shell/Membrane/Mindlin Plates

Shell solvers (e.g., `shell_membrane_static`) and Mindlin plates introduce transverse shear corrections. Document:

- Theories used (Kirchhoff vs. Mindlin) and their limitations.
- Shear-locking mitigation strategies (reduced integration, selective scaling).
- Example inputs for plate thickness, material sets, and loading (distributed pressure vs. nodal forces).

5.4 Verification and Benchmarks

To ensure accuracy, curate a benchmark suite:

Cantilever beam Compare tip displacement and moment diagrams against analytical solutions.

Cook's membrane Report energy norm errors for different mesh densities.

Portal frame Validate nonlinear buckling and post-buckling behavior.

Encourage contributors to run `python tools/smoke_examples.py -execute` after modifying solver code. A future release should include tables listing strain energy, reaction forces, and relative errors for each benchmark so the chapter contributes significantly to the overall page count.

Chapter 6

Thermal and Thermo-Mechanical Solvers

Thermal analyses are a core capability of PoDESL. This chapter describes steady and transient conduction solvers, convection boundary treatments, and coupled thermo-mechanical workflows. Each section should eventually include derivations, mesh diagrams, and validation plots.

6.1 Governing Equations

Steady-state conduction solves

$$-\nabla \cdot (k \nabla T) = Q, \tag{6.1}$$

while transient conduction adds the storage term

$$\rho c_p \frac{\partial T}{\partial t} - \nabla \cdot (k \nabla T) = Q. \tag{6.2}$$

Derive the weak form and connect the notation to DSL entries (`k`, `rho`, `cp`, `heat`, etc.). This section should eventually show the weak form derivation:

$$\int_\Omega k \nabla T \cdot \nabla \phi \, d\Omega = \int_\Omega Q \phi \, d\Omega + \int_{\Gamma_q} \bar{q} \phi \, d\Gamma + \int_{\Gamma_h} h(T_\infty - T) \phi \, d\Gamma, \tag{6.3}$$

and connect each boundary integral to DSL keywords (`flux`, `convection`). Include a table translating mathematical symbols to DSL fields and highlight units to avoid confusion when coupling with structural solvers.

6.2 1D and 2D Conduction

6.2.1 Mesh and Boundary Conditions

Document how nodes/elems are specified for 1D bars, 2D quads/triangles, and 2D axisymmetric meshes. Provide tables explaining each boundary keyword:

- `dirichlet` / `fix_temp`: enforced temperature.
- `flux`: Neumann heat flux.

- `convection`: Robin boundary with film coefficient and ambient temperature.

Add a sample input table mapping boundary IDs to physical faces. Include diagrams for simple meshes (1D bar with left/right boundaries, 2D plate with top flux, etc.).

6.2.2 Example: Heated Plate

Create a multi-page walkthrough of `examples/heat2d_q4_block.dsl`, including mesh diagrams, boundary annotations, and an explanation of the output fields (`T`, `flux`, VTK exports).

6.3 Transient Solvers

6.3.1 Time Integration

Most thermal solvers use backward Euler or Crank–Nicolson discretization. Explain how the timestep `dt` and end time `t_end` are chosen, include stability guidance, and show pseudocode for the update loop. Mention how the IDE's notebook mode can run parameter sweeps over time discretization. Add a table comparing schemes:

- Backward Euler (implicit, unconditionally stable).
- Crank–Nicolson (second-order, potential oscillations).
- Explicit Euler (rarely used; include warning about stability limit $\Delta t \leq k\Delta x^2/(2\kappa)$).

6.3.2 Coupled Thermal-Mechanical

Modules like `thermomech2d_ps_coupled_plate` solve thermal and structural problems sequentially:

1. Solve the thermal problem to obtain $T(\mathbf{x})$.
2. Convert temperature gradients into thermal strains $\epsilon^{\text{th}} = \alpha(T - T_{\text{ref}})$.
3. Pass the strains into the structural solver via shared DSL variables.

Detail how the DSL expresses this coupling (shared dictionaries, `REPORT` exports/imports). Include a flowchart showing the sequential solve order and note how to use the IDE to visualize both temperature and structural displacement fields.

6.4 3D Heat Solvers and Axisymmetric Cases

Discuss element choices (Hex8 for volumetric conduction, H8 with mass lumping for transient), memory considerations, and axisymmetric assumptions. Encourage adding a figure showing the meridional plane used in the axisymmetric module. Include a subsection describing boundary condition mapping for axisymmetric problems (e.g., specifying `nodes` in r–z plane and implicit revolution around the z-axis).

6.5 Post-processing

Provide guidance on interpreting arrays (e.g., `T`, `flux`), exporting to CSV/VTK, and linking results into the report builder. Suggest a future subsection with Matplotlib scripts or IDE chart presets for temperature profiles, contour plots, and energy balance checks. List common quantities to plot (temperature at critical nodes, heat flux through boundaries, total heat input). Mention how to use the IDE chart tool to pick expressions like `result["T"][:,0]` vs. `linspace(0, L, n)`.

Chapter 7

Testing, Build, and Release Process

This chapter documents quality assurance and packaging workflows so that future contributors can reproduce releases and validate new solvers.

7.1 Automated Testing

7.1.1 Unit Tests

Tests live under `tests/`. Contributors should run:

```
python -m pytest
```

before opening pull requests. Encourage the addition of regression tests for any parser or solver changes, and document how to mock solver environments for fast feedback. Future pages should include a table summarizing each test file and the modules it covers.

7.1.2 Smoke Tests

`tools/smoke_examples.py` and `tools/audit_examples.py` execute all DSL files to ensure no regressions slip in. Provide instructions for running them in CI and recording failures in `example_audit.json`. Consider adding a subsection listing long-running vs. quick smoke suites so that CI can run an abbreviated set on every commit and the full set nightly.

7.2 Build and Packaging

Describe how to produce the Windows bundle via `build_windows.ps1` (PyInstaller one-file executable plus example assets). Enumerate the directory structure of `release/podesl-win64` and document post-build verification steps (launching `podesl-cli.exe`, running a sample DSL). Include a TODO for Linux/macOS packaging so future releases can be multi-platform. Add diagrams of the release folder layout (bin, examples, docs) and brief instructions for signing binaries if needed.

7.3 Documentation Pipeline

Explain how this manual is built (LaTeX in `docs/manual`). Include instructions for installing `latexmk` or running `pdflatex` twice to resolve references. Encourage contributors to update both README and the manual when adding features so that the final PDF approaches the 200-page goal. Document how to use the helper script `docs/manual/build_pdf.py` if available, and mention common pitfalls (MiKTeX proxy prompts, missing packages).

Chapter 8

Solver Documentation

8.1 axisym_q4_static

Documentation for solver `axisym_q4_static`

Overview

The `axisym_q4_static` solver belongs to the continuum solid mechanics group. It implements a finite element formulation for axisymmetric elasticity in bodies of revolution, where all fields are independent of the circumferential angle. The primary unknown field is the displacement vector $\mathbf{u}$.

Governing equations

In the absence of inertia and damping the quasi-static equilibrium equations are

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \quad \text{in } \Omega,$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{b}$ denotes body forces per unit volume.

For linear elastic materials the constitutive relation is

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon},$$

with $\mathbb{C}$ the fourth-order elasticity tensor and $\boldsymbol{\varepsilon}$ the infinitesimal strain tensor

$$\boldsymbol{\varepsilon} = \tfrac{1}{2}\big(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathsf{T}}\big).$$

The particular kinematic assumptions (plane stress, plane strain, axisymmetry) are imposed at the element level through the strain–displacement matrices and the form of $\mathbb{C}$.

Boundary conditions

Boundary conditions are specified as:

- prescribed displacements $\mathbf{u} = \bar{\mathbf{u}}$ on Γ_u (Dirichlet),
- prescribed tractions $\boldsymbol{\sigma}\mathbf{n} = \bar{\mathbf{t}}$ on Γ_t (Neumann),

with $\Gamma_u \cup \Gamma_t = \partial\Omega$ and $\Gamma_u \cap \Gamma_t = \emptyset$.

Finite element formulation

The displacement field is interpolated using standard isoparametric shape functions,

$$\mathbf{u}_h(x) = \sum_i N_i(x)\mathbf{u}_i.$$

The weak form of the equilibrium equations leads to the standard finite element system

$$[K]\{\mathbf{U}\} = \{\mathbf{F}\},$$

with

$$[K]_e = \int_{\Omega_e} [B]^{\mathsf{T}}\mathbb{C}[B]\, d\Omega, \qquad \{\mathbf{F}\}_e = \int_{\Omega_e} N^{\mathsf{T}}\mathbf{b}\, d\Omega + \int_{\Gamma_{t,e}} N^{\mathsf{T}}\bar{\mathbf{t}}\, d\Gamma,$$

where $[B]$ is the strain–displacement matrix for the chosen element type.

Element stiffness matrices and load vectors are assembled into the global system and displacement boundary conditions are enforced by modifying the global matrix and right-hand side.

Input DSL structure

In the project DSL, use of `axisym_q4_static` involves specifying:

- material properties (e.g. Young's modulus, Poisson ratio),
- mesh topology (nodes and elements),
- boundary condition sets for prescribed displacements and tractions,
- body forces or equivalent nodal loads where needed.

The exact syntax is consistent with other structural solvers and allows combining this solver with post-processing, load stepping, and study drivers.

Output fields

Typical outputs include:

- nodal displacements,
- element or integration-point stresses and strains,
- reaction forces at constrained degrees of freedom,
- derived scalar measures (e.g. von Mises stress).

Typical use cases

- analysis of plane and axisymmetric components under mechanical loading,
- 3D solid analysis of parts and assemblies in structural mechanics,
- benchmark problems for code verification in elasticity.

Limitations and notes

- The formulation assumes small strains and rotations unless extended variants are used.
- Material behaviour is linear elastic in the base implementation; any nonlinear or inelastic behaviour belongs to specialised solvers.
- Mesh quality and appropriate boundary condition specification are critical for accurate results.

8.2 bar1d

Documentation for solver `bar1d`

Overview

The `bar1d` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

In the purely static case the assembled finite element equations reduce to

$$[K]\{U\} = \{F\},$$

with $[K]$ containing both axial and bending stiffness contributions from frame or beam elements.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `bar1d` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.3 bar1d_thermoelastic_static

Documentation for solver `bar1d_thermoelastic_static`

Overview

The `bar1d_thermoelastic_static` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

In the purely static case the assembled finite element equations reduce to

$$[K]\{U\} = \{F\},$$

with $[K]$ containing both axial and bending stiffness contributions from frame or beam elements.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `bar1d_thermoelastic_static` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.4 bc_utils

Documentation for solver `bc_utils`

Overview

The `bc_utils` module provides supporting functionality used by the physics solvers. It does not correspond to an independent physical model, but instead offers utilities such as:

- boundary condition handling and parsing,
- input/output helpers for meshes and results,
- material or section data management,
- linear algebra wrappers or post-processing routines.

Role in the codebase

Rather than assembling and solving its own finite element model, `bc_utils` is called by other solvers to perform common tasks and to avoid duplication of logic. The precise responsibilities can be seen from the module's public API (functions and classes).

Input and output

Inputs and outputs are specific to the helper functions provided and may include:

- dictionaries representing solver state or model data,
- arrays of nodal and elemental information,
- handles to files or streams for reading/writing results.

As such, this module is typically not invoked directly from the user-facing DSL, but rather used internally.

8.5 beam1d

Documentation for solver `beam1d`

Overview

The `beam1d` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

In the purely static case the assembled finite element equations reduce to

$$[K]\{U\} = \{F\},$$

with $[K]$ containing both axial and bending stiffness contributions from frame or beam elements.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `beam1d` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.6 beam1d_modal

Documentation for solver `beam1d_modal`

Overview

The `beam1d_modal` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

For free-vibration (modal) analysis, the discretised equations of motion

$$[M]\{\ddot{U}\} + [C]\{\dot{U}\} + [K]\{U\} = \{F\}$$

are recast as a homogeneous eigenproblem

$$[K]\{\phi\} = \lambda[M]\{\phi\},$$

to compute natural frequencies and mode shapes.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `beam1d_modal` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.7 beam_buckling

Documentation for solver `beam_buckling`

Overview

The `beam_buckling` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

For buckling analysis, the solver forms both the elastic stiffness matrix $[K]$ and the geometric stiffness matrix $[K_G]$ associated with a pre-stress state. A typical linearised buckling formulation solves

$$\big([K] + \lambda[K_G]\big)\{\phi\} = \mathbf{0},$$

where λ are critical load multipliers and $\{\phi\}$ the associated buckling mode shapes.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `beam_buckling` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.8 beam_modal

Documentation for solver `beam_modal`

Overview

The `beam_modal` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

For free-vibration (modal) analysis, the discretised equations of motion

$$[M]\{\ddot{U}\} + [C]\{\dot{U}\} + [K]\{U\} = \{F\}$$

are recast as a homogeneous eigenproblem

$$[K]\{\phi\} = \lambda[M]\{\phi\},$$

to compute natural frequencies and mode shapes.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `beam_modal` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.9 contact1d_gap

Documentation for solver `contact1d_gap`

Overview

The `contact1d_gap` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

In the purely static case the assembled finite element equations reduce to

$$[K]\{U\} = \{F\},$$

with $[K]$ containing both axial and bending stiffness contributions from frame or beam elements.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `contact1d_gap` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.10 contact1d_penalty

Documentation for solver `contact1d_penalty`

Overview

The `contact1d_penalty` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

In the purely static case the assembled finite element equations reduce to

$$[K]\{U\} = \{F\},$$

with $[K]$ containing both axial and bending stiffness contributions from frame or beam elements.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `contact1d_penalty` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.11 dynamics

Documentation for solver `dynamics`

Overview

The `dynamics` solver integrates second-order dynamical systems of the form

$$[M]\{\ddot{U}\} + [C]\{\dot{U}\} + [K]\{U\} = \{F(t)\},$$

which arise from finite element discretisation of structural dynamics, vibration, and related problems.

Governing equations

Here $[M]$ is the (possibly lumped) mass matrix, $[C]$ a damping matrix, $[K]$ a stiffness matrix, and $\{F(t)\}$ a time-dependent load vector. Initial conditions $\{U(0)\}$ and $\{\dot{U}(0)\}$ must be supplied.

Time integration

The solver advances the solution in time using a time-stepping method such as the Newmark family or other single-step schemes. The discrete update typically involves forming an effective stiffness matrix and load vector at each time step, then solving a linear system for the new displacements.

Input DSL structure

Input data include:

- system matrices or references to solvers that assemble them,
- time integration parameters (time step, total time, method settings),
- time histories of external forces,
- initial conditions for displacements and velocities.

Output fields

Outputs consist of:

- displacement and velocity histories,
- optionally, acceleration and internal force histories,
- derived response measures (e.g. peak values, response spectra) where implemented.

8.12 dynamics_mck

Documentation for solver `dynamics_mck`

Overview

The `dynamics_mck` solver integrates second-order dynamical systems of the form

$$[M]\{\ddot{U}\} + [C]\{\dot{U}\} + [K]\{U\} = \{F(t)\},$$

which arise from finite element discretisation of structural dynamics, vibration, and related problems.

Governing equations

Here $[M]$ is the (possibly lumped) mass matrix, $[C]$ a damping matrix, $[K]$ a stiffness matrix, and $\{F(t)\}$ a time-dependent load vector. Initial conditions $\{U(0)\}$ and $\{\dot{U}(0)\}$ must be supplied.

Time integration

The solver advances the solution in time using a time-stepping method such as the Newmark family or other single-step schemes. The discrete update typically involves forming an effective stiffness matrix and load vector at each time step, then solving a linear system for the new displacements.

Input DSL structure

Input data include:

- system matrices or references to solvers that assemble them,
- time integration parameters (time step, total time, method settings),
- time histories of external forces,
- initial conditions for displacements and velocities.

Output fields

Outputs consist of:

- displacement and velocity histories,
- optionally, acceleration and internal force histories,
- derived response measures (e.g. peak values, response spectra) where implemented.

8.13 eigen_generalized

Documentation for solver `eigen_generalized`

Overview

The `eigen_generalized` solver is devoted to solving generalised eigenvalue problems that arise in structural dynamics, stability, and other fields. It typically operates on matrices assembled by other solvers or pre-processing routines.

Governing equations

A generic generalised eigenproblem has the form

$$[K]\{\phi\} = \lambda[M]\{\phi\},$$

where:

- $[K]$ is a stiffness or system matrix,
- $[M]$ is a mass or weighting matrix,
- λ are eigenvalues, and $\{\phi\}$ the corresponding eigenvectors.

In structural dynamics, $\lambda = \omega^2$ yields natural circular frequencies ω, whereas in buckling analysis λ may represent critical load factors.

Numerical formulation

The solver interfaces with numerical eigensolvers available in the Python ecosystem (e.g. dense or sparse algorithms). Pre- and post-scaling, shift strategies, and selection of eigenvalue ranges are handled at this level or in higher-level driver code.

Input DSL structure

Inputs typically reference:

- matrices $[K]$ and $[M]$ assembled elsewhere,
- the number of modes to be computed,
- any shift or spectral transformation options.

Output fields

Outputs include:

- eigenvalues λ_i,
- eigenvectors/mode shapes $\{\phi_i\}$,
- optionally, normalised modes and effective modal masses.

Typical use cases

- modal analysis in structural dynamics,
- stability/buckling problems,
- reduced-order modelling and spectral decompositions.

8.14 elas2d_plane_strain

Documentation for solver `elas2d_plane_strain`

Overview

The `elas2d_plane_strain` solver belongs to the continuum solid mechanics group. It implements a finite element formulation for plane strain elasticity, where out-of-plane strain components vanish and the problem is effectively two-dimensional. The primary unknown field is the displacement vector $\mathbf{u}$.

Governing equations

In the absence of inertia and damping the quasi-static equilibrium equations are

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \quad \text{in } \Omega,$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{b}$ denotes body forces per unit volume.

For linear elastic materials the constitutive relation is

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon},$$

with $\mathbb{C}$ the fourth-order elasticity tensor and $\boldsymbol{\varepsilon}$ the infinitesimal strain tensor

$$\boldsymbol{\varepsilon} = \tfrac{1}{2}\big(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathsf{T}}\big).$$

The particular kinematic assumptions (plane stress, plane strain, axisymmetry) are imposed at the element level through the strain–displacement matrices and the form of $\mathbb{C}$.

Boundary conditions

Boundary conditions are specified as:

- prescribed displacements $\mathbf{u} = \bar{\mathbf{u}}$ on Γ_u (Dirichlet),
- prescribed tractions $\boldsymbol{\sigma}\mathbf{n} = \bar{\mathbf{t}}$ on Γ_t (Neumann),

with $\Gamma_u \cup \Gamma_t = \partial\Omega$ and $\Gamma_u \cap \Gamma_t = \emptyset$.

Finite element formulation

The displacement field is interpolated using standard isoparametric shape functions,

$$\mathbf{u}_h(x) = \sum_{i} N_i(x)\,\mathbf{u}_i.$$

The weak form of the equilibrium equations leads to the standard finite element system

$$[K]\{\mathbf{U}\} = \{\mathbf{F}\},$$

with

$$[K]_e = \int_{\Omega_e} [B]^{\mathsf{T}} \mathbb{C} [B]\, d\Omega, \qquad \{\mathbf{F}\}_e = \int_{\Omega_e} N^{\mathsf{T}} \mathbf{b}\, d\Omega + \int_{\Gamma_{t,e}} N^{\mathsf{T}} \bar{\mathbf{t}}\, d\Gamma,$$

where $[B]$ is the strain–displacement matrix for the chosen element type.

Element stiffness matrices and load vectors are assembled into the global system and displacement boundary conditions are enforced by modifying the global matrix and right-hand side.

Input DSL structure

In the project DSL, use of `elas2d_plane_strain` involves specifying:

- material properties (e.g. Young's modulus, Poisson ratio),
- mesh topology (nodes and elements),
- boundary condition sets for prescribed displacements and tractions,
- body forces or equivalent nodal loads where needed.

The exact syntax is consistent with other structural solvers and allows combining this solver with post-processing, load stepping, and study drivers.

Output fields

Typical outputs include:

- nodal displacements,
- element or integration-point stresses and strains,
- reaction forces at constrained degrees of freedom,
- derived scalar measures (e.g. von Mises stress).

Typical use cases

- analysis of plane and axisymmetric components under mechanical loading,
- 3D solid analysis of parts and assemblies in structural mechanics,
- benchmark problems for code verification in elasticity.

Limitations and notes

- The formulation assumes small strains and rotations unless extended variants are used.
- Material behaviour is linear elastic in the base implementation; any nonlinear or inelastic behaviour belongs to specialised solvers.
- Mesh quality and appropriate boundary condition specification are critical for accurate results.

8.15 elas2d_plane_stress_thermal

Documentation for solver `elas2d_plane_stress_thermal`

Overview

The `elas2d_plane_stress_thermal` solver belongs to the continuum solid mechanics group. It implements a finite element formulation for plane stress elasticity, appropriate for thin plates where out-of-plane stresses are negligible. The primary unknown field is the displacement vector $\mathbf{u}$.

Governing equations

In the absence of inertia and damping the quasi-static equilibrium equations are

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \quad \text{in } \Omega,$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{b}$ denotes body forces per unit volume.

For linear elastic materials the constitutive relation is

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon},$$

with $\mathbb{C}$ the fourth-order elasticity tensor and $\boldsymbol{\varepsilon}$ the infinitesimal strain tensor

$$\boldsymbol{\varepsilon} = \tfrac{1}{2}\big(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathsf{T}}\big).$$

The particular kinematic assumptions (plane stress, plane strain, axisymmetry) are imposed at the element level through the strain–displacement matrices and the form of $\mathbb{C}$.

Boundary conditions

Boundary conditions are specified as:

- prescribed displacements $\mathbf{u} = \bar{\mathbf{u}}$ on Γ_u (Dirichlet),
- prescribed tractions $\boldsymbol{\sigma}\mathbf{n} = \bar{\mathbf{t}}$ on Γ_t (Neumann),

with $\Gamma_u \cup \Gamma_t = \partial\Omega$ and $\Gamma_u \cap \Gamma_t = \emptyset$.

Finite element formulation

The displacement field is interpolated using standard isoparametric shape functions,

$$\mathbf{u}_h(x) = \sum_i N_i(x)\mathbf{u}_i.$$

The weak form of the equilibrium equations leads to the standard finite element system

$$[K]\{\mathbf{U}\} = \{\mathbf{F}\},$$

with

$$[K]_e = \int_{\Omega_e} [B]^{\mathsf{T}} \mathbb{C} [B] \, d\Omega, \qquad \{\mathbf{F}\}_e = \int_{\Omega_e} N^{\mathsf{T}} \mathbf{b} \, d\Omega + \int_{\Gamma_{t,e}} N^{\mathsf{T}} \bar{\mathbf{t}} \, d\Gamma,$$

where $[B]$ is the strain–displacement matrix for the chosen element type.

Element stiffness matrices and load vectors are assembled into the global system and displacement boundary conditions are enforced by modifying the global matrix and right-hand side.

Input DSL structure

In the project DSL, use of `elas2d_plane_stress_thermal` involves specifying:

- material properties (e.g. Young's modulus, Poisson ratio),
- mesh topology (nodes and elements),
- boundary condition sets for prescribed displacements and tractions,
- body forces or equivalent nodal loads where needed.

The exact syntax is consistent with other structural solvers and allows combining this solver with post-processing, load stepping, and study drivers.

Output fields

Typical outputs include:

- nodal displacements,
- element or integration-point stresses and strains,
- reaction forces at constrained degrees of freedom,
- derived scalar measures (e.g. von Mises stress).

Typical use cases

- analysis of plane and axisymmetric components under mechanical loading,
- 3D solid analysis of parts and assemblies in structural mechanics,
- benchmark problems for code verification in elasticity.

Limitations and notes

- The formulation assumes small strains and rotations unless extended variants are used.
- Material behaviour is linear elastic in the base implementation; any nonlinear or inelastic behaviour belongs to specialised solvers.
- Mesh quality and appropriate boundary condition specification are critical for accurate results.

8.16 elas2d_planestrain

Documentation for solver `elas2d_planestrain`

Overview

The `elas2d_planestrain` solver belongs to the continuum solid mechanics group. It implements a finite element formulation for plane strain elasticity, where out-of-plane strain components vanish and the problem is effectively two-dimensional. The primary unknown field is the displacement vector $\mathbf{u}$.

Governing equations

In the absence of inertia and damping the quasi-static equilibrium equations are

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \quad \text{in } \Omega,$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{b}$ denotes body forces per unit volume.

For linear elastic materials the constitutive relation is

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon},$$

with $\mathbb{C}$ the fourth-order elasticity tensor and $\boldsymbol{\varepsilon}$ the infinitesimal strain tensor

$$\boldsymbol{\varepsilon} = \tfrac{1}{2}\big(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathsf{T}}\big).$$

The particular kinematic assumptions (plane stress, plane strain, axisymmetry) are imposed at the element level through the strain–displacement matrices and the form of $\mathbb{C}$.

Boundary conditions

Boundary conditions are specified as:

- prescribed displacements $\mathbf{u} = \bar{\mathbf{u}}$ on Γ_u (Dirichlet),
- prescribed tractions $\boldsymbol{\sigma}\mathbf{n} = \bar{\mathbf{t}}$ on Γ_t (Neumann),

with $\Gamma_u \cup \Gamma_t = \partial\Omega$ and $\Gamma_u \cap \Gamma_t = \emptyset$.

Finite element formulation

The displacement field is interpolated using standard isoparametric shape functions,

$$\mathbf{u}_h(x) = \sum_i N_i(x)\mathbf{u}_i.$$

The weak form of the equilibrium equations leads to the standard finite element system

$$[K]\{\mathbf{U}\} = \{\mathbf{F}\},$$

with

$$[K]_e = \int_{\Omega_e} [B]^{\mathsf{T}} \mathbb{C} [B] \, d\Omega, \qquad \{\mathbf{F}\}_e = \int_{\Omega_e} N^{\mathsf{T}} \mathbf{b} \, d\Omega + \int_{\Gamma_{t,e}} N^{\mathsf{T}} \bar{\mathbf{t}} \, d\Gamma,$$

where $[B]$ is the strain–displacement matrix for the chosen element type.

Element stiffness matrices and load vectors are assembled into the global system and displacement boundary conditions are enforced by modifying the global matrix and right-hand side.

Input DSL structure

In the project DSL, use of `elas2d_planestrain` involves specifying:

- material properties (e.g. Young's modulus, Poisson ratio),
- mesh topology (nodes and elements),
- boundary condition sets for prescribed displacements and tractions,
- body forces or equivalent nodal loads where needed.

The exact syntax is consistent with other structural solvers and allows combining this solver with post-processing, load stepping, and study drivers.

Output fields

Typical outputs include:

- nodal displacements,
- element or integration-point stresses and strains,
- reaction forces at constrained degrees of freedom,
- derived scalar measures (e.g. von Mises stress).

Typical use cases

- analysis of plane and axisymmetric components under mechanical loading,
- 3D solid analysis of parts and assemblies in structural mechanics,
- benchmark problems for code verification in elasticity.

Limitations and notes

- The formulation assumes small strains and rotations unless extended variants are used.
- Material behaviour is linear elastic in the base implementation; any nonlinear or inelastic behaviour belongs to specialised solvers.
- Mesh quality and appropriate boundary condition specification are critical for accurate results.

8.17 elas2d_planestress

Documentation for solver `elas2d_planestress`

Overview

The `elas2d_planestress` solver belongs to the continuum solid mechanics group. It implements a finite element formulation for plane stress elasticity, appropriate for thin plates where out-of-plane stresses are negligible. The primary unknown field is the displacement vector $\mathbf{u}$.

Governing equations

In the absence of inertia and damping the quasi-static equilibrium equations are

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \quad \text{in } \Omega,$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{b}$ denotes body forces per unit volume.

For linear elastic materials the constitutive relation is

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon},$$

with $\mathbb{C}$ the fourth-order elasticity tensor and $\boldsymbol{\varepsilon}$ the infinitesimal strain tensor

$$\boldsymbol{\varepsilon} = \tfrac{1}{2}\big(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathsf{T}}\big).$$

The particular kinematic assumptions (plane stress, plane strain, axisymmetry) are imposed at the element level through the strain–displacement matrices and the form of $\mathbb{C}$.

Boundary conditions

Boundary conditions are specified as:

- prescribed displacements $\mathbf{u} = \bar{\mathbf{u}}$ on Γ_u (Dirichlet),
- prescribed tractions $\boldsymbol{\sigma}\mathbf{n} = \bar{\mathbf{t}}$ on Γ_t (Neumann),

with $\Gamma_u \cup \Gamma_t = \partial\Omega$ and $\Gamma_u \cap \Gamma_t = \emptyset$.

Finite element formulation

The displacement field is interpolated using standard isoparametric shape functions,

$$\mathbf{u}_h(x) = \sum_i N_i(x)\mathbf{u}_i.$$

The weak form of the equilibrium equations leads to the standard finite element system

$$[K]\{\mathbf{U}\} = \{\mathbf{F}\},$$

with

$$[K]_e = \int_{\Omega_e} [B]^{\mathsf{T}} \mathbb{C} [B] \, d\Omega, \qquad \{\mathbf{F}\}_e = \int_{\Omega_e} N^{\mathsf{T}} \mathbf{b} \, d\Omega + \int_{\Gamma_{t,e}} N^{\mathsf{T}} \bar{\mathbf{t}} \, d\Gamma,$$

where $[B]$ is the strain–displacement matrix for the chosen element type.

Element stiffness matrices and load vectors are assembled into the global system and displacement boundary conditions are enforced by modifying the global matrix and right-hand side.

Input DSL structure

In the project DSL, use of `elas2d_planestress` involves specifying:

- material properties (e.g. Young's modulus, Poisson ratio),
- mesh topology (nodes and elements),
- boundary condition sets for prescribed displacements and tractions,
- body forces or equivalent nodal loads where needed.

The exact syntax is consistent with other structural solvers and allows combining this solver with post-processing, load stepping, and study drivers.

Output fields

Typical outputs include:

- nodal displacements,
- element or integration-point stresses and strains,
- reaction forces at constrained degrees of freedom,
- derived scalar measures (e.g. von Mises stress).

Typical use cases

- analysis of plane and axisymmetric components under mechanical loading,
- 3D solid analysis of parts and assemblies in structural mechanics,
- benchmark problems for code verification in elasticity.

Limitations and notes

- The formulation assumes small strains and rotations unless extended variants are used.
- Material behaviour is linear elastic in the base implementation; any nonlinear or inelastic behaviour belongs to specialised solvers.
- Mesh quality and appropriate boundary condition specification are critical for accurate results.

8.18 elas2d_ps_edgeload

Documentation for solver `elas2d_ps_edgeload`

Overview

The `elas2d_ps_edgeload` solver belongs to the continuum solid mechanics group. It implements a finite element formulation for general three-dimensional small-strain linear elasticity using solid elements. The primary unknown field is the displacement vector $\mathbf{u}$.

Governing equations

In the absence of inertia and damping the quasi-static equilibrium equations are

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \quad \text{in } \Omega,$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{b}$ denotes body forces per unit volume.

For linear elastic materials the constitutive relation is

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon},$$

with $\mathbb{C}$ the fourth-order elasticity tensor and $\boldsymbol{\varepsilon}$ the infinitesimal strain tensor

$$\boldsymbol{\varepsilon} = \tfrac{1}{2}\big(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathsf{T}}\big).$$

The particular kinematic assumptions (plane stress, plane strain, axisymmetry) are imposed at the element level through the strain–displacement matrices and the form of $\mathbb{C}$.

Boundary conditions

Boundary conditions are specified as:

- prescribed displacements $\mathbf{u} = \bar{\mathbf{u}}$ on Γ_u (Dirichlet),
- prescribed tractions $\boldsymbol{\sigma}\mathbf{n} = \bar{\mathbf{t}}$ on Γ_t (Neumann),

with $\Gamma_u \cup \Gamma_t = \partial\Omega$ and $\Gamma_u \cap \Gamma_t = \emptyset$.

Finite element formulation

The displacement field is interpolated using standard isoparametric shape functions,

$$\mathbf{u}_h(x) = \sum_i N_i(x)\mathbf{u}_i.$$

The weak form of the equilibrium equations leads to the standard finite element system

$$[K]\{\mathbf{U}\} = \{\mathbf{F}\},$$

with

$$[K]_e = \int_{\Omega_e} [B]^{\mathsf{T}} \mathbb{C} [B] \, d\Omega, \qquad \{\mathbf{F}\}_e = \int_{\Omega_e} N^{\mathsf{T}} \mathbf{b} \, d\Omega + \int_{\Gamma_{t,e}} N^{\mathsf{T}} \bar{\mathbf{t}} \, d\Gamma,$$

where $[B]$ is the strain–displacement matrix for the chosen element type.

Element stiffness matrices and load vectors are assembled into the global system and displacement boundary conditions are enforced by modifying the global matrix and right-hand side.

Input DSL structure

In the project DSL, use of `elas2d_ps_edgeload` involves specifying:

- material properties (e.g. Young's modulus, Poisson ratio),
- mesh topology (nodes and elements),
- boundary condition sets for prescribed displacements and tractions,
- body forces or equivalent nodal loads where needed.

The exact syntax is consistent with other structural solvers and allows combining this solver with post-processing, load stepping, and study drivers.

Output fields

Typical outputs include:

- nodal displacements,
- element or integration-point stresses and strains,
- reaction forces at constrained degrees of freedom,
- derived scalar measures (e.g. von Mises stress).

Typical use cases

- analysis of plane and axisymmetric components under mechanical loading,
- 3D solid analysis of parts and assemblies in structural mechanics,
- benchmark problems for code verification in elasticity.

Limitations and notes

- The formulation assumes small strains and rotations unless extended variants are used.
- Material behaviour is linear elastic in the base implementation; any nonlinear or inelastic behaviour belongs to specialised solvers.
- Mesh quality and appropriate boundary condition specification are critical for accurate results.

8.19 elastic3d

Documentation for solver `elastic3d`

Overview

The `elastic3d` solver belongs to the continuum solid mechanics group. It implements a finite element formulation for general three-dimensional small-strain linear elasticity using solid elements. The primary unknown field is the displacement vector $\mathbf{u}$.

Governing equations

In the absence of inertia and damping the quasi-static equilibrium equations are

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \quad \text{in } \Omega,$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{b}$ denotes body forces per unit volume.

For linear elastic materials the constitutive relation is

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon},$$

with $\mathbb{C}$ the fourth-order elasticity tensor and $\boldsymbol{\varepsilon}$ the infinitesimal strain tensor

$$\boldsymbol{\varepsilon} = \tfrac{1}{2}\big(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathsf{T}}\big).$$

The particular kinematic assumptions (plane stress, plane strain, axisymmetry) are imposed at the element level through the strain–displacement matrices and the form of $\mathbb{C}$.

Boundary conditions

Boundary conditions are specified as:

- prescribed displacements $\mathbf{u} = \bar{\mathbf{u}}$ on Γ_u (Dirichlet),
- prescribed tractions $\boldsymbol{\sigma}\mathbf{n} = \bar{\mathbf{t}}$ on Γ_t (Neumann),

with $\Gamma_u \cup \Gamma_t = \partial\Omega$ and $\Gamma_u \cap \Gamma_t = \emptyset$.

Finite element formulation

The displacement field is interpolated using standard isoparametric shape functions,

$$\mathbf{u}_h(x) = \sum_{i} N_i(x)\,\mathbf{u}_i.$$

The weak form of the equilibrium equations leads to the standard finite element system

$$[K]\{\mathbf{U}\} = \{\mathbf{F}\},$$

with

$$[K]_e = \int_{\Omega_e} [B]^{\mathsf{T}} \mathbb{C} [B]\, d\Omega, \qquad \{\mathbf{F}\}_e = \int_{\Omega_e} N^{\mathsf{T}} \mathbf{b}\, d\Omega + \int_{\Gamma_{t,e}} N^{\mathsf{T}} \bar{\mathbf{t}}\, d\Gamma,$$

where $[B]$ is the strain–displacement matrix for the chosen element type.

Element stiffness matrices and load vectors are assembled into the global system and displacement boundary conditions are enforced by modifying the global matrix and right-hand side.

Input DSL structure

In the project DSL, use of `elastic3d` involves specifying:

- material properties (e.g. Young's modulus, Poisson ratio),
- mesh topology (nodes and elements),
- boundary condition sets for prescribed displacements and tractions,
- body forces or equivalent nodal loads where needed.

The exact syntax is consistent with other structural solvers and allows combining this solver with post-processing, load stepping, and study drivers.

Output fields

Typical outputs include:

- nodal displacements,
- element or integration-point stresses and strains,
- reaction forces at constrained degrees of freedom,
- derived scalar measures (e.g. von Mises stress).

Typical use cases

- analysis of plane and axisymmetric components under mechanical loading,
- 3D solid analysis of parts and assemblies in structural mechanics,
- benchmark problems for code verification in elasticity.

Limitations and notes

- The formulation assumes small strains and rotations unless extended variants are used.
- Material behaviour is linear elastic in the base implementation; any nonlinear or inelastic behaviour belongs to specialised solvers.
- Mesh quality and appropriate boundary condition specification are critical for accurate results.

8.20 frame2d

Documentation for solver `frame2d`

Overview

The `frame2d` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

In the purely static case the assembled finite element equations reduce to

$$[K]\{U\} = \{F\},$$

with $[K]$ containing both axial and bending stiffness contributions from frame or beam elements.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `frame2d` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.21 frame2d_beamcolumn

Documentation for solver `frame2d_beamcolumn`

Overview

The `frame2d_beamcolumn` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

In the purely static case the assembled finite element equations reduce to

$$[K]\{U\} = \{F\},$$

with $[K]$ containing both axial and bending stiffness contributions from frame or beam elements.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `frame2d_beamcolumn` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.22 frame2d_buckling

Documentation for solver `frame2d_buckling`

Overview

The `frame2d_buckling` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

For buckling analysis, the solver forms both the elastic stiffness matrix $[K]$ and the geometric stiffness matrix $[K_G]$ associated with a pre-stress state. A typical linearised buckling formulation solves

$$\big([K] + \lambda[K_G]\big)\{\phi\} = \mathbf{0},$$

where λ are critical load multipliers and $\{\phi\}$ the associated buckling mode shapes.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `frame2d_buckling` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.23 frame2d_modal

Documentation for solver `frame2d_modal`

Overview

The `frame2d_modal` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

For free-vibration (modal) analysis, the discretised equations of motion

$$[M]\{\ddot{U}\} + [C]\{\dot{U}\} + [K]\{U\} = \{F\}$$

are recast as a homogeneous eigenproblem

$$[K]\{\phi\} = \lambda[M]\{\phi\},$$

to compute natural frequencies and mode shapes.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `frame2d_modal` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.24 frame2d_modal_consistent

Documentation for solver `frame2d_modal_consistent`

Overview

The `frame2d_modal_consistent` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

For free-vibration (modal) analysis, the discretised equations of motion

$$[M]\{\ddot{U}\} + [C]\{\dot{U}\} + [K]\{U\} = \{F\}$$

are recast as a homogeneous eigenproblem

$$[K]\{\phi\} = \lambda[M]\{\phi\},$$

to compute natural frequencies and mode shapes.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `frame2d_modal_consistent` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.25 frame2d_pdelta

Documentation for solver `frame2d_pdelta`

Overview

The `frame2d_pdelta` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

For buckling analysis, the solver forms both the elastic stiffness matrix $[K]$ and the geometric stiffness matrix $[K_G]$ associated with a pre-stress state. A typical linearised buckling formulation solves

$$\big([K] + \lambda[K_G]\big)\{\phi\} = \mathbf{0},$$

where λ are critical load multipliers and $\{\phi\}$ the associated buckling mode shapes.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `frame2d_pdelta` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.26 frame2d_static

Documentation for solver `frame2d_static`

Overview

The `frame2d_static` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

In the purely static case the assembled finite element equations reduce to

$$[K]\{U\} = \{F\},$$

with $[K]$ containing both axial and bending stiffness contributions from frame or beam elements.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `frame2d_static` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.27 frame2d_static_nl

Documentation for solver `frame2d_static_nl`

Overview

The `frame2d_static_nl` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

In the purely static case the assembled finite element equations reduce to

$$[K]\{U\} = \{F\},$$

with $[K]$ containing both axial and bending stiffness contributions from frame or beam elements.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `frame2d_static_nl` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.28 frame2d_transient_newmark

Documentation for solver `frame2d_transient_newmark`

Overview

The `frame2d_transient_newmark` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

For transient dynamic analysis, the solver integrates the second-order system

$$[M]\{\ddot{U}\} + [C]\{\dot{U}\} + [K]\{U\} = \{F(t)\}$$

in time using a time-stepping scheme (for example, a Newmark-type method), given initial conditions on displacements and velocities.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `frame2d_transient_newmark` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.29 frame3d_beamcolumn

Documentation for solver `frame3d_beamcolumn`

Overview

The `frame3d_beamcolumn` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

In the purely static case the assembled finite element equations reduce to

$$[K]\{U\} = \{F\},$$

with $[K]$ containing both axial and bending stiffness contributions from frame or beam elements.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `frame3d_beamcolumn` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.30 fsi2d_coupled

Documentation for solver `fsi2d_coupled`

Overview

The `fsi2d_coupled` solver addresses fluid–structure interaction (FSI) problems, in which a deformable solid and a surrounding or internal fluid domain influence each other's motion. The solver couples a structural finite element model with a fluid model through interface conditions.

Governing equations

On the structural side, equilibrium of forces and appropriate constitutive relations govern the displacement field. On the fluid side, the flow may be described by incompressible or compressible Navier–Stokes equations, or by simplified models, depending on the implementation.

At the interface, kinematic and dynamic conditions enforce:

- continuity of velocities (no-slip or appropriate slip conditions),
- balance of tractions (fluid stresses equal and opposite to structural tractions).

Finite element / finite volume formulation

The structural domain is discretised by finite elements, while the fluid domain may be represented by finite elements, finite volumes, or a reduced model. The coupling is typically realised by exchanging interface tractions and displacements/velocities between the two solvers at each time step or load increment.

Boundary conditions and loading

Boundary conditions include those appropriate to both structural and fluid models, together with coupling conditions on the FSI interface. External loads may be applied via:

- imposed fluid inflow/outflow conditions,
- structural supports and external forces,
- time-dependent driving conditions.

Input DSL structure

The DSL description for `fsi2d_coupled` typically references:

- structural mesh, materials, and boundary conditions,
- fluid model parameters and boundary conditions,
- interface definitions linking structural and fluid boundaries,
- time integration and coupling algorithm settings.

Output fields

Outputs may include:

- structural displacements, stresses, and reaction forces,
- fluid pressures, velocities, and derived quantities,
- interface traction and displacement histories.

Typical use cases

- aeroelastic or hydroelastic response of flexible structures,
- interaction of internal flows with deformable walls,
- verification of simplified coupled models.

Limitations and notes

- The exact fluid formulation and coupling algorithm are solver-specific and should be checked in the implementation.
- Coupled problems can be sensitive to time step size and relaxation parameters.
- Mesh and interface quality are important for numerical stability.

8.31 heat1d

Documentation for solver `heat1d`

Overview

The `heat1d` solver performs one-dimensional steady-state heat conduction analysis with time-independent loading and boundary conditions.

Governing equations

The steady-state heat conduction problem is governed by

$$-\nabla \cdot (k \nabla T) = q \quad \text{in } \Omega,$$

where T is the temperature field, k the thermal conductivity, and q a volumetric heat source term. Time derivatives are absent in the steady case.

Boundary conditions

Boundary conditions supported by the various heat conduction solvers include:

- prescribed temperature (Dirichlet) on parts of the boundary,
- prescribed normal heat flux (Neumann),
- convection (Robin) of the form

$$-k \nabla T \cdot \mathbf{n} = h(T - T_\infty),$$

  where h is a convection coefficient and T_∞ an ambient reference temperature.

Finite element formulation

The temperature field is approximated by standard finite element shape functions N_i,

$$T_h(x) = \sum_i N_i(x)\, T_i,$$

and the Galerkin method is used to obtain a system of algebraic equations. For each element the conductivity matrix and (where applicable) capacity matrix are assembled as

$$[K]_e = \int_{\Omega_e} (\nabla N)^{\mathsf{T}} k \, \nabla N \, d\Omega, \qquad [C]_e = \int_{\Omega_e} \rho c \, N^{\mathsf{T}} N \, d\Omega,$$

with additional edge integrals for Neumann and Robin boundary conditions.

Assembly over all elements produces the global system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\},$$

which reduces to $[K]\{T\} = \{F\}$ in the steady-state case.

Input DSL structure

Analyses using `heat1d` are configured in the same DSL style as other solvers in the codebase. Users specify material properties (k, and for transient problems ρ, c), mesh data, volumetric sources q, and boundary condition groups (Dirichlet, Neumann, and where supported, convection data h, T_∞).

A schematic DSL fragment looks like:

```
solver heat1d {
  material steel {
    rho = 7850
    c   = 460
    k   = 45
  }

  region plate {
    mesh     = "meshes/plate_quad.msh"
    material = steel
  }

  boundary_conditions {
    dirichlet { ... }
    neumann   { ... }
    convection{ ... }    # if supported
  }

  loads {
    volumetric_heat { q = 1.0e6 }
  }

  # time_integration block for transient variants
}
```

Output fields

Typical outputs include:

- nodal temperatures,
- temperature distributions at selected times (transient),
- optionally, derived heat flux fields $\mathbf{q} = -k\nabla T$.

Typical use cases

- steady and transient temperature analysis of one-, two-, or three-dimensional solids,
- verification and benchmark problems for coupled thermo-mechanical simulations,
- parametric studies of material or boundary condition effects on thermal response.

Limitations and notes

- Material properties are typically treated as temperature-independent; strong temperature dependence requires explicit user handling.
- Only conduction and simple convection are modelled; radiation is not included unless added separately.

- Accuracy depends on mesh resolution and, for transient problems, on the chosen time-step size.

8.32 heat1d_linear_transient

Documentation for solver `heat1d_linear_transient`

Overview

The `heat1d_linear_transient` solver performs one-dimensional transient heat conduction analysis. The unknown field is the temperature as a function of space and time.

Governing equations

The governing equation for transient heat conduction is

$$\rho c \frac{\partial T}{\partial t} - \nabla \cdot (k \nabla T) = q \quad \text{in } \Omega,$$

where ρ is the density, c the specific heat capacity, k the thermal conductivity (possibly tensor-valued), and q a volumetric heat source term.

Boundary conditions

Boundary conditions supported by the various heat conduction solvers include:

- prescribed temperature (Dirichlet) on parts of the boundary,
- prescribed normal heat flux (Neumann),
- convection (Robin) of the form

  $$-k \nabla T \cdot \mathbf{n} = h (T - T_\infty),$$

  where h is a convection coefficient and T_∞ an ambient reference temperature.

Finite element formulation

The temperature field is approximated by standard finite element shape functions N_i,

$$T_h(x) = \sum_i N_i(x)\, T_i,$$

and the Galerkin method is used to obtain a system of algebraic equations. For each element the conductivity matrix and (where applicable) capacity matrix are assembled as

$$[K]_e = \int_{\Omega_e} (\nabla N)^{\mathsf{T}} k \, \nabla N \, d\Omega, \qquad [C]_e = \int_{\Omega_e} \rho c \, N^{\mathsf{T}} N \, d\Omega,$$

with additional edge integrals for Neumann and Robin boundary conditions.

Assembly over all elements produces the global system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\},$$

which reduces to $[K]\{T\} = \{F\}$ in the steady-state case.

Time discretisation

Using a standard single-step implicit time integration scheme, the semi-discrete system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\}$$

is advanced from time level n to $n+1$ according to

$$\left(\frac{1}{\Delta t}[C] + \theta[K]\right)\{T\}^{n+1} = \left(\frac{1}{\Delta t}[C] - (1-\theta)[K]\right)\{T\}^{n} + \theta\{F\}^{n+1} + (1-\theta)\{F\}^{n},$$

with $0.5 \leq \theta \leq 1$ (Crank–Nicolson to backward Euler).

Input DSL structure

Analyses using `heat1d_linear_transient` are configured in the same DSL style as other solvers in the codebase. Users specify material properties (k, and for transient problems ρ, c), mesh data, volumetric sources q, and boundary condition groups (Dirichlet, Neumann, and where supported, convection data h, T_∞).

A schematic DSL fragment looks like:

```
solver heat1d_linear_transient {
  material steel {
    rho = 7850
    c   = 460
    k   = 45
  }

  region plate {
    mesh     = "meshes/plate_quad.msh"
    material = steel
  }

  boundary_conditions {
    dirichlet { ... }
    neumann   { ... }
    convection{ ... }    # if supported
  }

  loads {
    volumetric_heat { q = 1.0e6 }
  }

  # time_integration block for transient variants
}
```

Output fields

Typical outputs include:

- nodal temperatures,

- temperature distributions at selected times (transient),
- optionally, derived heat flux fields $\mathbf{q} = -k\nabla T$.

Typical use cases

- steady and transient temperature analysis of one-, two-, or three-dimensional solids,
- verification and benchmark problems for coupled thermo-mechanical simulations,
- parametric studies of material or boundary condition effects on thermal response.

Limitations and notes

- Material properties are typically treated as temperature-independent; strong temperature dependence requires explicit user handling.
- Only conduction and simple convection are modelled; radiation is not included unless added separately.
- Accuracy depends on mesh resolution and, for transient problems, on the chosen time-step size.

8.33 heat1d_transient

Documentation for solver `heat1d_transient`

Overview

The `heat1d_transient` solver performs one-dimensional transient heat conduction analysis. The unknown field is the temperature as a function of space and time.

Governing equations

The governing equation for transient heat conduction is

$$\rho c \frac{\partial T}{\partial t} - \nabla \cdot (k \nabla T) = q \quad \text{in } \Omega,$$

where ρ is the density, c the specific heat capacity, k the thermal conductivity (possibly tensor-valued), and q a volumetric heat source term.

Boundary conditions

Boundary conditions supported by the various heat conduction solvers include:

- prescribed temperature (Dirichlet) on parts of the boundary,
- prescribed normal heat flux (Neumann),
- convection (Robin) of the form

  $$-k\nabla T \cdot \mathbf{n} = h(T - T_\infty),$$

  where h is a convection coefficient and T_∞ an ambient reference temperature.

Finite element formulation

The temperature field is approximated by standard finite element shape functions N_i,

$$T_h(x) = \sum_{i} N_i(x)\, T_i,$$

and the Galerkin method is used to obtain a system of algebraic equations. For each element the conductivity matrix and (where applicable) capacity matrix are assembled as

$$[K]_e = \int_{\Omega_e} (\nabla N)^{\mathsf{T}} k\, \nabla N \, d\Omega, \qquad [C]_e = \int_{\Omega_e} \rho c\, N^{\mathsf{T}} N \, d\Omega,$$

with additional edge integrals for Neumann and Robin boundary conditions.

Assembly over all elements produces the global system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\},$$

which reduces to $[K]\{T\} = \{F\}$ in the steady-state case.

Time discretisation

Using a standard single-step implicit time integration scheme, the semi-discrete system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\}$$

is advanced from time level n to $n+1$ according to

$$\left(\frac{1}{\Delta t}[C] + \theta[K]\right)\{T\}^{n+1} = \left(\frac{1}{\Delta t}[C] - (1-\theta)[K]\right)\{T\}^{n} + \theta\{F\}^{n+1} + (1-\theta)\{F\}^{n},$$

with $0.5 \le \theta \le 1$ (Crank–Nicolson to backward Euler).

Input DSL structure

Analyses using `heat1d_transient` are configured in the same DSL style as other solvers in the codebase. Users specify material properties (k, and for transient problems ρ, c), mesh data, volumetric sources q, and boundary condition groups (Dirichlet, Neumann, and where supported, convection data h, T_∞).

A schematic DSL fragment looks like:

```
solver heat1d_transient {
  material steel {
    rho = 7850
    c   = 460
    k   = 45
  }

  region plate {
    mesh     = "meshes/plate_quad.msh"
    material = steel
  }
```

```
  boundary_conditions {
    dirichlet { ... }
    neumann   { ... }
    convection{ ... }    # if supported
  }

  loads {
    volumetric_heat { q = 1.0e6 }
  }

  # time_integration block for transient variants
}
```

Output fields

Typical outputs include:

- nodal temperatures,
- temperature distributions at selected times (transient),
- optionally, derived heat flux fields $\mathbf{q} = -k\nabla T$.

Typical use cases

- steady and transient temperature analysis of one-, two-, or three-dimensional solids,
- verification and benchmark problems for coupled thermo-mechanical simulations,
- parametric studies of material or boundary condition effects on thermal response.

Limitations and notes

- Material properties are typically treated as temperature-independent; strong temperature dependence requires explicit user handling.
- Only conduction and simple convection are modelled; radiation is not included unless added separately.
- Accuracy depends on mesh resolution and, for transient problems, on the chosen time-step size.

8.34 heat2d

Documentation for solver `heat2d`

Overview

The `heat2d` solver performs two-dimensional steady-state heat conduction analysis with time-independent loading and boundary conditions.

Governing equations

The steady-state heat conduction problem is governed by

$$-\nabla \cdot (k \nabla T) = q \quad \text{in } \Omega,$$

where T is the temperature field, k the thermal conductivity, and q a volumetric heat source term. Time derivatives are absent in the steady case.

Boundary conditions

Boundary conditions supported by the various heat conduction solvers include:

- prescribed temperature (Dirichlet) on parts of the boundary,
- prescribed normal heat flux (Neumann),
- convection (Robin) of the form

$$-k \nabla T \cdot \mathbf{n} = h(T - T_\infty),$$

  where h is a convection coefficient and T_∞ an ambient reference temperature.

Finite element formulation

The temperature field is approximated by standard finite element shape functions N_i,

$$T_h(x) = \sum_i N_i(x)\, T_i,$$

and the Galerkin method is used to obtain a system of algebraic equations. For each element the conductivity matrix and (where applicable) capacity matrix are assembled as

$$[K]_e = \int_{\Omega_e} (\nabla N)^{\mathsf{T}} k \, \nabla N \, d\Omega, \qquad [C]_e = \int_{\Omega_e} \rho c \, N^{\mathsf{T}} N \, d\Omega,$$

with additional edge integrals for Neumann and Robin boundary conditions.

Assembly over all elements produces the global system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\},$$

which reduces to $[K]\{T\} = \{F\}$ in the steady-state case.

Input DSL structure

Analyses using `heat2d` are configured in the same DSL style as other solvers in the codebase. Users specify material properties (k, and for transient problems ρ, c), mesh data, volumetric sources q, and boundary condition groups (Dirichlet, Neumann, and where supported, convection data h, T_∞).

A schematic DSL fragment looks like:

```
solver heat2d {
  material steel {
    rho = 7850
    c   = 460
    k   = 45
  }

  region plate {
    mesh     = "meshes/plate_quad.msh"
    material = steel
  }

  boundary_conditions {
    dirichlet { ... }
    neumann   { ... }
    convection{ ... }    # if supported
  }

  loads {
    volumetric_heat { q = 1.0e6 }
  }

  # time_integration block for transient variants
}
```

Output fields

Typical outputs include:

- nodal temperatures,
- temperature distributions at selected times (transient),
- optionally, derived heat flux fields $\mathbf{q} = -k\nabla T$.

Typical use cases

- steady and transient temperature analysis of one-, two-, or three-dimensional solids,
- verification and benchmark problems for coupled thermo-mechanical simulations,
- parametric studies of material or boundary condition effects on thermal response.

Limitations and notes

- Material properties are typically treated as temperature-independent; strong temperature dependence requires explicit user handling.
- Only conduction and simple convection are modelled; radiation is not included unless added separately.

- Accuracy depends on mesh resolution and, for transient problems, on the chosen time-step size.

8.35 heat2d_planar_transient

Documentation for solver `heat2d_planar_transient`

Overview

The `heat2d_planar_transient` solver performs two-dimensional transient heat conduction analysis. The unknown field is the temperature as a function of space and time.

Governing equations

The governing equation for transient heat conduction is

$$\rho c \frac{\partial T}{\partial t} - \nabla \cdot (k \nabla T) = q \quad \text{in } \Omega,$$

where ρ is the density, c the specific heat capacity, k the thermal conductivity (possibly tensor-valued), and q a volumetric heat source term.

Boundary conditions

Boundary conditions supported by the various heat conduction solvers include:

- prescribed temperature (Dirichlet) on parts of the boundary,
- prescribed normal heat flux (Neumann),
- convection (Robin) of the form

  $$-k \nabla T \cdot \mathbf{n} = h (T - T_\infty),$$

  where h is a convection coefficient and T_∞ an ambient reference temperature.

Finite element formulation

The temperature field is approximated by standard finite element shape functions N_i,

$$T_h(x) = \sum_i N_i(x)\, T_i,$$

and the Galerkin method is used to obtain a system of algebraic equations. For each element the conductivity matrix and (where applicable) capacity matrix are assembled as

$$[K]_e = \int_{\Omega_e} (\nabla N)^{\mathsf{T}} k \, \nabla N \, d\Omega, \qquad [C]_e = \int_{\Omega_e} \rho c \, N^{\mathsf{T}} N \, d\Omega,$$

with additional edge integrals for Neumann and Robin boundary conditions.

Assembly over all elements produces the global system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\},$$

which reduces to $[K]\{T\} = \{F\}$ in the steady-state case.

Time discretisation

Using a standard single-step implicit time integration scheme, the semi-discrete system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\}$$

is advanced from time level n to $n+1$ according to

$$\left(\frac{1}{\Delta t}[C] + \theta[K]\right)\{T\}^{n+1} = \left(\frac{1}{\Delta t}[C] - (1-\theta)[K]\right)\{T\}^{n} + \theta\{F\}^{n+1} + (1-\theta)\{F\}^{n},$$

with $0.5 \leq \theta \leq 1$ (Crank–Nicolson to backward Euler).

Input DSL structure

Analyses using `heat2d_planar_transient` are configured in the same DSL style as other solvers in the codebase. Users specify material properties (k, and for transient problems ρ, c), mesh data, volumetric sources q, and boundary condition groups (Dirichlet, Neumann, and where supported, convection data h, T_∞).

A schematic DSL fragment looks like:

```
solver heat2d_planar_transient {
  material steel {
    rho = 7850
    c   = 460
    k   = 45
  }

  region plate {
    mesh     = "meshes/plate_quad.msh"
    material = steel
  }

  boundary_conditions {
    dirichlet { ... }
    neumann   { ... }
    convection{ ... }    # if supported
  }

  loads {
    volumetric_heat { q = 1.0e6 }
  }

  # time_integration block for transient variants
}
```

Output fields

Typical outputs include:

- nodal temperatures,

- temperature distributions at selected times (transient),
- optionally, derived heat flux fields $\mathbf{q} = -k\nabla T$.

Typical use cases

- steady and transient temperature analysis of one-, two-, or three-dimensional solids,
- verification and benchmark problems for coupled thermo-mechanical simulations,
- parametric studies of material or boundary condition effects on thermal response.

Limitations and notes

- Material properties are typically treated as temperature-independent; strong temperature dependence requires explicit user handling.
- Only conduction and simple convection are modelled; radiation is not included unless added separately.
- Accuracy depends on mesh resolution and, for transient problems, on the chosen time-step size.

8.36 heat2d_q4_static

Documentation for solver `heat2d_q4_static`

Overview

The `heat2d_q4_static` solver performs two-dimensional steady-state heat conduction analysis with time-independent loading and boundary conditions.

Governing equations

The steady-state heat conduction problem is governed by

$$-\nabla \cdot (k\nabla T) = q \quad \text{in } \Omega,$$

where T is the temperature field, k the thermal conductivity, and q a volumetric heat source term. Time derivatives are absent in the steady case.

Boundary conditions

Boundary conditions supported by the various heat conduction solvers include:

- prescribed temperature (Dirichlet) on parts of the boundary,
- prescribed normal heat flux (Neumann),
- convection (Robin) of the form

  $$-k\nabla T \cdot \mathbf{n} = h(T - T_\infty),$$

  where h is a convection coefficient and T_∞ an ambient reference temperature.

Finite element formulation

The temperature field is approximated by standard finite element shape functions N_i,

$$T_h(x) = \sum_i N_i(x)\, T_i,$$

and the Galerkin method is used to obtain a system of algebraic equations. For each element the conductivity matrix and (where applicable) capacity matrix are assembled as

$$[K]_e = \int_{\Omega_e} (\nabla N)^{\mathsf{T}} k\, \nabla N \, d\Omega, \qquad [C]_e = \int_{\Omega_e} \rho c\, N^{\mathsf{T}} N \, d\Omega,$$

with additional edge integrals for Neumann and Robin boundary conditions.

Assembly over all elements produces the global system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\},$$

which reduces to $[K]\{T\} = \{F\}$ in the steady-state case.

Input DSL structure

Analyses using `heat2d_q4_static` are configured in the same DSL style as other solvers in the codebase. Users specify material properties (k, and for transient problems ρ, c), mesh data, volumetric sources q, and boundary condition groups (Dirichlet, Neumann, and where supported, convection data h, T_∞).

A schematic DSL fragment looks like:

```
solver heat2d_q4_static {
  material steel {
    rho = 7850
    c   = 460
    k   = 45
  }

  region plate {
    mesh     = "meshes/plate_quad.msh"
    material = steel
  }

  boundary_conditions {
    dirichlet { ... }
    neumann   { ... }
    convection{ ... }    # if supported
  }

  loads {
    volumetric_heat { q = 1.0e6 }
  }

  # time_integration block for transient variants
}
```

Output fields

Typical outputs include:

- nodal temperatures,
- temperature distributions at selected times (transient),
- optionally, derived heat flux fields $\mathbf{q} = -k\nabla T$.

Typical use cases

- steady and transient temperature analysis of one-, two-, or three-dimensional solids,
- verification and benchmark problems for coupled thermo-mechanical simulations,
- parametric studies of material or boundary condition effects on thermal response.

Limitations and notes

- Material properties are typically treated as temperature-independent; strong temperature dependence requires explicit user handling.
- Only conduction and simple convection are modelled; radiation is not included unless added separately.
- Accuracy depends on mesh resolution and, for transient problems, on the chosen time-step size.

8.37 heat2d_steady

Documentation for solver `heat2d_steady`

Overview

The `heat2d_steady` solver performs two-dimensional steady-state heat conduction analysis with time-independent loading and boundary conditions.

Governing equations

The steady-state heat conduction problem is governed by

$$-\nabla \cdot (k\nabla T) = q \quad \text{in } \Omega,$$

where T is the temperature field, k the thermal conductivity, and q a volumetric heat source term. Time derivatives are absent in the steady case.

Boundary conditions

Boundary conditions supported by the various heat conduction solvers include:

- prescribed temperature (Dirichlet) on parts of the boundary,
- prescribed normal heat flux (Neumann),
- convection (Robin) of the form

$$-k\nabla T \cdot \mathbf{n} = h(T - T_\infty),$$

  where h is a convection coefficient and T_∞ an ambient reference temperature.

Finite element formulation

The temperature field is approximated by standard finite element shape functions N_i,

$$T_h(x) = \sum_i N_i(x)\, T_i,$$

and the Galerkin method is used to obtain a system of algebraic equations. For each element the conductivity matrix and (where applicable) capacity matrix are assembled as

$$[K]_e = \int_{\Omega_e} (\nabla N)^{\mathsf{T}} k\, \nabla N \, d\Omega, \qquad [C]_e = \int_{\Omega_e} \rho c\, N^{\mathsf{T}} N \, d\Omega,$$

with additional edge integrals for Neumann and Robin boundary conditions.

Assembly over all elements produces the global system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\},$$

which reduces to $[K]\{T\} = \{F\}$ in the steady-state case.

Input DSL structure

Analyses using `heat2d_steady` are configured in the same DSL style as other solvers in the codebase. Users specify material properties (k, and for transient problems ρ, c), mesh data, volumetric sources q, and boundary condition groups (Dirichlet, Neumann, and where supported, convection data h, T_∞).

A schematic DSL fragment looks like:

```
solver heat2d_steady {
  material steel {
    rho = 7850
    c   = 460
    k   = 45
  }

  region plate {
    mesh     = "meshes/plate_quad.msh"
    material = steel
```

```
  }

  boundary_conditions {
    dirichlet { ... }
    neumann   { ... }
    convection{ ... }    # if supported
  }

  loads {
    volumetric_heat { q = 1.0e6 }
  }

  # time_integration block for transient variants
}
```

Output fields

Typical outputs include:

- nodal temperatures,
- temperature distributions at selected times (transient),
- optionally, derived heat flux fields $\mathbf{q} = -k\nabla T$.

Typical use cases

- steady and transient temperature analysis of one-, two-, or three-dimensional solids,
- verification and benchmark problems for coupled thermo-mechanical simulations,
- parametric studies of material or boundary condition effects on thermal response.

Limitations and notes

- Material properties are typically treated as temperature-independent; strong temperature dependence requires explicit user handling.
- Only conduction and simple convection are modelled; radiation is not included unless added separately.
- Accuracy depends on mesh resolution and, for transient problems, on the chosen time-step size.

8.38 heat2d_steady_bc

Documentation for solver `heat2d_steady_bc`

Overview

The `heat2d_steady_bc` solver performs two-dimensional steady-state heat conduction analysis with time-independent loading and boundary conditions.

Governing equations

The steady-state heat conduction problem is governed by

$$-\nabla \cdot (k \nabla T) = q \quad \text{in } \Omega,$$

where T is the temperature field, k the thermal conductivity, and q a volumetric heat source term. Time derivatives are absent in the steady case.

Boundary conditions

Boundary conditions supported by the various heat conduction solvers include:

- prescribed temperature (Dirichlet) on parts of the boundary,
- prescribed normal heat flux (Neumann),
- convection (Robin) of the form

  $$-k \nabla T \cdot \mathbf{n} = h(T - T_\infty),$$

  where h is a convection coefficient and T_∞ an ambient reference temperature.

Finite element formulation

The temperature field is approximated by standard finite element shape functions N_i,

$$T_h(x) = \sum_i N_i(x)\, T_i,$$

and the Galerkin method is used to obtain a system of algebraic equations. For each element the conductivity matrix and (where applicable) capacity matrix are assembled as

$$[K]_e = \int_{\Omega_e} (\nabla N)^{\mathsf{T}} k \, \nabla N \, d\Omega, \qquad [C]_e = \int_{\Omega_e} \rho c \, N^{\mathsf{T}} N \, d\Omega,$$

with additional edge integrals for Neumann and Robin boundary conditions.

Assembly over all elements produces the global system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\},$$

which reduces to $[K]\{T\} = \{F\}$ in the steady-state case.

Input DSL structure

Analyses using `heat2d_steady_bc` are configured in the same DSL style as other solvers in the codebase. Users specify material properties (k, and for transient problems ρ, c), mesh data, volumetric sources q, and boundary condition groups (Dirichlet, Neumann, and where supported, convection data h, T_∞).

A schematic DSL fragment looks like:

```
solver heat2d_steady_bc {
  material steel {
    rho = 7850
    c   = 460
    k   = 45
  }

  region plate {
    mesh     = "meshes/plate_quad.msh"
    material = steel
  }

  boundary_conditions {
    dirichlet { ... }
    neumann   { ... }
    convection{ ... }    # if supported
  }

  loads {
    volumetric_heat { q = 1.0e6 }
  }

  # time_integration block for transient variants
}
```

Output fields

Typical outputs include:

- nodal temperatures,
- temperature distributions at selected times (transient),
- optionally, derived heat flux fields $\mathbf{q} = -k\nabla T$.

Typical use cases

- steady and transient temperature analysis of one-, two-, or three-dimensional solids,
- verification and benchmark problems for coupled thermo-mechanical simulations,
- parametric studies of material or boundary condition effects on thermal response.

Limitations and notes

- Material properties are typically treated as temperature-independent; strong temperature dependence requires explicit user handling.
- Only conduction and simple convection are modelled; radiation is not included unless added separately.
- Accuracy depends on mesh resolution and, for transient problems, on the chosen time-step size.

8.39 heat2d_transient

Documentation for solver `heat2d_transient`

Overview

The `heat2d_transient` solver performs two-dimensional transient heat conduction analysis. The unknown field is the temperature as a function of space and time.

Governing equations

The governing equation for transient heat conduction is

$$\rho c \frac{\partial T}{\partial t} - \nabla \cdot (k \nabla T) = q \quad \text{in } \Omega,$$

where ρ is the density, c the specific heat capacity, k the thermal conductivity (possibly tensor-valued), and q a volumetric heat source term.

Boundary conditions

Boundary conditions supported by the various heat conduction solvers include:

- prescribed temperature (Dirichlet) on parts of the boundary,
- prescribed normal heat flux (Neumann),
- convection (Robin) of the form

  $$-k \nabla T \cdot \mathbf{n} = h(T - T_\infty),$$

  where h is a convection coefficient and T_∞ an ambient reference temperature.

Finite element formulation

The temperature field is approximated by standard finite element shape functions N_i,

$$T_h(x) = \sum_i N_i(x)\, T_i,$$

and the Galerkin method is used to obtain a system of algebraic equations. For each element the conductivity matrix and (where applicable) capacity matrix are assembled as

$$[K]_e = \int_{\Omega_e} (\nabla N)^{\mathsf{T}} k \, \nabla N \, d\Omega, \qquad [C]_e = \int_{\Omega_e} \rho c \, N^{\mathsf{T}} N \, d\Omega,$$

with additional edge integrals for Neumann and Robin boundary conditions.

Assembly over all elements produces the global system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\},$$

which reduces to $[K]\{T\} = \{F\}$ in the steady-state case.

Time discretisation

Using a standard single-step implicit time integration scheme, the semi-discrete system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\}$$

is advanced from time level n to $n+1$ according to

$$\left(\frac{1}{\Delta t}[C] + \theta[K]\right)\{T\}^{n+1} = \left(\frac{1}{\Delta t}[C] - (1-\theta)[K]\right)\{T\}^{n} + \theta\{F\}^{n+1} + (1-\theta)\{F\}^{n},$$

with $0.5 \le \theta \le 1$ (Crank–Nicolson to backward Euler).

Input DSL structure

Analyses using `heat2d_transient` are configured in the same DSL style as other solvers in the codebase. Users specify material properties (k, and for transient problems ρ, c), mesh data, volumetric sources q, and boundary condition groups (Dirichlet, Neumann, and where supported, convection data h, T_∞).

A schematic DSL fragment looks like:

```
solver heat2d_transient {
  material steel {
    rho = 7850
    c   = 460
    k   = 45
  }

  region plate {
    mesh     = "meshes/plate_quad.msh"
    material = steel
  }

  boundary_conditions {
    dirichlet { ... }
    neumann   { ... }
    convection{ ... }    # if supported
  }
```

```
  loads {
    volumetric_heat { q = 1.0e6 }
  }

  # time_integration block for transient variants
}
```

Output fields

Typical outputs include:

- nodal temperatures,
- temperature distributions at selected times (transient),
- optionally, derived heat flux fields $\mathbf{q} = -k\nabla T$.

Typical use cases

- steady and transient temperature analysis of one-, two-, or three-dimensional solids,
- verification and benchmark problems for coupled thermo-mechanical simulations,
- parametric studies of material or boundary condition effects on thermal response.

Limitations and notes

- Material properties are typically treated as temperature-independent; strong temperature dependence requires explicit user handling.
- Only conduction and simple convection are modelled; radiation is not included unless added separately.
- Accuracy depends on mesh resolution and, for transient problems, on the chosen time-step size.

8.40 heat2d_transient_bc

Documentation for solver `heat2d_transient_bc`

Overview

The `heat2d_transient_bc` solver performs two-dimensional transient heat conduction analysis. The unknown field is the temperature as a function of space and time.

Governing equations

The governing equation for transient heat conduction is

$$\rho c \frac{\partial T}{\partial t} - \nabla \cdot (k \nabla T) = q \quad \text{in } \Omega,$$

where ρ is the density, c the specific heat capacity, k the thermal conductivity (possibly tensor-valued), and q a volumetric heat source term.

Boundary conditions

Boundary conditions supported by the various heat conduction solvers include:

- prescribed temperature (Dirichlet) on parts of the boundary,
- prescribed normal heat flux (Neumann),
- convection (Robin) of the form

  $$-k \nabla T \cdot \mathbf{n} = h (T - T_\infty),$$

  where h is a convection coefficient and T_∞ an ambient reference temperature.

Finite element formulation

The temperature field is approximated by standard finite element shape functions N_i,

$$T_h(x) = \sum_i N_i(x)\, T_i,$$

and the Galerkin method is used to obtain a system of algebraic equations. For each element the conductivity matrix and (where applicable) capacity matrix are assembled as

$$[K]_e = \int_{\Omega_e} (\nabla N)^{\mathsf{T}} k \, \nabla N \, d\Omega, \qquad [C]_e = \int_{\Omega_e} \rho c \, N^{\mathsf{T}} N \, d\Omega,$$

with additional edge integrals for Neumann and Robin boundary conditions.

Assembly over all elements produces the global system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\},$$

which reduces to $[K]\{T\} = \{F\}$ in the steady-state case.

Time discretisation

Using a standard single-step implicit time integration scheme, the semi-discrete system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\}$$

is advanced from time level n to $n+1$ according to

$$\left(\frac{1}{\Delta t}[C] + \theta [K] \right) \{T\}^{n+1} = \left(\frac{1}{\Delta t}[C] - (1-\theta)[K] \right) \{T\}^{n} + \theta \{F\}^{n+1} + (1-\theta)\{F\}^{n},$$

with $0.5 \le \theta \le 1$ (Crank–Nicolson to backward Euler).

Input DSL structure

Analyses using `heat2d_transient_bc` are configured in the same DSL style as other solvers in the codebase. Users specify material properties (k, and for transient problems ρ, c), mesh data, volumetric sources q, and boundary condition groups (Dirichlet, Neumann, and where supported, convection data h, T_∞).

A schematic DSL fragment looks like:

```
solver heat2d_transient_bc {
  material steel {
    rho = 7850
    c   = 460
    k   = 45
  }

  region plate {
    mesh     = "meshes/plate_quad.msh"
    material = steel
  }

  boundary_conditions {
    dirichlet { ... }
    neumann   { ... }
    convection{ ... }    # if supported
  }

  loads {
    volumetric_heat { q = 1.0e6 }
  }

  # time_integration block for transient variants
}
```

Output fields

Typical outputs include:

- nodal temperatures,
- temperature distributions at selected times (transient),
- optionally, derived heat flux fields $\mathbf{q} = -k\nabla T$.

Typical use cases

- steady and transient temperature analysis of one-, two-, or three-dimensional solids,
- verification and benchmark problems for coupled thermo-mechanical simulations,
- parametric studies of material or boundary condition effects on thermal response.

Limitations and notes

- Material properties are typically treated as temperature-independent; strong temperature dependence requires explicit user handling.
- Only conduction and simple convection are modelled; radiation is not included unless added separately.
- Accuracy depends on mesh resolution and, for transient problems, on the chosen time-step size.

8.41 heat2d_transient_conv

Documentation for solver `heat2d_transient_conv`

Overview

The `heat2d_transient_conv` solver performs two-dimensional transient heat conduction analysis. The unknown field is the temperature as a function of space and time.

Governing equations

The governing equation for transient heat conduction is

$$\rho c \frac{\partial T}{\partial t} - \nabla \cdot (k \nabla T) = q \quad \text{in } \Omega,$$

where ρ is the density, c the specific heat capacity, k the thermal conductivity (possibly tensor-valued), and q a volumetric heat source term.

Boundary conditions

Boundary conditions supported by the various heat conduction solvers include:

- prescribed temperature (Dirichlet) on parts of the boundary,
- prescribed normal heat flux (Neumann),
- convection (Robin) of the form

  $$-k \nabla T \cdot \mathbf{n} = h(T - T_\infty),$$

  where h is a convection coefficient and T_∞ an ambient reference temperature.

Finite element formulation

The temperature field is approximated by standard finite element shape functions N_i,

$$T_h(x) = \sum_i N_i(x)\, T_i,$$

and the Galerkin method is used to obtain a system of algebraic equations. For each element the conductivity matrix and (where applicable) capacity matrix are assembled as

$$[K]_e = \int_{\Omega_e} (\nabla N)^{\mathsf{T}} k \, \nabla N \, d\Omega, \qquad [C]_e = \int_{\Omega_e} \rho c \, N^{\mathsf{T}} N \, d\Omega,$$

with additional edge integrals for Neumann and Robin boundary conditions.

Assembly over all elements produces the global system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\},$$

which reduces to $[K]\{T\} = \{F\}$ in the steady-state case.

Time discretisation

Using a standard single-step implicit time integration scheme, the semi-discrete system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\}$$

is advanced from time level n to $n+1$ according to

$$\left(\frac{1}{\Delta t}[C] + \theta[K]\right)\{T\}^{n+1} = \left(\frac{1}{\Delta t}[C] - (1-\theta)[K]\right)\{T\}^{n} + \theta\{F\}^{n+1} + (1-\theta)\{F\}^{n},$$

with $0.5 \le \theta \le 1$ (Crank–Nicolson to backward Euler).

Input DSL structure

Analyses using `heat2d_transient_conv` are configured in the same DSL style as other solvers in the codebase. Users specify material properties (k, and for transient problems ρ, c), mesh data, volumetric sources q, and boundary condition groups (Dirichlet, Neumann, and where supported, convection data h, T_∞).

A schematic DSL fragment looks like:

```
solver heat2d_transient_conv {
  material steel {
    rho = 7850
    c   = 460
    k   = 45
  }

  region plate {
    mesh     = "meshes/plate_quad.msh"
    material = steel
  }

  boundary_conditions {
    dirichlet { ... }
    neumann   { ... }
    convection{ ... }    # if supported
  }
```

```
  loads {
    volumetric_heat { q = 1.0e6 }
  }

  # time_integration block for transient variants
}
```

Output fields

Typical outputs include:

- nodal temperatures,
- temperature distributions at selected times (transient),
- optionally, derived heat flux fields $\mathbf{q} = -k\nabla T$.

Typical use cases

- steady and transient temperature analysis of one-, two-, or three-dimensional solids,
- verification and benchmark problems for coupled thermo-mechanical simulations,
- parametric studies of material or boundary condition effects on thermal response.

Limitations and notes

- Material properties are typically treated as temperature-independent; strong temperature dependence requires explicit user handling.
- Only conduction and simple convection are modelled; radiation is not included unless added separately.
- Accuracy depends on mesh resolution and, for transient problems, on the chosen time-step size.

8.42 heat3d_h8_transient

Documentation for solver `heat3d_h8_transient`

Overview

The `heat3d_h8_transient` solver performs three-dimensional transient heat conduction analysis. The unknown field is the temperature as a function of space and time.

Governing equations

The governing equation for transient heat conduction is

$$\rho c \frac{\partial T}{\partial t} - \nabla \cdot (k \nabla T) = q \quad \text{in } \Omega,$$

where ρ is the density, c the specific heat capacity, k the thermal conductivity (possibly tensor-valued), and q a volumetric heat source term.

Boundary conditions

Boundary conditions supported by the various heat conduction solvers include:

- prescribed temperature (Dirichlet) on parts of the boundary,
- prescribed normal heat flux (Neumann),
- convection (Robin) of the form

  $$-k \nabla T \cdot \mathbf{n} = h (T - T_\infty),$$

  where h is a convection coefficient and T_∞ an ambient reference temperature.

Finite element formulation

The temperature field is approximated by standard finite element shape functions N_i,

$$T_h(x) = \sum_i N_i(x)\, T_i,$$

and the Galerkin method is used to obtain a system of algebraic equations. For each element the conductivity matrix and (where applicable) capacity matrix are assembled as

$$[K]_e = \int_{\Omega_e} (\nabla N)^{\mathsf{T}} k \, \nabla N \, d\Omega, \qquad [C]_e = \int_{\Omega_e} \rho c \, N^{\mathsf{T}} N \, d\Omega,$$

with additional edge integrals for Neumann and Robin boundary conditions.

Assembly over all elements produces the global system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\},$$

which reduces to $[K]\{T\} = \{F\}$ in the steady-state case.

Time discretisation

Using a standard single-step implicit time integration scheme, the semi-discrete system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\}$$

is advanced from time level n to $n+1$ according to

$$\left(\frac{1}{\Delta t}[C] + \theta [K] \right) \{T\}^{n+1} = \left(\frac{1}{\Delta t}[C] - (1-\theta)[K] \right) \{T\}^{n} + \theta \{F\}^{n+1} + (1-\theta)\{F\}^{n},$$

with $0.5 \le \theta \le 1$ (Crank–Nicolson to backward Euler).

Input DSL structure

Analyses using `heat3d_h8_transient` are configured in the same DSL style as other solvers in the codebase. Users specify material properties (k, and for transient problems ρ, c), mesh data, volumetric sources q, and boundary condition groups (Dirichlet, Neumann, and where supported, convection data h, T_∞).

A schematic DSL fragment looks like:

```
solver heat3d_h8_transient {
  material steel {
    rho = 7850
    c   = 460
    k   = 45
  }

  region plate {
    mesh     = "meshes/plate_quad.msh"
    material = steel
  }

  boundary_conditions {
    dirichlet { ... }
    neumann   { ... }
    convection{ ... }    # if supported
  }

  loads {
    volumetric_heat { q = 1.0e6 }
  }

  # time_integration block for transient variants
}
```

Output fields

Typical outputs include:

- nodal temperatures,
- temperature distributions at selected times (transient),
- optionally, derived heat flux fields $\mathbf{q} = -k\nabla T$.

Typical use cases

- steady and transient temperature analysis of one-, two-, or three-dimensional solids,
- verification and benchmark problems for coupled thermo-mechanical simulations,
- parametric studies of material or boundary condition effects on thermal response.

Limitations and notes

- Material properties are typically treated as temperature-independent; strong temperature dependence requires explicit user handling.
- Only conduction and simple convection are modelled; radiation is not included unless added separately.
- Accuracy depends on mesh resolution and, for transient problems, on the chosen time-step size.

8.43 heat3d_transient

Documentation for solver `heat3d_transient`

Overview

The `heat3d_transient` solver performs three-dimensional transient heat conduction analysis. The unknown field is the temperature as a function of space and time.

Governing equations

The governing equation for transient heat conduction is

$$\rho c \frac{\partial T}{\partial t} - \nabla \cdot (k \nabla T) = q \quad \text{in } \Omega,$$

where ρ is the density, c the specific heat capacity, k the thermal conductivity (possibly tensor-valued), and q a volumetric heat source term.

Boundary conditions

Boundary conditions supported by the various heat conduction solvers include:

- prescribed temperature (Dirichlet) on parts of the boundary,
- prescribed normal heat flux (Neumann),
- convection (Robin) of the form

$$-k \nabla T \cdot \mathbf{n} = h(T - T_\infty),$$

  where h is a convection coefficient and T_∞ an ambient reference temperature.

Finite element formulation

The temperature field is approximated by standard finite element shape functions N_i,

$$T_h(x) = \sum_i N_i(x)\, T_i,$$

and the Galerkin method is used to obtain a system of algebraic equations. For each element the conductivity matrix and (where applicable) capacity matrix are assembled as

$$[K]_e = \int_{\Omega_e} (\nabla N)^{\mathsf{T}} k \, \nabla N \, d\Omega, \qquad [C]_e = \int_{\Omega_e} \rho c \, N^{\mathsf{T}} N \, d\Omega,$$

with additional edge integrals for Neumann and Robin boundary conditions.

Assembly over all elements produces the global system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\},$$

which reduces to $[K]\{T\} = \{F\}$ in the steady-state case.

Time discretisation

Using a standard single-step implicit time integration scheme, the semi-discrete system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\}$$

is advanced from time level n to $n+1$ according to

$$\left(\frac{1}{\Delta t}[C] + \theta[K]\right)\{T\}^{n+1} = \left(\frac{1}{\Delta t}[C] - (1-\theta)[K]\right)\{T\}^{n} + \theta\{F\}^{n+1} + (1-\theta)\{F\}^{n},$$

with $0.5 \le \theta \le 1$ (Crank–Nicolson to backward Euler).

Input DSL structure

Analyses using `heat3d_transient` are configured in the same DSL style as other solvers in the codebase. Users specify material properties (k, and for transient problems ρ, c), mesh data, volumetric sources q, and boundary condition groups (Dirichlet, Neumann, and where supported, convection data h, T_∞).

A schematic DSL fragment looks like:

```
solver heat3d_transient {
  material steel {
    rho = 7850
    c   = 460
    k   = 45
  }

  region plate {
    mesh     = "meshes/plate_quad.msh"
    material = steel
  }

  boundary_conditions {
    dirichlet { ... }
    neumann   { ... }
    convection{ ... }    # if supported
  }
```

```
  loads {
    volumetric_heat { q = 1.0e6 }
  }

  # time_integration block for transient variants
}
```

Output fields

Typical outputs include:

- nodal temperatures,
- temperature distributions at selected times (transient),
- optionally, derived heat flux fields $\mathbf{q} = -k\nabla T$.

Typical use cases

- steady and transient temperature analysis of one-, two-, or three-dimensional solids,
- verification and benchmark problems for coupled thermo-mechanical simulations,
- parametric studies of material or boundary condition effects on thermal response.

Limitations and notes

- Material properties are typically treated as temperature-independent; strong temperature dependence requires explicit user handling.
- Only conduction and simple convection are modelled; radiation is not included unless added separately.
- Accuracy depends on mesh resolution and, for transient problems, on the chosen time-step size.

8.44 heat_axi_transient

Documentation for solver `heat_axi_transient`

Overview

The `heat_axi_transient` solver performs two-dimensional transient heat conduction analysis. The unknown field is the temperature as a function of space and time.

Governing equations

The governing equation for transient heat conduction is

$$\rho c \frac{\partial T}{\partial t} - \nabla \cdot (k \nabla T) = q \quad \text{in } \Omega,$$

where ρ is the density, c the specific heat capacity, k the thermal conductivity (possibly tensor-valued), and q a volumetric heat source term.

Boundary conditions

Boundary conditions supported by the various heat conduction solvers include:

- prescribed temperature (Dirichlet) on parts of the boundary,
- prescribed normal heat flux (Neumann),
- convection (Robin) of the form

  $$-k \nabla T \cdot \mathbf{n} = h (T - T_\infty),$$

  where h is a convection coefficient and T_∞ an ambient reference temperature.

Finite element formulation

The temperature field is approximated by standard finite element shape functions N_i,

$$T_h(x) = \sum_i N_i(x)\, T_i,$$

and the Galerkin method is used to obtain a system of algebraic equations. For each element the conductivity matrix and (where applicable) capacity matrix are assembled as

$$[K]_e = \int_{\Omega_e} (\nabla N)^{\mathsf{T}} k \, \nabla N \, d\Omega, \qquad [C]_e = \int_{\Omega_e} \rho c \, N^{\mathsf{T}} N \, d\Omega,$$

with additional edge integrals for Neumann and Robin boundary conditions.

Assembly over all elements produces the global system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\},$$

which reduces to $[K]\{T\} = \{F\}$ in the steady-state case.

Time discretisation

Using a standard single-step implicit time integration scheme, the semi-discrete system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\}$$

is advanced from time level n to $n+1$ according to

$$\left(\frac{1}{\Delta t}[C] + \theta [K] \right) \{T\}^{n+1} = \left(\frac{1}{\Delta t}[C] - (1-\theta)[K] \right) \{T\}^{n} + \theta \{F\}^{n+1} + (1-\theta)\{F\}^{n},$$

with $0.5 \le \theta \le 1$ (Crank–Nicolson to backward Euler).

Input DSL structure

Analyses using `heat_axi_transient` are configured in the same DSL style as other solvers in the codebase. Users specify material properties (k, and for transient problems ρ, c), mesh data, volumetric sources q, and boundary condition groups (Dirichlet, Neumann, and where supported, convection data h, T_∞).

A schematic DSL fragment looks like:

```
solver heat_axi_transient {
  material steel {
    rho = 7850
    c   = 460
    k   = 45
  }

  region plate {
    mesh     = "meshes/plate_quad.msh"
    material = steel
  }

  boundary_conditions {
    dirichlet { ... }
    neumann   { ... }
    convection{ ... }    # if supported
  }

  loads {
    volumetric_heat { q = 1.0e6 }
  }

  # time_integration block for transient variants
}
```

Output fields

Typical outputs include:

- nodal temperatures,
- temperature distributions at selected times (transient),
- optionally, derived heat flux fields $\mathbf{q} = -k\nabla T$.

Typical use cases

- steady and transient temperature analysis of one-, two-, or three-dimensional solids,
- verification and benchmark problems for coupled thermo-mechanical simulations,
- parametric studies of material or boundary condition effects on thermal response.

Limitations and notes

- Material properties are typically treated as temperature-independent; strong temperature dependence requires explicit user handling.
- Only conduction and simple convection are modelled; radiation is not included unless added separately.
- Accuracy depends on mesh resolution and, for transient problems, on the chosen time-step size.

8.45 input_utils

Documentation for solver `input_utils`

Overview

The `input_utils` module provides supporting functionality used by the physics solvers. It does not correspond to an independent physical model, but instead offers utilities such as:

- boundary condition handling and parsing,
- input/output helpers for meshes and results,
- material or section data management,
- linear algebra wrappers or post-processing routines.

Role in the codebase

Rather than assembling and solving its own finite element model, `input_utils` is called by other solvers to perform common tasks and to avoid duplication of logic. The precise responsibilities can be seen from the module's public API (functions and classes).

Input and output

Inputs and outputs are specific to the helper functions provided and may include:

- dictionaries representing solver state or model data,
- arrays of nodal and elemental information,
- handles to files or streams for reading/writing results.

As such, this module is typically not invoked directly from the user-facing DSL, but rather used internally.

8.46 linalg

Documentation for solver `linalg`

Overview

The `linalg` module provides supporting functionality used by the physics solvers. It does not correspond to an independent physical model, but instead offers utilities such as:

- boundary condition handling and parsing,
- input/output helpers for meshes and results,
- material or section data management,
- linear algebra wrappers or post-processing routines.

Role in the codebase

Rather than assembling and solving its own finite element model, `linalg` is called by other solvers to perform common tasks and to avoid duplication of logic. The precise responsibilities can be seen from the module's public API (functions and classes).

Input and output

Inputs and outputs are specific to the helper functions provided and may include:

- dictionaries representing solver state or model data,
- arrays of nodal and elemental information,
- handles to files or streams for reading/writing results.

As such, this module is typically not invoked directly from the user-facing DSL, but rather used internally.

8.47 materials

Documentation for solver `materials`

Overview

The `materials` module provides supporting functionality used by the physics solvers. It does not correspond to an independent physical model, but instead offers utilities such as:

- boundary condition handling and parsing,
- input/output helpers for meshes and results,
- material or section data management,
- linear algebra wrappers or post-processing routines.

Role in the codebase

Rather than assembling and solving its own finite element model, `materials` is called by other solvers to perform common tasks and to avoid duplication of logic. The precise responsibilities can be seen from the module's public API (functions and classes).

Input and output

Inputs and outputs are specific to the helper functions provided and may include:

- dictionaries representing solver state or model data,
- arrays of nodal and elemental information,
- handles to files or streams for reading/writing results.

As such, this module is typically not invoked directly from the user-facing DSL, but rather used internally.

8.48 mesh_rect

Documentation for solver `mesh_rect`

Overview

The `mesh_rect` module provides supporting functionality used by the physics solvers. It does not correspond to an independent physical model, but instead offers utilities such as:

- boundary condition handling and parsing,
- input/output helpers for meshes and results,
- material or section data management,
- linear algebra wrappers or post-processing routines.

Role in the codebase

Rather than assembling and solving its own finite element model, `mesh_rect` is called by other solvers to perform common tasks and to avoid duplication of logic. The precise responsibilities can be seen from the module's public API (functions and classes).

Input and output

Inputs and outputs are specific to the helper functions provided and may include:

- dictionaries representing solver state or model data,
- arrays of nodal and elemental information,
- handles to files or streams for reading/writing results.

As such, this module is typically not invoked directly from the user-facing DSL, but rather used internally.

8.49 nl_core

Documentation for solver `nl_core`

Overview

The `nl_core` solver belongs to the continuum solid mechanics group. It implements a finite element formulation for general three-dimensional small-strain linear elasticity using solid elements. The primary unknown field is the displacement vector $\mathbf{u}$.

Governing equations

In the absence of inertia and damping the quasi-static equilibrium equations are

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \quad \text{in } \Omega,$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{b}$ denotes body forces per unit volume.

For linear elastic materials the constitutive relation is

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon},$$

with $\mathbb{C}$ the fourth-order elasticity tensor and $\boldsymbol{\varepsilon}$ the infinitesimal strain tensor

$$\boldsymbol{\varepsilon} = \tfrac{1}{2}\big(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathsf{T}}\big).$$

The particular kinematic assumptions (plane stress, plane strain, axisymmetry) are imposed at the element level through the strain–displacement matrices and the form of $\mathbb{C}$.

Boundary conditions

Boundary conditions are specified as:

- prescribed displacements $\mathbf{u} = \bar{\mathbf{u}}$ on Γ_u (Dirichlet),
- prescribed tractions $\boldsymbol{\sigma}\mathbf{n} = \bar{\mathbf{t}}$ on Γ_t (Neumann),

with $\Gamma_u \cup \Gamma_t = \partial\Omega$ and $\Gamma_u \cap \Gamma_t = \emptyset$.

Finite element formulation

The displacement field is interpolated using standard isoparametric shape functions,

$$\mathbf{u}_h(x) = \sum_i N_i(x)\mathbf{u}_i.$$

The weak form of the equilibrium equations leads to the standard finite element system

$$[K]\{\mathbf{U}\} = \{\mathbf{F}\},$$

with

$$[K]_e = \int_{\Omega_e} [B]^{\mathsf{T}} \mathbb{C} [B] \, d\Omega, \qquad \{\mathbf{F}\}_e = \int_{\Omega_e} N^{\mathsf{T}} \mathbf{b} \, d\Omega + \int_{\Gamma_{t,e}} N^{\mathsf{T}} \bar{\mathbf{t}} \, d\Gamma,$$

where $[B]$ is the strain–displacement matrix for the chosen element type.

Element stiffness matrices and load vectors are assembled into the global system and displacement boundary conditions are enforced by modifying the global matrix and right-hand side.

Input DSL structure

In the project DSL, use of `nl_core` involves specifying:

- material properties (e.g. Young's modulus, Poisson ratio),
- mesh topology (nodes and elements),
- boundary condition sets for prescribed displacements and tractions,
- body forces or equivalent nodal loads where needed.

The exact syntax is consistent with other structural solvers and allows combining this solver with post-processing, load stepping, and study drivers.

Output fields

Typical outputs include:

- nodal displacements,
- element or integration-point stresses and strains,
- reaction forces at constrained degrees of freedom,
- derived scalar measures (e.g. von Mises stress).

Typical use cases

- analysis of plane and axisymmetric components under mechanical loading,
- 3D solid analysis of parts and assemblies in structural mechanics,
- benchmark problems for code verification in elasticity.

Limitations and notes

- The formulation assumes small strains and rotations unless extended variants are used.
- Material behaviour is linear elastic in the base implementation; any nonlinear or inelastic behaviour belongs to specialised solvers.
- Mesh quality and appropriate boundary condition specification are critical for accurate results.

8.50 plane2d_q4_modal

Documentation for solver `plane2d_q4_modal`

Overview

The `plane2d_q4_modal` solver belongs to the continuum solid mechanics group. It implements a finite element formulation for general three-dimensional small-strain linear elasticity using solid elements. The primary unknown field is the displacement vector **u**.

Governing equations

In the absence of inertia and damping the quasi-static equilibrium equations are

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \quad \text{in } \Omega,$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{b}$ denotes body forces per unit volume.

For linear elastic materials the constitutive relation is

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon},$$

with $\mathbb{C}$ the fourth-order elasticity tensor and $\boldsymbol{\varepsilon}$ the infinitesimal strain tensor

$$\boldsymbol{\varepsilon} = \tfrac{1}{2}\big(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathsf{T}}\big).$$

The particular kinematic assumptions (plane stress, plane strain, axisymmetry) are imposed at the element level through the strain–displacement matrices and the form of $\mathbb{C}$.

Boundary conditions

Boundary conditions are specified as:

- prescribed displacements $\mathbf{u} = \bar{\mathbf{u}}$ on Γ_u (Dirichlet),
- prescribed tractions $\boldsymbol{\sigma}\mathbf{n} = \bar{\mathbf{t}}$ on Γ_t (Neumann),

with $\Gamma_u \cup \Gamma_t = \partial\Omega$ and $\Gamma_u \cap \Gamma_t = \emptyset$.

Finite element formulation

The displacement field is interpolated using standard isoparametric shape functions,

$$\mathbf{u}_h(x) = \sum_{i} N_i(x)\,\mathbf{u}_i.$$

The weak form of the equilibrium equations leads to the standard finite element system

$$[K]\{\mathbf{U}\} = \{\mathbf{F}\},$$

with

$$[K]_e = \int_{\Omega_e} [B]^{\mathsf{T}} \mathbb{C} [B]\, d\Omega, \qquad \{\mathbf{F}\}_e = \int_{\Omega_e} N^{\mathsf{T}} \mathbf{b}\, d\Omega + \int_{\Gamma_{t,e}} N^{\mathsf{T}} \bar{\mathbf{t}}\, d\Gamma,$$

where $[B]$ is the strain–displacement matrix for the chosen element type.

Element stiffness matrices and load vectors are assembled into the global system and displacement boundary conditions are enforced by modifying the global matrix and right-hand side.

Input DSL structure

In the project DSL, use of `plane2d_q4_modal` involves specifying:

- material properties (e.g. Young's modulus, Poisson ratio),
- mesh topology (nodes and elements),
- boundary condition sets for prescribed displacements and tractions,
- body forces or equivalent nodal loads where needed.

The exact syntax is consistent with other structural solvers and allows combining this solver with post-processing, load stepping, and study drivers.

Output fields

Typical outputs include:

- nodal displacements,
- element or integration-point stresses and strains,
- reaction forces at constrained degrees of freedom,
- derived scalar measures (e.g. von Mises stress).

Typical use cases

- analysis of plane and axisymmetric components under mechanical loading,
- 3D solid analysis of parts and assemblies in structural mechanics,
- benchmark problems for code verification in elasticity.

Limitations and notes

- The formulation assumes small strains and rotations unless extended variants are used.
- Material behaviour is linear elastic in the base implementation; any nonlinear or inelastic behaviour belongs to specialised solvers.
- Mesh quality and appropriate boundary condition specification are critical for accurate results.

8.51 plane2d_q4_static

Documentation for solver `plane2d_q4_static`

Overview

The `plane2d_q4_static` solver belongs to the continuum solid mechanics group. It implements a finite element formulation for general three-dimensional small-strain linear elasticity using solid elements. The primary unknown field is the displacement vector **u**.

Governing equations

In the absence of inertia and damping the quasi-static equilibrium equations are

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \quad \text{in } \Omega,$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{b}$ denotes body forces per unit volume.

For linear elastic materials the constitutive relation is

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon},$$

with $\mathbb{C}$ the fourth-order elasticity tensor and $\boldsymbol{\varepsilon}$ the infinitesimal strain tensor

$$\boldsymbol{\varepsilon} = \tfrac{1}{2}\big(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathsf{T}}\big).$$

The particular kinematic assumptions (plane stress, plane strain, axisymmetry) are imposed at the element level through the strain–displacement matrices and the form of $\mathbb{C}$.

Boundary conditions

Boundary conditions are specified as:

- prescribed displacements $\mathbf{u} = \bar{\mathbf{u}}$ on Γ_u (Dirichlet),
- prescribed tractions $\boldsymbol{\sigma}\mathbf{n} = \bar{\mathbf{t}}$ on Γ_t (Neumann),

with $\Gamma_u \cup \Gamma_t = \partial\Omega$ and $\Gamma_u \cap \Gamma_t = \emptyset$.

Finite element formulation

The displacement field is interpolated using standard isoparametric shape functions,

$$\mathbf{u}_h(x) = \sum_{i} N_i(x)\,\mathbf{u}_i.$$

The weak form of the equilibrium equations leads to the standard finite element system

$$[K]\{\mathbf{U}\} = \{\mathbf{F}\},$$

with

$$[K]_e = \int_{\Omega_e} [B]^{\mathsf{T}} \mathbb{C} [B]\, d\Omega, \qquad \{\mathbf{F}\}_e = \int_{\Omega_e} N^{\mathsf{T}} \mathbf{b}\, d\Omega + \int_{\Gamma_{t,e}} N^{\mathsf{T}} \bar{\mathbf{t}}\, d\Gamma,$$

where $[B]$ is the strain–displacement matrix for the chosen element type.

Element stiffness matrices and load vectors are assembled into the global system and displacement boundary conditions are enforced by modifying the global matrix and right-hand side.

Input DSL structure

In the project DSL, use of `plane2d_q4_static` involves specifying:

- material properties (e.g. Young's modulus, Poisson ratio),
- mesh topology (nodes and elements),
- boundary condition sets for prescribed displacements and tractions,
- body forces or equivalent nodal loads where needed.

The exact syntax is consistent with other structural solvers and allows combining this solver with post-processing, load stepping, and study drivers.

Output fields

Typical outputs include:

- nodal displacements,
- element or integration-point stresses and strains,
- reaction forces at constrained degrees of freedom,
- derived scalar measures (e.g. von Mises stress).

Typical use cases

- analysis of plane and axisymmetric components under mechanical loading,
- 3D solid analysis of parts and assemblies in structural mechanics,
- benchmark problems for code verification in elasticity.

Limitations and notes

- The formulation assumes small strains and rotations unless extended variants are used.
- Material behaviour is linear elastic in the base implementation; any nonlinear or inelastic behaviour belongs to specialised solvers.
- Mesh quality and appropriate boundary condition specification are critical for accurate results.

8.52 plane2d_q4_thermoelastic

Documentation for solver `plane2d_q4_thermoelastic`

Overview

The `plane2d_q4_thermoelastic` solver belongs to the continuum solid mechanics group. It implements a finite element formulation for general three-dimensional small-strain linear elasticity using solid elements. The primary unknown field is the displacement vector $\mathbf{u}$.

Governing equations

In the absence of inertia and damping the quasi-static equilibrium equations are

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \quad \text{in } \Omega,$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{b}$ denotes body forces per unit volume.

For linear elastic materials the constitutive relation is

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon},$$

with $\mathbb{C}$ the fourth-order elasticity tensor and $\boldsymbol{\varepsilon}$ the infinitesimal strain tensor

$$\boldsymbol{\varepsilon} = \tfrac{1}{2}\big(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathsf{T}}\big).$$

The particular kinematic assumptions (plane stress, plane strain, axisymmetry) are imposed at the element level through the strain–displacement matrices and the form of $\mathbb{C}$.

Boundary conditions

Boundary conditions are specified as:

- prescribed displacements $\mathbf{u} = \bar{\mathbf{u}}$ on Γ_u (Dirichlet),
- prescribed tractions $\boldsymbol{\sigma}\mathbf{n} = \bar{\mathbf{t}}$ on Γ_t (Neumann),

with $\Gamma_u \cup \Gamma_t = \partial\Omega$ and $\Gamma_u \cap \Gamma_t = \emptyset$.

Finite element formulation

The displacement field is interpolated using standard isoparametric shape functions,

$$\mathbf{u}_h(x) = \sum_{i} N_i(x)\,\mathbf{u}_i.$$

The weak form of the equilibrium equations leads to the standard finite element system

$$[K]\{\mathbf{U}\} = \{\mathbf{F}\},$$

with

$$[K]_e = \int_{\Omega_e} [B]^{\mathsf{T}} \mathbb{C} [B] \, d\Omega, \qquad \{\mathbf{F}\}_e = \int_{\Omega_e} N^{\mathsf{T}} \mathbf{b} \, d\Omega + \int_{\Gamma_{t,e}} N^{\mathsf{T}} \bar{\mathbf{t}} \, d\Gamma,$$

where $[B]$ is the strain–displacement matrix for the chosen element type.

Element stiffness matrices and load vectors are assembled into the global system and displacement boundary conditions are enforced by modifying the global matrix and right-hand side.

Input DSL structure

In the project DSL, use of `plane2d_q4_thermoelastic` involves specifying:

- material properties (e.g. Young's modulus, Poisson ratio),
- mesh topology (nodes and elements),
- boundary condition sets for prescribed displacements and tractions,
- body forces or equivalent nodal loads where needed.

The exact syntax is consistent with other structural solvers and allows combining this solver with post-processing, load stepping, and study drivers.

Output fields

Typical outputs include:

- nodal displacements,
- element or integration-point stresses and strains,
- reaction forces at constrained degrees of freedom,
- derived scalar measures (e.g. von Mises stress).

Typical use cases

- analysis of plane and axisymmetric components under mechanical loading,
- 3D solid analysis of parts and assemblies in structural mechanics,
- benchmark problems for code verification in elasticity.

Limitations and notes

- The formulation assumes small strains and rotations unless extended variants are used.
- Material behaviour is linear elastic in the base implementation; any nonlinear or inelastic behaviour belongs to specialised solvers.
- Mesh quality and appropriate boundary condition specification are critical for accurate results.

8.53 plane_strain2d

Documentation for solver `plane_strain2d`

Overview

The `plane_strain2d` solver belongs to the continuum solid mechanics group. It implements a finite element formulation for plane strain elasticity, where out-of-plane strain components vanish and the problem is effectively two-dimensional. The primary unknown field is the displacement vector $\mathbf{u}$.

Governing equations

In the absence of inertia and damping the quasi-static equilibrium equations are

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \quad \text{in } \Omega,$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{b}$ denotes body forces per unit volume.

For linear elastic materials the constitutive relation is

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon},$$

with $\mathbb{C}$ the fourth-order elasticity tensor and $\boldsymbol{\varepsilon}$ the infinitesimal strain tensor

$$\boldsymbol{\varepsilon} = \tfrac{1}{2}\big(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathsf{T}}\big).$$

The particular kinematic assumptions (plane stress, plane strain, axisymmetry) are imposed at the element level through the strain–displacement matrices and the form of $\mathbb{C}$.

Boundary conditions

Boundary conditions are specified as:

- prescribed displacements $\mathbf{u} = \bar{\mathbf{u}}$ on Γ_u (Dirichlet),
- prescribed tractions $\boldsymbol{\sigma}\mathbf{n} = \bar{\mathbf{t}}$ on Γ_t (Neumann),

with $\Gamma_u \cup \Gamma_t = \partial\Omega$ and $\Gamma_u \cap \Gamma_t = \emptyset$.

Finite element formulation

The displacement field is interpolated using standard isoparametric shape functions,

$$\mathbf{u}_h(x) = \sum_i N_i(x)\mathbf{u}_i.$$

The weak form of the equilibrium equations leads to the standard finite element system

$$[K]\{\mathbf{U}\} = \{\mathbf{F}\},$$

with

$$[K]_e = \int_{\Omega_e} [B]^{\mathsf{T}} \mathbb{C} [B] \, d\Omega, \qquad \{\mathbf{F}\}_e = \int_{\Omega_e} N^{\mathsf{T}} \mathbf{b} \, d\Omega + \int_{\Gamma_{t,e}} N^{\mathsf{T}} \bar{\mathbf{t}} \, d\Gamma,$$

where $[B]$ is the strain–displacement matrix for the chosen element type.

Element stiffness matrices and load vectors are assembled into the global system and displacement boundary conditions are enforced by modifying the global matrix and right-hand side.

Input DSL structure

In the project DSL, use of `plane_strain2d` involves specifying:

- material properties (e.g. Young's modulus, Poisson ratio),
- mesh topology (nodes and elements),
- boundary condition sets for prescribed displacements and tractions,
- body forces or equivalent nodal loads where needed.

The exact syntax is consistent with other structural solvers and allows combining this solver with post-processing, load stepping, and study drivers.

Output fields

Typical outputs include:

- nodal displacements,
- element or integration-point stresses and strains,
- reaction forces at constrained degrees of freedom,
- derived scalar measures (e.g. von Mises stress).

Typical use cases

- analysis of plane and axisymmetric components under mechanical loading,
- 3D solid analysis of parts and assemblies in structural mechanics,
- benchmark problems for code verification in elasticity.

Limitations and notes

- The formulation assumes small strains and rotations unless extended variants are used.
- Material behaviour is linear elastic in the base implementation; any nonlinear or inelastic behaviour belongs to specialised solvers.
- Mesh quality and appropriate boundary condition specification are critical for accurate results.

8.54 plane_strain_q4_static

Documentation for solver `plane_strain_q4_static`

Overview

The `plane_strain_q4_static` solver belongs to the continuum solid mechanics group. It implements a finite element formulation for plane strain elasticity, where out-of-plane strain components vanish and the problem is effectively two-dimensional. The primary unknown field is the displacement vector $\mathbf{u}$.

Governing equations

In the absence of inertia and damping the quasi-static equilibrium equations are

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \quad \text{in } \Omega,$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{b}$ denotes body forces per unit volume.

For linear elastic materials the constitutive relation is

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon},$$

with $\mathbb{C}$ the fourth-order elasticity tensor and $\boldsymbol{\varepsilon}$ the infinitesimal strain tensor

$$\boldsymbol{\varepsilon} = \tfrac{1}{2}\big(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathsf{T}}\big).$$

The particular kinematic assumptions (plane stress, plane strain, axisymmetry) are imposed at the element level through the strain–displacement matrices and the form of $\mathbb{C}$.

Boundary conditions

Boundary conditions are specified as:

- prescribed displacements $\mathbf{u} = \bar{\mathbf{u}}$ on Γ_u (Dirichlet),
- prescribed tractions $\boldsymbol{\sigma}\mathbf{n} = \bar{\mathbf{t}}$ on Γ_t (Neumann),

with $\Gamma_u \cup \Gamma_t = \partial\Omega$ and $\Gamma_u \cap \Gamma_t = \emptyset$.

Finite element formulation

The displacement field is interpolated using standard isoparametric shape functions,

$$\mathbf{u}_h(x) = \sum_i N_i(x)\mathbf{u}_i.$$

The weak form of the equilibrium equations leads to the standard finite element system

$$[K]\{\mathbf{U}\} = \{\mathbf{F}\},$$

with

$$[K]_e = \int_{\Omega_e} [B]^{\mathsf{T}} \mathbb{C} [B] \, d\Omega, \qquad \{\mathbf{F}\}_e = \int_{\Omega_e} N^{\mathsf{T}} \mathbf{b} \, d\Omega + \int_{\Gamma_{t,e}} N^{\mathsf{T}} \bar{\mathbf{t}} \, d\Gamma,$$

where $[B]$ is the strain–displacement matrix for the chosen element type.

Element stiffness matrices and load vectors are assembled into the global system and displacement boundary conditions are enforced by modifying the global matrix and right-hand side.

Input DSL structure

In the project DSL, use of `plane_strain_q4_static` involves specifying:

- material properties (e.g. Young's modulus, Poisson ratio),
- mesh topology (nodes and elements),
- boundary condition sets for prescribed displacements and tractions,
- body forces or equivalent nodal loads where needed.

The exact syntax is consistent with other structural solvers and allows combining this solver with post-processing, load stepping, and study drivers.

Output fields

Typical outputs include:

- nodal displacements,
- element or integration-point stresses and strains,
- reaction forces at constrained degrees of freedom,
- derived scalar measures (e.g. von Mises stress).

Typical use cases

- analysis of plane and axisymmetric components under mechanical loading,
- 3D solid analysis of parts and assemblies in structural mechanics,
- benchmark problems for code verification in elasticity.

Limitations and notes

- The formulation assumes small strains and rotations unless extended variants are used.
- Material behaviour is linear elastic in the base implementation; any nonlinear or inelastic behaviour belongs to specialised solvers.
- Mesh quality and appropriate boundary condition specification are critical for accurate results.

8.55 plane_stress2d

Documentation for solver `plane_stress2d`

Overview

The `plane_stress2d` solver belongs to the continuum solid mechanics group. It implements a finite element formulation for plane stress elasticity, appropriate for thin plates where out-of-plane stresses are negligible. The primary unknown field is the displacement vector $\mathbf{u}$.

Governing equations

In the absence of inertia and damping the quasi-static equilibrium equations are

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \quad \text{in } \Omega,$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{b}$ denotes body forces per unit volume.

For linear elastic materials the constitutive relation is

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon},$$

with $\mathbb{C}$ the fourth-order elasticity tensor and $\boldsymbol{\varepsilon}$ the infinitesimal strain tensor

$$\boldsymbol{\varepsilon} = \tfrac{1}{2}\big(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathsf{T}}\big).$$

The particular kinematic assumptions (plane stress, plane strain, axisymmetry) are imposed at the element level through the strain–displacement matrices and the form of $\mathbb{C}$.

Boundary conditions

Boundary conditions are specified as:

- prescribed displacements $\mathbf{u} = \bar{\mathbf{u}}$ on Γ_u (Dirichlet),
- prescribed tractions $\boldsymbol{\sigma}\mathbf{n} = \bar{\mathbf{t}}$ on Γ_t (Neumann),

with $\Gamma_u \cup \Gamma_t = \partial\Omega$ and $\Gamma_u \cap \Gamma_t = \emptyset$.

Finite element formulation

The displacement field is interpolated using standard isoparametric shape functions,

$$\mathbf{u}_h(x) = \sum_i N_i(x)\mathbf{u}_i.$$

The weak form of the equilibrium equations leads to the standard finite element system

$$[K]\{\mathbf{U}\} = \{\mathbf{F}\},$$

with

$$[K]_e = \int_{\Omega_e} [B]^{\mathsf{T}} \mathbb{C} [B] \, d\Omega, \qquad \{\mathbf{F}\}_e = \int_{\Omega_e} N^{\mathsf{T}} \mathbf{b} \, d\Omega + \int_{\Gamma_{t,e}} N^{\mathsf{T}} \bar{\mathbf{t}} \, d\Gamma,$$

where $[B]$ is the strain–displacement matrix for the chosen element type.

Element stiffness matrices and load vectors are assembled into the global system and displacement boundary conditions are enforced by modifying the global matrix and right-hand side.

Input DSL structure

In the project DSL, use of `plane_stress2d` involves specifying:

- material properties (e.g. Young's modulus, Poisson ratio),
- mesh topology (nodes and elements),
- boundary condition sets for prescribed displacements and tractions,
- body forces or equivalent nodal loads where needed.

The exact syntax is consistent with other structural solvers and allows combining this solver with post-processing, load stepping, and study drivers.

Output fields

Typical outputs include:

- nodal displacements,
- element or integration-point stresses and strains,
- reaction forces at constrained degrees of freedom,
- derived scalar measures (e.g. von Mises stress).

Typical use cases

- analysis of plane and axisymmetric components under mechanical loading,
- 3D solid analysis of parts and assemblies in structural mechanics,
- benchmark problems for code verification in elasticity.

Limitations and notes

- The formulation assumes small strains and rotations unless extended variants are used.
- Material behaviour is linear elastic in the base implementation; any nonlinear or inelastic behaviour belongs to specialised solvers.
- Mesh quality and appropriate boundary condition specification are critical for accurate results.

8.56 plane_stress_q4_static

Documentation for solver `plane_stress_q4_static`

Overview

The `plane_stress_q4_static` solver belongs to the continuum solid mechanics group. It implements a finite element formulation for plane stress elasticity, appropriate for thin plates where out-of-plane stresses are negligible. The primary unknown field is the displacement vector $\mathbf{u}$.

Governing equations

In the absence of inertia and damping the quasi-static equilibrium equations are

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \quad \text{in } \Omega,$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{b}$ denotes body forces per unit volume.

For linear elastic materials the constitutive relation is

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon},$$

with $\mathbb{C}$ the fourth-order elasticity tensor and $\boldsymbol{\varepsilon}$ the infinitesimal strain tensor

$$\boldsymbol{\varepsilon} = \tfrac{1}{2}\big(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathsf{T}}\big).$$

The particular kinematic assumptions (plane stress, plane strain, axisymmetry) are imposed at the element level through the strain–displacement matrices and the form of $\mathbb{C}$.

Boundary conditions

Boundary conditions are specified as:

- prescribed displacements $\mathbf{u} = \bar{\mathbf{u}}$ on Γ_u (Dirichlet),
- prescribed tractions $\boldsymbol{\sigma}\mathbf{n} = \bar{\mathbf{t}}$ on Γ_t (Neumann),

with $\Gamma_u \cup \Gamma_t = \partial\Omega$ and $\Gamma_u \cap \Gamma_t = \emptyset$.

Finite element formulation

The displacement field is interpolated using standard isoparametric shape functions,

$$\mathbf{u}_h(x) = \sum_i N_i(x)\mathbf{u}_i.$$

The weak form of the equilibrium equations leads to the standard finite element system

$$[K]\{\mathbf{U}\} = \{\mathbf{F}\},$$

with

$$[K]_e = \int_{\Omega_e} [B]^{\mathsf{T}} \mathbb{C} [B] \, d\Omega, \qquad \{\mathbf{F}\}_e = \int_{\Omega_e} N^{\mathsf{T}} \mathbf{b} \, d\Omega + \int_{\Gamma_{t,e}} N^{\mathsf{T}} \bar{\mathbf{t}} \, d\Gamma,$$

where $[B]$ is the strain–displacement matrix for the chosen element type.

Element stiffness matrices and load vectors are assembled into the global system and displacement boundary conditions are enforced by modifying the global matrix and right-hand side.

Input DSL structure

In the project DSL, use of `plane_stress_q4_static` involves specifying:

- material properties (e.g. Young's modulus, Poisson ratio),
- mesh topology (nodes and elements),
- boundary condition sets for prescribed displacements and tractions,
- body forces or equivalent nodal loads where needed.

The exact syntax is consistent with other structural solvers and allows combining this solver with post-processing, load stepping, and study drivers.

Output fields

Typical outputs include:

- nodal displacements,
- element or integration-point stresses and strains,
- reaction forces at constrained degrees of freedom,
- derived scalar measures (e.g. von Mises stress).

Typical use cases

- analysis of plane and axisymmetric components under mechanical loading,
- 3D solid analysis of parts and assemblies in structural mechanics,
- benchmark problems for code verification in elasticity.

Limitations and notes

- The formulation assumes small strains and rotations unless extended variants are used.
- Material behaviour is linear elastic in the base implementation; any nonlinear or inelastic behaviour belongs to specialised solvers.
- Mesh quality and appropriate boundary condition specification are critical for accurate results.

8.57 planestress2d

Documentation for solver `planestress2d`

Overview

The `planestress2d` solver belongs to the continuum solid mechanics group. It implements a finite element formulation for plane stress elasticity, appropriate for thin plates where out-of-plane stresses are negligible. The primary unknown field is the displacement vector $\mathbf{u}$.

Governing equations

In the absence of inertia and damping the quasi-static equilibrium equations are

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \quad \text{in } \Omega,$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{b}$ denotes body forces per unit volume.

For linear elastic materials the constitutive relation is

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon},$$

with $\mathbb{C}$ the fourth-order elasticity tensor and $\boldsymbol{\varepsilon}$ the infinitesimal strain tensor

$$\boldsymbol{\varepsilon} = \tfrac{1}{2}\big(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathsf{T}}\big).$$

The particular kinematic assumptions (plane stress, plane strain, axisymmetry) are imposed at the element level through the strain–displacement matrices and the form of $\mathbb{C}$.

Boundary conditions

Boundary conditions are specified as:

- prescribed displacements $\mathbf{u} = \bar{\mathbf{u}}$ on Γ_u (Dirichlet),
- prescribed tractions $\boldsymbol{\sigma}\mathbf{n} = \bar{\mathbf{t}}$ on Γ_t (Neumann),

with $\Gamma_u \cup \Gamma_t = \partial\Omega$ and $\Gamma_u \cap \Gamma_t = \emptyset$.

Finite element formulation

The displacement field is interpolated using standard isoparametric shape functions,

$$\mathbf{u}_h(x) = \sum_i N_i(x)\mathbf{u}_i.$$

The weak form of the equilibrium equations leads to the standard finite element system

$$[K]\{\mathbf{U}\} = \{\mathbf{F}\},$$

with

$$[K]_e = \int_{\Omega_e} [B]^{\mathsf{T}} \mathbb{C} [B] \, d\Omega, \qquad \{\mathbf{F}\}_e = \int_{\Omega_e} N^{\mathsf{T}} \mathbf{b} \, d\Omega + \int_{\Gamma_{t,e}} N^{\mathsf{T}} \bar{\mathbf{t}} \, d\Gamma,$$

where $[B]$ is the strain–displacement matrix for the chosen element type.

Element stiffness matrices and load vectors are assembled into the global system and displacement boundary conditions are enforced by modifying the global matrix and right-hand side.

Input DSL structure

In the project DSL, use of `planestress2d` involves specifying:

- material properties (e.g. Young's modulus, Poisson ratio),
- mesh topology (nodes and elements),
- boundary condition sets for prescribed displacements and tractions,
- body forces or equivalent nodal loads where needed.

The exact syntax is consistent with other structural solvers and allows combining this solver with post-processing, load stepping, and study drivers.

Output fields

Typical outputs include:

- nodal displacements,
- element or integration-point stresses and strains,
- reaction forces at constrained degrees of freedom,
- derived scalar measures (e.g. von Mises stress).

Typical use cases

- analysis of plane and axisymmetric components under mechanical loading,
- 3D solid analysis of parts and assemblies in structural mechanics,
- benchmark problems for code verification in elasticity.

Limitations and notes

- The formulation assumes small strains and rotations unless extended variants are used.
- Material behaviour is linear elastic in the base implementation; any nonlinear or inelastic behaviour belongs to specialised solvers.
- Mesh quality and appropriate boundary condition specification are critical for accurate results.

8.58 plate2d_mindlin_modal

Documentation for solver `plate2d_mindlin_modal`

Overview

The `plate2d_mindlin_modal` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

For free-vibration (modal) analysis, the discretised equations of motion

$$[M]\{\ddot{U}\} + [C]\{\dot{U}\} + [K]\{U\} = \{F\}$$

are recast as a homogeneous eigenproblem

$$[K]\{\phi\} = \lambda[M]\{\phi\},$$

to compute natural frequencies and mode shapes.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `plate2d_mindlin_modal` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.59 plate2d_mindlin_static

Documentation for solver `plate2d_mindlin_static`

Overview

The `plate2d_mindlin_static` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

In the purely static case the assembled finite element equations reduce to

$$[K]\{U\} = \{F\},$$

with $[K]$ containing both axial and bending stiffness contributions from frame or beam elements.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `plate2d_mindlin_static` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.60 plate_mindlin2d

Documentation for solver `plate_mindlin2d`

Overview

The `plate_mindlin2d` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

In the purely static case the assembled finite element equations reduce to

$$[K]\{U\} = \{F\},$$

with $[K]$ containing both axial and bending stiffness contributions from frame or beam elements.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `plate_mindlin2d` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.61 plate_mindlin_static

Documentation for solver `plate_mindlin_static`

Overview

The `plate_mindlin_static` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

In the purely static case the assembled finite element equations reduce to

$$[K]\{U\} = \{F\},$$

with $[K]$ containing both axial and bending stiffness contributions from frame or beam elements.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `plate_mindlin_static` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.62 plate_mindlin_winkler

Documentation for solver `plate_mindlin_winkler`

Overview

The `plate_mindlin_winkler` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

In the purely static case the assembled finite element equations reduce to

$$[K]\{U\} = \{F\},$$

with $[K]$ containing both axial and bending stiffness contributions from frame or beam elements.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `plate_mindlin_winkler` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.63 ps_linear_fe

Documentation for solver `ps_linear_fe`

Overview

The `ps_linear_fe` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

In the purely static case the assembled finite element equations reduce to

$$[K]\{U\} = \{F\},$$

with $[K]$ containing both axial and bending stiffness contributions from frame or beam elements.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `ps_linear_fe` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.64 ps_vonmises_fe

Documentation for solver `ps_vonmises_fe`

Overview

The `ps_vonmises_fe` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

In the purely static case the assembled finite element equations reduce to

$$[K]\{U\} = \{F\},$$

with $[K]$ containing both axial and bending stiffness contributions from frame or beam elements.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `ps_vonmises_fe` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.65 shell2d_mindlin_static

Documentation for solver `shell2d_mindlin_static`

Overview

The `shell2d_mindlin_static` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

In the purely static case the assembled finite element equations reduce to

$$[K]\{U\} = \{F\},$$

with $[K]$ containing both axial and bending stiffness contributions from frame or beam elements.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `shell2d_mindlin_static` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.66 shell_flat_dkt_proxy

Documentation for solver `shell_flat_dkt_proxy`

Overview

The `shell_flat_dkt_proxy` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

In the purely static case the assembled finite element equations reduce to

$$[K]\{U\} = \{F\},$$

with $[K]$ containing both axial and bending stiffness contributions from frame or beam elements.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `shell_flat_dkt_proxy` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.67 shell_membrane_static

Documentation for solver `shell_membrane_static`

Overview

The `shell_membrane_static` solver belongs to the beam and frame element family. It is used to analyse structural members that can carry axial force, bending moment, and shear force, organised in one- or two-dimensional frames.

Depending on the specific variant, the solver can perform static, dynamic, buckling, or modal analyses of frame structures.

Governing equations

The governing equations follow from equilibrium of forces and moments in combination with the appropriate beam theory (e.g. Euler–Bernoulli or Timoshenko). At the continuum level this leads to differential equations for deflection and rotation along the member.

After finite element discretisation, the member behaviour is described by element stiffness matrices that include axial, bending, and possibly shear components. Axial deformation is governed by relations of the form

$$N = EA\frac{du}{dx},$$

while bending response involves

$$M = EI\frac{d^2w}{dx^2},$$

with E Young's modulus, A cross-sectional area, I second moment of area, u axial displacement, and w transverse deflection.

In the purely static case the assembled finite element equations reduce to

$$[K]\{U\} = \{F\},$$

with $[K]$ containing both axial and bending stiffness contributions from frame or beam elements.

Finite element formulation

Frame and beam elements interpolate displacements and rotations along the member using appropriate shape functions. Typical element degrees of freedom include axial displacements and transverse displacements/rotations at the element nodes. Element stiffness matrices are first derived in local member coordinates and then transformed to the global coordinate system using direction cosines.

The solver assembles:

- stiffness matrices from member properties (E, A, I, length),
- mass matrices from density and cross-sectional properties (for dynamic and modal analyses),
- geometric stiffness matrices (for buckling or second-order effects) where implemented.

Boundary conditions and loading

Boundary conditions are applied as constraints on nodal degrees of freedom (displacements and rotations) to represent common support types such as:

- fixed, pinned, or roller supports,
- internal hinges or releases,
- imposed displacements or rotations.

Loads may be specified as:

- nodal forces and moments,
- distributed loads along members (converted internally to equivalent nodal loads),
- time-dependent loads for dynamic analyses.

Input DSL structure

Using `shell_membrane_static` in the DSL typically involves defining:

- material properties and cross-sections,
- node coordinates and frame element connectivity,
- boundary condition groups (supports, releases),
- load cases and, where appropriate, time histories.

The DSL syntax mirrors that of other structural solvers and allows re-use of the same mesh and material definitions across different study types.

Output fields

Typical outputs include:

- nodal displacements and rotations,
- element end forces and bending moments,
- internal force diagrams (shear, moment, axial force),
- reaction forces at supports,
- natural frequencies and mode shapes (modal analyses),
- critical load factors and buckling modes (buckling analyses).

Typical use cases

- analysis of planar and space frame structures in buildings and bridges,
- dynamic response of frames to time-varying loads,
- stability and buckling checks for frame members.

Limitations and notes

- The underlying beam theory (Euler–Bernoulli or Timoshenko) determines the accuracy for slender or deep beams.
- Large rotation and nonlinear effects are only covered in specialised variants that explicitly implement them.
- Accurate representation of releases and boundary conditions is critical for realistic internal force distributions.

8.68 solid3d_hex20

Documentation for solver `solid3d_hex20`

Overview

The `solid3d_hex20` solver belongs to the continuum solid mechanics group. It implements a finite element formulation for general three-dimensional small-strain linear elasticity using solid elements. The primary unknown field is the displacement vector $\mathbf{u}$.

Governing equations

In the absence of inertia and damping the quasi-static equilibrium equations are

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \quad \text{in } \Omega,$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{b}$ denotes body forces per unit volume.

For linear elastic materials the constitutive relation is

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon},$$

with $\mathbb{C}$ the fourth-order elasticity tensor and $\boldsymbol{\varepsilon}$ the infinitesimal strain tensor

$$\boldsymbol{\varepsilon} = \tfrac{1}{2}\big(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathsf{T}}\big).$$

The particular kinematic assumptions (plane stress, plane strain, axisymmetry) are imposed at the element level through the strain–displacement matrices and the form of $\mathbb{C}$.

Boundary conditions

Boundary conditions are specified as:

- prescribed displacements $\mathbf{u} = \bar{\mathbf{u}}$ on Γ_u (Dirichlet),
- prescribed tractions $\boldsymbol{\sigma}\mathbf{n} = \bar{\mathbf{t}}$ on Γ_t (Neumann),

with $\Gamma_u \cup \Gamma_t = \partial\Omega$ and $\Gamma_u \cap \Gamma_t = \emptyset$.

Finite element formulation

The displacement field is interpolated using standard isoparametric shape functions,

$$\mathbf{u}_h(x) = \sum_i N_i(x)\mathbf{u}_i.$$

The weak form of the equilibrium equations leads to the standard finite element system

$$[K]\{\mathbf{U}\} = \{\mathbf{F}\},$$

with

$$[K]_e = \int_{\Omega_e} [B]^{\mathsf{T}}\mathbb{C}[B]\, d\Omega, \qquad \{\mathbf{F}\}_e = \int_{\Omega_e} N^{\mathsf{T}}\mathbf{b}\, d\Omega + \int_{\Gamma_{t,e}} N^{\mathsf{T}}\bar{\mathbf{t}}\, d\Gamma,$$

where $[B]$ is the strain–displacement matrix for the chosen element type.

Element stiffness matrices and load vectors are assembled into the global system and displacement boundary conditions are enforced by modifying the global matrix and right-hand side.

Input DSL structure

In the project DSL, use of `solid3d_hex20` involves specifying:

- material properties (e.g. Young's modulus, Poisson ratio),
- mesh topology (nodes and elements),
- boundary condition sets for prescribed displacements and tractions,
- body forces or equivalent nodal loads where needed.

The exact syntax is consistent with other structural solvers and allows combining this solver with post-processing, load stepping, and study drivers.

Output fields

Typical outputs include:

- nodal displacements,
- element or integration-point stresses and strains,
- reaction forces at constrained degrees of freedom,
- derived scalar measures (e.g. von Mises stress).

Typical use cases

- analysis of plane and axisymmetric components under mechanical loading,
- 3D solid analysis of parts and assemblies in structural mechanics,
- benchmark problems for code verification in elasticity.

Limitations and notes

- The formulation assumes small strains and rotations unless extended variants are used.
- Material behaviour is linear elastic in the base implementation; any nonlinear or inelastic behaviour belongs to specialised solvers.
- Mesh quality and appropriate boundary condition specification are critical for accurate results.

8.69 solid3d_hex8_nonlinear

Documentation for solver `solid3d_hex8_nonlinear`

Overview

The `solid3d_hex8_nonlinear` solver belongs to the continuum solid mechanics group. It implements a finite element formulation for general three-dimensional small-strain linear elasticity using solid elements. The primary unknown field is the displacement vector $\mathbf{u}$.

Governing equations

In the absence of inertia and damping the quasi-static equilibrium equations are

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \quad \text{in } \Omega,$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{b}$ denotes body forces per unit volume.

For linear elastic materials the constitutive relation is

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon},$$

with $\mathbb{C}$ the fourth-order elasticity tensor and $\boldsymbol{\varepsilon}$ the infinitesimal strain tensor

$$\boldsymbol{\varepsilon} = \tfrac{1}{2}\big(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathsf{T}}\big).$$

The particular kinematic assumptions (plane stress, plane strain, axisymmetry) are imposed at the element level through the strain–displacement matrices and the form of $\mathbb{C}$.

Boundary conditions

Boundary conditions are specified as:

- prescribed displacements $\mathbf{u} = \bar{\mathbf{u}}$ on Γ_u (Dirichlet),
- prescribed tractions $\boldsymbol{\sigma}\mathbf{n} = \bar{\mathbf{t}}$ on Γ_t (Neumann),

with $\Gamma_u \cup \Gamma_t = \partial\Omega$ and $\Gamma_u \cap \Gamma_t = \emptyset$.

Finite element formulation

The displacement field is interpolated using standard isoparametric shape functions,

$$\mathbf{u}_h(x) = \sum_i N_i(x)\mathbf{u}_i.$$

The weak form of the equilibrium equations leads to the standard finite element system

$$[K]\{\mathbf{U}\} = \{\mathbf{F}\},$$

with

$$[K]_e = \int_{\Omega_e} [B]^{\mathsf{T}}\mathbb{C}[B]\, d\Omega, \qquad \{\mathbf{F}\}_e = \int_{\Omega_e} N^{\mathsf{T}}\mathbf{b}\, d\Omega + \int_{\Gamma_{t,e}} N^{\mathsf{T}}\bar{\mathbf{t}}\, d\Gamma,$$

where $[B]$ is the strain–displacement matrix for the chosen element type.

Element stiffness matrices and load vectors are assembled into the global system and displacement boundary conditions are enforced by modifying the global matrix and right-hand side.

Input DSL structure

In the project DSL, use of `solid3d_hex8_nonlinear` involves specifying:

- material properties (e.g. Young's modulus, Poisson ratio),
- mesh topology (nodes and elements),
- boundary condition sets for prescribed displacements and tractions,
- body forces or equivalent nodal loads where needed.

The exact syntax is consistent with other structural solvers and allows combining this solver with post-processing, load stepping, and study drivers.

Output fields

Typical outputs include:

- nodal displacements,
- element or integration-point stresses and strains,
- reaction forces at constrained degrees of freedom,
- derived scalar measures (e.g. von Mises stress).

Typical use cases

- analysis of plane and axisymmetric components under mechanical loading,
- 3D solid analysis of parts and assemblies in structural mechanics,
- benchmark problems for code verification in elasticity.

Limitations and notes

- The formulation assumes small strains and rotations unless extended variants are used.
- Material behaviour is linear elastic in the base implementation; any nonlinear or inelastic behaviour belongs to specialised solvers.
- Mesh quality and appropriate boundary condition specification are critical for accurate results.

8.70 solid3d_hex8_transient

Documentation for solver `solid3d_hex8_transient`

Overview

The `solid3d_hex8_transient` solver belongs to the continuum solid mechanics group. It implements a finite element formulation for general three-dimensional small-strain linear elasticity using solid elements. The primary unknown field is the displacement vector $\mathbf{u}$.

Governing equations

In the absence of inertia and damping the quasi-static equilibrium equations are

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \quad \text{in } \Omega,$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{b}$ denotes body forces per unit volume.

For linear elastic materials the constitutive relation is

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon},$$

with $\mathbb{C}$ the fourth-order elasticity tensor and $\boldsymbol{\varepsilon}$ the infinitesimal strain tensor

$$\boldsymbol{\varepsilon} = \tfrac{1}{2}\big(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathsf{T}}\big).$$

The particular kinematic assumptions (plane stress, plane strain, axisymmetry) are imposed at the element level through the strain–displacement matrices and the form of $\mathbb{C}$.

Boundary conditions

Boundary conditions are specified as:

- prescribed displacements $\mathbf{u} = \bar{\mathbf{u}}$ on Γ_u (Dirichlet),
- prescribed tractions $\boldsymbol{\sigma}\mathbf{n} = \bar{\mathbf{t}}$ on Γ_t (Neumann),

with $\Gamma_u \cup \Gamma_t = \partial\Omega$ and $\Gamma_u \cap \Gamma_t = \emptyset$.

Finite element formulation

The displacement field is interpolated using standard isoparametric shape functions,

$$\mathbf{u}_h(x) = \sum_i N_i(x)\mathbf{u}_i.$$

The weak form of the equilibrium equations leads to the standard finite element system

$$[K]\{\mathbf{U}\} = \{\mathbf{F}\},$$

with

$$[K]_e = \int_{\Omega_e} [B]^{\mathsf{T}}\mathbb{C}[B]\, d\Omega, \qquad \{\mathbf{F}\}_e = \int_{\Omega_e} N^{\mathsf{T}}\mathbf{b}\, d\Omega + \int_{\Gamma_{t,e}} N^{\mathsf{T}}\bar{\mathbf{t}}\, d\Gamma,$$

where $[B]$ is the strain–displacement matrix for the chosen element type.

Element stiffness matrices and load vectors are assembled into the global system and displacement boundary conditions are enforced by modifying the global matrix and right-hand side.

Input DSL structure

In the project DSL, use of `solid3d_hex8_transient` involves specifying:

- material properties (e.g. Young's modulus, Poisson ratio),
- mesh topology (nodes and elements),
- boundary condition sets for prescribed displacements and tractions,
- body forces or equivalent nodal loads where needed.

The exact syntax is consistent with other structural solvers and allows combining this solver with post-processing, load stepping, and study drivers.

Output fields

Typical outputs include:

- nodal displacements,
- element or integration-point stresses and strains,
- reaction forces at constrained degrees of freedom,
- derived scalar measures (e.g. von Mises stress).

Typical use cases

- analysis of plane and axisymmetric components under mechanical loading,
- 3D solid analysis of parts and assemblies in structural mechanics,
- benchmark problems for code verification in elasticity.

Limitations and notes

- The formulation assumes small strains and rotations unless extended variants are used.
- Material behaviour is linear elastic in the base implementation; any nonlinear or inelastic behaviour belongs to specialised solvers.
- Mesh quality and appropriate boundary condition specification are critical for accurate results.

8.71 solid_axisym_q4_static

Documentation for solver `solid_axisym_q4_static`

Overview

The `solid_axisym_q4_static` solver belongs to the continuum solid mechanics group. It implements a finite element formulation for axisymmetric elasticity in bodies of revolution, where all fields are independent of the circumferential angle. The primary unknown field is the displacement vector $\mathbf{u}$.

Governing equations

In the absence of inertia and damping the quasi-static equilibrium equations are

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \quad \text{in } \Omega,$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{b}$ denotes body forces per unit volume.

For linear elastic materials the constitutive relation is

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon},$$

with $\mathbb{C}$ the fourth-order elasticity tensor and $\boldsymbol{\varepsilon}$ the infinitesimal strain tensor

$$\boldsymbol{\varepsilon} = \tfrac{1}{2}\big(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathsf{T}}\big).$$

The particular kinematic assumptions (plane stress, plane strain, axisymmetry) are imposed at the element level through the strain–displacement matrices and the form of $\mathbb{C}$.

Boundary conditions

Boundary conditions are specified as:

- prescribed displacements $\mathbf{u} = \bar{\mathbf{u}}$ on Γ_u (Dirichlet),
- prescribed tractions $\boldsymbol{\sigma}\mathbf{n} = \bar{\mathbf{t}}$ on Γ_t (Neumann),

with $\Gamma_u \cup \Gamma_t = \partial\Omega$ and $\Gamma_u \cap \Gamma_t = \emptyset$.

Finite element formulation

The displacement field is interpolated using standard isoparametric shape functions,

$$\mathbf{u}_h(x) = \sum_i N_i(x)\mathbf{u}_i.$$

The weak form of the equilibrium equations leads to the standard finite element system

$$[K]\{\mathbf{U}\} = \{\mathbf{F}\},$$

with

$$[K]_e = \int_{\Omega_e} [B]^{\mathsf{T}}\mathbb{C}[B]\, d\Omega, \qquad \{\mathbf{F}\}_e = \int_{\Omega_e} N^{\mathsf{T}}\mathbf{b}\, d\Omega + \int_{\Gamma_{t,e}} N^{\mathsf{T}}\bar{\mathbf{t}}\, d\Gamma,$$

where $[B]$ is the strain–displacement matrix for the chosen element type.

Element stiffness matrices and load vectors are assembled into the global system and displacement boundary conditions are enforced by modifying the global matrix and right-hand side.

Input DSL structure

In the project DSL, use of `solid_axisym_q4_static` involves specifying:

- material properties (e.g. Young's modulus, Poisson ratio),
- mesh topology (nodes and elements),
- boundary condition sets for prescribed displacements and tractions,
- body forces or equivalent nodal loads where needed.

The exact syntax is consistent with other structural solvers and allows combining this solver with post-processing, load stepping, and study drivers.

Output fields

Typical outputs include:

- nodal displacements,
- element or integration-point stresses and strains,
- reaction forces at constrained degrees of freedom,
- derived scalar measures (e.g. von Mises stress).

Typical use cases

- analysis of plane and axisymmetric components under mechanical loading,
- 3D solid analysis of parts and assemblies in structural mechanics,
- benchmark problems for code verification in elasticity.

Limitations and notes

- The formulation assumes small strains and rotations unless extended variants are used.
- Material behaviour is linear elastic in the base implementation; any nonlinear or inelastic behaviour belongs to specialised solvers.
- Mesh quality and appropriate boundary condition specification are critical for accurate results.

8.72 study_drivers

Documentation for solver `study_drivers`

Overview

The `study_drivers` module provides supporting functionality used by the physics solvers. It does not correspond to an independent physical model, but instead offers utilities such as:

- boundary condition handling and parsing,
- input/output helpers for meshes and results,
- material or section data management,
- linear algebra wrappers or post-processing routines.

Role in the codebase

Rather than assembling and solving its own finite element model, `study_drivers` is called by other solvers to perform common tasks and to avoid duplication of logic. The precise responsibilities can be seen from the module's public API (functions and classes).

Input and output

Inputs and outputs are specific to the helper functions provided and may include:

- dictionaries representing solver state or model data,
- arrays of nodal and elemental information,
- handles to files or streams for reading/writing results.

As such, this module is typically not invoked directly from the user-facing DSL, but rather used internally.

8.73 study_fusion

Documentation for solver `study_fusion`

Overview

The `study_fusion` module provides supporting functionality used by the physics solvers. It does not correspond to an independent physical model, but instead offers utilities such as:

- boundary condition handling and parsing,
- input/output helpers for meshes and results,
- material or section data management,
- linear algebra wrappers or post-processing routines.

Role in the codebase

Rather than assembling and solving its own finite element model, `study_fusion` is called by other solvers to perform common tasks and to avoid duplication of logic. The precise responsibilities can be seen from the module's public API (functions and classes).

Input and output

Inputs and outputs are specific to the helper functions provided and may include:

- dictionaries representing solver state or model data,
- arrays of nodal and elemental information,
- handles to files or streams for reading/writing results.

As such, this module is typically not invoked directly from the user-facing DSL, but rather used internally.

8.74 thermal1d

Documentation for solver `thermal1d`

Overview

The `thermal1d` solver performs one-dimensional steady-state heat conduction analysis with time-independent loading and boundary conditions.

Governing equations

The steady-state heat conduction problem is governed by

$$-\nabla \cdot (k \nabla T) = q \quad \text{in } \Omega,$$

where T is the temperature field, k the thermal conductivity, and q a volumetric heat source term. Time derivatives are absent in the steady case.

Boundary conditions

Boundary conditions supported by the various heat conduction solvers include:

- prescribed temperature (Dirichlet) on parts of the boundary,
- prescribed normal heat flux (Neumann),
- convection (Robin) of the form

$$-k \nabla T \cdot \mathbf{n} = h(T - T_\infty),$$

  where h is a convection coefficient and T_∞ an ambient reference temperature.

Finite element formulation

The temperature field is approximated by standard finite element shape functions N_i,

$$T_h(x) = \sum_i N_i(x)\, T_i,$$

and the Galerkin method is used to obtain a system of algebraic equations. For each element the conductivity matrix and (where applicable) capacity matrix are assembled as

$$[K]_e = \int_{\Omega_e} (\nabla N)^{\mathsf{T}} k \, \nabla N \, d\Omega, \qquad [C]_e = \int_{\Omega_e} \rho c \, N^{\mathsf{T}} N \, d\Omega,$$

with additional edge integrals for Neumann and Robin boundary conditions.

Assembly over all elements produces the global system

$$[C]\{\dot{T}\} + [K]\{T\} = \{F\},$$

which reduces to $[K]\{T\} = \{F\}$ in the steady-state case.

Input DSL structure

Analyses using `thermal1d` are configured in the same DSL style as other solvers in the codebase. Users specify material properties (k, and for transient problems ρ, c), mesh data, volumetric sources q, and boundary condition groups (Dirichlet, Neumann, and where supported, convection data h, T_∞).

A schematic DSL fragment looks like:

```
solver thermal1d {
  material steel {
    rho = 7850
    c   = 460
    k   = 45
  }

  region plate {
    mesh     = "meshes/plate_quad.msh"
    material = steel
  }

  boundary_conditions {
    dirichlet { ... }
    neumann   { ... }
    convection{ ... }    # if supported
  }

  loads {
    volumetric_heat { q = 1.0e6 }
  }

  # time_integration block for transient variants
}
```

Output fields

Typical outputs include:

- nodal temperatures,
- temperature distributions at selected times (transient),
- optionally, derived heat flux fields $\mathbf{q} = -k\nabla T$.

Typical use cases

- steady and transient temperature analysis of one-, two-, or three-dimensional solids,
- verification and benchmark problems for coupled thermo-mechanical simulations,
- parametric studies of material or boundary condition effects on thermal response.

Limitations and notes

- Material properties are typically treated as temperature-independent; strong temperature dependence requires explicit user handling.
- Only conduction and simple convection are modelled; radiation is not included unless added separately.

- Accuracy depends on mesh resolution and, for transient problems, on the chosen time-step size.

8.75 thermomech2d_ps_coupled

Documentation for solver `thermomech2d_ps_coupled`

Overview

The `thermomech2d_ps_coupled` solver performs coupled thermo-mechanical analysis in which temperature and displacement fields interact. Thermal loads generate thermal strains, which in turn induce stresses and deformations in the structure.

Governing equations

The coupled problem combines:

- the equilibrium equations of solid mechanics

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0},$$

- the heat conduction equation

$$\rho c \dot{T} - \nabla \cdot (k \nabla T) = q,$$

with a constitutive law that includes thermal strain,

$$\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}^{\text{mech}} + \boldsymbol{\varepsilon}^{\text{th}}, \qquad \boldsymbol{\varepsilon}^{\text{th}} = \alpha (T - T_{\text{ref}}) \mathbf{I},$$

where α is a coefficient of thermal expansion and T_{ref} a reference temperature.

Finite element formulation

The finite element discretisation introduces nodal temperatures and displacements as primary unknowns. After assembling the mechanical and thermal contributions, the global system can be written schematically as

$$\begin{bmatrix} K_{uu} & K_{uT} \\ K_{Tu} & K_{TT} \end{bmatrix} \begin{Bmatrix} U \\ T \end{Bmatrix} = \begin{Bmatrix} F_u \\ F_T \end{Bmatrix},$$

where:

- K_{uu} is the mechanical stiffness matrix,
- K_{TT} is the thermal conduction (and, in transient cases, capacity) contribution,
- K_{uT} and K_{Tu} represent thermo-mechanical coupling terms.

Depending on the specific implementation, the thermal problem may be solved first and used as input to a purely mechanical step, or the full coupled system may be solved simultaneously.

Boundary conditions and loading

Boundary conditions combine those of structural mechanics (prescribed displacements and tractions) with those of heat transfer (prescribed temperature, flux, and convection). Thermal loads can be created by:

- prescribing nonuniform temperature fields on the structure,
- applying internal heat sources or boundary heat fluxes,
- defining convection to ambient conditions.

Input DSL structure

In the DSL, use of `thermomech2d_ps_coupled` involves specifying:

- mechanical material properties (elastic moduli, Poisson ratio),
- thermal properties (ρ, c, k, α),
- mesh and element topology,
- mechanical and thermal boundary conditions and loads,
- , for transient problems, time integration parameters.

Output fields

Typical outputs include:

- nodal displacement and temperature histories,
- mechanical stresses and strains (including thermal components),
- reaction forces and heat fluxes at boundaries.

Typical use cases

- analysis of thermally loaded structures (e.g. heated plates or beams),
- evaluation of thermal stresses in components exposed to temperature gradients,
- verification of uncoupled approaches that apply thermal fields from separate analyses.

Limitations and notes

- The coupling is typically one-way (thermal problem not influenced by mechanical dissipation) unless otherwise documented in the source code.
- Material properties may be temperature-dependent, but the level of support depends on the specific solver implementation.
- Appropriate time-stepping and convergence controls are important for accuracy in transient and nonlinear cases.

8.76 thermomech3d_coupled

Documentation for solver `thermomech3d_coupled`

Overview

The `thermomech3d_coupled` solver performs coupled thermo-mechanical analysis in which temperature and displacement fields interact. Thermal loads generate thermal strains, which in turn induce stresses and deformations in the structure.

Governing equations

The coupled problem combines:

- the equilibrium equations of solid mechanics

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0},$$

- the heat conduction equation

$$\rho c \dot{T} - \nabla \cdot (k \nabla T) = q,$$

with a constitutive law that includes thermal strain,

$$\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}^{\text{mech}} + \boldsymbol{\varepsilon}^{\text{th}}, \qquad \boldsymbol{\varepsilon}^{\text{th}} = \alpha (T - T_{\text{ref}}) \mathbf{I},$$

where α is a coefficient of thermal expansion and T_{ref} a reference temperature.

Finite element formulation

The finite element discretisation introduces nodal temperatures and displacements as primary unknowns. After assembling the mechanical and thermal contributions, the global system can be written schematically as

$$\begin{bmatrix} K_{uu} & K_{uT} \\ K_{Tu} & K_{TT} \end{bmatrix} \begin{Bmatrix} U \\ T \end{Bmatrix} = \begin{Bmatrix} F_u \\ F_T \end{Bmatrix},$$

where:

- K_{uu} is the mechanical stiffness matrix,
- K_{TT} is the thermal conduction (and, in transient cases, capacity) contribution,
- K_{uT} and K_{Tu} represent thermo-mechanical coupling terms.

Depending on the specific implementation, the thermal problem may be solved first and used as input to a purely mechanical step, or the full coupled system may be solved simultaneously.

Boundary conditions and loading

Boundary conditions combine those of structural mechanics (prescribed displacements and tractions) with those of heat transfer (prescribed temperature, flux, and convection). Thermal loads can be created by:

- prescribing nonuniform temperature fields on the structure,
- applying internal heat sources or boundary heat fluxes,
- defining convection to ambient conditions.

Input DSL structure

In the DSL, use of `thermomech3d_coupled` involves specifying:

- mechanical material properties (elastic moduli, Poisson ratio),
- thermal properties (ρ, c, k, α),
- mesh and element topology,
- mechanical and thermal boundary conditions and loads,
- , for transient problems, time integration parameters.

Output fields

Typical outputs include:

- nodal displacement and temperature histories,
- mechanical stresses and strains (including thermal components),
- reaction forces and heat fluxes at boundaries.

Typical use cases

- analysis of thermally loaded structures (e.g. heated plates or beams),
- evaluation of thermal stresses in components exposed to temperature gradients,
- verification of uncoupled approaches that apply thermal fields from separate analyses.

Limitations and notes

- The coupling is typically one-way (thermal problem not influenced by mechanical dissipation) unless otherwise documented in the source code.
- Material properties may be temperature-dependent, but the level of support depends on the specific solver implementation.
- Appropriate time-stepping and convergence controls are important for accuracy in transient and nonlinear cases.

8.77 truss2d

Documentation for solver `truss2d`

Overview

The `truss2d` solver implements a finite element formulation for truss structures. Members are idealised as straight bars that carry axial force only (no bending or shear), and nodes act as frictionless pins. In two- or three-dimensional space each node has translational degrees of freedom and elements are oriented arbitrarily according to their nodal coordinates.

Typical applications include planar and space truss structures in buildings, bridges, and lattice frameworks used in mechanical and civil engineering.

Governing equations

For a bar of length L, cross-sectional area A, and Young's modulus E, the axial force–displacement relation in local coordinates is

$$N = \frac{EA}{L}(u_2 - u_1),$$

where u_1 and u_2 are the axial displacements of the bar end nodes.

In matrix form, for local degrees of freedom $\{u\}_{\text{loc}} = \{u_1, u_2\}^{\mathsf{T}}$,

$$[k]_{\text{loc}} = \frac{EA}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}.$$

The transformation to global coordinates introduces direction cosines based on the nodal coordinates. For a 2D truss member with direction cosines c, s, the global element stiffness is

$$[k]_{\text{glob}} = \frac{EA}{L}\begin{bmatrix} c^2 & cs & -c^2 & -cs \\ cs & s^2 & -cs & -s^2 \\ -c^2 & -cs & c^2 & cs \\ -cs & -s^2 & cs & s^2 \end{bmatrix}.$$

At the global level the assembled linear system is

$$[K]\{U\} = \{F\},$$

with $[K]$ the global stiffness matrix, $\{U\}$ the nodal displacement vector, and $\{F\}$ the nodal load vector.

Finite element formulation

The solver assembles element stiffness (and mass or geometric stiffness, when applicable) matrices from the bar properties and connectivity. For each element:

- nodal coordinates define the element length and orientation,
- material properties provide E (and density ρ in dynamic or modal analyses),

- section data provide the cross-sectional area A.

Element matrices are transformed from local to global coordinates using direction cosine matrices and then added into the global system according to the element connectivity array.

Boundary conditions and loading

Boundary conditions are applied as constraints on translational degrees of freedom at nodes:

- fixed supports: all translational DOFs constrained,
- roller or pin supports: one or more DOFs left free,
- free nodes: no kinematic constraints.

Loads are typically applied as:

- nodal forces at joints,
- equivalent nodal forces obtained from distributed loads when such options are present in higher-level drivers.

Prescribed displacements can also be enforced by modifying the global system matrix and right-hand side.

Input DSL structure

In the surrounding project DSL, an analysis using `truss2d` is configured with blocks that define materials, sections, nodes, elements, supports, and loads. A schematic example is:

```
solver truss2d {
  material steel {
    E   = 210e9
    rho = 7850
  }

  section bar {
    area = 0.003
  }

  nodes {
    n1 = (0.0, 0.0)
    n2 = (5.0, 0.0)
    n3 = (2.5, 3.0)
  }

  elements {
    e1: truss2d (n1, n3) material=steel section=bar
    e2: truss2d (n3, n2) material=steel section=bar
```

```
  }

  supports {
    n1.ux = 0.0
    n1.uy = 0.0
    n2.uy = 0.0
  }

  loads {
    n3.Fy = -50e3
  }
}
```

The exact syntax is governed by the DSL front-end, but the solver expects coherent references to nodal sets, element sets, and material/section data.

Output fields

Typical outputs of the truss solvers include:

- nodal displacements,
- element axial forces,
- element axial stresses $\sigma = N/A$,
- reaction forces at constrained nodes,
- in modal or buckling variants, eigenvalues and mode shapes.

These quantities can be post-processed to generate deformed shape plots, force diagrams, or tables for design checks.

Typical use cases

- analysis and design of planar and space truss structures,
- benchmark problems for verification of more advanced finite element formulations,
- parametric studies of stiffness, load paths, and member utilisation.

Limitations and notes

- The basic formulation assumes pin-jointed members carrying axial force only; bending and shear are neglected unless a more general frame element is used instead.
- Geometric and material nonlinearity are only included in specialised variants that explicitly implement them.
- Accurate results require a truss-like structural behaviour; using a truss model for frame-like structures can be misleading.

8.78 truss2d_buckling_linear

Documentation for solver `truss2d_buckling_linear`

Overview

The `truss2d_buckling_linear` solver implements a finite element formulation for truss structures. Members are idealised as straight bars that carry axial force only (no bending or shear), and nodes act as frictionless pins. In two- or three-dimensional space each node has translational degrees of freedom and elements are oriented arbitrarily according to their nodal coordinates.

Typical applications include planar and space truss structures in buildings, bridges, and lattice frameworks used in mechanical and civil engineering.

Governing equations

For a bar of length L, cross-sectional area A, and Young's modulus E, the axial force–displacement relation in local coordinates is

$$N = \frac{EA}{L}(u_2 - u_1),$$

where u_1 and u_2 are the axial displacements of the bar end nodes.

In matrix form, for local degrees of freedom $\{u\}_{\text{loc}} = \{u_1, u_2\}^{\mathsf{T}}$,

$$[k]_{\text{loc}} = \frac{EA}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}.$$

The transformation to global coordinates introduces direction cosines based on the nodal coordinates. For a 2D truss member with direction cosines c, s, the global element stiffness is

$$[k]_{\text{glob}} = \frac{EA}{L}\begin{bmatrix} c^2 & cs & -c^2 & -cs \\ cs & s^2 & -cs & -s^2 \\ -c^2 & -cs & c^2 & cs \\ -cs & -s^2 & cs & s^2 \end{bmatrix}.$$

For linear buckling analysis the solver evaluates the geometric stiffness matrix $[K_G]$ associated with the pre-stress state and solves the eigenvalue problem

$$\big([K] + \lambda[K_G]\big)\{\phi\} = \mathbf{0},$$

where λ are the critical load multipliers and $[K]$ is the elastic stiffness matrix.

Finite element formulation

The solver assembles element stiffness (and mass or geometric stiffness, when applicable) matrices from the bar properties and connectivity. For each element:

- nodal coordinates define the element length and orientation,
- material properties provide E (and density ρ in dynamic or modal analyses),

- section data provide the cross-sectional area A.

Element matrices are transformed from local to global coordinates using direction cosine matrices and then added into the global system according to the element connectivity array.

Boundary conditions and loading

Boundary conditions are applied as constraints on translational degrees of freedom at nodes:

- fixed supports: all translational DOFs constrained,
- roller or pin supports: one or more DOFs left free,
- free nodes: no kinematic constraints.

Loads are typically applied as:

- nodal forces at joints,
- equivalent nodal forces obtained from distributed loads when such options are present in higher-level drivers.

Prescribed displacements can also be enforced by modifying the global system matrix and right-hand side.

Input DSL structure

In the surrounding project DSL, an analysis using `truss2d_buckling_linear` is configured with blocks that define materials, sections, nodes, elements, supports, and loads. A schematic example is:

```
solver truss2d_buckling_linear {
  material steel {
    E   = 210e9
    rho = 7850
  }

  section bar {
    area = 0.003
  }

  nodes {
    n1 = (0.0, 0.0)
    n2 = (5.0, 0.0)
    n3 = (2.5, 3.0)
  }

  elements {
    e1: truss2d (n1, n3) material=steel section=bar
```

```
    e2: truss2d (n3, n2) material=steel section=bar
  }

  supports {
    n1.ux = 0.0
    n1.uy = 0.0
    n2.uy = 0.0
  }

  loads {
    n3.Fy = -50e3
  }
}
```

The exact syntax is governed by the DSL front-end, but the solver expects coherent references to nodal sets, element sets, and material/section data.

Output fields

Typical outputs of the truss solvers include:

- nodal displacements,
- element axial forces,
- element axial stresses $\sigma = N/A$,
- reaction forces at constrained nodes,
- in modal or buckling variants, eigenvalues and mode shapes.

These quantities can be post-processed to generate deformed shape plots, force diagrams, or tables for design checks.

Typical use cases

- analysis and design of planar and space truss structures,
- benchmark problems for verification of more advanced finite element formulations,
- parametric studies of stiffness, load paths, and member utilisation.

Limitations and notes

- The basic formulation assumes pin-jointed members carrying axial force only; bending and shear are neglected unless a more general frame element is used instead.
- Geometric and material nonlinearity are only included in specialised variants that explicitly implement them.
- Accurate results require a truss-like structural behaviour; using a truss model for frame-like structures can be misleading.

8.79 truss2d_ep_bilin

Documentation for solver `truss2d_ep_bilin`

Overview

The `truss2d_ep_bilin` solver implements a finite element formulation for truss structures. Members are idealised as straight bars that carry axial force only (no bending or shear), and nodes act as frictionless pins. In two- or three-dimensional space each node has translational degrees of freedom and elements are oriented arbitrarily according to their nodal coordinates.

Typical applications include planar and space truss structures in buildings, bridges, and lattice frameworks used in mechanical and civil engineering.

This particular variant extends the basic truss formulation to include nonlinear effects or specialised response types.

Governing equations

For a bar of length L, cross-sectional area A, and Young's modulus E, the axial force–displacement relation in local coordinates is

$$N = \frac{EA}{L}(u_2 - u_1),$$

where u_1 and u_2 are the axial displacements of the bar end nodes.

In matrix form, for local degrees of freedom $\{u\}_{\text{loc}} = \{u_1, u_2\}^{\mathsf{T}}$,

$$[k]_{\text{loc}} = \frac{EA}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}.$$

The transformation to global coordinates introduces direction cosines based on the nodal coordinates. For a 2D truss member with direction cosines c, s, the global element stiffness is

$$[k]_{\text{glob}} = \frac{EA}{L}\begin{bmatrix} c^2 & cs & -c^2 & -cs \\ cs & s^2 & -cs & -s^2 \\ -c^2 & -cs & c^2 & cs \\ -cs & -s^2 & cs & s^2 \end{bmatrix}.$$

The global equilibrium is written as

$$\mathbf{r}(\mathbf{U}) = \mathbf{F}_{\text{ext}} - \mathbf{F}_{\text{int}}(\mathbf{U}) = \mathbf{0},$$

where $\mathbf{F}_{\text{int}}$ depends nonlinearly on the current deformation state.

In this variant, geometric and/or material nonlinearities are taken into account. The internal force vector becomes a nonlinear function of the nodal displacements, and the global equilibrium is solved by iterative methods (e.g. Newton–Raphson) instead of a single linear solve.

Finite element formulation

The solver assembles element stiffness (and mass or geometric stiffness, when applicable) matrices from the bar properties and connectivity. For each element:

- nodal coordinates define the element length and orientation,
- material properties provide E (and density ρ in dynamic or modal analyses),
- section data provide the cross-sectional area A.

Element matrices are transformed from local to global coordinates using direction cosine matrices and then added into the global system according to the element connectivity array.

Boundary conditions and loading

Boundary conditions are applied as constraints on translational degrees of freedom at nodes:

- fixed supports: all translational DOFs constrained,
- roller or pin supports: one or more DOFs left free,
- free nodes: no kinematic constraints.

Loads are typically applied as:

- nodal forces at joints,
- equivalent nodal forces obtained from distributed loads when such options are present in higher-level drivers.

Prescribed displacements can also be enforced by modifying the global system matrix and right-hand side.

Input DSL structure

In the surrounding project DSL, an analysis using `truss2d_ep_bilin` is configured with blocks that define materials, sections, nodes, elements, supports, and loads. A schematic example is:

```
solver truss2d_ep_bilin {
  material steel {
    E   = 210e9
    rho = 7850
  }

  section bar {
    area = 0.003
  }

  nodes {
```

```
    n1 = (0.0, 0.0)
    n2 = (5.0, 0.0)
    n3 = (2.5, 3.0)
  }

  elements {
    e1: truss2d (n1, n3) material=steel section=bar
    e2: truss2d (n3, n2) material=steel section=bar
  }

  supports {
    n1.ux = 0.0
    n1.uy = 0.0
    n2.uy = 0.0
  }

  loads {
    n3.Fy = -50e3
  }
}
```

The exact syntax is governed by the DSL front-end, but the solver expects coherent references to nodal sets, element sets, and material/section data.

Output fields

Typical outputs of the truss solvers include:

- nodal displacements,
- element axial forces,
- element axial stresses $\sigma = N/A$,
- reaction forces at constrained nodes,
- in modal or buckling variants, eigenvalues and mode shapes.

These quantities can be post-processed to generate deformed shape plots, force diagrams, or tables for design checks.

Typical use cases

- analysis and design of planar and space truss structures,
- benchmark problems for verification of more advanced finite element formulations,
- parametric studies of stiffness, load paths, and member utilisation.

Limitations and notes

- The basic formulation assumes pin-jointed members carrying axial force only; bending and shear are neglected unless a more general frame element is used instead.
- Geometric and material nonlinearity are only included in specialised variants that explicitly implement them.
- Accurate results require a truss-like structural behaviour; using a truss model for frame-like structures can be misleading.

8.80 truss2d_geom_nl

Documentation for solver `truss2d_geom_nl`

Overview

The `truss2d_geom_nl` solver implements a finite element formulation for truss structures. Members are idealised as straight bars that carry axial force only (no bending or shear), and nodes act as frictionless pins. In two- or three-dimensional space each node has translational degrees of freedom and elements are oriented arbitrarily according to their nodal coordinates.

Typical applications include planar and space truss structures in buildings, bridges, and lattice frameworks used in mechanical and civil engineering.

This particular variant extends the basic truss formulation to include nonlinear effects or specialised response types.

Governing equations

For a bar of length L, cross-sectional area A, and Young's modulus E, the axial force–displacement relation in local coordinates is

$$N = \frac{EA}{L}(u_2 - u_1),$$

where u_1 and u_2 are the axial displacements of the bar end nodes.

In matrix form, for local degrees of freedom $\{u\}_{\text{loc}} = \{u_1, u_2\}^{\mathsf{T}}$,

$$[k]_{\text{loc}} = \frac{EA}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}.$$

The transformation to global coordinates introduces direction cosines based on the nodal coordinates. For a 2D truss member with direction cosines c, s, the global element stiffness is

$$[k]_{\text{glob}} = \frac{EA}{L}\begin{bmatrix} c^2 & cs & -c^2 & -cs \\ cs & s^2 & -cs & -s^2 \\ -c^2 & -cs & c^2 & cs \\ -cs & -s^2 & cs & s^2 \end{bmatrix}.$$

The global equilibrium is written as

$$\mathbf{r}(\mathbf{U}) = \mathbf{F}_{\text{ext}} - \mathbf{F}_{\text{int}}(\mathbf{U}) = \mathbf{0},$$

where $\mathbf{F}_{\text{int}}$ depends nonlinearly on the current deformation state.

In this variant, geometric and/or material nonlinearities are taken into account. The internal force vector becomes a nonlinear function of the nodal displacements, and the global equilibrium is solved by iterative methods (e.g. Newton–Raphson) instead of a single linear solve.

Finite element formulation

The solver assembles element stiffness (and mass or geometric stiffness, when applicable) matrices from the bar properties and connectivity. For each element:

- nodal coordinates define the element length and orientation,
- material properties provide E (and density ρ in dynamic or modal analyses),
- section data provide the cross-sectional area A.

Element matrices are transformed from local to global coordinates using direction cosine matrices and then added into the global system according to the element connectivity array.

Boundary conditions and loading

Boundary conditions are applied as constraints on translational degrees of freedom at nodes:

- fixed supports: all translational DOFs constrained,
- roller or pin supports: one or more DOFs left free,
- free nodes: no kinematic constraints.

Loads are typically applied as:

- nodal forces at joints,
- equivalent nodal forces obtained from distributed loads when such options are present in higher-level drivers.

Prescribed displacements can also be enforced by modifying the global system matrix and right-hand side.

Input DSL structure

In the surrounding project DSL, an analysis using `truss2d_geom_nl` is configured with blocks that define materials, sections, nodes, elements, supports, and loads. A schematic example is:

```
solver truss2d_geom_nl {
  material steel {
    E   = 210e9
    rho = 7850
  }
```

```
  section bar {
    area = 0.003
  }

  nodes {
    n1 = (0.0, 0.0)
    n2 = (5.0, 0.0)
    n3 = (2.5, 3.0)
  }

  elements {
    e1: truss2d (n1, n3) material=steel section=bar
    e2: truss2d (n3, n2) material=steel section=bar
  }

  supports {
    n1.ux = 0.0
    n1.uy = 0.0
    n2.uy = 0.0
  }

  loads {
    n3.Fy = -50e3
  }
}
```

The exact syntax is governed by the DSL front-end, but the solver expects coherent references to nodal sets, element sets, and material/section data.

Output fields

Typical outputs of the truss solvers include:

- nodal displacements,
- element axial forces,
- element axial stresses $\sigma = N/A$,
- reaction forces at constrained nodes,
- in modal or buckling variants, eigenvalues and mode shapes.

These quantities can be post-processed to generate deformed shape plots, force diagrams, or tables for design checks.

Typical use cases

- analysis and design of planar and space truss structures,

- benchmark problems for verification of more advanced finite element formulations,
- parametric studies of stiffness, load paths, and member utilisation.

Limitations and notes

- The basic formulation assumes pin-jointed members carrying axial force only; bending and shear are neglected unless a more general frame element is used instead.
- Geometric and material nonlinearity are only included in specialised variants that explicitly implement them.
- Accurate results require a truss-like structural behaviour; using a truss model for frame-like structures can be misleading.

8.81 truss2d_linear

Documentation for solver `truss2d_linear`

Overview

The `truss2d_linear` solver implements a finite element formulation for truss structures. Members are idealised as straight bars that carry axial force only (no bending or shear), and nodes act as frictionless pins. In two- or three-dimensional space each node has translational degrees of freedom and elements are oriented arbitrarily according to their nodal coordinates.

Typical applications include planar and space truss structures in buildings, bridges, and lattice frameworks used in mechanical and civil engineering.

Governing equations

For a bar of length L, cross-sectional area A, and Young's modulus E, the axial force–displacement relation in local coordinates is

$$N = \frac{EA}{L}(u_2 - u_1),$$

where u_1 and u_2 are the axial displacements of the bar end nodes.

In matrix form, for local degrees of freedom $\{u\}_{\text{loc}} = \{u_1, u_2\}^{\mathsf{T}}$,

$$[k]_{\text{loc}} = \frac{EA}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}.$$

The transformation to global coordinates introduces direction cosines based on the nodal coordinates. For a 2D truss member with direction cosines c, s, the global element stiffness is

$$[k]_{\text{glob}} = \frac{EA}{L}\begin{bmatrix} c^2 & cs & -c^2 & -cs \\ cs & s^2 & -cs & -s^2 \\ -c^2 & -cs & c^2 & cs \\ -cs & -s^2 & cs & s^2 \end{bmatrix}.$$

At the global level the assembled linear system is

$$[K]\{U\} = \{F\},$$

with $[K]$ the global stiffness matrix, $\{U\}$ the nodal displacement vector, and $\{F\}$ the nodal load vector.

Finite element formulation

The solver assembles element stiffness (and mass or geometric stiffness, when applicable) matrices from the bar properties and connectivity. For each element:

- nodal coordinates define the element length and orientation,
- material properties provide E (and density ρ in dynamic or modal analyses),
- section data provide the cross-sectional area A.

Element matrices are transformed from local to global coordinates using direction cosine matrices and then added into the global system according to the element connectivity array.

Boundary conditions and loading

Boundary conditions are applied as constraints on translational degrees of freedom at nodes:

- fixed supports: all translational DOFs constrained,
- roller or pin supports: one or more DOFs left free,
- free nodes: no kinematic constraints.

Loads are typically applied as:

- nodal forces at joints,
- equivalent nodal forces obtained from distributed loads when such options are present in higher-level drivers.

Prescribed displacements can also be enforced by modifying the global system matrix and right-hand side.

Input DSL structure

In the surrounding project DSL, an analysis using `truss2d_linear` is configured with blocks that define materials, sections, nodes, elements, supports, and loads. A schematic example is:

```
solver truss2d_linear {
  material steel {
    E   = 210e9
    rho = 7850
  }

  section bar {
    area = 0.003
  }

  nodes {
    n1 = (0.0, 0.0)
    n2 = (5.0, 0.0)
    n3 = (2.5, 3.0)
  }

  elements {
    e1: truss2d (n1, n3) material=steel section=bar
    e2: truss2d (n3, n2) material=steel section=bar
  }

  supports {
    n1.ux = 0.0
    n1.uy = 0.0
    n2.uy = 0.0
  }

  loads {
    n3.Fy = -50e3
  }
}
```

The exact syntax is governed by the DSL front-end, but the solver expects coherent references to nodal sets, element sets, and material/section data.

Output fields

Typical outputs of the truss solvers include:

- nodal displacements,
- element axial forces,
- element axial stresses $\sigma = N/A$,
- reaction forces at constrained nodes,
- in modal or buckling variants, eigenvalues and mode shapes.

These quantities can be post-processed to generate deformed shape plots, force diagrams, or tables for design checks.

Typical use cases

- analysis and design of planar and space truss structures,
- benchmark problems for verification of more advanced finite element formulations,
- parametric studies of stiffness, load paths, and member utilisation.

Limitations and notes

- The basic formulation assumes pin-jointed members carrying axial force only; bending and shear are neglected unless a more general frame element is used instead.
- Geometric and material nonlinearity are only included in specialised variants that explicitly implement them.
- Accurate results require a truss-like structural behaviour; using a truss model for frame-like structures can be misleading.

8.82 truss2d_modal

Documentation for solver `truss2d_modal`

Overview

The `truss2d_modal` solver implements a finite element formulation for truss structures. Members are idealised as straight bars that carry axial force only (no bending or shear), and nodes act as frictionless pins. In two- or three-dimensional space each node has translational degrees of freedom and elements are oriented arbitrarily according to their nodal coordinates.

Typical applications include planar and space truss structures in buildings, bridges, and lattice frameworks used in mechanical and civil engineering.

Governing equations

For a bar of length L, cross-sectional area A, and Young's modulus E, the axial force–displacement relation in local coordinates is

$$N = \frac{EA}{L}(u_2 - u_1),$$

where u_1 and u_2 are the axial displacements of the bar end nodes.

In matrix form, for local degrees of freedom $\{u\}_{\text{loc}} = \{u_1, u_2\}^{\mathsf{T}}$,

$$[k]_{\text{loc}} = \frac{EA}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}.$$

The transformation to global coordinates introduces direction cosines based on the nodal coordinates. For a 2D truss member with direction cosines c, s, the global element

stiffness is

$$[k]_{\text{glob}} = \frac{EA}{L}\begin{bmatrix} c^2 & cs & -c^2 & -cs \\ cs & s^2 & -cs & -s^2 \\ -c^2 & -cs & c^2 & cs \\ -cs & -s^2 & cs & s^2 \end{bmatrix}.$$

For free vibration (modal) analysis the solver assembles the consistent mass matrix $[M]$ in addition to the stiffness matrix $[K]$ and solves the generalized eigenproblem

$$[K]\{\phi\} = \lambda[M]\{\phi\},$$

where $\lambda = \omega^2$ are the eigenvalues and ω the natural circular frequencies.

Finite element formulation

The solver assembles element stiffness (and mass or geometric stiffness, when applicable) matrices from the bar properties and connectivity. For each element:

- nodal coordinates define the element length and orientation,
- material properties provide E (and density ρ in dynamic or modal analyses),
- section data provide the cross-sectional area A.

Element matrices are transformed from local to global coordinates using direction cosine matrices and then added into the global system according to the element connectivity array.

Boundary conditions and loading

Boundary conditions are applied as constraints on translational degrees of freedom at nodes:

- fixed supports: all translational DOFs constrained,
- roller or pin supports: one or more DOFs left free,
- free nodes: no kinematic constraints.

Loads are typically applied as:

- nodal forces at joints,
- equivalent nodal forces obtained from distributed loads when such options are present in higher-level drivers.

Prescribed displacements can also be enforced by modifying the global system matrix and right-hand side.

Input DSL structure

In the surrounding project DSL, an analysis using `truss2d_modal` is configured with blocks that define materials, sections, nodes, elements, supports, and loads. A schematic example is:

```
solver truss2d_modal {
  material steel {
    E   = 210e9
    rho = 7850
  }

  section bar {
    area = 0.003
  }

  nodes {
    n1 = (0.0, 0.0)
    n2 = (5.0, 0.0)
    n3 = (2.5, 3.0)
  }

  elements {
    e1: truss2d (n1, n3) material=steel section=bar
    e2: truss2d (n3, n2) material=steel section=bar
  }

  supports {
    n1.ux = 0.0
    n1.uy = 0.0
    n2.uy = 0.0
  }

  loads {
    n3.Fy = -50e3
  }
}
```

The exact syntax is governed by the DSL front-end, but the solver expects coherent references to nodal sets, element sets, and material/section data.

Output fields

Typical outputs of the truss solvers include:

- nodal displacements,
- element axial forces,
- element axial stresses $\sigma = N/A$,

- reaction forces at constrained nodes,
- in modal or buckling variants, eigenvalues and mode shapes.

These quantities can be post-processed to generate deformed shape plots, force diagrams, or tables for design checks.

Typical use cases

- analysis and design of planar and space truss structures,
- benchmark problems for verification of more advanced finite element formulations,
- parametric studies of stiffness, load paths, and member utilisation.

Limitations and notes

- The basic formulation assumes pin-jointed members carrying axial force only; bending and shear are neglected unless a more general frame element is used instead.
- Geometric and material nonlinearity are only included in specialised variants that explicitly implement them.
- Accurate results require a truss-like structural behaviour; using a truss model for frame-like structures can be misleading.

8.83 truss2d_nonlinear

Documentation for solver `truss2d_nonlinear`

Overview

The `truss2d_nonlinear` solver implements a finite element formulation for truss structures. Members are idealised as straight bars that carry axial force only (no bending or shear), and nodes act as frictionless pins. In two- or three-dimensional space each node has translational degrees of freedom and elements are oriented arbitrarily according to their nodal coordinates.

Typical applications include planar and space truss structures in buildings, bridges, and lattice frameworks used in mechanical and civil engineering.

This particular variant extends the basic truss formulation to include nonlinear effects or specialised response types.

Governing equations

For a bar of length L, cross-sectional area A, and Young's modulus E, the axial force–displacement relation in local coordinates is

$$N = \frac{EA}{L}(u_2 - u_1),$$

where u_1 and u_2 are the axial displacements of the bar end nodes.

In matrix form, for local degrees of freedom $\{u\}_{\text{loc}} = \{u_1, u_2\}^{\mathsf{T}}$,

$$[k]_{\text{loc}} = \frac{EA}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}.$$

The transformation to global coordinates introduces direction cosines based on the nodal coordinates. For a 2D truss member with direction cosines c, s, the global element stiffness is

$$[k]_{\text{glob}} = \frac{EA}{L}\begin{bmatrix} c^2 & cs & -c^2 & -cs \\ cs & s^2 & -cs & -s^2 \\ -c^2 & -cs & c^2 & cs \\ -cs & -s^2 & cs & s^2 \end{bmatrix}.$$

The global equilibrium is written as

$$\mathbf{r}(\mathbf{U}) = \mathbf{F}_{\text{ext}} - \mathbf{F}_{\text{int}}(\mathbf{U}) = \mathbf{0},$$

where $\mathbf{F}_{\text{int}}$ depends nonlinearly on the current deformation state.

In this variant, geometric and/or material nonlinearities are taken into account. The internal force vector becomes a nonlinear function of the nodal displacements, and the global equilibrium is solved by iterative methods (e.g. Newton–Raphson) instead of a single linear solve.

Finite element formulation

The solver assembles element stiffness (and mass or geometric stiffness, when applicable) matrices from the bar properties and connectivity. For each element:

- nodal coordinates define the element length and orientation,
- material properties provide E (and density ρ in dynamic or modal analyses),
- section data provide the cross-sectional area A.

Element matrices are transformed from local to global coordinates using direction cosine matrices and then added into the global system according to the element connectivity array.

Boundary conditions and loading

Boundary conditions are applied as constraints on translational degrees of freedom at nodes:

- fixed supports: all translational DOFs constrained,
- roller or pin supports: one or more DOFs left free,
- free nodes: no kinematic constraints.

Loads are typically applied as:

- nodal forces at joints,
- equivalent nodal forces obtained from distributed loads when such options are present in higher-level drivers.

Prescribed displacements can also be enforced by modifying the global system matrix and right-hand side.

Input DSL structure

In the surrounding project DSL, an analysis using `truss2d_nonlinear` is configured with blocks that define materials, sections, nodes, elements, supports, and loads. A schematic example is:

```
solver truss2d_nonlinear {
  material steel {
    E   = 210e9
    rho = 7850
  }

  section bar {
    area = 0.003
  }

  nodes {
    n1 = (0.0, 0.0)
    n2 = (5.0, 0.0)
    n3 = (2.5, 3.0)
  }

  elements {
    e1: truss2d (n1, n3) material=steel section=bar
    e2: truss2d (n3, n2) material=steel section=bar
  }

  supports {
    n1.ux = 0.0
    n1.uy = 0.0
    n2.uy = 0.0
  }

  loads {
    n3.Fy = -50e3
  }
}
```

The exact syntax is governed by the DSL front-end, but the solver expects coherent references to nodal sets, element sets, and material/section data.

Output fields

Typical outputs of the truss solvers include:

- nodal displacements,
- element axial forces,
- element axial stresses $\sigma = N/A$,

- reaction forces at constrained nodes,
- in modal or buckling variants, eigenvalues and mode shapes.

These quantities can be post-processed to generate deformed shape plots, force diagrams, or tables for design checks.

Typical use cases

- analysis and design of planar and space truss structures,
- benchmark problems for verification of more advanced finite element formulations,
- parametric studies of stiffness, load paths, and member utilisation.

Limitations and notes

- The basic formulation assumes pin-jointed members carrying axial force only; bending and shear are neglected unless a more general frame element is used instead.
- Geometric and material nonlinearity are only included in specialised variants that explicitly implement them.
- Accurate results require a truss-like structural behaviour; using a truss model for frame-like structures can be misleading.

8.84 truss2d_static

Documentation for solver `truss2d_static`

Overview

The `truss2d_static` solver implements a finite element formulation for truss structures. Members are idealised as straight bars that carry axial force only (no bending or shear), and nodes act as frictionless pins. In two- or three-dimensional space each node has translational degrees of freedom and elements are oriented arbitrarily according to their nodal coordinates.

Typical applications include planar and space truss structures in buildings, bridges, and lattice frameworks used in mechanical and civil engineering.

Governing equations

For a bar of length L, cross-sectional area A, and Young's modulus E, the axial force–displacement relation in local coordinates is

$$N = \frac{EA}{L}(u_2 - u_1),$$

where u_1 and u_2 are the axial displacements of the bar end nodes.

In matrix form, for local degrees of freedom $\{u\}_{\text{loc}} = \{u_1, u_2\}^{\mathsf{T}}$,

$$[k]_{\text{loc}} = \frac{EA}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}.$$

The transformation to global coordinates introduces direction cosines based on the nodal coordinates. For a 2D truss member with direction cosines c, s, the global element stiffness is

$$[k]_{\text{glob}} = \frac{EA}{L}\begin{bmatrix} c^2 & cs & -c^2 & -cs \\ cs & s^2 & -cs & -s^2 \\ -c^2 & -cs & c^2 & cs \\ -cs & -s^2 & cs & s^2 \end{bmatrix}.$$

At the global level the assembled linear system is

$$[K]\{U\} = \{F\},$$

with $[K]$ the global stiffness matrix, $\{U\}$ the nodal displacement vector, and $\{F\}$ the nodal load vector.

Finite element formulation

The solver assembles element stiffness (and mass or geometric stiffness, when applicable) matrices from the bar properties and connectivity. For each element:

- nodal coordinates define the element length and orientation,
- material properties provide E (and density ρ in dynamic or modal analyses),
- section data provide the cross-sectional area A.

Element matrices are transformed from local to global coordinates using direction cosine matrices and then added into the global system according to the element connectivity array.

Boundary conditions and loading

Boundary conditions are applied as constraints on translational degrees of freedom at nodes:

- fixed supports: all translational DOFs constrained,
- roller or pin supports: one or more DOFs left free,
- free nodes: no kinematic constraints.

Loads are typically applied as:

- nodal forces at joints,
- equivalent nodal forces obtained from distributed loads when such options are present in higher-level drivers.

Prescribed displacements can also be enforced by modifying the global system matrix and right-hand side.

Input DSL structure

In the surrounding project DSL, an analysis using `truss2d_static` is configured with blocks that define materials, sections, nodes, elements, supports, and loads. A schematic example is:

```
solver truss2d_static {
  material steel {
    E   = 210e9
    rho = 7850
  }

  section bar {
    area = 0.003
  }

  nodes {
    n1 = (0.0, 0.0)
    n2 = (5.0, 0.0)
    n3 = (2.5, 3.0)
  }

  elements {
    e1: truss2d (n1, n3) material=steel section=bar
    e2: truss2d (n3, n2) material=steel section=bar
  }

  supports {
    n1.ux = 0.0
    n1.uy = 0.0
    n2.uy = 0.0
  }

  loads {
    n3.Fy = -50e3
  }
}
```

The exact syntax is governed by the DSL front-end, but the solver expects coherent references to nodal sets, element sets, and material/section data.

Output fields

Typical outputs of the truss solvers include:

- nodal displacements,
- element axial forces,
- element axial stresses $\sigma = N/A$,

- reaction forces at constrained nodes,
- in modal or buckling variants, eigenvalues and mode shapes.

These quantities can be post-processed to generate deformed shape plots, force diagrams, or tables for design checks.

Typical use cases

- analysis and design of planar and space truss structures,
- benchmark problems for verification of more advanced finite element formulations,
- parametric studies of stiffness, load paths, and member utilisation.

Limitations and notes

- The basic formulation assumes pin-jointed members carrying axial force only; bending and shear are neglected unless a more general frame element is used instead.
- Geometric and material nonlinearity are only included in specialised variants that explicitly implement them.
- Accurate results require a truss-like structural behaviour; using a truss model for frame-like structures can be misleading.

8.85 truss3d

Documentation for solver `truss3d`

Overview

The `truss3d` solver implements a finite element formulation for truss structures. Members are idealised as straight bars that carry axial force only (no bending or shear), and nodes act as frictionless pins. In two- or three-dimensional space each node has translational degrees of freedom and elements are oriented arbitrarily according to their nodal coordinates.

Typical applications include planar and space truss structures in buildings, bridges, and lattice frameworks used in mechanical and civil engineering.

Governing equations

For a bar of length L, cross-sectional area A, and Young's modulus E, the axial force–displacement relation in local coordinates is

$$N = \frac{EA}{L}(u_2 - u_1),$$

where u_1 and u_2 are the axial displacements of the bar end nodes.

In matrix form, for local degrees of freedom $\{u\}_{\text{loc}} = \{u_1, u_2\}^{\mathsf{T}}$,

$$[k]_{\text{loc}} = \frac{EA}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}.$$

The transformation to global coordinates introduces direction cosines based on the nodal coordinates. For a 2D truss member with direction cosines c, s, the global element stiffness is

$$[k]_{\text{glob}} = \frac{EA}{L}\begin{bmatrix} c^2 & cs & -c^2 & -cs \\ cs & s^2 & -cs & -s^2 \\ -c^2 & -cs & c^2 & cs \\ -cs & -s^2 & cs & s^2 \end{bmatrix}.$$

At the global level the assembled linear system is

$$[K]\{U\} = \{F\},$$

with $[K]$ the global stiffness matrix, $\{U\}$ the nodal displacement vector, and $\{F\}$ the nodal load vector.

Finite element formulation

The solver assembles element stiffness (and mass or geometric stiffness, when applicable) matrices from the bar properties and connectivity. For each element:

- nodal coordinates define the element length and orientation,
- material properties provide E (and density ρ in dynamic or modal analyses),
- section data provide the cross-sectional area A.

Element matrices are transformed from local to global coordinates using direction cosine matrices and then added into the global system according to the element connectivity array.

Boundary conditions and loading

Boundary conditions are applied as constraints on translational degrees of freedom at nodes:

- fixed supports: all translational DOFs constrained,
- roller or pin supports: one or more DOFs left free,
- free nodes: no kinematic constraints.

Loads are typically applied as:

- nodal forces at joints,
- equivalent nodal forces obtained from distributed loads when such options are present in higher-level drivers.

Prescribed displacements can also be enforced by modifying the global system matrix and right-hand side.

Input DSL structure

In the surrounding project DSL, an analysis using `truss3d` is configured with blocks that define materials, sections, nodes, elements, supports, and loads. A schematic example is:

```
solver truss3d {
  material steel {
    E   = 210e9
    rho = 7850
  }

  section bar {
    area = 0.003
  }

  nodes {
    n1 = (0.0, 0.0)
    n2 = (5.0, 0.0)
    n3 = (2.5, 3.0)
  }

  elements {
    e1: truss2d (n1, n3) material=steel section=bar
    e2: truss2d (n3, n2) material=steel section=bar
  }

  supports {
    n1.ux = 0.0
    n1.uy = 0.0
    n2.uy = 0.0
  }

  loads {
    n3.Fy = -50e3
  }
}
```

The exact syntax is governed by the DSL front-end, but the solver expects coherent references to nodal sets, element sets, and material/section data.

Output fields

Typical outputs of the truss solvers include:

- nodal displacements,
- element axial forces,
- element axial stresses $\sigma = N/A$,
- reaction forces at constrained nodes,

- in modal or buckling variants, eigenvalues and mode shapes.

These quantities can be post-processed to generate deformed shape plots, force diagrams, or tables for design checks.

Typical use cases

- analysis and design of planar and space truss structures,
- benchmark problems for verification of more advanced finite element formulations,
- parametric studies of stiffness, load paths, and member utilisation.

Limitations and notes

- The basic formulation assumes pin-jointed members carrying axial force only; bending and shear are neglected unless a more general frame element is used instead.
- Geometric and material nonlinearity are only included in specialised variants that explicitly implement them.
- Accurate results require a truss-like structural behaviour; using a truss model for frame-like structures can be misleading.

8.86 vtk_writer

Documentation for solver `vtk_writer`

Overview

The `vtk_writer` module provides supporting functionality used by the physics solvers. It does not correspond to an independent physical model, but instead offers utilities such as:

- boundary condition handling and parsing,
- input/output helpers for meshes and results,
- material or section data management,
- linear algebra wrappers or post-processing routines.

Role in the codebase

Rather than assembling and solving its own finite element model, `vtk_writer` is called by other solvers to perform common tasks and to avoid duplication of logic. The precise responsibilities can be seen from the module's public API (functions and classes).

Input and output

Inputs and outputs are specific to the helper functions provided and may include:

- dictionaries representing solver state or model data,
- arrays of nodal and elemental information,
- handles to files or streams for reading/writing results.

As such, this module is typically not invoked directly from the user-facing DSL, but rather used internally.

Appendix A

Appendix B: DSL Example Directory

This appendix maps every file under `examples/` to its description, required inputs, and expected outputs. Contributors can expand each entry into a full page with figures and result tables.

A.1 Structure

Each entry should include:

1. File path and PROBLEM code.
2. Summary diagram or mesh screenshot.
3. Key parameters and boundary conditions.
4. Sample report output (displacements, temperatures, etc.).

A.2 Example Entry Template

```
### examples/truss2d_simple.dsl
- Problem: "Solid : Truss2D : Static"
- Purpose: sanity check of 2-bar truss
- Inputs: nodes, elems, E, A, loads, fix
- Outputs: U, element_forces
- Notes: include plot of axial force distribution
```

A.3 Suggested Expansion Plan

- Group entries by domain (structural, thermal, multiphysics) and dedicate 1–2 pages per example.
- For flagship cases (Cook's membrane, fusion studies), add multi-figure walkthroughs showing IDE screenshots, DSL snippets, and solver outputs.
- Provide QR codes or hyperlinks to the IDE for quick loading of each example.

A.4 Future Enhancements

1. Add cross-links from each appendix entry to the corresponding sections in the solver chapters.
2. Include "expected runtime" and "complexity" indicators so users know which examples are suitable for quick tests vs. regression suites.
3. Provide script snippets for batch execution (e.g., running all structural examples via `python tools/smoke_examples.py -filter structural`).

A.5 Example Listing

Appendix B

Appendix A: Formula Reference

This appendix will eventually collect derivations, element matrices, and material models referenced throughout the manual. Use it to avoid duplicating long formulas inside the main chapters.

B.1 Element Matrices

B.1.1 Truss Elements

Provide full derivations of local/global stiffness matrices, transformation matrices, and expressions for axial stress/strain. Include symbolic examples for two-bar trusses and a worked numeric example with actual values substituted.

B.1.2 Beam and Frame Elements

Document cubic Hermitian shape functions, bending stiffness matrices, and shear deformation corrections. Suggest including plots of mode shapes for cantilever beams.

B.1.3 Solid Elements

List the $\mathbf{B}$ matrices for Hex8/Hex20 elements and constitutive matrices for isotropic, orthotropic, and thermoelastic materials. Future pages should add integration point coordinates and Jacobian determinants.

B.2 Integration Schemes

Summaries of Newmark parameters, damping ratios, and stability limits. Provide a table comparing explicit vs. implicit schemes and recommended timestep limits for common problems (frame vibration, transient heat).

B.3 Material Models

- Plasticity parameters for bilinear and multilinear models.
- Thermal coefficients (expansion, conductivity) for common alloys.

- Placeholder subsections for data-driven material models introduced in future releases.

Expand each bullet into full subsections with tables listing parameters (yield stress, hardening modulus, etc.). Encourage contributors to include derivations for conversion between engineering constants and tensor components.

examples/axisym_q4_ring.dsl

```
PROBLEM "Solid : Axisymmetric : Q4 : Static"
GIVEN
  nodes = [[0.5,0.0],[1.0,0.0],[1.0,1.0],[0.5,1.0]]
  elems = [[0,1,2,3]]
  E = 210e9
  nu = 0.3
  fix = [[0,"ur",0.0],[1,"ur",0.0]]
  loads = [[2,0.0,-1000.0]]
REPORT
  print U
```

examples/bar1d_static.dsl

```
PROBLEM "Solid : Bar1D : Static"
GIVEN
  L = 2.0
  E = 210e9
  A = 2.0e-4
  nel = 20
  p = 1000.0
  left = "fixed"
  right = "fixed"
REPORT
  print u
  export "bar_u.csv" x u
```

examples/bar1d_thermoelastic_example.dsl

```
PROBLEM "Solid : Bar1D : ThermoElastic : Static"
GIVEN
  nodes = [0.0, 1.0]
  elems = [[0,1]]
  E = 210e9
  A = 1.0e-4
  alpha = 1.2e-5
  T = [20.0, 120.0]
  Tref = 20.0
  fix = [[0,0.0]]
REPORT
  print U
```

examples/beam1d_modal_cantilever.dsl

```
PROBLEM "Solid : Beam1D : Modal"
```

```
GIVEN
  nodes = [0.0, 0.5, 1.0]
  elems = [[0,1],[1,2]]
  E = 210e9
  I = 8.0e-6
  rho = 7800.0
  A = 4.0e-4
  fix = [[0,"both",0.0]]
  nmodes = 3
REPORT
  print freq
```

examples/beam_buckling_pinned_pinned.dsl

```
PROBLEM "Solid : Beam : Buckling"
GIVEN
  L = 2.0
  E = 210e9
  I = 8.0e-6
  nel = 40
  Nref = 1.0
  left = "pinned"
  right = "pinned"
  nmodes = 3
REPORT
  print Pcr
```

examples/beam_modal_cantilever.dsl

```
PROBLEM "Solid : Beam : Modal"
GIVEN
  L = 2.0
  E = 210e9
  I = 8.0e-6
  rho = 7800.0
  A = 3.0e-4
  nel = 40
  left = "clamped"
  right = "free"
  nmodes = 4
REPORT
  print freq
```

examples/beam_plot_export.dsl

```
PROBLEM "Solid : Beam : Static"
GIVEN
```

```
  L = 2.0
  E = 210e9
  I = 8.0e-6
  nel = 40
  q = 1000.0
  left = "clamped"
  right = "clamped"
REPORT
  plot x w
  export "beam_w.csv" x w
```

examples/beam_static.dsl

```
PROBLEM "Solid : Beam : Static"
GIVEN
  L = 2.0
  E = 210e9
  I = 8.0e-6
  nel = 40
  q = 1000.0
  left = "clamped"
  right = "clamped"
REPORT
  print w
  print reactions
```

examples/contact1d_basic.dsl

```
PROBLEM "Contact : 1D : Penalty"
GIVEN
  L = 1.0
  E = 200e9
  A = 1.0e-4
  g = 0.001
  P = 1.0e5
  kpen = 1.0e9
REPORT
  print U
```

examples/contact1d_gap.dsl

```
PROBLEM "Solid : Contact1D : Static"
GIVEN
  nn = 3
  springs = [[0,1,1e6],[1,2,1e6]]
  supports = [[0,1]]
  loads = [[2, -5.0e2]]
```

```
  contact_gaps = [[1,0, 0.001, 1e9]]
REPORT
  export "contact1d_gap_U.csv" U
  print active_contacts
```

examples/cook_membrane.dsl

```
PROBLEM "Solid : PlaneStrain : LinearFE"
GIVEN
  Lx = 48.0e-3
  Ly = 44.0e-3
  t  = 1.0e-3
  nx = 24
  ny = 22
  E  = 70e9
  nu = 0.3
  traction_right_x = 1.0e6
  traction_right_y = 0.0
  clamp_left = True
REPORT
  export "cook_ps_linear.vtk" vtk
```

examples/dynamics_mck.dsl

```
PROBLEM "Dynamics : MCKSystem : Transient"
GIVEN
  M = [[2,0],[0,1]]
  C = [[0.02,0],[0,0.02]]
  K = [[2000,-1000],[-1000,1000]]
  u0 = [0,0]
  v0 = [0,0]
  dt = 0.001
  t_end = 0.5
  f = [0.0, 1.0]
REPORT
  print U
```

examples/eigen_generalized_2x2.dsl

```
PROBLEM "LinearAlgebra : Eigen : Generalized"
GIVEN
  K = [[2,0],[0,3]]
  M = [[1,0],[0,1]]
  nmodes = 2
REPORT
  print freq
```

examples/elas2d_plane_strain_rect.dsl

```
PROBLEM "Solid : Elasticity2D : PlaneStrain"
GIVEN
  Lx=1.0 Ly=0.5 nx=12 ny=6
  E=210e9 nu=0.3 t=1.0
  bx=0.0 by=-1000.0
  clamp_left=1
  tx_right=1e5
  ty_right=0.0
REPORT
  export "elas2d_plane_strain_U.csv" U
  export "elas2d_plane_strain_stress.csv" stress
```

examples/elas2d_plane_stress_thermal_rect.dsl

```
PROBLEM "Solid : Elasticity2D : PlaneStress"
GIVEN
  Lx=1.0 Ly=1.0 nx=10 ny=10
  E=70e9 nu=0.33 t=1.0
  alpha=1.2e-5
  dT=50.0
  clamp_left=1
REPORT
  export "elas2d_ps_thermal_U.csv" U
  export "elas2d_ps_thermal_stress.csv" stress
```

examples/elas2d_ps_edgeload_rect.dsl

```
PROBLEM "Solid : Elasticity2D : PlaneStressEdgeLoad"
GIVEN
  Lx = 2.0
  Ly = 1.0
  nx = 20
  ny = 10
  t = 0.01
  E = 210e9
  nu = 0.3
  clamp_left = 1
  # traction on right edge (downward 5kN/m)
  tx_right = 0.0
  ty_right = -5000.0
REPORT
  export "elas2d_ps_edgeload_U.csv" U
  export "elas2d_ps_edgeload_stress.csv" stress
```

examples/elas2d_ps_rect.dsl

```
PROBLEM "Solid : Elasticity2D : PlaneStress"
GIVEN
  Lx = 2.0
  Ly = 1.0
  nx = 12
  ny = 6
  t = 0.01
  E = 210e9
  nu = 0.3
  clamp_left = 1
  supports = []
  loads = [[(nx+1)*ny + nx, 0.0, -1000.0]]
REPORT
  export "elas2d_U.csv" U
  export "elas2d_stress.csv" stress
```

examples/elas2d_ps_train.dsl

```
PROBLEM "Solid : Elasticity2D : PlaneStrain"
GIVEN
  Lx = 1.0
  Ly = 1.0
  nx = 8
  ny = 8
  t = 1.0
  E = 70e9
  nu = 0.25
  clamp_left = 1
  loads = [[(nx+1)*ny + nx, 0.0, -1000.0]]
REPORT
  export "elas2d_ps_U.csv" U
  export "elas2d_ps_stress.csv" stress
```

examples/elastic3d_hex8.dsl

```
PROBLEM "Solid : Elastic3D : Linear"
GIVEN
  etype = "HEX8"
  nodes = [[0,0,0],[1,0,0],[1,1,0],[0,1,0],
           [0,0,1],[1,0,1],[1,1,1],[0,1,1]]
  elems = [[0,1,2,3,4,5,6,7]]
  E = 70e9
  nu = 0.33
  fix = [[0,'ux'],[0,'uy'],[0,'uz'],
```

```
        [3,'ux'],[3,'uy'],[3,'uz'],
        [4,'ux'],[4,'uy'],[4,'uz'],
        [7,'ux'],[7,'uy'],[7,'uz']]
  point_loads = [[6, 0, 0, 1e4]]
REPORT
  export "elastic3d_hex8.vtk" vtk
```

examples/elastic3d_tet4.dsl

```
PROBLEM "Solid : Elastic3D : Linear"
GIVEN
  etype = "TET4"
  nodes = [[0,0,0],[1,0,0],[0,1,0],[0,0,1]]
  elems = [[0,1,2,3]]
  E = 200e9
  nu = 0.28
  body = [0,0,-9.81]
  rho = 7800
  fix = [[0,'ux'],[0,'uy'],[0,'uz']]
REPORT
  export "elastic3d_tet4.vtk" vtk
```

examples/equations_linear.dsl

```
PROBLEM "LinearAlgebra : Equations : Direct"
EQUATIONS
  unknowns = ["u0","u1","u2"]
  eq " 2*u0 - 1*u1      =  5"
  eq "-1*u0 + 2*u1 - 1*u2 = -1"
  eq "        -1*u1 + 2*u2 =  1"
REPORT
  print x
  export "eqs_x.csv" x
```

examples/frame2d_beamcolumn_nonlinear.dsl

```
PROBLEM "Frame2D : BeamColumn : Nonlinear"
GIVEN
  nodes = [[0,0],[0,3]]
  elements = [[0,1]]
  E = 210e9
  A = 4e-4
  I = 8e-6
  fix = [[0,"ux",0.0],[0,"uy",0.0],[0,"rz",0.0]]
  loads = [[1,0.0,-2.0e3,0.0]]
  load_steps = 2
```

```
  max_iter = 10
REPORT
  print U
```

examples/frame2d_beamcolumn_static.dsl

```
PROBLEM "Frame2D : BeamColumn : Static"
GIVEN
  nodes = [[0,0],[0,3]]
  elements = [[0,1]]
  E = 210e9
  A = 4e-4
  I = 8e-6
  fix = [[0,"ux",0.0],[0,"uy",0.0],[0,"rz",0.0]]
  loads = [[1,0.0,-1000.0,0.0]]
REPORT
  print U
```

examples/frame2d_buckling_col.dsl

```
PROBLEM "Solid : Frame2D : Buckling"
GIVEN
  nodes = [[0,0],[3,0]]
  elems = [[0,1]]
  E = 210e9
  A = 4.0e-4
  I = 3.3e-6
  supports = [[0,1,1,1],[1,0,1,0]]
  N_axial = [1.0e5]
REPORT
  export "frame2d_buckling_lambda.csv" lambda_cr
```

examples/frame2d_buckling_L.dsl

```
PROBLEM "Solid : Frame2D : Buckling : Linear"
GIVEN
  nodes = [[0,0],[3,0],[3,3]]
  elems = [[0,1],[1,2]]
  E = 210e9
  A = 4e-4
  I = 8e-6
  Nref = [1.0e5, 2.0e5]
  fix = [[0,"both",0.0],[1,"uy",0.0]]
  nmodes = 3
REPORT
  print lambda_cr
```

examples/frame2d_cantilever.dsl

```
PROBLEM "Solid : Frame2D : Static"
GIVEN
  nodes = [[0,0],[2,0]]
  elems = [[0,1]]
  E = 210e9
  A = 4.0e-4
  I = 3.3e-6
  supports = [[0,1,1,1]]
  w_dist = [[0, -1000.0]]
REPORT
  export "frame2d_cantilever_U.csv" U
```

examples/frame2d_cantilever_udl.dsl

```
PROBLEM "Solid : Frame2D : Static"
GIVEN
  nodes = [[0,0],[4,0]]
  elems = [[0,1]]
  E = 210e9
  A = 4e-4
  I = 8e-6
  edist = [[0, 0.0, -1000.0]]
  fix = [[0,"both",0.0]]
REPORT
  print U
```

examples/frame2d_L_modal.dsl

```
PROBLEM "Solid : Frame2D : Modal"
GIVEN
  nodes = [[0,0],[3,0],[3,3]]
  elems = [[0,1],[1,2]]
  E = 210e9
  A = 4e-4
  I = 8e-6
  rho = 7850.0
  fix = [[0, "both", 0.0], [1, "uy", 0.0]]
  nmodes = 3
REPORT
  print freq
```

examples/frame2d_L_static.dsl

```
PROBLEM "Solid : Frame2D : Static"
```

```
GIVEN
  nodes = [[0,0],[3,0],[3,3]]
  elems = [[0,1],[1,2]]
  E = 210e9
  A = 4e-4
  I = 8e-6
  loads = [[2, 0.0, -10000.0, 0.0]]
  fix = [[0, "both", 0.0], [1, "uy", 0.0]]
REPORT
  print U
```

examples/frame2d_linear.dsl

```
PROBLEM "Solid : Frame2D : Static"
GIVEN
  nodes = [[0,0],[4,0],[4,3]]
  # elems: n1, n2, A, E, I, wloc(optional, local -y). Here no distributed load.
  elems = [[0,1, 3.2e-3, 210e9, 8.0e-6],
           [1,2, 3.2e-3, 210e9, 8.0e-6]]
  # supports: node, fix_u, fix_v, fix_theta
  supports = [[0,1,1,1],[2,0,1,0]]
  # nodal loads: node, Fx, Fy, Mz
  loads = [[1, 0.0, -10000.0, 0.0]]
REPORT
  print U
  export "frame_U.csv" U
```

examples/frame2d_loadstep.dsl

```
PROBLEM "Solid : Frame2D : StaticNL"
GIVEN
  nodes = [[0,0],[3,0],[3,3]]
  elems = [[0,1],[1,2]]
  E = 210e9
  A = 4.0e-4
  I = 3.3e-6
  supports = [[0,1,1,1],[2,0,1,0]]
  loads = [[1, 0.0, -2.0e4, 0.0]]
  CONTROLS = {"method":"loadstep","load_steps":20,"max_iter":30,"tol":1e-9,"line_searc
REPORT
  export "frame2d_loadstep_path.csv" path
  export "frame2d_loadstep_U.csv" U
```

examples/frame2d_modal_beam.dsl

```
PROBLEM "Solid : Frame2D : Modal"
GIVEN
```

```
  nodes = [[0,0],[3,0]]
  elems = [[0,1]]
  E = 210e9
  A = 4.0e-4
  I = 3.3e-6
  rho = 7850
  supports = [[0,1,1,1]]
REPORT
  export "frame2d_modal_freq.csv" freq
```

examples/frame2d_modal_consistent_L.dsl

```
PROBLEM "Solid : Frame2D : Modal : ConsistentMass"
GIVEN
  nodes = [[0,0],[3,0],[3,3]]
  elems = [[0,1],[1,2]]
  E = 210e9
  A = 4e-4
  I = 8e-6
  rho = 7850.0
  fix = [[0,"both",0.0],[1,"uy",0.0]]
  nmodes = 3
REPORT
  print freq
```

examples/frame2d_pdelta.dsl

```
PROBLEM "Solid : Frame2D : Static"
GIVEN
  nodes = [[0,0],[4,0],[4,3]]
  elems = [[0,1, 3.2e-3, 210e9, 8.0e-6, -2000.0],   # uniform downward local load -200
           [1,2, 3.2e-3, 210e9, 8.0e-6,  0.0]]
  supports = [[0,1,1,1],[2,0,1,0]]
  loads = [[1, 0.0, -15000.0, 0.0]]
  nonlinear = true
  max_iter = 50
  tol = 1e-9
REPORT
  print U
  export "frame_member_forces.csv" member_forces
```

examples/frame2d_pdelta_L.dsl

```
PROBLEM "Solid : Frame2D : StaticNonlinear : PDelta"
GIVEN
  nodes = [[0,0],[3,0],[3,3]]
  elems = [[0,1],[1,2]]
```

```
  E = 210e9
  A = 4e-4
  I = 8e-6
  N = [1.0e5, 2.0e5]  # compression positive
  loads = [[2, 0.0, -10000.0, 0.0]]
  fix = [[0, "both", 0.0], [1, "uy", 0.0]]
REPORT
  print U
```

examples/frame2d_pdelta_portal.dsl

```
PROBLEM "Solid : Frame2D : PDelta"
GIVEN
  nodes = [[0,0],[3,0],[0,3],[3,3]]
  elems = [[0,1],[0,2],[1,3]]
  E = 210e9
  A = 4.0e-4
  I = 8.0e-6
  udl = [[0, 0.0, 5000.0]]
  point_loads = [[3, 0.0, -30000.0, 0.0]]
  fix = [[0,'ux'],[0,'uy'],[0,'rz'], [1,'ux'],[1,'uy'],[1,'rz']]
  max_iter = 40
  tol = 1.0e-9
REPORT
  print U
```

examples/frame2d_portal.dsl

```
PROBLEM "Solid : Frame2D : Static"
GIVEN
  nodes = [[0,0],[3,0],[0,3],[3,3]]
  elems = [[0,1],[0,2],[1,3]]
  E = 210e9
  A = 4.0e-4
  I = 8.0e-6
  point_loads = [[3, 0.0, -20000.0, 0.0]]
  fix = [[0,'ux'],[0,'uy'],[0,'rz'], [1,'ux'],[1,'uy'],[1,'rz']]
REPORT
  export "frame2d_portal.vtk" vtk
```

examples/frame2d_portal_point.dsl

```
PROBLEM "Solid : Frame2D : Linear"
GIVEN
```

```
  nodes = [[0,0],[0,3],[4,3]]
  elems = [[0,1],[1,2]]
  E = 200e9
  A = 1.0e-3
  I = 2.0e-6
  point_loads = [[2, 0.0, -1000.0, 0.0]]
  fix = [[0,'ux'],[0,'uy'],[0,'rz']]
REPORT
  print U
```

examples/frame2d_portal_udl.dsl

```
PROBLEM "Solid : Frame2D : Static"
GIVEN
  nodes = [[0,0],[3,0],[0,3],[3,3]]
  elems = [[0,1],[0,2],[1,3]]
  E = 210e9
  A = 4.0e-4
  I = 8.0e-6
  udl = [[0, 0.0, 5000.0]]
  point_loads = [[3, 0.0, -20000.0, 0.0]]
  fix = [[0,'ux'],[0,'uy'],[0,'rz'], [1,'ux'],[1,'uy'],[1,'rz']]
REPORT
  export "frame2d_portal.vtk" vtk
```

examples/frame2d_postbuckling.dsl

```
PROBLEM "Solid : Frame2D : StaticNL"
GIVEN
  nodes = [[0,0],[0,3]]
  elems = [[0,1]]
  E = 210e9
  A = 6.0e-4
  I = 2.0e-6
  supports = [[0,1,1,1],[1,0,1,0]]
  loads = [[1, 0.0, -1.0e4, 0.0]]
  CONTROLS = {"method":"arc","load_steps":50,"arc_len":5e-4,"max_iter":30,"tol":1e-8,
REPORT
  export "frame2d_postbuckling_path.csv" path
  export "frame2d_postbuckling_U.csv" U
```

examples/frame2d_staticnl_portal.dsl

```
PROBLEM "Solid : Frame2D : StaticNL"
GIVEN
  nodes = [[0,0],[3,0],[3,3]]
```

```
  elems = [[0,1],[1,2]]
  E = 210e9
  A = 4.0e-4
  I = 3.3e-6
  supports = [[0,1,1,1],[2,0,1,0]]
  loads = [[1, 0.0, -2.0e4, 0.0]]
REPORT
  export "frame2d_staticnl_U.csv" U
  export "frame2d_staticnl_Reac.csv" reactions
```

examples/frame2d_transient_impulse.dsl

```
PROBLEM "Solid : Frame2D : Transient"
GIVEN
  nodes = [[0,0],[3,0]]
  elems = [[0,1]]
  E = 210e9
  A = 4.0e-4
  I = 3.3e-6
  rho = 7850
  supports = [[0,1,1,1],[1,0,1,0]]
  dt=0.002 t_end=0.2 beta=0.25 gamma=0.5
  rayleigh_a0=0.0 rayleigh_a1=0.0005
  loads_t = [[0.0, 1, 0, -1e3, 0],
             [0.02,1, 0, -1e3, 0],
             [0.021,1, 0, 0,    0],
             [0.2, 1, 0, 0,    0]]
REPORT
  export "frame2d_transient_Uhist.npy" U_hist
```

examples/frame3d_beamcolumn_modal.dsl

```
PROBLEM "Frame3D : BeamColumn : Modal"
GIVEN
  nodes = [[0,0,0],[0,0,3]]
  elements = [[0,1]]
  E = 210e9
  G = 80e9
  A = 4e-4
  Iy = 8e-6
  Iz = 8e-6
  J = 1e-5
  rho = 7800.0
  nmodes = 2
  bcs = [
    {"node":0,"dof":"ux","value":0.0},
    {"node":0,"dof":"uy","value":0.0},
```

```
    {"node":0,"dof":"uz","value":0.0},
    {"node":0,"dof":"rx","value":0.0},
    {"node":0,"dof":"ry","value":0.0},
    {"node":0,"dof":"rz","value":0.0}
  ]
REPORT
  print freq
```

examples/frame3d_beamcolumn_static.dsl

```
PROBLEM "Frame3D : BeamColumn : Static"
GIVEN
  nodes = [[0,0,0],[0,0,3]]
  elements = [[0,1]]
  E = 210e9
  G = 80e9
  A = 4e-4
  Iy = 8e-6
  Iz = 8e-6
  J = 1e-5
  bcs = [
    {"node":0,"dof":"ux","value":0.0},
    {"node":0,"dof":"uy","value":0.0},
    {"node":0,"dof":"uz","value":0.0},
    {"node":0,"dof":"rx","value":0.0},
    {"node":0,"dof":"ry","value":0.0},
    {"node":0,"dof":"rz","value":0.0}
  ]
  loads = [
    {"node":1,"dof":"uy","value":-1000.0}
  ]
REPORT
  print u
```

examples/heat1d_bar.dsl

```
PROBLEM "Heat : OneD : Linear : Transient"
GIVEN
  nodes = [0.0, 0.5, 1.0]
  elems = [[0,1],[1,2]]
  k = 10.0
  rho = 7800.0
  cp = 500.0
  dt = 0.1
  t_end = 1.0
  T0 = 0.0
  fixT = [[0,100.0]]
```

```
REPORT
  print times
```

examples/heat1d_bar_transient.dsl

```
PROBLEM "Heat : Conduction1D : Linear : Transient"
GIVEN
  nodes = [0.0, 0.5, 1.0]
  elems = [[0,1],[1,2]]
  k = 20.0
  rho = 7800.0
  cp = 500.0
  dt = 0.1
  t_end = 1.0
  theta = 1.0
  hedge = [[1,1,10.0,300.0]]
  fix = [[0, 400.0]]
REPORT
  print T_hist
```

examples/heat1d_plot.dsl

```
PROBLEM "Thermal : Heat1D : Steady"
GIVEN
  L = 1.0
  k = 200.0
  A = 1.0
  qdot = 0.0
  nel = 20
  left = "Dirichlet"; T_left = 100.0
  right = "Robin"; h_right = 10.0; Tinf_right = 20.0
REPORT
  plot x T
  export "heat_T.csv" x T
```

examples/heat1d_steady.dsl

```
PROBLEM "Thermal : Heat1D : Steady"
GIVEN
  L = 1.0
  k = 200.0
  A = 1.0
  qdot = 0.0
  nel = 10
  left = "Dirichlet"; T_left = 100.0
  right = "Robin"; h_right = 10.0; Tinf_right = 20.0
REPORT
```

```
  print T
```

examples/heat2d_q4_block.dsl

```
PROBLEM "Heat : Conduction2D : Linear : Steady"
GIVEN
  etype = "Q4"
  nodes = [[0,0],[1,0],[1,1],[0,1]]
  elems = [[0,1,2,3]]
  k = 20.0
  q = 1e5
  hedge = [[0,1,10.0,300.0]]
  fix = [[0,300.0],[3,300.0]]
REPORT
  print T
```

examples/heat2d_q4_plate.dsl

```
PROBLEM "Heat : TwoD : Q4 : Static"
GIVEN
  nodes = [[0,0],[1,0],[1,1],[0,1]]
  elems = [[0,1,2,3]]
  k = 10.0
  q = 1000.0
  fixT = [[0,0.0],[1,0.0]]
REPORT
  print T
```

examples/heat2d_q4_transient.dsl

```
PROBLEM "Heat : Conduction2D : Linear : Transient"
GIVEN
  etype = "Q4"
  nodes = [[0,0],[1,0],[1,1],[0,1]]
  elems = [[0,1,2,3]]
  k = 35.0
  rho = 7800.0
  cp = 500.0
  dt = 0.1
  t_end = 1.0
  theta = 0.5
  hedge = [[0,1,20.0,300.0]]
  fix = [[0,300.0],[3,300.0]]
REPORT
  print T_hist
```

examples/heat2d_rect_steady.dsl

```
PROBLEM "Thermal : Heat2D : Steady"
GIVEN
  Lx = 1.0
  Ly = 1.0
  nx = 10
  ny = 10
  k = 10.0
  t = 1.0
  q = 0.0
  T_left = 100.0
  T_right = 0.0
  T_bottom = 0.0
  T_top = 0.0
REPORT
  export "heat2d_T.csv" nodes[:,0] nodes[:,1] T
```

examples/heat2d_square_transient.dsl

```
PROBLEM "Heat : Planar : Linear : Transient"
GIVEN
  nodes = [[0,0],[1,0],[1,1],[0,1]]
  elems = [[0,1,2,3]]
  k = 20.0
  rho = 7800.0
  cp = 500.0
  dt = 0.1
  t_end = 1.0
  theta = 1.0
  hedge = [[0,1,10.0,300.0]]
  fix = [[3,400.0]]
REPORT
  print T_hist
```

examples/heat2d_steadybc_rect.dsl

```
PROBLEM "Thermal : Heat2D : SteadyBC"
GIVEN
  Lx = 1.0
  Ly = 1.0
  nx = 16
  ny = 16
  k = 15.0
  t = 1.0
  q = 0.0
  # Dirichlet
```

```
  T_left = 100.0
  # Convection on right edge
  h_right = 25.0
  Tinf_right = 20.0
  # Insulated top/bottom (  )
REPORT
  export "heat2d_steadybc_T.csv" nodes[:,0] nodes[:,1] T
```

examples/heat2d_t3_patch.dsl

```
PROBLEM "Heat : Conduction2D : Linear : Steady"
GIVEN
  etype = "T3"
  nodes = [[0,0],[2,0],[2,1],[0,1]]
  elems = [[0,1,2],[0,2,3]]
  k = 45.0
  qedge = [[0,1,1000.0]]
  fix = [[0,350.0],[3,350.0]]
REPORT
  print T
```

examples/heat2d_t3_transient.dsl

```
PROBLEM "Heat : Conduction2D : Linear : Transient"
GIVEN
  etype = "T3"
  nodes = [[0,0],[2,0],[2,1],[0,1]]
  elems = [[0,1,2],[0,2,3]]
  k = 15.0
  rho = 8900.0
  cp = 385.0
  dt = 0.05
  t_end = 0.5
  theta = 1.0
  qedge = [[0,1,1000.0]]
  fix = [[0,300.0],[3,300.0]]
REPORT
  print T_hist
```

examples/heat2d_transient_bc_rect.dsl

```
PROBLEM "Thermal : Heat2D : TransientBC"
GIVEN
  Lx = 1.0
  Ly = 1.0
```

```
  nx = 12
  ny = 12
  rho = 7800.0
  c = 500.0
  k = 45.0
  t = 0.01
  q = 0.0
  dt = 0.1
  t_end = 2.0
  theta = 0.5
  T_init = 20.0
  # Dirichlet
  T_left = 100.0
  # Robin    h(t)   10  40
  h_right_series = [[0.0, 10.0], [2.0, 40.0]]
  Tinf_right = 20.0
REPORT
  export "heat2d_transientBC_Tend.csv" T[-1]
```

examples/heat2d_transient_conv_square.dsl

```
PROBLEM "Solid : Heat2D : TransientConv"
GIVEN
  Lx=1.0 Ly=1.0 nx=20 ny=20
  kx=45 ky=45 rho=7800 cp=500
  dt=0.02 t_end=0.5 theta=1.0
  T0=100.0
  qdot=2.0e5
  h_left=50  Tinf_left=20
  h_right=50 Tinf_right=20
  h_bottom=50 Tinf_bottom=20
  h_top=50   Tinf_top=20
REPORT
  export "heat2d_conv_Thist.npy" T_hist
```

examples/heat2d_transient_q4.dsl

```
PROBLEM "Heat : Conduction2D : Transient"
GIVEN
  etype = "Q4"
  nodes = [[0,0],[1,0],[1,1],[0,1]]
  elems = [[0,1,2,3]]
  k = 45.0
  rho = 7800
  c = 500
  t = 0.01
```

```
  dt = 0.05
  t_end = 0.5
  theta = 0.5
  fixT = [[0,100],[3,100],[1,0],[2,0]]
  store_every = 1
REPORT
  export "heat2d_transient_q4.vtk" vtk
```

examples/heat2d_transient_rect.dsl

```
PROBLEM "Thermal : Heat2D : Transient"
GIVEN
  Lx = 1.0
  Ly = 1.0
  nx = 10
  ny = 10
  rho = 7800.0
  c = 500.0
  k = 45.0
  t = 0.01
  q = 0.0
  dt = 0.05
  t_end = 1.0
  theta = 0.5
  T_init = 20.0
  T_left = 100.0
  T_right = 20.0
REPORT
  export "heat2d_transient_Tend.csv" T[-1]
```

examples/heat2d_transient_square.dsl

```
PROBLEM "Solid : Heat2D : Transient"
GIVEN
  Lx=1.0 Ly=1.0 nx=20 ny=20
  kx=45 ky=45 rho=7800 cp=500
  dt=0.02 t_end=0.5 theta=1.0
  T0=100.0
  T_left=0.0
  T_right=0.0
  T_bottom=0.0
  T_top=0.0
REPORT
  export "heat2d_T_final.csv" T
```

examples/heat2d_transient_t3.dsl

```
PROBLEM "Heat : Conduction2D : Transient"
GIVEN
  etype = "T3"
  nodes = [[0,0],[1,0],[1,1],[0,1]]
  elems = [[0,1,2],[0,2,3]]
  k = 15.0
  rho = 2700
  c = 900
  t = 0.005
  dt = 0.02
  t_end = 0.2
  q = 1e5
  qedge = [[0,1,2000]]
  fixT = [[0,20]]
  T0 = 20
  store_every = 1
REPORT
  export "heat2d_transient_t3.vtk" vtk
```

examples/heat3d_cube.dsl

```
PROBLEM "Heat : ThreeD : H8 : Transient"
GIVEN
  nodes = [[0,0,0],[1,0,0],[1,1,0],[0,1,0],[0,0,1],[1,0,1],[1,1,1],[0,1,1]]
  elems = [[0,1,2,3,4,5,6,7]]
  k = 10.0
  rho = 7800.0
  cp = 500.0
  dt = 0.1
  t_end = 1.0
  T0 = 0.0
  fixT = [[0,100.0]]
REPORT
  print times
```

examples/heat3d_transient_hex8.dsl

```
PROBLEM "Heat : Conduction3D : Transient"
GIVEN
  etype = "HEX8"
  nodes = [[0,0,0],[1,0,0],[1,1,0],[0,1,0],[0,0,1],[1,0,1],[1,1,1],[0,1,1]]
  elems = [[0,1,2,3,4,5,6,7]]
  k = 45.0
```

```
  rho = 7800
  c = 500
  dt = 0.05
  t_end = 0.5
  fixT = [[0,100],[3,100],[4,100],[7,100],[1,0],[2,0],[5,0],[6,0]]
  store_every = 1
REPORT
  export "heat3d_transient_hex8.vtk" vtk
```

examples/heat3d_transient_tet4.dsl

```
PROBLEM "Heat : Conduction3D : Transient"
GIVEN
  etype = "TET4"
  nodes = [[0,0,0],[1,0,0],[0,1,0],[0,0,1]]
  elems = [[0,1,2,3]]
  k = 15.0
  rho = 2700
  c = 900
  dt = 0.02
  t_end = 0.2
  q = 1e5
  qface = [[0,0,2000]]
  fixT = [[0,20]]
  T0 = 20
  store_every = 1
REPORT
  export "heat3d_transient_tet4.vtk" vtk
```

examples/heat_axi_transient_ring.dsl

```
PROBLEM "Heat : Axisymmetric : Linear : Transient"
GIVEN
  nodes = [[0.1,0],[0.2,0],[0.2,0.05],[0.1,0.05]]
  elems = [[0,1,2,3]]
  k = 20.0
  rho = 7800.0
  cp = 500.0
  dt = 0.1
  t_end = 1.0
  theta = 1.0
  hedge = [[0,1,10.0,300.0]]
  fix = [[0,300.0],[3,300.0]]
REPORT
  print T_hist
```

examples/help_in_dsl.dsl

```
PROBLEM "Solid : Beam : Static"
GIVEN
  L = 1.0
  E = 200e9
  I = 1.0e-6
  nel = 10
REPORT
  print help("Solid : Beam : Static")
```

examples/linear_system.dsl

```
PROBLEM "LinearAlgebra : AxEqualsb : Direct"
GIVEN
  A = [[3, -1, 0],
       [-1, 2, -1],
       [0, -1, 3]]
  b = [2, -2, 2]
SOLVE
  method = "LU"
REPORT
  print x
```

examples/plane2d_q4_modal_block.dsl

```
PROBLEM "Solid : Plane2D : Q4 : Modal"
GIVEN
  nodes = [[0,0],[1,0],[1,1],[0,1]]
  elems = [[0,1,2,3]]
  E = 210e9
  nu = 0.3
  rho = 7800.0
  t = 0.1
  plane = "stress"
  fix = [[0,"both",0.0],[3,"both",0.0]]
  nmodes = 3
REPORT
  print freq
```

examples/plane2d_q4_static_block.dsl

```
PROBLEM "Solid : Plane2D : Q4 : Static"
GIVEN
  nodes = [[0,0],[1,0],[1,1],[0,1]]
  elems = [[0,1,2,3]]
```

```
  E = 210e9
  nu = 0.3
  t = 0.1
  plane = "stress"
  fix = [[0,"both",0.0],[3,"both",0.0]]
  loads = [[1,1000.0,0.0],[2,1000.0,0.0]]
REPORT
  print U
```

examples/plane2d_q4_thermoelastic_block.dsl

```
PROBLEM "Solid : Plane2D : Q4 : ThermoElastic"
GIVEN
  nodes = [[0,0],[1,0],[1,1],[0,1]]
  elems = [[0,1,2,3]]
  E = 210e9
  nu = 0.3
  alpha = 1.2e-5
  T = [20.0,120.0,120.0,20.0]
  Tref = 20.0
  t = 0.1
  plane = "stress"
  fix = [[0,"both",0.0],[3,"both",0.0]]
REPORT
  print U
```

examples/plane_strain_plate.dsl

```
PROBLEM "Solid : PlaneStrain : LinearFE"
GIVEN
  Lx = 48.0e-3
  Ly = 44.0e-3
  t  = 1.0e-3
  nx = 12
  ny = 11
  E  = 70e9
  nu = 0.33
  traction_right_x = 1.0e6
  traction_right_y = 0.0
  clamp_left = True
REPORT
  export "ps_linear_result.vtk" vtk
  export "ps_linear_vm.csv" sigma_vm
```

examples/plane_strain_q4.dsl

```
PROBLEM "Solid : PlaneStrain2D : Linear"
GIVEN
  etype = "Q4"
  nodes = [[0,0],[2,0],[2,1],[0,1]]
  elems = [[0,1,2,3]]
  E = 210e9
  nu = 0.30
  t = 0.1
  traction = [[0, 2, 0.0, 1e6]]
  fix = [[0,'ux'],[0,'uy'],[3,'ux'],[3,'uy']]
REPORT
  export "plane_strain_q4.vtk" vtk
```

examples/plane_strain_q4_block.dsl

```
PROBLEM "Solid : PlaneStrain : Q4 : Static"
GIVEN
  nodes = [[0,0],[2,0],[2,1],[0,1]]
  elems = [[0,1,2,3]]
  E = 30e9
  nu = 0.25
  fix = [[0,"both",0.0],[3,"both",0.0]]
REPORT
  print U
```

examples/plane_strain_t3.dsl

```
PROBLEM "Solid : PlaneStrain2D : Linear"
GIVEN
  etype = "T3"
  nodes = [[0,0],[2,0],[2,1],[0,1]]
  elems = [[0,1,2],[0,2,3]]
  E = 70e9
  nu = 0.33
  t = 0.05
  traction = [[0, 1, 0.0, 5e5]]
  fix = [[0,'ux'],[0,'uy'],[3,'ux'],[3,'uy']]
REPORT
  export "plane_strain_t3.vtk" vtk
```

examples/plane_stress_q4_cantilever.dsl

```
PROBLEM "Solid : PlaneStress : Q4 : Static"
```

```
GIVEN
  nodes = [[0,0],[1,0],[1,1],[0,1]]
  elems = [[0,1,2,3]]
  E = 210e9
  nu = 0.3
  bf = [0, -1000]
  fix = [[0,"both",0.0],[3,"both",0.0]]
REPORT
  print U
```

examples/planestress_q4_plate.dsl

```
PROBLEM "Solid : PlaneStress2D : Linear"
GIVEN
  etype = "Q4"
  nodes = [[0,0],[1,0],[1,1],[0,1]]
  elems = [[0,1,2,3]]
  E = 210e9
  nu = 0.3
  t = 0.01
  tedge = [[0,1,0.0,-1e6]]
  fix = [[0,'ux',0.0],[0,'uy',0.0],[3,'ux',0.0],[3,'uy',0.0]]
REPORT
  print U
```

examples/planestress_t3_cantilever.dsl

```
PROBLEM "Solid : PlaneStress2D : Linear"
GIVEN
  etype = "T3"
  nodes = [[0,0],[2,0],[2,1],[1,1],[0,1]]
  elems = [[0,1,3],[0,3,4],[1,2,3]]
  E = 70e9
  nu = 0.33
  t = 0.02
  b = [0.0, -1000.0]
  fix = [[0,'ux',0],[0,'uy',0],[4,'ux',0],[4,'uy',0]]
REPORT
  print U
```

examples/plastic_tension.dsl

```
PROBLEM "Materials : VonMises : Uniaxial"
GIVEN
  E = 210e9
```

```
  nu = 0.3
  sigma_y0 = 250e6
  H = 1.0e9
  eps_max = 0.02
  nsteps = 200
REPORT
  export "uniaxial_eps.csv" eps
  export "uniaxial_sigma.csv" sigma
  export "uniaxial_epsp.csv" eps_p
  export "uniaxial_Et.csv" Et
```

examples/plate2d_mindlin_modal_block.dsl

```
PROBLEM "Solid : Plate2D : Mindlin : Modal"
GIVEN
  nodes = [[0,0],[1,0],[1,1],[0,1]]
  elems = [[0,1,2,3]]
  E = 210e9
  nu = 0.3
  t = 0.01
  rho = 7800.0
  kappa = 0.8333
  fix = [[0,"all",0.0],[1,"all",0.0],[3,"all",0.0]]
  nmodes = 3
REPORT
  print freq
```

examples/plate2d_mindlin_modal_square.dsl

```
PROBLEM "Solid : Plate2D : Mindlin : Modal"
GIVEN
  nodes = [[0,0],[1,0],[1,1],[0,1]]
  elems = [[0,1,2,3]]
  E = 210e9
  nu = 0.3
  t = 0.01
  rho = 7800.0
  kappa = 0.8333333
  fix = [[0,"both",0.0],[1,"both",0.0],[2,"both",0.0],[3,"both",0.0]]
  nmodes = 6
REPORT
  print freq
```

examples/plate2d_mindlin_square.dsl

```
PROBLEM "Solid : Plate2D : Mindlin : Static"
GIVEN
```

```
  nodes = [[0,0],[1,0],[1,1],[0,1]]
  elems = [[0,1,2,3]]
  E = 210e9
  nu = 0.3
  t = 0.01
  p = 1000.0
  fix = [[0,"both",0.0],[1,"both",0.0],[2,"both",0.0],[3,"both",0.0]]
REPORT
  print U
```

examples/plate2d_mindlin_static_block.dsl

```
PROBLEM "Solid : Plate2D : Mindlin : Static"
GIVEN
  nodes = [[0,0],[1,0],[1,1],[0,1]]
  elems = [[0,1,2,3]]
  E = 210e9
  nu = 0.3
  t = 0.01
  kappa = 0.8333
  fix = [[0,"all",0.0],[1,"all",0.0],[3,"all",0.0]]
  loads = [[2,-1000.0]]
REPORT
  print U
```

examples/plate_bending_q4.dsl

```
PROBLEM "Solid : PlateMindlin2D : Linear"
GIVEN
  nodes = [[0,0],[1,0],[1,1],[0,1]]
  elems = [[0,1,2,3]]
  E = 210e9
  nu = 0.30
  t = 0.01
  kappa = 0.8333333333
  q = 1e4
  fixW = [[0,0],[1,0],[2,0],[3,0]]
  fixRot = [[0,'rx',0],[0,'ry',0],[1,'rx',0],[1,'ry',0],[2,'rx',0],[2,'ry',0],[3,'rx'
REPORT
  export "plate_bending_q4.vtk" vtk
```

examples/plate_heat_q4.dsl

```
PROBLEM "Heat : Conduction2D : Steady"
GIVEN
```

```
  etype = "Q4"
  nodes = [[0,0],[1,0],[1,1],[0,1]]
  elems = [[0,1,2,3]]
  k = 45.0
  t = 0.01
  fixT = [[0,100],[3,100],[1,0],[2,0]]
REPORT
  export "plate_heat_q4.vtk" vtk
```

examples/plate_heat_t3.dsl

```
PROBLEM "Heat : Conduction2D : Steady"
GIVEN
  etype = "T3"
  nodes = [[0,0],[1,0],[1,1],[0,1]]
  elems = [[0,1,2],[0,2,3]]
  k = 20.0
  t = 0.02
  qedge = [[0, 1, 1000.0]]
  fixT = [[0,0],[1,0]]
REPORT
  export "plate_heat_t3.vtk" vtk
```

examples/plate_mindlin_rect.dsl

```
PROBLEM "Solid : PlateMindlin : Static"
GIVEN
  Lx = 1.0
  Ly = 1.0
  nx = 12
  ny = 12
  t = 0.02
  E = 210e9
  nu = 0.3
  q = 1000.0     # N/m^2
  clamp_left = 1
  clamp_right = 1
  clamp_bottom = 1
  clamp_top = 1
REPORT
  export "plate_mindlin_U.csv" U
```

examples/plate_mindlin_winkler_rect.dsl

```
PROBLEM "Solid : PlateMindlin : Winkler"
GIVEN
```

```
  Lx = 1.0
  Ly = 1.0
  nx = 12
  ny = 12
  t = 0.02
  E = 210e9
  nu = 0.3
  q = 1000.0
  kw = 1.0e6   # N/m^3
  clamp_left = 1
  clamp_right = 1
  clamp_bottom = 1
  clamp_top = 1
REPORT
  export "plate_mindlin_winkler_U.csv" U
```

examples/plate_q4.dsl

```
PROBLEM "Solid : PlaneStress2D : Linear"
GIVEN
  etype = "Q4"
  nodes = [[0,0],[1,0],[1,1],[0,1]]
  elems = [[0,1,2,3]]
  E = 210e9
  nu = 0.30
  t = 0.01
  point_loads = [[2, 0.0, -1000.0]]
  fix = [[0,'ux'],[0,'uy'],[3,'ux'],[3,'uy']]
REPORT
  export "plate_q4.vtk" vtk
```

examples/plate_q4_traction.dsl

```
PROBLEM "Solid : PlaneStress2D : Linear"
GIVEN
  etype = "Q4"
  nodes = [[0,0],[2,0],[2,1],[0,1]]
  elems = [[0,1,2,3]]
  E = 210e9
  nu = 0.30
  t = 0.01
  traction = [[0, 2, 0.0, 500.0]]
  fix = [[0,'ux'],[0,'uy'],[3,'ux'],[3,'uy']]
REPORT
  export "plate_q4_traction.vtk" vtk
```

examples/plate_t3.dsl

```
PROBLEM "Solid : PlaneStress2D : Linear"
GIVEN
  etype = "T3"
  nodes = [[0,0],[1,0],[1,1],[0,1]]
  elems = [[0,1,2],[0,2,3]]
  E = 210e9
  nu = 0.30
  t = 0.01
  point_loads = [[2, 0.0, -1000.0]]
  fix = [[0,'ux'],[0,'uy'],[3,'ux'],[3,'uy']]
REPORT
  export "plate_t3.vtk" vtk
```

examples/plate_t3_traction.dsl

```
PROBLEM "Solid : PlaneStress2D : Linear"
GIVEN
  etype = "T3"
  nodes = [[0,0],[2,0],[2,1],[0,1]]
  elems = [[0,1,2],[0,2,3]]
  E = 210e9
  nu = 0.30
  t = 0.01
  traction = [[0, 1, 0.0, 500.0]]
  fix = [[0,'ux'],[0,'uy'],[3,'ux'],[3,'uy']]
REPORT
  export "plate_t3_traction.vtk" vtk
```

examples/ps_plate_plastic.dsl

```
PROBLEM "Materials : VonMises : PlaneStressFE"
GIVEN
  Lx = 1.0
  Ly = 1.0
  t = 0.01
  nx = 1
  ny = 1
  E = 210e9
  nu = 0.3
  sigma_y0 = 250e6
  H = 1.0e9
  ux_bar = 0.004
  load_steps = 40
```

```
  max_iter = 40
  tol = 1e-8
  penalty = 1e9
REPORT
  export "ps_path.csv" path
  export "ps_U.csv" U
```

examples/ps_plate_plastic_mesh.dsl

```
PROBLEM "Materials : VonMises : PlaneStressFE"
GIVEN
  Lx = 1.0
  Ly = 1.0
  t  = 0.01
  nx = 8
  ny = 8
  E = 210e9
  nu = 0.3
  sigma_y0 = 250e6
  H = 1.0e9
  # Option A: uniform traction on right edge
  traction_right_x = 0.0
  traction_right_y = -5.0e7
  # Option B: prescribed displacement (penalty)
  # ux_bar = 0.002
  load_steps = 50
  max_iter   = 50
  tol        = 1e-8
  penalty    = 1e8
REPORT
  export "ps_path.csv" path
  export "ps_U.csv" U
  export "ps_sigma.csv" sigma
  export "ps_ep.csv" ep
```

examples/ps_plate_plastic_vm.dsl

```
PROBLEM "Materials : VonMises : PlaneStressFE"
GIVEN
  Lx = 1.0
  Ly = 1.0
  t  = 0.01
  nx = 8
  ny = 8
  E = 210e9
  nu = 0.3
  sigma_y0 = 250e6
```

```
  H = 1.0e9
  traction_right_x = 0.0
  traction_right_y = -5.0e7
  load_steps = 50
  max_iter   = 50
  tol        = 1e-8
  penalty    = 1e8
REPORT
  export "ps_path.csv" path
  export "ps_U.csv" U
  export "ps_sigma.csv" sigma
  export "ps_ep.csv" ep
  export "ps_vm.csv" sigma_vm
```

examples/ps_vonmises_vtk.dsl

```
PROBLEM "Materials : VonMises : PlaneStressFE"
GIVEN
  Lx = 1.0
  Ly = 1.0
  t  = 0.01
  nx = 8
  ny = 8
  E = 210e9
  nu = 0.3
  sigma_y0 = 250e6
  H = 1.0e9
  traction_right_x = 0.0
  traction_right_y = -5.0e7
  load_steps = 30
  max_iter   = 40
  tol        = 1e-8
REPORT
  export "ps_result.vtk" vtk
  export "ps_vm.csv" sigma_vm
  export "ps_U.csv" U
```

examples/shell2d_mindlin_plate.dsl

```
PROBLEM "Solid : Shell2D : Static"
GIVEN
  Lx=1.0 Ly=1.0 nx=8 ny=8
  E=210e9 nu=0.3 t=0.01
  q=1000.0
  clamp_left=1
  clamp_right=1
```

```
  clamp_bottom=1
  clamp_top=1
REPORT
  export "shell2d_plate_U.csv" U
```

examples/shell_flat_dkt_rect.dsl

```
PROBLEM "Solid : Shell : FlatDKT"
GIVEN
  Lx = 1.0
  Ly = 1.0
  nx = 10
  ny = 10
  t = 0.01
  E = 70e9
  nu = 0.25
  q = 500.0
  clamp_left = 1
  clamp_right = 1
  clamp_bottom = 1
  clamp_top = 1
REPORT
  export "shell_flat_dkt_U.csv" U
```

examples/shell_membrane_panel.dsl

```
PROBLEM "Shell : Membrane : Static"
GIVEN
  nodes = [
    [0.0, 0.0],
    [1.0, 0.0],
    [0.0, 1.0]
  ]
  elements = [
    [0, 1, 2]
  ]
  thickness = 0.01
  E = 210e9
  nu = 0.3
  bcs = [
    {"node": 0, "dof": "ux", "value": 0.0},
    {"node": 0, "dof": "uy", "value": 0.0},
    {"node": 1, "dof": "uy", "value": 0.0}
  ]
  loads = [
    {"node": 2, "dof": "uy", "value": -1000.0}
  ]
```

```
REPORT
  print u
  print element_stress
```

examples/solid3d_hex20_modal.dsl

```
PROBLEM "Solid3D : Hex20 : Modal"
GIVEN
  nodes = [
    [0,0,0],[1,0,0],[1,1,0],[0,1,0],
    [0,0,1],[1,0,1],[1,1,1],[0,1,1],
    [0.5,0,0],[1,0.5,0],[0.5,1,0],[0,0.5,0],
    [0,0,0.5],[1,0,0.5],[1,1,0.5],[0,1,0.5],
    [0.5,0,1],[1,0.5,1],[0.5,1,1],[0,0.5,1]
  ]
  elements = [[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]]
  E = 210e9
  nu = 0.30
  rho = 7800.0
  nmodes = 3
  bcs = [
    {"node":0,"dof":"ux","value":0.0},
    {"node":0,"dof":"uy","value":0.0},
    {"node":0,"dof":"uz","value":0.0}
  ]
REPORT
  print freq
```

examples/solid3d_hex20_static.dsl

```
PROBLEM "Solid3D : Hex20 : Static"
GIVEN
  nodes = [
    [0,0,0],[1,0,0],[1,1,0],[0,1,0],
    [0,0,1],[1,0,1],[1,1,1],[0,1,1],
    [0.5,0,0],[1,0.5,0],[0.5,1,0],[0,0.5,0],
    [0,0,0.5],[1,0,0.5],[1,1,0.5],[0,1,0.5],
    [0.5,0,1],[1,0.5,1],[0.5,1,1],[0,0.5,1]
  ]
  elements = [[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]]
  E = 210e9
  nu = 0.30
  loads = [[6,"uz",-1e5]]
  bcs = [
    {"node":0,"dof":"ux","value":0.0},
    {"node":0,"dof":"uy","value":0.0},
    {"node":0,"dof":"uz","value":0.0}
```

```
  ]
REPORT
  print u
```

examples/solid3d_hex8_nonlinear.dsl

```
PROBLEM "Solid3D : Hex8 : Nonlinear"
GIVEN
  nodes = [
    [0,0,0],[1,0,0],[1,1,0],[0,1,0],
    [0,0,1],[1,0,1],[1,1,1],[0,1,1]
  ]
  elements = [[0,1,2,3,4,5,6,7]]
  material = {"E":210e9,"nu":0.30,"nonlinear_coef":1e6}
  loads = [[6,"uy",-1e5]]
  bcs = [
    {"node":0,"dof":"ux","value":0.0},
    {"node":0,"dof":"uy","value":0.0},
    {"node":0,"dof":"uz","value":0.0},
    {"node":1,"dof":"ux","value":0.0},
    {"node":1,"dof":"uy","value":0.0},
    {"node":1,"dof":"uz","value":0.0},
    {"node":3,"dof":"ux","value":0.0},
    {"node":3,"dof":"uy","value":0.0},
    {"node":3,"dof":"uz","value":0.0},
    {"node":2,"dof":"ux","value":0.0},
    {"node":2,"dof":"uy","value":0.0},
    {"node":2,"dof":"uz","value":0.0}
  ]
  max_iter = 10
  load_steps = 2
REPORT
  print u
```

examples/solid3d_hex8_transient.dsl

```
PROBLEM "Solid3D : Hex8 : Transient"
GIVEN
  nodes = [
    [0,0,0],[1,0,0],[1,1,0],[0,1,0],
    [0,0,1],[1,0,1],[1,1,1],[0,1,1]
  ]
  elements = [[0,1,2,3,4,5,6,7]]
  E = 210e9
  nu = 0.30
  rho = 7800.0
  damping_ratio = 0.02
```

```
  dt = 1e-4
  t_end = 1e-3
  fixed_dofs = [0,1,2]
  load_func = lambda t: (
    [0.0, 0.0, 0.0, 0.0, 0.0, -5e4*math.sin(2*math.pi*400*t)]
    + [0.0]*18
  )
REPORT
  print result["time"]
  print result["u"][-1]
```

examples/solid_axisym_q4_ring.dsl

```
PROBLEM "Solid : Axisymmetric : Q4 : Static"
GIVEN
  nodes = [[0.5,0.0],[1.0,0.0],[1.0,1.0],[0.5,1.0]]
  elems = [[0,1,2,3]]
  E = 200e9
  nu = 0.3
  fix = [[0,"ur",0.0],[3,"ur",0.0]]
REPORT
  print U
```

examples/study_fusion_steady.dsl

```
PROBLEM "Study : Fusion : Steady"
GIVEN
  subsystems = [
    {
      "name": "thermal",
      "solver": "podesl.solvers.heat1d:solve_heat1d_steady",
      "model": {
        "L": 1.0,
        "A": 1.0,
        "k": 10.0,
        "nel": 10,
        "T_left": 300.0,
        "T_right": 350.0
      }
    },
    {
      "name": "bar",
      "solver": "podesl.solvers.bar1d:solve_bar1d_static",
      "model": {
        "L": 1.0,
        "A": 1.0e-4,
        "E": 210e9,
```

```
        "nel": 10,
        "left": "fixed",
        "right": "traction",
        "traction_right": 0.0
      }
    }
  ]
  couplings = [
    {
      "src": "thermal",
      "src_field": "T",
      "dst": "bar",
      "dst_field": "traction_right",
      "map": "lambda T: 1.0e2 * (T[-1] - 300.0)",
      "description": "convert temp difference to axial load"
    }
  ]
  mode = "steady"
  max_iter = 3
  tol = 1e-6
REPORT
  print results["bar"]["u"][-1]
```

examples/study_fusion_transient.dsl

```
PROBLEM "Study : Fusion : Transient"
GIVEN
  subsystems = [
    {
      "name": "thermal",
      "solver": "podesl.solvers.heat1d_transient:solve_heat1d_transient",
      "model": {
        "L": 1.0,
        "A": 1.0,
        "k": 10.0,
        "rho": 7800.0,
        "cp": 500.0,
        "nel": 4,
        "nodes": [0.0, 0.25, 0.5, 0.75, 1.0],
        "elems": [[0,1],[1,2],[2,3],[3,4]],
        "dt": 0.1,
        "t_end": 0.0,
        "T_left": 300.0,
        "T_right": 300.0,
        "Tinit": 300.0,
        "fix": [[0, 300.0], [4, 300.0]]
      }
    },
```

```
    {
      "name": "bar",
      "solver": "podesl.solvers.bar1d:solve_bar1d_static",
      "model": {
        "L": 1.0,
        "A": 1.0e-4,
        "E": 210e9,
        "nel": 4,
        "left": "fixed",
        "right": "traction",
        "traction_right": 0.0
      }
    }
  ]
  couplings = [
    {
      "src": "thermal",
      "src_field": "T",
      "dst": "bar",
      "dst_field": "traction_right",
      "map": "lambda T: 5e1 * (T[-1][-1] - 300.0)",
      "description": "convert tip temperature into axial load"
    }
  ]
  mode = "transient"
  nsteps = 3
  dt = 0.1
  time_update = "lambda step, t, subsystems: subsystems['thermal'].model.update({'T_r
REPORT
  export "study_fusion_transient_times.csv" times
  export "study_fusion_transient_tip.csv" [step["u"][-1] for step in history["bar"]]
```

examples/study_montecarlo_bar.dsl

```
PROBLEM "Study : MonteCarlo : Run"
GIVEN
  child_problem = "Solid : Bar1D : Static"
  base_env = {
    "L": 1.0,
    "A": 1.0e-4,
    "E": 210e9,
    "nel": 4,
    "left": "fixed",
    "right": "traction",
    "traction_right": 1000.0
  }
  param_defs = {
    "traction_right": {"dist": "uniform", "low": 900.0, "high": 1100.0},
```

```
    "E": {"dist": "normal", "mean": 210e9, "std": 5e9}
  }
  outputs = ["u"]
  nsamples = 8
  rng_seed = 123
REPORT
  export "study_montecarlo_mean.csv" mean
  export "study_montecarlo_std.csv" std
  print mean
  print std
```

examples/study_optimize_bar.dsl

```
PROBLEM "Study : Optimize : Gradient"
GIVEN
  child_problem = "Solid : Bar1D : Static"
  base_env = {
    "L": 1.0,
    "A": 1.0e-4,
    "E": 210e9,
    "nel": 4,
    "left": "fixed",
    "right": "traction",
    "traction_right": 500.0
  }
  design_vars = [
    {"path": "traction_right"}
  ]
  x0 = [500.0]
  objective_expr = "(u[-1]*1e6)**2"
  max_iter = 40
  step_size = 0.5
  tol = 1e-8
REPORT
  export "study_optimize_x_opt.csv" x_opt
  export "study_optimize_history.csv" history["f"]
  print x_opt
  print f_opt
```

examples/study_scenarios_bar.dsl

```
PROBLEM "Study : Scenario : Compare"
GIVEN
  child_problem = "Solid : Bar1D : Static"
  base_env = {
    "L": 1.0,
    "A": 1.0e-4,
```

```
    "E": 210e9,
    "nel": 4,
    "left": "fixed",
    "right": "traction",
    "traction_right": 1000.0
  }
  scenarios = [
    {"name": "baseline", "modifications": []},
    {
      "name": "overload",
      "modifications": [
        {"path": "traction_right", "scale": 1.5}
      ]
    }
  ]
  metrics = [
    {"name": "tip_disp", "expr": "u[-1]"},
    {"name": "max_stress", "expr": "sigma.max()"}
  ]
REPORT
  export "study_scenarios_tip_disp.csv" [s["metrics"]["tip_disp"] for s in scenarios]
  print scenarios
```

examples/thermal1d_steady.dsl

```
PROBLEM "Thermal : 1DConduction : Steady"
GIVEN
  L = 1.0
  A = 1.0
  k = 12.0
  nel = 8
  T_left = 100.0
  T_right = 50.0
REPORT
  print T
```

examples/thermal1d_transient.dsl

```
PROBLEM "Thermal : 1DConduction : Transient"
GIVEN
  L = 1.0
  A = 1.0
  k = 12.0
  rho = 7800.0
  c = 500.0
  nel = 8
  dt = 0.1
```

```
  t_end = 0.5
  T_left = 100.0
  T_right = 50.0
REPORT
  print T[-1]
```

examples/thermomech2d_ps_coupled_plate.dsl

```
PROBLEM "Solid : ThermoMech2D : PS"
GIVEN
  Lx=1.0 Ly=1.0 nx=12 ny=12
  E=70e9 nu=0.33 t=1.0
  alpha=1.2e-5
  dT=60.0
  supports = [[0,1,1]]
REPORT
  export "thermomech2d_ps_U.csv" U
```

examples/truss2d.dsl

```
PROBLEM "Solid : Truss2D : Static"
GIVEN
  nodes = [[0,0],[1,0],[1,1]]
  elems = [[0,1, 1e-4, 210e9],
           [1,2, 1e-4, 210e9],
           [0,2, 1e-4, 210e9]]
  supports = [[0, 1,1], [1, 0,1]]  # [node, fix_ux, fix_uy]
  loads = [[2, 0, -1000]]
REPORT
  print U
  print reactions
```

examples/truss2d_buckling_simple.dsl

```
PROBLEM "Solid : Truss2D : BucklingLinear"
GIVEN
  nodes = [[0,0],[1,0],[2,0]]
  elems = [[0,1],[1,2]]
  E = 210e9
  A = 1.0e-4
  Nref = -1.0e4
  fix = [[0,"both",0.0],[2,"uy",0.0]]
  nmodes = 3
REPORT
  print lambda
```

examples/truss2d_cantilever.dsl

```
PROBLEM "Solid : Truss2D : Static"
GIVEN
  nodes = [[0,0],[1,0],[2,0],[3,0],[4,0]]
  elems = [[0,1],[1,2],[2,3],[3,4]]
  E = 210e9
  A = 3.0e-4
  point_loads = [[4, 0.0, -1000.0]]
  fix = [[0,'x'],[0,'y']]
REPORT
  export "truss2d_cantilever.vtk" vtk
```

examples/truss2d_ep_pull.dsl

```
PROBLEM "Solid : Truss2D : ElastoPlastic"
GIVEN
  nodes = [[0,0],[1,0]]
  elems = [[0,1]]
  E = 210e9
  A = 1.0e-4
  sigy = 250e6
  H = 1.0e9
  supports = [[0,1,1]]
  loads = [[1, 1.0e5, 0.0]]
REPORT
  export "truss2d_ep_U.csv" U
  export "truss2d_ep_N.csv" N
  export "truss2d_ep_ep.csv" ep
```

examples/truss2d_export.dsl

```
PROBLEM "Solid : Truss2D : Static"
GIVEN
  nodes = [[0,0],[1,0],[1,1]]
  elems = [[0,1, 1e-4, 210e9],
           [1,2, 1e-4, 210e9],
           [0,2, 1e-4, 210e9]]
  supports = [[0, 1,1], [1, 0,1]]
  loads = [[2, 0, -1000]]
REPORT
  export "truss_u.csv" U
```

examples/truss2d_geomnl_arc.dsl

```
PROBLEM "Solid : Truss2D : GeomNL"
```

```
GIVEN
  nodes = [[0,0],[1,0]]
  elems = [[0,1]]
  E = 210e9
  A = 1.0e-4
  supports = [[0,1,1]]
  loads = [[1, 0.0, -5.0e4]]
REPORT
  export "truss2d_geomnl_U.csv" U
  export "truss2d_geomnl_Reac.csv" reactions
```

examples/truss2d_linear_bridge.dsl

```
PROBLEM "Solid : Truss2D : Static"
GIVEN
  nodes = [[0,0],[1,0],[0.5,0.8]]
  elems = [[0,1],[0,2],[1,2]]
  E = 210e9
  A = 2.0e-4
  supports = [[0,1,1],[1,1,1]]
  loads = [[2, 0.0, -1000.0]]
REPORT
  export "truss2d_linear_U.csv" U
  export "truss2d_linear_forces.csv" element_forces
```

examples/truss2d_modal_bar.dsl

```
PROBLEM "Solid : Truss2D : Modal"
GIVEN
  nodes = [[0,0],[2,0]]
  elems = [[0,1]]
  E=210e9 A=1.0e-4 rho=7800
  supports = [[0,1,1]]
REPORT
  export "truss2d_modal_freq.csv" freq
```

examples/truss2d_modal_simple.dsl

```
PROBLEM "Solid : Truss2D : Modal"
GIVEN
  nodes = [[0,0],[1,0],[2,0]]
  elems = [[0,1],[1,2]]
  E = 210e9
  A = 1.0e-4
  rho = 7800.0
  fix = [[0,"both",0.0]]
  nmodes = 3
```

```
REPORT
  print freq
```

examples/truss2d_modal_two_bar.dsl

```
PROBLEM "Solid : Truss2D : Modal"
GIVEN
  nodes = [[0,0],[1,0],[1,1]]
  elems = [[0,1],[1,2]]
  E = 210e9
  A = 1.0e-4
  rho = 7850.0
  fix = [[0,"both",0.0],[1,"uy",0.0]]
  nmodes = 3
REPORT
  print freq
```

examples/truss2d_nonlinear_pull.dsl

```
PROBLEM "Solid : Truss2D : Nonlinear"
GIVEN
  nodes = [[0,0],[1,0]]
  elems = [[0,1]]
  E = 210e9
  A = 2.0e-4
  supports = [[0,1,1]]
  loads = [[1, 10000.0, 0.0]]
  max_iter = 40
  tol = 1e-10
REPORT
  export "truss2d_nonlinear_U.csv" U
  export "truss2d_nonlinear_N.csv" N
```

examples/truss2d_simple.dsl

```
PROBLEM "Solid : Truss2D : Static"
GIVEN
  nodes = [[0,0],[1,0],[1,1]]
  elems = [[0,1],[1,2]]
  E = 210e9
  A = 1.0e-4
  loads = [[2, 0.0, -1000.0]]
  fix = [[0,"both",0.0],[1,"uy",0.0]]
REPORT
  print U
```

examples/truss2d_snap.dsl

```
PROBLEM "Solid : Truss2D : GeomNL"
GIVEN
  nodes = [[0,0],[1,0],[2,0]]
  elems = [[0,1],[1,2]]
  E = 210e9
  A = 1.0e-4
  supports = [[0,1,1],[2,1,1]]
  loads = [[1, 0.0, -2.0e5]]
  CONTROLS = {"method":"arc","load_steps":60,"arc_len":1e-3,"arc_minmax":[1e-5,1e-2],"
REPORT
  export "truss2d_snap_path.csv" path
  export "truss2d_snap_U.csv" U
```

examples/truss2d_static.dsl

```
PROBLEM "Solid : Truss2D : Static"
GIVEN
  nodes = [[0,0],[3,0],[6,0],[3,3]]
  elems = [
    [0,1, 4.0e-4, 210e9],
    [1,2, 4.0e-4, 210e9],
    [0,3, 4.0e-4, 210e9],
    [1,3, 4.0e-4, 210e9],
    [2,3, 4.0e-4, 210e9]
  ]
  supports = [[0,1,1],[2,1,1]]
  loads = [[3, 0.0, -20000.0]]
REPORT
  print U
  export "truss_U.csv" U
  export "truss_member_forces.csv" member_forces
```

examples/truss2d_tri.dsl

```
PROBLEM "Solid : Truss2D : Static"
GIVEN
  nodes = [[0,0],[1,0],[0,1]]
  elems = [[0,1],[1,2],[2,0]]
  E = [210e9, 210e9, 210e9]
  A = [8e-5, 8e-5, 8e-5]
  loads = [[1, 1000.0, -500.0]]
  fix = [[0,"both",0.0],[2,"uy",0.0]]
REPORT
  print U
  print element_forces
```

examples/truss2d_two_bar.dsl

```
PROBLEM "Solid : Truss2D : Static"
GIVEN
  nodes = [[0,0],[1,0],[1,1]]
  elems = [[0,1],[1,2]]
  E = 210e9
  A = 1.0e-4
  loads = [[2, 0.0, -1000.0]]
  fix = [[0,"both",0.0],[1,"uy",0.0]]
REPORT
  print U
  print element_forces
```

examples/truss3d_single_bar.dsl

```
PROBLEM "Solid : Truss3D : Linear"
GIVEN
  nodes = [[0,0,0],[1,0,0]]
  elems = [[0,1]]
  A = 1.0e-4
  E = 200e9
  fix = [[0,'ux'],[0,'uy'],[0,'uz']]
  point_loads = [[1, 1000.0, 0.0, 0.0]]
REPORT
  print U
```

examples/truss3d_square_braced.dsl

```
PROBLEM "Solid : Truss3D : Linear"
GIVEN
  nodes = [[0,0,0],[1,0,0],[1,1,0],[0,1,0]]
  elems = [[0,1,1e-4,200e9],[1,2,1e-4,200e9],[2,3,1e-4,200e9],[3,0,1e-4,200e9],[0,2,1e
  fix = [[0,'ux'],[0,'uy'],[0,'uz'],[3,'ux'],[3,'uy'],[3,'uz']]
  point_loads = [[1, 0.0, -1000.0, 0.0],[2, 0.0, -1000.0, 0.0]]
REPORT
  print U
```

examples/truss3d_v.dsl

```
PROBLEM "Solid : Truss3D : Static"
GIVEN
  nodes = [[0,0,0],[1,0,0],[1,1,0]]
  elems = [[0,1],[1,2]]
```

```
  E = 210e9
  A = 1.0e-4
  loads = [[2, 0.0, -1000.0, 0.0]]
  fix = [[0,"both",0.0],[1,"uy",0.0]]
REPORT
  print U
```

examples/truss3d_v_modal.dsl

```
PROBLEM "Solid : Truss3D : Modal"
GIVEN
  nodes = [[0,0,0],[1,0,0],[1,1,0]]
  elems = [[0,1],[1,2]]
  E = 210e9
  A = 1.0e-4
  rho = 7850.0
  fix = [[0,"both",0.0],[1,"uy",0.0]]
  nmodes = 3
REPORT
  print freq
```

Document Roadmap