Guide 1: FlightStream and pyfts, an overview

What the solver does, how a simulation is built, and where each guide takes over

Written for pyflightstream 0.58.2.

Sources: the FlightStream user manual, the panel-method literature, and the package pages "The workspace and the workflow" and "Storage and sync" [1, 2, 6, 7, 8, 10].

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.

Who reads what

You are new to FlightStream

- Part 1: what the tool is, and what it will not do.
- Part 2: how a surface vorticity panel solver works, steady, unsteady and rotating.
- Part 3: the simulation workflow, stage by stage.

You know FlightStream, and meet pyfs

- Part 4: how pyflightstream carries each stage: the matrix, the references, the setups, the pproc, run and post, the four workflows, storage and sync.
- Part 5: the other eight guides, and when to read each.

Takeaway *This deck orients; it defines nothing. Every key, command and refusal it names is defined in the guide it points to.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Contents

FlightStream

How a panel solver works

The simulation workflow

pyflightstream on the workflow

The guides

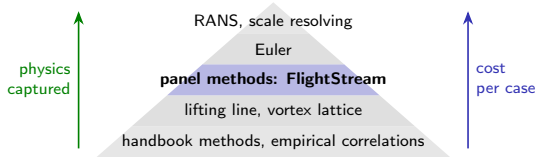
Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 1

FlightStream

Where a panel method sits

- Aerodynamic tools form a **fidelity ladder**: each level resolves more physics and costs more per case.
- A panel method sits in the middle: the **whole configuration** in three dimensions, in seconds to minutes per point on a workstation.
- That is the regime of **interference and installation**: where a propeller sits, what a fuselage does to a wing, a pitching moment slope.



Takeaway *A panel method buys full-configuration aerodynamics at a cost that still allows design iteration.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

What FlightStream is

- A **surface vorticity** panel solver: the potential-flow problem, with vorticity carried on the surface mesh [7, 8]. It tolerates [REF-21], [REF-27] unstructured, engineering-quality meshes and reads the common exchange formats [6, pp. 72-75].
- The core is **inviscid**. The rest is correction models on top of it: a boundary layer, with optional viscous coupling, for friction drag and decambering [6, pp. 204-211]; subsonic compressibility, chosen at initialisation [6, pp. 218-221], [12]. [REF-31]
- Steady, unsteady and rotating solutions, driven from a graphical interface or from a **script** of commands.

Check the validity envelope

- Deep stall and buffet need separate validation.
- Transonic options need build-specific validation.
- Acoustics needs the separate toolbox and acoustic exports.

And it does not warn you

A case that leaves the validity envelope still converges and still prints coefficients.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The build number is part of the answer

- FlightStream is released as **builds**; this series names build 26.123 in its examples. The scripting language changes between builds of one version: a command that runs on one build may be absent, renamed or spelled differently on another.
- The manual ships inside each build, and **editions are not page aligned**. A page number is quoted with its edition, or not at all.
- A coefficient without the build that produced it is **not reproducible**. Record the build next to every result.

How pyflightstream holds this

Every command it writes comes from a **command database per build**, and every row of a study names its build in the column `FS_BUILD`. `plan` checks the whole study against that build's database before any seat is spent (Part 4, Guide 2).

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The tool's own view of the workflow

- The manual closes with three checklists: the **general** workflow [6, p. 411], the workflow from **CAD cross sections** (CCS) [6, p. 412], and the **rotor and turbine** workflow [6, p. 413].
- The general one has five steps: geometry preparation, half-model setup, surface detection, boundary conditions, simulation setup.
- Boundary conditions come **before** anything a user would call setup. Part 3 explains why: initialisation locks them.

A half model is expected

The manual recommends the half model under a mirror plane, and free edges along that plane are **expected**, not a defect [6, p. 411].

A reference case ships with it

A rotor in hover and cruise [6, p. 401]: the fastest check that an installation behaves.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The method, and where it has been measured

What it solves

- The integrated-circulation formulation, in full [7]. [REF-27]
- The surface vorticity panel method in the archival literature: the reference to cite when someone asks what FlightStream solves [8]. [REF-21]
- The classical panel-method background: [9, 10]. [REF-28], [REF-29]

Where it has been compared with experiment

- Stability and control of a vehicle against wind-tunnel data: derivatives agree acceptably, control effectiveness is overpredicted, propellers off [13]. [REF-32]
- A high-lift wing with distributed propulsion, against experiment: the closest published case to a powered airframe [14]. [REF-33]

Takeaway *Convergence against a theoretical anchor is verification; agreement with measured data is validation. Say which one a result shows.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 2

How a panel solver works

Potential flow and its boundary conditions

- Assumptions: **incompressible, irrotational, inviscid**. The velocity derives from a potential, $V = \nabla\phi$, and mass conservation becomes the Laplace equation, $\nabla^2\phi = 0$ [9, 10]. [REF-28], [REF-29]
- The equation is **linear**: elementary solutions superpose, and the flow about a body is assembled from them.
- Two conditions close it: **flow tangency** on the surface, and **decay** of the disturbance far away.

What is absent

No volume grid, so **no farfield to size** and no blockage to correct: the disturbance decays because the elementary solutions decay [6, p. 164].

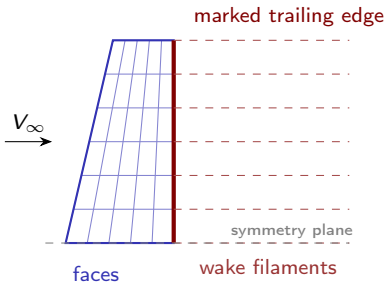
What it can predict

Pressure effects: lift, induced drag, moments, interference. Friction drag does not come from the potential-flow solution; it comes from the optional boundary-layer correction.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

From source and doublet panels to surface vorticity

- A classic panel method gives each surface panel a source or doublet strength, enforces tangency at one point per panel, and solves a dense linear system [9, 10]. [REF-28], [REF-29]
- FlightStream carries **vorticity** on the surface mesh instead [7, 8]. Degenerate faces are removed in preprocessing, and [REF-21], [REF-27] flow-aligned quadrilaterals are preferred on lifting surfaces [6, pp. 75, 156].
- Whatever the singularity: **surface resolution drives the induced velocities**, and cost grows steeply with the panel count.



Takeaway *Different singularity, same physics: enforce tangency, resolve the circulation, integrate the pressures.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The Kutta condition and the wake

- Potential flow alone **does not fix lift**: any circulation satisfies the equations. The physical statement that selects the real solution is smooth flow leaving a sharp trailing edge [10, 11]. [REF-29], [REF-30]
- In the solver, **marking a trailing edge** is that statement: wake filaments are shed from marked edges, and an unmarked edge sheds nothing at all [6, p. 165].
- The wake carries the trailed vorticity downstream. Its strength mirrors the spanwise load gradient, and **induced drag** is the price of that vorticity.

The most critical input

An unmarked trailing edge raises no error. It returns a converged solution with the wrong lift; the manual calls the trailing edges by far the most critical input [6, p. 164].

Copyright (c) 2026 Geovana Neves, CC BY 4.0

How much circulation is shed: the factor K

- The strength of the wake filament leaving a marked edge is

$$\Gamma_{s_t} = \frac{K}{2}\left(q_u^2 - q_l^2\right)\Delta t,$$

  with q_u and q_l the velocities above and below the filament [6, p. 165].
- **Standard Kutta**, $K = 1$: the full net circulation is shed. The default, and the right choice at any real lifting trailing edge.
- **Relaxed Kutta**, $K = 0.5$: half is shed. For an edge that is not sharp, typically the aft end of a rounded fuselage, so that the body carries an equivalent lifting load rather than none.
- Two more types exist: exhaust outflow, and vortex shedding, which has no vortex-bursting model and is not recommended beyond 25 to 30 degrees of incidence [6, p. 165].

Takeaway *Standard against relaxed is not a numerical knob: it states whether an edge is a real trailing edge.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

When a trailing edge is not a Kutta edge

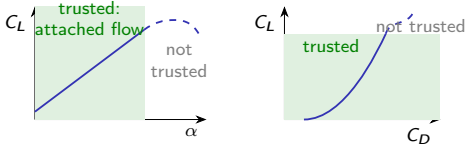
- A **blunt** trailing edge is a separated base: the solver applies a constant pressure over a **base region** [6, p. 164].
- Rule of thumb: thicker than **2 percent of the chord**, keep it blunt with a base region, and mark the base perimeter as trailing edges [6, pp. 164, 167]. Thinner, collapse it to a sharp edge: no loss of accuracy, faster and cleaner convergence.
- A body protruding through a wing trailing edge by a length L, on a chord c: the carry-over decision turns on $L > c/2$ [6, p. 166].

Taken in CAD

Collapsed or blunt is decided **before the mesh is made**, and it governs every stage after it. Part 3 places it in the geometry stage.

What the method captures, and what it does not

- **Inviscid core**: lift slope, induced drag, spanwise loading, thickness effects, interference between components.
- **Correction models**: friction drag by the boundary layer, decoupled by default, coupled through transpiration for nonlinear decambering [6, pp. 204-211]; subcritical Mach effects by Prandtl-Glauert [6, pp. 218-221], [12]. [REF-31]
- **Not captured** under any setting: deep stall, buffet, shock stall.

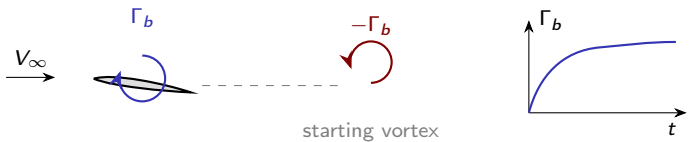


Takeaway *A polar is trustworthy where the flow is attached, and nothing in the output says where that ends.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Unsteady: the wake remembers

- **Kelvin**: in inviscid flow the circulation of a material contour is conserved. When the bound circulation changes, an equal and opposite vortex is shed into the wake [10]. [REF-29]
- The **starting vortex** is the canonical picture: a wing set impulsively in motion leaves behind a vortex that mirrors the circulation it gains.
- The wake stops being a fixed sheet and becomes part of the solution: it carries the history of every change of lift, and induces velocity back on the wing.



Copyright (c) 2026 Geovana Neves, CC BY 4.0

The time step is a mesh dimension

- The solver marches in **physical time**: each step runs a block of iterations under the steady settings, and the wake grows one shed segment per step [6, pp. 212-213].
- The streamwise length of a shed segment is the convection velocity times the step: **the step sets the wake resolution** as the panel size sets the surface resolution. Choose it from a physical time the geometry has, for example the chord's convective time $t_c = c/V$.
- The judge is the **reduced frequency** [15, 10]: [REF-29], [REF-34]

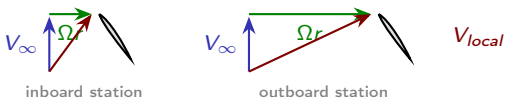
$$k = \frac{\omega c}{2V_\infty}.$$

  At low k the flow adjusts quasi-statically; as k grows, wake memory and phase lag become the answer.

Takeaway *Run unsteady only when the question demands it: gusts, oscillation, dynamic derivatives, rotors. A slow angle sweep is a sequence of steady points.*

Rotation added to the unsteady problem

- Each blade section sees the sum of Ωr and the free stream; the inflow angle $\varphi = \arctan\left(V_\infty/(\Omega r)\right)$ varies along the radius, which is why blades are twisted [16, 17]. [REF-35], [REF-36]
- Blade geometry and section loads live in the **rotating** frame; the wake, the airframe and the performance numbers in the **inertial** one. Name the frame when you report a number.
- Everything unsteady still holds. Rotation adds geometry to the wake, not new principles.



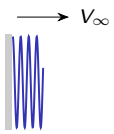
Propeller performance, and the helical wake

$$J = \frac{V_\infty}{nD}, \qquad C_T = \frac{T}{\rho n^2 D^4},$$
$$C_P = \frac{P}{\rho n^3 D^5}, \qquad \eta = \frac{J C_T}{C_P}$$

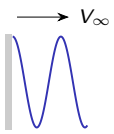
n in revolutions per second, D the diameter [16, 17] [REF-35], [REF-36]. At fixed J the non-dimensional picture is fixed; the rotation rate alone only rescales the Reynolds and tip Mach numbers.

The solver reports forces

It reports forces and moments normalised on the reference values you set; J, C_T, C_P and η are built by you, or by the post-processing (Guide 7).



low J: tight helix



high J: open helix

- The wake is a **helix** whose pitch is set by J: a low J packs the turns behind the disc, and each blade meets the wake of the others.
- The clock is the shaft: resolution in degrees per step, about 10 to 12 degrees, run length in revolutions, reference velocity the tip speed [6, p. 413].

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 3

The simulation workflow

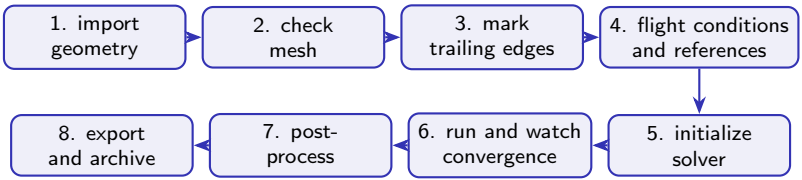
Ten stages, in the order decisions constrain each other

#	Stage	The decision it owns
1	Case study and scope	what question the case answers, and what it does not
2	Matrix definition	which points are run, and why those
3	Water-tight geometry	blunt or sharp trailing edge, and where the components split
4	Surface mesh	where resolution is spent
5	Boundary conditions	which edges shed vorticity, and how much
6	Solver setup	what the solver is allowed to assume
7	Execution and monitoring	whether the answer has converged
8	Output definition and export	what evidence the run leaves behind
9	Verification and validation	whether the answer is right
10	Engineering analyses	what the answer means for the design

Takeaway *The stages are not a checklist to tick: a decision taken out of order has to be taken again.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

One session in the interface, and the lock in it



- **Initialisation is a lock.** It freezes the solver against the mesh and the boundary conditions as they stand, and fixes the solver mode and the symmetry claim [6, pp. 218-221].
- A runtime setting (angle, velocity, iterations) can change after it. A mesh or boundary change needs the initialisation removed and taken again. That is why boundary conditions precede setup [6, p. 411].

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Stages 3 and 4: geometry and surface mesh

Geometry

- **Water-tight** means the body closes, not the domain.
- Collapsed or blunt trailing edge: decided in CAD (Part 2).
- Components are split by the **boundary condition they will receive**, and by the loads you will want per component, never by CAD convenience.

Surface mesh

- Four sources: the tool's cross-section meshers, transfinite interpolation, the wrapper, or an imported mesh [6, pp. 72-75]. Only the first two regenerate in seconds.
- Faces go where the gradients are: leading edge, tip, junction, the row the wake leaves from [6, p. 62].
- A mesh convergence study is an obligation, and it is affordable.

Takeaway *The mesh carries the boundary conditions. A boundary is a decision about what you will be able to ask for later.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Stage 5: boundary conditions are marking

- **Farfield**: there is none to set. **Walls**: every face is a slip wall once meshed, switched to no slip by the solver when the boundary layer is active [6, p. 164].
- What remains is **marking** where that default is wrong: trailing edges, with their type (Part 2); base regions; wake termination nodes; and, when the case needs them, actuator discs, inlets and outlets, custom free-stream fields [6, pp. 164-197].

Auto-detection is a proposal

The solver can detect trailing edges; it proposes, you decide. Inspect every marked edge before initialisation: a split trailing edge is two edges, and a missing one is silent.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Stage 6: what the solver may assume

Symmetry	What it asserts
None	nothing: the mesh in front of the solver is the whole problem
Mirror	a symmetry plane in the flow, not only in the geometry; false as soon as the sideslip is not zero
Periodic	an onset flow the same at every azimuth, plus a copy count; false as soon as the angle of attack is not zero

- **Coordinate systems first**: loads are reported in one, and a moment without its frame and point is not a number.
- **Reference velocity, area and length** normalise every coefficient. Set them; never inherit them.
- **Convergence is three controls**: an iteration cap, a residual threshold, and how long the residual must stay below it [6, pp. 202-203]. Viscous settings change the answer; leaving them off is a decision too.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Stages 7 and 8: run, watch, and declare the outputs

Execution and monitoring

- **Converged is a state that was held**, not a number that was touched. A run stopped at its cap ran out of iterations.
- Unsteady asks twice: the residual inside each step, and the history of the quantity you want. They are independent.
- Periodic: judge the **per-revolution** statistic, not the sample.

Output definition and export

- **An output not declared was not computed**: sections, probes and plots are set before the run.
- Symmetry-load doubling changes what the solver computes. Two runs set differently are not the same quantity.
- Every coefficient carries a normalisation: record it with the number.

Takeaway *Recovering a variable that was never declared means running the case again, which for a rotor is the most expensive mistake available.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Stages 9 and 10: is it right, and what does it mean

- **Verification**: the solution against a theoretical anchor and against itself. A finite-wing lift slope, a span efficiency, a mesh and a time-step study, a sector against the full wheel.
- **Validation**: the solution against measured data, inside the envelope where the method has been compared with experiment [13, 14]. [REF-32], [REF-33]
- An error that scales the whole solution divides out of a ratio. An efficiency can agree while thrust and torque are both off: check the forces, not only their ratio.
- **Engineering analyses** use the answer: derivatives, trim, installation deltas, component breakdowns. They inherit every assumption of stages 3 to 8.

Takeaway *A mid-fidelity number is defensible when its build, its settings, its convergence and its verification travel with it.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 4

pyflightstream on the workflow

What pyflightstream is

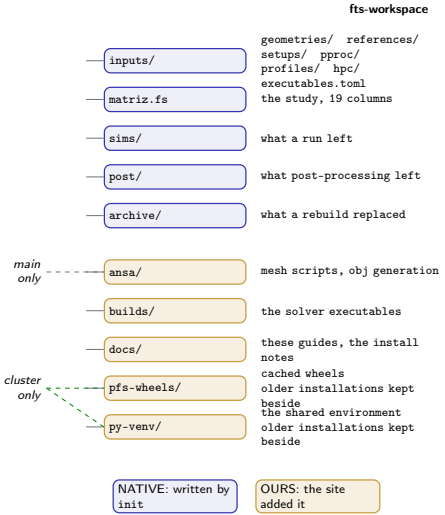
- A Python driver for FlightStream, MIT licensed, installed from PyPI (`pip install pyflightstream`). FlightStream itself, and its licence, are separate.
- It **writes** the solver script from a command database per build, **runs** it, **reads** what the solver exported, and **post-processes** it into tables with their provenance.
- A study is a **workspace**: a folder with a run matrix and the input files the matrix cites. No Python is written to run it.



Takeaway *A row names things, and the package makes the link: a row never carries a path.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The workspace

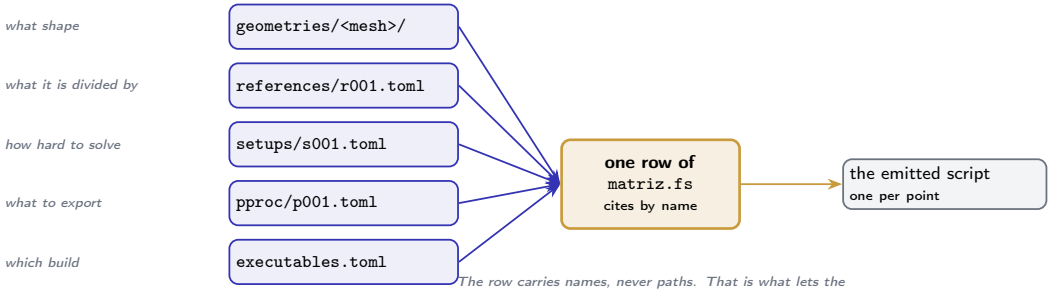


The ten stages, and where each one lives

Stage	In the workspace	Guide
1, 2 Scope and matrix	one row of `matriz.fs` per study point or sweep: `DESCRIPTION`, `FLIGHT_CONDITION`, `SWEEP_VALUES`, `RUN`	2
3, 4 Geometry and mesh	made outside the package; the file sits in `inputs/geometries/`, the row names it in `GEOMETRY`; `pyfs-mesh inventory` reads its boundaries	2, 3
5 Boundary conditions	the trailing edge by file or by detection, base regions and wake nodes as keys; rotors and actuator discs in the reference file	3, 5
6 Solver setup	`inputs/setups/s<id>.toml` (column `SET`); lengths, moment point and frames in `inputs/references/r<id>.toml` (`REF`); `SYMMETRY` and `WORKFLOW` on the row	5, 6
7 Execution	`plan`, `run`, `collect`; `FS_BUILD`, `NCPUS`, `WALLTIME` on the row	2, 4
8 Output definition	`inputs/pproc/p<id>.toml` (`PPROC`): what the run exports, declared before it runs	6, 7
9, 10 Verification and analyses	`post` writes `post/`: polars, series, sections, probes, provenance; FSI couples a structure	7, 8

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The row names things, the files define them



Takeaway *`REF`, `SET` and `PPROC` carry a file stem and `GEOMETRY` names the mesh: stated once, in one file, and every row citing it demonstrably used it.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Reference, setup, pproc: three files a row cites

`r<id>.toml`

What the coefficients divide by, and about which point: reference area and lengths, the moment point, aliases for groups of surfaces, custom frames, and one block per rotor or actuator disc (Guide 5).

`s<id>.toml`

How hard to solve and when to stop: iterations and convergence, the boundary layer and its coupling, the model and its initialisation, wake settings, and a raw table for any command without a key (Guide 6).

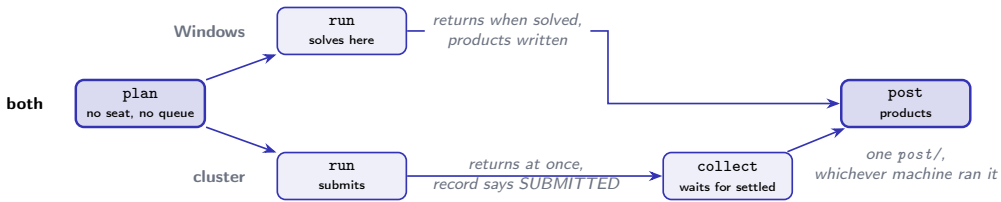
`p<id>.toml`

What the run exports, and how post reads it: loads, sections, probes, plots, the log last; the groups and derived columns post reduces with. The averaging window belongs to the row (Guides 6, 7).

- The flight condition is a cell of the row, not a file: Mach, speed, Reynolds number, altitude, a temperature offset, each a constraint on one flow state [1].
- Any key only some rows need goes in the row's free cell, `VAR_NAMES_VALUES`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Four stages of a run




```
pyfs-matrix plan    matriz.fs --workspace . --fs-version 26.123   # no seat, no queue
pyfs-matrix run     matriz.fs --workspace . --fs-version 26.123   # solves, or submits
pyfs-matrix collect --workspace .                                 # cluster: settled outputs
pyfs-matrix post    matriz.fs --workspace .                       # rebuilds the products
```

`plan` validates every row against the build's command database and writes the receipt `run` refuses to start without (Guide 2).

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Four workflows, one per kind of physics

WORKFLOW	What is solved	The physics of Part 2	Cost
steady	one converged steady solve per point; a sweep is one job	the steady problem: tangency, Kutta, the fixed wake	seconds per point
unsteady	a march in physical time, nothing turning	Kelvin and the starting vortex; the step as a wake dimension; a window to average	minutes
unsteady_rotor	the blades turn, the wake is shed in time	the helical wake; degrees per step, revolutions, a per-revolution window; the reference for any rotor	minutes to hours
qsteady_rotor	blades held still, the free stream turning about the shaft; sector or wheel	an isolated, axisymmetric rotor where the reduced frequency is small	seconds

- The row names its workflow; a key another workflow owns is refused by name at `plan`. Costs are orders of magnitude on one workstation [20].
- Guide 2 carries the decision chart, the quasi-steady rotor in depth, and when its answer stands for the turning blade.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Storage and sync

What a run leaves

- `sims/`: one folder per matrix row, `sim_<POL>`, with the scripts and one datapoint folder per point: the saved simulation and the solver's exports.
- `post/`: the products, rebuilt by `post` with no seat.
- `archive/`: whatever a rebuild replaced.
- `runs.json`: the record of every run, with its state.

Two workspaces, one main

- A workstation workspace for planning, one point, looking at results; a cluster workspace for long marches and many points.
- `pyfs-workspace sync`, run on the main, brings the cluster's runs home at a chosen level; it previews unless told `--apply`, and never copies a mesh [2].

Takeaway *You never edit anything under `sims/` or `post/`. If a number there is wrong, an input is wrong.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 5

The guides

Nine guides, and when to read each

	File `pyfts-guide-`	What it covers	Read it when
1	`01-fts-overview`	the solver, the workflow, the map	before anything else
2	`02-workspaces`	the tree, the inputs, choosing a workflow, running, two workspaces and sync	you set up or run a study
3	`03-gui-to-pyfs`	every step of a GUI session, and the key that takes it	you know the interface
4	`04-cheatsheet`	every command and option, then one page per stage	you need a command, not the explanation
5	`05-references`	lengths, the moment point, aliases, frames, rotors, actuator discs	you write a reference file
6	`06-solver-setup`	the setup file, its keys and the solver commands behind them	you tune how a point is solved
7	`07-pproc-definitions`	what every product, reduction and window means	you read or plot a result
8	`08-fsi`	the fluid-structure loop, the structural input, calibration, limits [22]	a structure deforms under load
9	`09-python-environment-offline`	Python, a virtual environment and the package with no internet	once per offline machine

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Four ways through them

A first study

1, then 2 (Parts 1 to 11), then 5 and 6 for the files you cite, then 7 before you plot, with 4 beside you. The package page "Getting started" is the shortest path to a first run [21].

From the interface to the workspace

1 (Parts 3 and 4), then 3 side by side with your own session, then 6.

A rotor

1 (Part 2, rotation), 5 (the rotor block), 2 (choosing a workflow and the quasi-steady rotor), 7 (per-blade, phase-locked and window products); 8 for a blade that deforms.

Installing and operating

9 on each machine without internet, then 2 (the cluster, two workspaces and sync, space on disk).

Takeaway *Each guide is written for one version of the package; its title page says which.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Where to practise, and how to cite

A public practice geometry

- **SMI**: an airframe geometry, wing and wing-body, published for practice [18].

Where to take a question

Solver behaviour to the vendor, with the build and the page. The driver to the repository, as an issue.

Citing the tool

Geovana Neves. *pyflightstream: version-aware, didactic Python driver for the FlightStream panel-method solver*, 2026. `https://github.com/nevesgeovana/pyflightstream`. Concept DOI: `10.5281/zenodo.21482924`.

The Python library

For scripts written in Python rather than a matrix: the library guide, `pyflightstream_user_guide.tex` in the repository's `guide/` folder.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

References

[1] Geovana Neves, *pyflightstream documentation*, page "The workspace and the workflow". https://nevesgeovana.github.io/pyflightstream/latest/workspace-and-workflows/

[2] Geovana Neves, *pyflightstream documentation*, page "Storage and sync". https://nevesgeovana.github.io/pyflightstream/latest/storage-and-sync/

[3] Geovana Neves, *pyflightstream documentation*, page "From the GUI to pyfs". https://nevesgeovana.github.io/pyflightstream/latest/gui-to-pyfs/

[4] Geovana Neves, *pyflightstream documentation*, page "Post-processing definitions". https://nevesgeovana.github.io/pyflightstream/latest/post-processing-definitions/

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

[6] Altair Engineering, *FlightStream User Manual*, version 26.1 (build 26121), 2026. Cited with its page, as [6, p. 164].

REF-27 V. Ahuja, *Aerodynamic Loads over Arbitrary Bodies by Method of Integrated Circulation*, PhD dissertation, Auburn University, Auburn, 2013.

REF-21 D. J. Pate, B. J. German, "A surface vorticity panel method", *AIAA Journal*, 56(12), 2018. DOI 10.2514/1.J057120.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

References (continued)

REF-28 J. L. Hess, A. M. O. Smith, "Calculation of potential flow about arbitrary bodies", *Progress in Aerospace Sciences*, 8, pp. 1-138, 1967.

REF-29 J. Katz, A. Plotkin, *Low-Speed Aerodynamics*, 2nd ed., Cambridge University Press, Cambridge, 2001.

REF-30 W. M. Kutta, "Auftriebskräfte in strömenden Flüssigkeiten", *Illustrierte Aeronautische Mitteilungen*, 6, pp. 133-135, 1902.

REF-31 H. Glauert, "The effect of compressibility on the lift of an aerofoil", *Proceedings of the Royal Society of London, Series A*, 118(779), pp. 113-119, 1928.

REF-32 B. M. Simmons, S. C. Geuther, V. Ahuja, "Validation of a mid-fidelity approach for aircraft stability and control characterization", AIAA AVIATION Forum, 2023. DOI 10.2514/6.2023-4076.

REF-33 A. Gothow, F. Gavrila, J. Weiss, V. Ahuja, C. Gates, "Comparative analysis: validating FlightStream with experimental results of a high-lift configured wing model equipped with distributed electric propulsion", AIAA AVIATION Forum, 2024.

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

REF-35 J. G. Leishman, *Principles of Helicopter Aerodynamics*, 2nd ed., Cambridge University Press, Cambridge, 2006.

REF-36 B. W. McCormick, *Aerodynamics, Aeronautics, and Flight Mechanics*, 2nd ed., Wiley, New York, 1995.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

References (continued)

[18] G. Neves, "SMI: airframe geometry", version 1.0.0, Zenodo, 2026. [REF-58]
`https://zenodo.org/records/23025105`

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

[21] Geovana Neves, *pyflightstream documentation*, page "Getting started".
`https://nevesgeovana.github.io/pyflightstream/latest/getting-started/`

[22] Geovana Neves, *pyflightstream documentation*, page "FSI in a workspace".
`https://nevesgeovana.github.io/pyflightstream/latest/fsi-workspace/`

Copyright (c) 2026 Geovana Neves, CC BY 4.0