Guide 8: Fluid-structure interaction

Fluid-structure interaction, as pyfs does it

Written for pyflightstream 0.58.2.

Sources: the package page "FSI in a workspace", the FSI tutorial and the FSI examples [1].

pyflightstream is the work of Geovana Neves: `https://github.com/nevesgeovana/pyflightstream`, concept DOI `10.5281/zenodo.21482924`.
Copyright (c) 2026 Geovana Neves. Licensed under CC BY 4.0
(`https://creativecommons.org/licenses/by/4.0/`). See LICENSE-AND-AUTHORSHIP.md.

What FSI means here, and how to read this deck

- **FSI** couples the aerodynamic loads on a blade or a wing to its structural deflection: FlightStream solves the flow, pyflightstream solves the structure, and the solver deforms its surface with the displacements it is handed.
- The structure is **one Euler beam per blade**, or one along the wing (PyNite [5]), in flap and torsion, solved quasi-statically at every coupling call.
- A matrix row selects its structure by name, `FSI: f001`, and calibrates it from the matrix.
- Each frame names its source at its foot: the examples' numbers are the package's, on synthetic blades; the solver's behaviour is measured on 26.124 [2].

The optional extra

The beam needs PyNite, from the `[fsi]` extra: `pip install "pyflightstream[fsi]"`. Without it the package says how to install it.

Takeaway *FSI runs on `steady`, `qsteady_rotor` (sector) and `unsteady` rows. A turning rotor, `unsteady_rotor`, is refused: that coupling is still being debugged.*

Source: pyflightstream [1]; FSI tutorial (`src/pyflightstream/fsi/README.md`); `fsi/beam.py`; `extras.py`; the page "FSI in a workspace".

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Where FSI runs, and what the structure carries

Workflow	FSI	What the structure carries, and why
`steady`	allowed	a fixed wing, one clamped beam: the aerodynamic loads and the structure's **own weight**; nothing turns, so no centrifugal load. At most 50 coupling iterations
`qsteady_rotor`, sector	allowed	one blade held still in a free stream turning at the rotor's speed: the structural solver applies the **centrifugal** loads and stiffening at the **free-stream rotation's** rpm, $\Omega = \lvert\mathrm{rpm}\rvert\,\pi/30$
`qsteady_rotor`, wheel	refused	several steady clockings averaged; a blade deformed once per clocking is not the state of one structure
`unsteady`	allowed	a fixed wing, nothing turning: the loads and its own weight, one coupling call per time step
`unsteady_rotor`	refused	"FSI on unsteady_rotor is still in debug on this release": on 26.124 the morph of a turning blade is applied to the blade at its imported azimuth [2]

A fixed wing's FSI input states `[config.wing]` and couples on `steady` or `unsteady` only; a blade's input, without it, couples on the rotor route only. A quasi-steady answer is valid only for an isolated, axisymmetric rotor, and only while the flow at each station changes slowly beside the revolution (Guide 2, the workflows; Guide 7).

Source: pyflightstream [1]; `cases/fsi_workspace.py`, `FSI_WORKFLOW_STATE`, `validate_workspace_fsi`, `STEADY_AEROELASTIC_ITERATIONS`; `cases/workflows.py`; RPT-093 [2].

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Contents

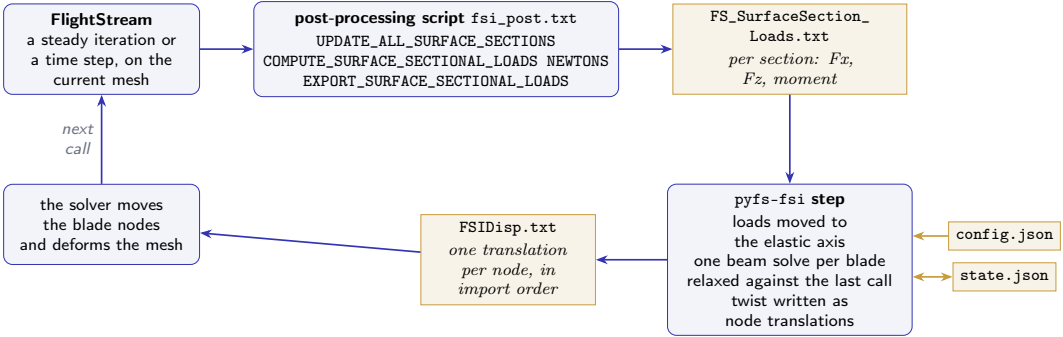
The coupling loop
The structural input
Calibration and provenance
From the row to the solver
What is refused, and why
Worked examples
Outputs, checklist and limits

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 1

The coupling loop

The loop, one coupling call



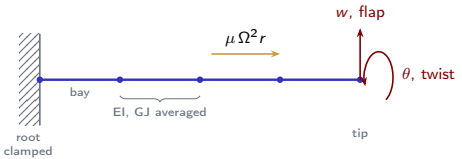
FlightStream's Aeroelastic Toolbox calls the executable once per aeroelastic iteration of a steady row, or once per time step of an unsteady one (`SET_AEROELASTIC_ITERATIONS 1`). Each call is stateless: what persists is in `state.json`. Loads and displacements are in each blade's own frame, or the wing's; the package never sees azimuth.

Source: pyflightstream [1]; FSI tutorial, "The loop in one paragraph", "cli"; `cases/fsi_workspace.py`, `wire_workspace_fsi`; `fsi/__init__.py`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The structure: one Euler beam per blade, or along the wing

- One node per radial station, on the elastic axis; one member per bay, with the bay-averaged EI and GJ; the root station clamped.
- Unit elastic moduli, $E = G = 1$: the section constants handed to PyNite are the configuration's EI and GJ, so no modulus is applied twice.
- Flap deflection w and elastic twist θ carry mass ($\mu\, l$ and $(I_1 + I_2)\, l$); every other degree of freedom is condensed out of the eigenproblem exactly.
- Static solves are linear, or P-Delta whenever axial tension is present [6, 7]. [REF-08], [REF-52]



Takeaway *The stiffnesses in* `config.json` *are the beam's stiffnesses, typed or generated, exactly as written.*

Source: pyflightstream [1]; FSI tutorial, "beam"; `fsi/beam.py` module docstring.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

What rotation adds

- **Tension.** The distributed axial force $\mu\,\Omega^2 r$ makes the tension $N(r)$ emerge in the solver, and P-Delta turns it into bending stiffness, with no manual correction terms: the centrifugal stiffening of a rotating blade [8, 9, 10]. [REF-05], [REF-53], [REF-54]
- **Propeller moment.** $-\Omega^2(I_1 - I_2)\sin\theta\cos\theta$ drives each section toward flat pitch. It depends on the total pitch, geometric plus elastic, so each call re-solves the beam a few times until the twist stops moving [9, 12]. [REF-05], [REF-56]
- **In-plane softening.** The part of the flap deflection that lies in the plane of rotation feels the centrifugal field pulling it outward, a term $\mu\,\Omega^2 \sin^2\beta\, w$, β the section's pitch, that lowers the stiffness; it is applied with the tension [8, 9]. [REF-05], [REF-53]
- **Not modelled: the trapeze effect.** A declared limitation; omitting it overpredicts the elastic twist, which is conservative.

Source: pyflightstream [1]; FSI tutorial, "The loop in one paragraph" and "centrifugal"; `fsi/centrifugal.py` module docstring.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

A fixed wing: its own weight, and where the model stops

- **A wing that does not turn** has no centrifugal load. The structural solver applies the structure's **own weight**, gravity on its mass distribution, beside the aerodynamic loads, on a `steady` or an `unsteady` row; the loop is the same.
- **A blade held still** in a turning free stream (`qsteady_rotor`, sector) is the opposite case: the aerodynamics see the rotation through the free stream, which reproduces the turning blade's axial force to 0.35 % [3], and the structural solver still applies the centrifugal loads at the free-stream rotation's rpm.
- **Quasi-static.** The mean deformation is exact within the beam idealisation; the once-per-revolution content is trustworthy only well below resonance, roughly $n\Omega/\omega_n \leq 0.3$ [12, 13]. [REF-34], [REF-56] The convergence log prints this boundary in its header.

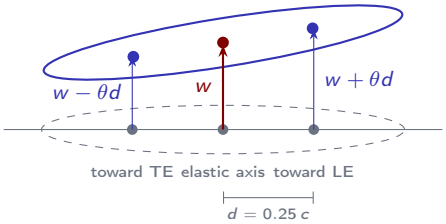
The formulas are transcribed, not re-verified

The primary sources of the model have not been independently checked against the implemented formulas; each formula lives in one small function that cites its source, so a correction stays local.

Source: pyflightstream [1]; `fsi/wing.py` module docstring (gravity $-z$ by default, `[config.wing]`); `fsi/centrifugal.py`; the page "FSI in a workspace".

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Twist travels as three nodes per station



- `FSIDisp.txt` carries translations only, so twist is encoded by three nodes per station, $d = 0.25\,c$ apart (`node_offset_chord_fraction`).
- Rigid section kinematics, $\delta y = w + \theta\, d$: linear, so the inverse is exact to machine precision.
- One generator makes the node file, the ordering map and the `FSIDisp.txt` rows: the solver applies them strictly in import order, and a drift would corrupt the geometry in silence.
- One node file serves every blade, imported once per blade frame, in blade order.

Source: pyflightstream [1]; FSI tutorial, "kinematics" and "nodes"; `fsi/nodes.py` module docstring; `fsi/config.py`, `FsiConfig.node_offset_chord_fraction`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The offline driver: four phases, keyed on the step counter

This schedule belongs to the standalone driver. The workspace routes allowed for FSI do not run this four-phase schedule or establish its convergence judgement.

Phase	Behaviour, and its key in `config.json` `phases`
1, wake development	zero displacements on the rigid blade; `wake_development_revolutions` (1.0)
2, averaged coupling	loads averaged over the window, relaxed updates $d_{new} = d_{old} + \lambda(d_{calc} - d_{old})$; `averaging_window_revolutions` (1.0), `coupling_relaxation` (0.4)
3, convergence watch	as 2, and converged when BOTH hold across consecutive revolutions: tip twist change below `tip_twist_tolerance_deg` (0.05) and relative change of the integrated normal force below `thrust_tolerance_fraction` (0.02)
4, recording	instantaneous loads, $\lambda = 1$ by design, twist recorded per step; `recording_revolutions` (1.0)

- **Freshness is asserted**: each call must see an advancing solver iteration in the loads header; a repeat is refused, since it means more than one FSI iteration per time step.
- **Frozen mode**: a `fsi_frozen_displacements.txt` in the run folder is replayed verbatim at every call, with no loads and no solve.

Source: pyflightstream [1]; FSI tutorial, "driver and state"; `fsi/config.py`, `PhaseSchedule` (defaults in brackets).

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The structural nodes, and how the surface follows them

- **Inside the component.** Where the input gives section contours, the nodes lie on the camber line of each station, turned by its local twist, inside the blade or the wing; a node outside is refused at plan.
- **The node frame.** Check the stored node coordinates and frame in the probe evidence [2]; the requested frame alone does not prove which frame the solver stored.
- **The surface by its boundary ID**: an OBJ import numbers its first boundary 2 and an STL 1; a list holding the tree position maps 0 vertices, in silence [2].
- **After the structural call**: the exports that must show the deformation run in the post-processing script.

A radial basis function carries the nodes to the panels

The displacement known at the nodes is interpolated onto every aerodynamic panel [14, 15, 16]. A beam is a **line** of nodes: a [REF-44], [REF-45], [REF-47] compactly supported kernel reaches only the panels near it. The coupled row uses the **multiquadric** [14], whose support is global: on one beam line the measured maximum error reached 24.19 mm for the imposed bend. The compact kernel gave 64 % of the imposed bend and the coupled run diverged [2]. These are measured cases, not a general accuracy bound.

Source: pyflightstream [1]; `fsi/nodes.py`; `cases/fsi_workspace.py`; Guide 6, `aeroelastic_rbf_type`; RPT-093 [2], facts 1, 2 and 7.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Route B: the direct transform, a sanity check

- The same beam, solved on the solver's sectional loads, gives flap $w(r)$ and twist $\theta(r)$; **every mesh vertex is moved** about the elastic axis by that w and θ, and the deformed geometry is solved again, rigid. No interpolation, no in-run coupling.
- Repeated until the tip flap and twist stop moving, it is a partitioned static iteration: quasi-steady by construction.
- A zero deformation through the same chain reproduces the rigid answer, which is its own control.

What route B is for

A simplified run, a check of the coupled route against an independent one, and an FSI option where the coupled route is not available. It is **not** the coupled route: it carries no dynamics and no in-run deformation.

Source: pyflightstream [1]; `fsi/loads.py`, `to_elastic_axis`; `fsi/centrifugal.py`, `solve_rotating_static`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

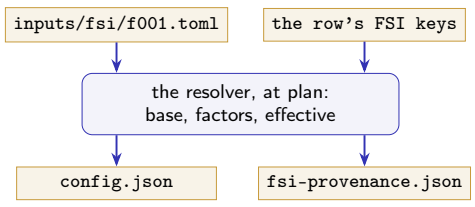
Part 2

The structural input

One structural input, selected by name

- One configuration per file, `inputs/fsi/f001.toml`: `f`, then letters, digits, _ or -; no extension in the row.
- The row selects it in `VAR_NAMES_VALUES`, with any calibration beside it.
- The resolver attaches the *effective* `FsiConfig` and its provenance to the case. No solver runs while loading or planning.
- The run writes both files in the point's working directory and records their hashes. The source is hashed too: changing it means planning again.

```
FSI: f001 / FSI_BENDING_STIFFNESS_N_M2_FACTOR: 1.15
```



written and hashed in the point's working directory

Takeaway *The geometry library is never a destination for these generated files.*

Source: pyflightstream [1]; `docs/fsi-workspace.md`, **opening section**; `workspace/fsi_setup.py`, `resolve_row_fsi`; `docs/srs/data-model.md`, the `inputs/` tree.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Two ways to state the blade

	`mode = "supplied"`	`mode = "calculated"`
What you write	the `FsiConfig` scalars under `[config]` and the complete `BladeProperties` arrays under `[config.blade]`	the `[config]` scalars, a `material`, and `[sections]`: one closed contour per station, in metres
What the package computes	nothing: the distributions are the file's, then calibrated	area, centroid, principal inertias, the torsion constant, then EI, GJ, running mass and mass inertias from the material
Where the numbers come from	you: the example distributions are synthetic, not a material or airframe dataset	the geometry and ONE sourced material, recorded with the numbers
Refused	a `material` or `[sections]` table; any material factor	a `[config.blade]` table; no material; no `[sections]`

Takeaway *Either way the result is the existing solid homogeneous Euler beam: this input adds no shell, spar, hollow-section, shear-centre or second structural model.*

Source: pyflightstream [1]; `docs/fsi-workspace.md`, "Supplied distributions" and "Calculated solid sections"; `workspace/fsi_setup.py`, `resolve_fsi_setup`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Supplied distributions

```toml
mode = "supplied"

[calibration]
bending_stiffness_n_m2 = 1.05

[config]
blade_count = 2
omega_rad_per_s = 0.0

[config.blade]
station_radii_m = [0.1, 0.5]
chord_m = [0.04, 0.04]
mass_per_length_kg_per_m = [1.0, 2.0]
inertia_major_kg_m = [0.001, 0.002]
inertia_minor_kg_m = [0.0001, 0.0002]
bending_stiffness_n_m2 = [10.0, 20.0]
torsion_stiffness_n_m2 = [5.0, 10.0]
elastic_axis_offset_chordwise_m = [0.0, 0.0]
elastic_axis_offset_normal_m = [0.0, 0.0]
cg_offset_chordwise_m = [0.0, 0.0]
cg_offset_normal_m = [0.0, 0.0]
geometric_pitch_deg = [0.0, 0.0]
```

Array	Unit
`station_radii_m`, strictly increasing	m
`mass_per_length_kg_per_m`	kg/m
`bending_stiffness_n_m2` (EI, flap)	$\mathrm{N\,m^2}$
`torsion_stiffness_n_m2` (GJ)	$\mathrm{N\,m^2}$
`inertia_major_kg_m`, `_minor_kg_m`	kg m
offsets, `chord_m`	m

- Every array has the length of `station_radii_m`; values are finite; mass, EI, GJ and chord positive; principal inertias not negative.
- With the row factor 1.15 of the previous part, the effective EI is `[11.5, 23.0]`: the row replaces the file's 1.05, it does not multiply it.

A zero-speed input like this one is a valid structural input for offline checks, not a runnable coupled rotor row.

Source: pyflightstream [1]; `docs/fsi-workspace.md`, "Supplied distributions"; `fsi/config.py`, `BladeProperties`.

Calculated solid sections

```
mode = "calculated"
material = "ti-6al-4v-grade5-annealed"

[config]
blade_count = 2
omega_rad_per_s = 0.0

[sections]
station_radii_m = [0.1, 0.5]
chord_m = [0.04, 0.04]
geometric_pitch_deg = [0.0, 0.0]
geometry_source = "Synthetic rectangular sections in SI
      units"
torsion_grid_cells = 16
sections_m = [
  [[-0.02, -0.002], [0.02, -0.002],
   [0.02, 0.002], [-0.02, 0.002]],
  [[-0.02, -0.002], [0.02, -0.002],
   [0.02, 0.002], [-0.02, 0.002]]
]
```

- `sections_m`: one closed polygon per station, chordwise and normal metres about the pitch axis.
- `geometry_source`: what the contours were built from, in words; it travels with the numbers.
- `torsion_grid_cells`: cells across the thickness for the torsion solve, 8 to 512, 64 when omitted.
- No `[config.blade]`: this mode derives it.

Takeaway *A 40 by 4 mm titanium strip; example 1 turns these contours into EI.*

Source: pyflightstream [1]; `docs/fsi-workspace.md`, "Calculated solid sections"; `workspace/fsi_setup.py`, `SolidSections`, `FSI_TEMPLATE`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The material database: no number without a source

Key	ρ [kg/m^3]	E [GPa]	G [GPa]	ν	Source, as recorded
`ti-6al-4v-grade5-annealed`	4430	113.8	44.0	0.342	ASM Aerospace Specification Metals data sheet, annealed; G tabulated
`al-7075-t6`	2810	71.7	26.9	0.33	ASM Aerospace Specification Metals data sheet, T6 typical values; G tabulated

- Each entry's four numbers come from ONE cited data set, with its table, condition, address and reading date. Database version 1; a changed number is a new version.
- Your own material: a `[material]` table stating every field, with a `[material.source]` of `document`, `table`, `condition`, `url`, `consulted`. Density, E, G finite and positive; $-1 < \nu < 0.5$.
- Room-temperature typical values for a structural model of a blade, not the design allowables of a certified part.

Source: pyflightstream [1]; `fsi/materials.py`; `docs/fsi-workspace.md`, "Calculated solid sections"; FSI tutorial, "sections and materials".

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The solid-section calculator

Field	Generated as
`mass_per_length_kg_per_m`	ρA
`inertia_major_kg_m, _minor`	ρ times the larger, smaller principal second moment
`bending_stiffness_n_m2`	E times the second moment about the chordwise centroidal axis
`torsion_stiffness_n_m2`	GJ, J computed numerically
`elastic_axis_offset_*`	the centroid's position from the pitch axis
`cg_offset_*`	zero

- Area, centroid, second moments: exact closed forms of the polygon. A contour that crosses or touches itself, repeats a vertex or folds back is refused by name.
- J: the Prandtl stress function by finite differences cut exactly at the polygon [6, 7]. Check grid convergence for your section; the [REF-08], [REF-52] blade-properties example prints values at each resolution.
- The thin-section $\frac{1}{3}\int t^3\,ds$ is kept as a cross-check, never as the value.

Stated hypotheses, recorded with every generated blade

SOLID, HOMOGENEOUS sections; the elastic axis at the CENTROID, the shear centre not computed; a zero elastic-axis-to-CG offset, so no bend-twist coupling from it; flap bending about the chordwise axis.

Source: pyflightstream [1]; FSI tutorial, "sections and materials".

Copyright (c) 2026 Geovana Neves, CC BY 4.0

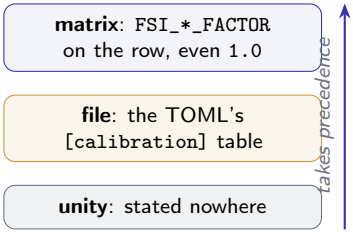
Part 3

Calibration and provenance

Calibration factors: unity, the file, the matrix

Level	Property names
Distributions, either mode	`mass_per_length_kg_per_m`, `inertia_major_kg_m`, `inertia_minor_kg_m`, `bending_stiffness_n_m2`, `torsion_stiffness_n_m2`
Orientation	`geometric_pitch_deg`
Offsets	`elastic_axis_offset_chordwise_m`, `_normal_m`, `cg_offset_chordwise_m`, `_normal_m`
Material, calculated mode only	`density_kg_per_m3`, `youngs_modulus_pa`, `shear_modulus_pa`, `poisson_ratio`

The matrix key is `FSI_` + the name in capitals + `_FACTOR`: `FSI_TORSION_STIFFNESS_N_M2_FACTOR`.



- Every factor is dimensionless, finite and positive.
- An explicit matrix value replaces the file factor exactly once; it is never multiplied by it.

Source: pyflightstream [1]; `docs/fsi-workspace.md`, "Calibration and provenance"; `_fsi_calibration.py`, `MATRIX_FACTORS`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

One property, scaled once

Calibrating a source and a property derived from it would scale the same number twice. Refused, in calculated mode:

Source factor	with any of
density	mass per length, either mass inertia
Young's modulus E	EI
shear modulus G	GJ
E or ν, when G is derived	a separate G or GJ factor

"…: calibrating youngs_modulus_pa and its derived bending_stiffness_n_m2 would scale the same property twice; choose one calibration level."

- The legacy `config.stiffness_scale_factor` must be 1.0 in a workspace input: calibrate EI and GJ explicitly instead of a second scaling at solve time.
- Supplied mode takes no material factor: calibrate the distribution itself.
- Every effective property is validated again, including major principal inertia not below minor.

File 1.05, row 1.15

Effective EI = base × 1.15, not base × 1.2075. The origin recorded is `matrix`.

Source: pyflightstream [1]; `workspace/fsi_setup.py`, `resolve_fsi_setup`; `docs/fsi-workspace.md`, "Calibration and provenance".

Copyright (c) 2026 Geovana Neves, CC BY 4.0

A tabulated shear modulus stays tabulated

- A material either takes G as its source tabulates it, or derives it as $E/(2(1+\nu))$ and says so (`shear_modulus_basis`).
- When the source tabulates G, changing E or ν keeps that G. Only a material declaring a derived G follows the formula.
- The two contracts stay distinct even at unity calibration: nothing substitutes the isotropic formula for a tabulated value.
- Calibration never rewrites the source database; the base and effective material both go to the provenance.

The titanium entry

Tabulated: $G = 44.0$ GPa.
From the same sheet, $E/(2(1+\nu)) = 42.4$ GPa.

The entry keeps 44.0, about 4 % apart, and a torsion stiffness built from it keeps it too.

Takeaway *Unity calibration, or a density, E or ν change, does not substitute the formula for a tabulated G.*

Source: pyflightstream [1]; `docs/fsi-workspace.md`, "Calculated solid sections" and "Calibration and provenance"; `fsi/materials.py`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The pitch factor scales angles, once

- `FSI_GEOMETRIC_PITCH_DEG_FACTOR` scales each signed structural angle in degrees once; zero stays zero.
- It changes the orientation of the spinning beam's sections used to generate the structural nodes and to project the loads; section properties, radii and node order stay.
- It is not an angular offset. It does not turn the section contours again, and it does not re-pitch the aerodynamic mesh.
- Keep the aerodynamic geometry consistent with the intended structural orientation, and regenerate the node file and ordering map from the effective configuration.

File factor 2.0, row factor 1.5

`geometric_pitch_deg = [-60, 20]`

becomes `[-90, 30]`,
not `[-180, 60]`.

Alone, it validates nothing

This parameter alone does not establish coupled aerodynamic validation.

Source: pyflightstream [1]; `docs/fsi-workspace.md`, "Calibration and provenance".

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Provenance: every number says where it came from

`fsi-provenance.json`**, per point**

- the source path and its sha256, and the mode;
- the unscaled (base) and effective configurations;
- every factor, and whether it came from `unity`, `file` or `matrix`;
- calculated mode: the base and effective material properties and the section algorithm's provenance;
- `section_model`: `"existing solid homogeneous Euler beam"`.

Inside a generated blade:

`BladeProperties.provenance`

- the material entry, its source and the database version;
- the geometry file's NAME and sha256, never its path;
- the hypotheses, the torsion method and its grid per station, and the thin-section ratio per station.

`config_sha256`

- over the sorted, whitespace-free JSON: it identifies the physics, not the file layout;
- on every row of the convergence log. A typed configuration has no provenance and keeps the digest it always had.

Source: pyflightstream [1]; `workspace/fsi_setup.py`, `ResolvedFsiSetup.provenance`; `fsi/config.py`, `config_sha256`; FSI tutorial, "config".

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 4

From the row to the solver

What a coupled row stages

A row that states `FSI: f001` on a workflow that allows it prepares the driver's complete input interface. Planning derives the nodes and the maps from the effective configuration and resolves the boundaries and the frames of each blade or of the wing; the run writes and hashes:

File, in the point's directory	What it is
`config.json`, `fsi-provenance.json`	the effective configuration, and where every number came from
`fsi_nodes.csv`	the structural nodes, inside the component, in its own frame; one identical file imported once per blade
`fsi_node_map.json`	the node ordering map (`node_map_file`), from the same generator
`fsi_family_map.json`	the emitted section-load order: which rows of the loads export belong to which blade
`fsi_post.txt`	the native loads-export callback: update the sections, compute their loads in Newtons, export them
`fsi_callback.py`	the Python callback: one `pyfs-fsi` step in the point's directory

The callback runs under the interpreter that prepared the run (its `pythonw.exe` sibling on Windows): the package, with its `[fsi]` extra, must be installed for that interpreter on the machine that executes the point.

Source: pyflightstream [1]; `docs/fsi-workspace.md`, "Automatic workspace coupling"; `cases/fsi_workspace.py`, `wire_workspace_fsi`; `fsi/beam.py` (PyNite from the `[fsi]` extra).

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The native commands, in the order they are emitted

Command	What the row gives it
`AEROELASTIC_RBF_TYPE`	`MULTI_QUADRATIC`, for a beam line of nodes, unless the setup states `aeroelastic_rbf_type`
`DELETE_AEROELASTIC_STRUCTURAL_NODES`	nothing: a clean start
`ASSIGN_AEROELASTIC_SURFACES`	the coupled boundaries by their **boundary ID**: the tree position plus 1 for an OBJ import, plus 0 for an STL
`ASSIGN_AEROELASTIC_COORDINATE_SYSTEMS`	each blade's own frame, or the frame the wing's sections are cut in
`IMPORT_AEROELASTIC_STRUCTURAL_NODES`	per blade, in blade order: its frame, `DISABLE`, `fsi_nodes.csv`
`SET_AEROELASTIC_WORKING_DIRECTORY`	`.`, the point's directory
`SET_AEROELASTIC_POST_PROCESSING_SCRIPT`	`fsi_post.txt`
`SET_AEROELASTIC_STRUCTURAL_EXECUTION_COMMAND`	the interpreter and `fsi_callback.py`
`SET_AEROELASTIC_ITERATIONS`	steady: 50, a cap; unsteady: 1 per time step
`SET_AEROELASTIC_COUPLING_IN_UNSTEADY`	an unsteady row: `ENABLE`, before `START_SOLVER`
`EXECUTE_AEROELASTIC_ANALYSIS`	a steady row: the **last line** of the script

Source: pyflightstream [1]; `cases/fsi_workspace.py`, `_stage_and_emit`, `emit_steady_aeroelastic_analysis`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

A steady coupled script ends at the analysis

- In a script the analysis **returns at once**: an export after it would write the rigid surface, and a `CLOSE_FLIGHTSTREAM` would end the analysis [2]. So it is the last line; the plan refuses any line after it.
- The exports run in `fsi_post.txt`, after every coupling iteration, the last included: the files left are the deformed ones.
- The run waits for `Aeroelastic solver run time`, then stops the process.
- Unsteady: `fsi_post.txt` runs twice per step, before and after the structural call.

`fsi_callback.py`, whole:

```
from pyflightstream.fsi.cli import main
raise SystemExit(main(['step', '--dir', '.']))
```

Check evidence for each command and build

The command reference records status per command and build, including verified and removed entries. A family-wide status cannot replace that lookup. Coupled-run measurements (RPT-093 [2]) have their own scope.

Source: pyflightstream [1]; `cases/fsi_workspace.py`, `refuse_lines_after_steady_analysis`, `STEADY_AEROELASTIC_COMPLETION`, `aeroelastic_post_script`; `commands/aeroelastic_coupling.yaml`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The pproc sections a coupled row needs

A one-blade rotor whose family is `Blade1`:

```
[sections]
count = 50
include_symmetry = false
[[sections.distributions]]
families = ["Blade1"]
frame = "LOCAL_AXIS"
planes = ["XY"]
```

Takeaway *The loads export is one flat table; the family map tells the driver which rows are which blade.*

- Exactly one XY distribution per blade, emitted in blade order, each in that blade's own frame (span along Z).
- A wing: exactly one distribution, over its one family, on the XZ plane, in a frame the run creates with the reference axes and its origin on the wing's `origin_m` along the span.
- Several blades: `LOCAL_AXIS` expands the rotor's blades in the declared order; `blade_count` counts every explicit blade.
- Duplicate cuts, aggregated families, other planes or more distributions are refused, never guessed.

Source: pyflightstream [1]; `docs/fsi-workspace.md`, "Automatic workspace coupling"; FSI tutorial, "loads".

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 5

What is refused, and why

Refused at plan: the shape of the run

The automatic route is deliberately bounded: each of these refuses the row when it is planned, before it can run. The workflow is judged first.

The row	Why it is refused
turns a rotor (`unsteady_rotor`)	"FSI on unsteady_rotor is still in debug (the morph is applied to the un-rotated blade, reported to the vendor)" [2]
is a `qsteady_rotor` wheel	several clockings averaged are not the state of one structure; the coupled quasi-steady case is the sector
names a wing input on a rotor workflow, or a blade input on `steady` or `unsteady`	a fixed wing (`[config.wing]`) couples on `steady` and `unsteady`; a blade on the rotor route
continues a stopped run	the structural state and the section order of a continuation are not proved

Source: pyflightstream [1]; `cases/fsi_workspace.py`, `validate_workspace_fsi`, `fsi_workflow_refusal`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Refused at plan: the geometry and what the script may carry

The row	Why it is refused
opens a saved `.fsm`	"Workspace FSI requires a fresh mesh import": import the mesh, with its boundary sidecar
places nodes in a unit other than metres	other native node units have not been measured
carries a raw command or a custom flag	the section and frame order can no longer be proved
copies blades by symmetry, outside the `qsteady_rotor` sector	every coupled blade must be explicit, with its own sections
is a steady coupled row asking for probes, a volume section or loads selections	the script never starts the solver after the analysis, so they would be exported empty
puts a structural node outside the component	the interpolation cannot place a surface around a node that is not within it

Source: pyflightstream [1]; `cases/fsi_workspace.py`, `validate_workspace_fsi`; `fsi/nodes.py`, `refuse_nodes_outside_sections`; `cases/workflows.py`, `_refuse_what_a_steady_coupled_run_cannot_export`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Refused at plan: blades, frames and the speed

- **The count.** The structural `blade_count` must equal the rotor's blades, every one explicit.
- **The speed.** $\Omega = |\mathrm{rpm}|\,\pi/30$ must equal the row's rotor speed; on a `qsteady_rotor` row that is the speed of its free-stream rotation. A structure that does not turn has $\Omega = 0$ and carries its own weight instead.
- **The clock.** Where the structure states a time increment, it must equal the solver's time step.
- **The basis.** A rotor's shaft is X and its blade datum Z; a wing's sections are cut in a frame with the reference axes, its origin on the wing's `origin_m` along the span.
- **The mapping.** Blade, frame and boundary are one-to-one; there is exactly one section distribution per blade, in blade import order, each in its own frame with XY cuts, span along Z (a wing: one, XZ).

Geometry and structural pitch must describe the same blade: the adapter infers no basis and re-pitches no mesh.

Source: pyflightstream [1]; `cases/fsi_workspace.py`, `wire_workspace_fsi`, `wire_fixed_wing_fsi`, `validate_workspace_fsi`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Refused while reading the input

The input	The message says
a factor key the package does not know	"unknown FSI matrix keys [...]; use FSI_GEOMETRIC_PITCH_DEG_FACTOR, ..."
factors on a row with no `FSI`	"matrix calibration states no FSI input; add FSI: f001 in VAR_NAMES_VALUES."
a code that is not `f` + name	"FSI='...'; name an f-prefixed code without an extension, such as f001."
a zero or negative factor	"... use a finite positive factor."
a stiffness scale other than 1 in `[config]`	"... config.stiffness_scale_factor must be 1.0 in workspace inputs; calibrate bending_stiffness_n_m2 and torsion_stiffness_n_m2 explicitly ..."
material factors in supplied mode	"supplied mode cannot apply material factors [...] to already supplied distributions; calibrate the distribution itself."
a blade table in calculated mode	"calculated mode derives config.blade; remove the supplied blade table."
a material the database lacks	"no material '...' in the materials database (version 1); it holds: al-7075-t6, ti-6al-4v-grade5-annealed. ..."

Source: pyflightstream [1]; `workspace/fsi_setup.py`, `resolve_row_fsi`, `resolve_fsi_setup`; `fsi/materials.py`, `material`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 6

Worked examples

Example 1: calculated and supplied inputs agree

`docs/examples/_sources/workspace_fsi_calibration.py`, offline, no solver:

1. Writes `inputs/fsi/f_calculated.toml` from the package template (the titanium strip), with the file factor 1.05 on EI.
2. Resolves it and reads the *unscaled* base distributions.
3. Writes `f_supplied.toml` with those base distributions and the same file factor 1.05. Reusing the effective values here would apply the calibration a second time.
4. Resolves both with `FSI_BENDING_STIFFNESS_N_M2_FACTOR: 1.15`.
5. Asserts every distribution equal across the two modes, effective EI equal to base EI times 1.15, and the origin `matrix`.

```
python workspace_fsi_calibration.py /absolute/path/to/new-
      workspace
```

Printed	Value
base EI, each station	24.277 N m^2
effective EI, each station	27.919 N m^2
file factor, matrix factor	1.05, 1.15
`factor_origin`	`matrix`
`mode_equivalence`	verified
`solver_executed`	`false`

Check it by hand

$I = bh^3/12 = 0.04 \times 0.004^3/12 = 2.133 \times 10^{-10}$ m^4; $EI = 113.8$ GPa $\times\, I = 24.28$ N m^2; $\times 1.15 = 27.92$, not $\times 1.05 \times 1.15$.

Choose a new destination: the example refuses to overwrite its two inputs.

Source: pyflightstream [1]; `docs/examples/_sources/workspace_fsi_calibration.py`, run at the tag; `workspace/fsi_setup.py`, `FSI_TEMPLATE`.

Example 2: one section, and its torsion constant

`docs/examples/_sources/fsi_solid_blade_properties.py` builds a synthetic blade: seven stations from 0.15 to 0.75 m, chord 80 to 40 mm, pitch 35 to 11 deg, NACA 2415 at the root thinning to NACA 2409 at the tip, written to a section file and read back, the file's sha256 into the provenance.

Root section, NACA 2415	
area	654.144 mm^2
centroid from the pitch axis	−13.422 mm chordwise
I flap, I chord	5512.2, 227784.7 mm^4
principal angle	−0.100 deg

Grid, cells across	J [mm^4]
16	20926.02
32	20956.81
64, the default	20962.52
thin-section estimate	21721.14

- Doubling the cells divides the error by about five (the change falls from 30.79 to 5.71): at 64 cells about a quarter of the change from 32 to 64 is left.
- The thin-section formula is 3.6 % high here: a cross-check, never the value.

Source: pyflightstream [1]; `docs/examples/_sources/fsi_solid_blade_properties.py`, run at the tag.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Example 2: the whole blade, in titanium

r [m]	μ [kg/m]	EI [N m^2]	GJ [N m^2]	I_1 [kg m]	I_2 [kg m]	e_c [mm]	thin/J
0.15	2.8979	627.29	922.35	1.009e-03	2.442e-05	−13.42	1.036
0.25	2.2727	361.01	531.76	6.650e-04	1.405e-05	−12.30	1.032
0.35	1.7441	198.04	291.97	4.217e-04	7.708e-06	−11.19	1.028
0.45	1.3040	102.60	151.24	2.554e-04	3.993e-06	−10.07	1.024
0.55	0.9445	49.58	72.98	1.462e-04	1.930e-06	−8.95	1.020
0.65	0.6574	21.98	32.24	7.789e-05	8.553e-07	−7.83	1.017
0.75	0.4347	8.73	12.72	3.784e-05	3.395e-07	−6.71	1.014

- Ti-6Al-4V grade 5, annealed: ρ 4430 kg/m^3, E 113.8 GPa, G 44.0 GPa, ν 0.342, database version 1.
- The configuration round-trips through `config.json` unchanged, with its provenance and its `config_sha256`.
- e_c: the centroid ahead of or behind the pitch axis, which the elastic axis is assumed to follow.

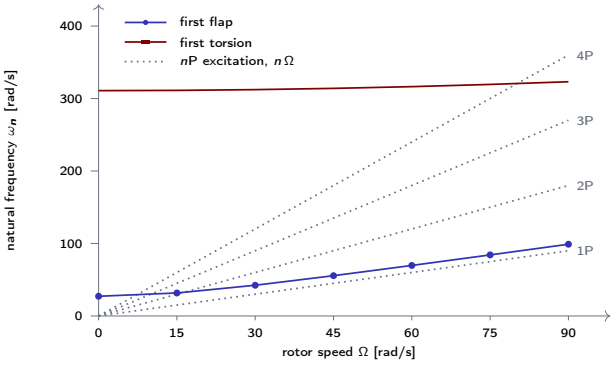
At rest, from the beam

first flap: 30.6 Hz
first torsion: 601.9 Hz

Source: pyflightstream [1]; `docs/examples/_sources/fsi_solid_blade_properties.py`, sections 3 and 4, run at the tag.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Example 3: the Campbell diagram of a rotating blade



A uniform synthetic blade, 1 m, 15 stations, swept from 0 to 90 rad/s. At every speed: static solve under its own centrifugal loads, then the modes with the tension's geometric stiffness.

Southwell fit	**flap**	**torsion**
ω_0 [rad/s]	27.96	310.90
S	1.118	0.961
r^2	0.99986	1.00000

$\omega_n^2 = \omega_0^2 + S\,\Omega^2$. Flap S sits in the expected 1.1 to 1.3; torsion near $(I_1 - I_2)/(I_1 + I_2) = 0.961$. A crossing of a track and a ray flags a potential resonance [10, 11].

[REF-54], [REF-55]

Source: pyflightstream [1]; `docs/examples/_sources/fsi_campbell_diagram.py`, run at the tag; FSI tutorial, "centrifugal".

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 7

Outputs, checklist and limits

Outputs, all in the point's working directory

File, written at every call	What it holds
`FS_SurfaceSection_Loads.txt`	by the solver: the section loads, computed in Newtons, line densities along the span; the reader asserts units and completeness
`FSIDisp.txt`	one translation per node, in metres, blade frame, import order
`state.json`	the structural state, written atomically, so a killed call resumes
`fsi_convergence_log.csv`	the validity boundary in its header, `config_sha256` on every row
`fsi_archive/call_NNNN/`	that call's loads and displacements, for offline replay
`pyfs_fsi_calls.log`, `_error.log`	a line per call; on failure the traceback, and a non-zero exit, whose effect on the solver has not been measured

- `pyfs-fsi step --dir /absolute/path` runs one driver call when current loads and maps already exist; stale solver iterations are refused.
- Changing a calibration requires a new resolved point and regenerated nodes.

Source: pyflightstream [1]; `docs/fsi-workspace.md`, "Automatic workspace coupling"; `fsi/cli.py`; `fsi/driver.py`; FSI tutorial, "loads", "driver and state", "cli".

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Before planning an FSI row

1. **The input**: `inputs/fsi/f<id>.toml` in ONE mode, SI units; calculated mode names a database material or a fully sourced one, and closed simple contours in metres about the pitch axis.
2. **The row**: `WORKFLOW` `steady`, `unsteady`, or `qsteady_rotor` with periodic symmetry (the sector); `FSI: f<id>` and any `FSI_*_FACTOR` in `VAR_NAMES_VALUES`; no `RAW`, no custom flags.
3. **The geometry**: a fresh OBJ or STL import with its boundary sidecar, in `METER`; never a saved `.fsm`.
4. **The rotor**: one isolated, axisymmetric rotor; shaft X, blade datum Z; `blade_count` equal to its blades; `omega_rad_per_s` equal to the row's free-stream rotation. A wing: `[config.wing]`, $\Omega = 0$, `blade_count = 1`.
5. **The pproc**: a rotor uses one XY distribution per blade, `LOCAL_AXIS`, in blade order, `include_symmetry = false`. A wing uses its spanwise distribution in XZ in the wing frame; follow the wing example rather than copying the blade shape.
6. **Then** `pyfs-matrix plan`, where the refusals of this deck fire; after the run, `fsi-provenance.json` in the point's directory gives each factor's value and origin.

Source: pyflightstream [1]; `docs/fsi-workspace.md`; `cases/fsi_workspace.py`; `workspace/fsi_setup.py`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Limits: what the coupling does not establish

- **Wiring is not validation**: the staging and callback checks do not establish native two-way rotor deformation, convergence or aerodynamic validity.
- **The native evidence so far** (RPT-093 [2]): on 26.124 a blade held still in a turning free stream, steady, mapped every surface vertex and exported the imposed flap to 0.1 mm, its rigid axial force within 0.35 % of the same blade turning; a fixed wing mapped and exported the bend, steady and unsteady. A turning blade is morphed at its imported azimuth, which is why `unsteady_rotor` with FSI is refused. These are imposed-displacement probes, not coupled accuracy.
- **Open**: saved-simulation and continuation use, and native coupled validation; staging or command acceptance implies neither.
- **The zero-speed example inputs** are for offline checks, not runnable coupled rotor rows.

The package's own capability table

FSI beam and modal analysis: experimental. The coupled driver: experimental, replayed offline on archived fixtures, never run against a live solver in CI. Rotary two-way coupling: not validated.

Source: pyflightstream [1]; `docs/fsi-workspace.md`, "Automatic workspace coupling"; `README.md`, the capability status table; `docs/srs/architecture-srs.md`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Limits: the structural model

- **An Euler beam**, one per blade, in flap and torsion (w, θ): only those two carry mass. This input adds no shell, spar, hollow-section, shear-centre or second structural model; hollow and spar sections are a future option, not built.
- **Solid, homogeneous sections** from ONE material, when calculated.
- **The elastic axis at the centroid**; the shear centre is not computed. A generated blade has a zero CG offset, so the bend-twist coupling that offset drives is absent.
- **No trapeze effect**: the elastic twist is overpredicted, which is conservative.
- **Quasi-static**: the once-per-revolution content is trustworthy only well below resonance, $n\,\Omega/\omega_n \lesssim 0.3$.
- **Sources**: the formulas are transcribed from the coupling plan, not re-checked against the primary papers; the materials are typical values, not design allowables.

Source: pyflightstream [1]; `docs/fsi-workspace.md`; FSI tutorial, "The loop in one paragraph", "sections and materials", "beam", "centrifugal"; `fsi/materials.py`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The physics behind the coupling, and where to read it

- **The beam**: Euler bending and Saint-Venant torsion of a slender member, and the section constants of a solid contour [6, 7].
- **The rotating blade**: centrifugal tension and stiffening, the propeller moment toward flat pitch, the in-plane softening, and the natural frequencies that rise with the speed [8, 9, 10].
- **The Campbell diagram**: the blade's frequencies against the rotor speed, beside the lines of the rotor's harmonics [11].
- **Quasi-static coupling**: the unsteady aerodynamics of an oscillating section, and the reduced frequency below which a steady answer holds [12, 13].
- **The interpolation**: radial basis functions from a set of structural nodes to an aerodynamic surface, and the multiquadric kernel [14, 15, 16].

Copyright (c) 2026 Geovana Neves, CC BY 4.0

References

[1] Geovana Neves, *pyflightstream documentation*, page "FSI in a workspace". `https://nevesgeovana.github.io/pyflightstream/latest/fsi-workspace/`; and the FSI tutorial, `src/pyflightstream/fsi/README.md` of the repository.

[2] Geovana Neves, "The Aeroelastic Coupling Toolbox on 26.124: nine measured facts", report RPT-093, 2026, `reports/` of `https://github.com/nevesgeovana/pyflightstream`.

[3] Geovana Neves, "The quasi-steady rotor against the unsteady rotor, measured", report RPT-089, 2026, `reports/` of `https://github.com/nevesgeovana/pyflightstream`.

[4] Geovana Neves, *pyflightstream: measurement records of the solver's behaviour*, 2026. `https://github.com/nevesgeovana/pyflightstream`, concept DOI `10.5281/zenodo.21482924`.

[5] D. C. Brinck, *PyNite*: a 3D structural finite element library for Python. `https://github.com/JWock82/Pynite`

REF-08 S. P. Timoshenko, J. N. Goodier, *Theory of Elasticity*, 3rd ed., McGraw-Hill, New York, 1970.

REF-52 W. D. Pilkey, *Analysis and Design of Elastic Beams: Computational Methods*, Wiley, New York, 2002.

REF-53 D. H. Hodges, E. H. Dowell, "Nonlinear equations of motion for the elastic bending and torsion of twisted nonuniform rotor blades", NASA TN D-7818, 1974.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

References (continued)

REF-05 R. L. Bielawa, *Rotary Wing Structural Dynamics and Aeroelasticity*, AIAA Education Series, AIAA, Washington, 1992.

REF-54 R. V. Southwell, B. S. Gough, "On the free transverse vibration of airscrew blades", Aeronautical Research Committee, Reports and Memoranda 766, 1921.

REF-55 W. Campbell, "The protection of steam-turbine disk wheels from axial vibration", *Transactions of the ASME*, 46, pp. 31-160, 1924.

REF-56 R. L. Bisplinghoff, H. Ashley, R. L. Halfman, *Aeroelasticity*, Addison-Wesley, Reading, 1955.

REF-34 T. Theodorsen, "General theory of aerodynamic instability and the mechanism of flutter", NACA Report 496, 1935.

REF-47 R. L. Hardy, "Multiquadric equations of topography and other irregular surfaces", *Journal of Geophysical Research*, 76(8), pp. 1905-1915, 1971.

REF-44 A. Beckert, H. Wendland, "Multivariate interpolation for fluid-structure-interaction problems using radial basis functions", *Aerospace Science and Technology*, 5(2), pp. 125-134, 2001.

REF-45 A. de Boer, M. S. van der Schoot, H. Bijl, "Mesh deformation based on radial basis function interpolation", *Computers and Structures*, 85(11-14), pp. 784-795, 2007.

Copyright (c) 2026 Geovana Neves, CC BY 4.0