Guide 2: Workspaces

One tree, two workspaces, and every file in it

Written for pyflightstream 0.58.2.

Sources: the package pages "The workspace and the workflow" and "Storage and sync" [1, 2].

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 going to USE the workspace

- Parts 1 and 2: the tree and the map.
- Parts 3 and 4: first time on Windows; on the cluster it is one line.
- Parts 5 and 6: inputs and site folders.
- Parts 7 and 8: choosing a workflow, and the quasi-steady rotor.
- Parts 9 to 11: running, what a run and its post leave.
- Parts 12 and 13: Windows and an HPC cluster, two workspaces and sync; space on disk.
- Parts 14 and 15: version control, and a reference.

The other guides

- 1: FlightStream and pyfts, an overview. Read it first.
- 3: from the GUI to pyfs. 4: the cheatsheet.
- 5: the reference file. 6: solver setup.
- 7: post-processing definitions.
- 8: fluid-structure interaction.
- 9: installing Python and the package on an offline machine, Windows and Linux. It happens once per machine.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Contents

One tree, two workspaces
The map
First time on Windows
The cluster
The input files
Site folders
Choosing a workflow
The quasi-steady rotor
Running
Grouped runs
What a run leaves
What post-processing leaves
Windows and an HPC cluster: two workspaces and sync
Space on disk
Version control
Reference

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 1

One tree, two workspaces

A workspace is a folder

- Everything a study needs is **inside one folder**: the matrix, the inputs it cites by name, the meshes, the results.
- Nothing lives in a database, a registry or somebody's head.
- A study moves between machines by being **opened** there.

Machine	What it is for
Windows workstation	planning, one point, looking at it
HPC cluster	a long march, many points

main: `C:/<work-folder>/fts-workspace`
cluster: `<cluster-root>/fts-unix/fts-workspace`

Takeaway *The solver seat and the queue are both scarce, in different ways. Use each machine for what it is good at.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The path is not a preference

`C:/<work-folder>/fts-workspace`

one agreed root, the same for everybody

Why

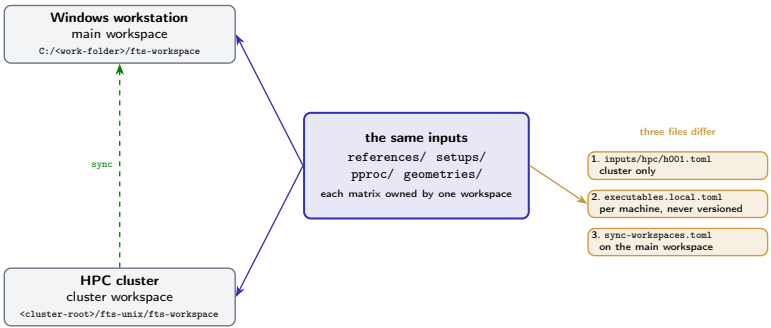
- The solver is reached by an **absolute path**.
- Paths are **recorded inside runs**; another root stops finding them months later.

How it gets broken

- A workspace in `Documents`. Move it, do not fix it up.
- A mapped drive letter, `Z:/...`. Per user, per session.
- The agreed root cannot be created? **Stop and raise it.**

Copyright (c) 2026 Geovana Neves, CC BY 4.0

What differs between the two machines



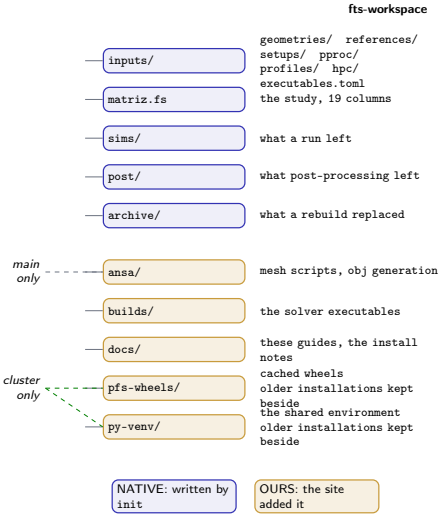
- `inputs/hpc/h001.toml`: cluster only, how the cluster runs a job.
- `inputs/executables.local.toml`: per machine, never versioned.
- `inputs/sync-workspaces.toml`: on the main workspace, which brings the cluster's runs home (Part 13).

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 2

The map

The whole tree



Copyright (c) 2026 Geovana Neves, CC BY 4.0

NATIVE or OURS, and three writers

Two kinds of folder

- **NATIVE**: created by `pyfs-workspace init`. Renaming one breaks the tool.
- **OURS**: added by the site (for example `ansa/`, `builds/`, `docs/`). Renaming one is a team decision.

Writer	Writes
You	`inputs/`, `matriz.fs`, `ansa/`, `docs/`
The tool	`sims/`, `post/`, `archive/`; two pproc guides in `inputs/pproc/`
The solver	inside a datapoint folder

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 3

First time on Windows

Four steps before the first plan

1. **Get the workspace** into `C:/<work-folder>/fts-workspace`: a checkout from your version control, or a copy of the study's folder.
2. **Install** Python 3.12 or newer and the package in a virtual environment (offline machine: Guide 9):

```
py -3.12 -m venv .venv
.venv\Scripts\activate
pip install pyflightstream==0.58.2
```

3. **Point at the solver** in `inputs/executables.local.toml`, forward slashes, absolute path.
4. **Read the boundary order off the mesh**, never type it:

```
pyfs-mesh inventory inputs/geometries/<mesh>/<mesh>.fsm
```

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Plan first, always

```
pyfs-matrix plan matriz.fs --workspace . --fs-version 26.123
pyfs-matrix plan matriz.fs --workspace . --fs-version 26.123 --cost
```

- Spends **no seat and no queue**.
- Writes the receipt `run` refuses to start without.
- The receipt carries the matrix digest: an edit between plan and run is visible.

- `--cost`: mesh size, run type, steps, processors, and an *expected* time.
- The time is an extrapolation from this workspace's own runs, and says so. No comparable run, no number.
- A machine's cost file, `inputs/costs/c<NNN>.toml`, prices it instead; `--cost-file NAME` picks one of several (Part 5).
- It warns when the points still to run may not fit on the disk (Part 14).

Copyright (c) 2026 Geovana Neves, CC BY 4.0

A healthy first run

The rotor row states `LAST_REVS_AVG: 1`; its flight condition lists Mach, alpha, beta and advance ratio, in that order.

```
pyfs-matrix run matriz.fs --workspace . --fs-version 26.123

  -> fts-workspace/sim_0001/M144AL+000BE+000J+100  [unsteady_rotor]  building and running
     fts-workspace/sim_0001/M144AL+000BE+000J+100  CONVERGED
  ...
```

- Each line names the **run id**; the point tag ends it.
- The products are under `post/matriz/`; the `TOTAL` group files hold each polar.
- Read `skipped`: it may name an inapplicable reduction or missing data needed by a product.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 4

The cluster

Using it is one line

```
source <cluster-root>/fts-unix/bin/pfs0410-env
```

- At the top of a session **or of a job script**.
- Nobody installs, nobody builds an environment, nobody asks permission.
- One environment for everybody, which is what makes two people's results comparable.

An alias is not available to a job

A job script does not expand aliases and fails after the queue wait with `command not found`. Keep an alias as a convenience, never as the mechanism.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Leaving the environment

You sourced	To leave
`py-venv/pfs0410/bin/activate`	`deactivate`
`bin/pfs0410-env`	a new shell, or `exit`

The shared file only puts one folder at the front of `PATH`. To keep the shell open, take it off again:

```
export PATH=${PATH#<cluster-root>/fts-unix/py-venv/pfs0410/bin:}
which pyfs-matrix      # must no longer name py-venv/pfs0410
```

Takeaway *Leave one environment before sourcing another.* `which pyfs-matrix` *says which one you are in.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Installing is another guide

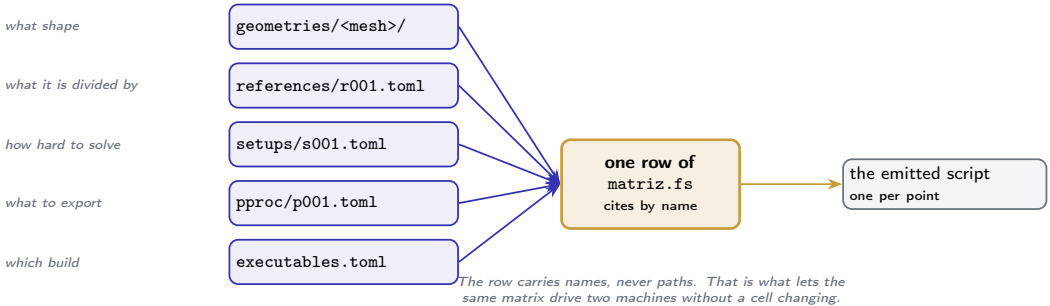
- Python 3.12, the wheels and the shared environment are installed **once, by one person**, from a connected machine: Guide 9, *Python environment for offline machines*.
- Everybody else sources one line, above.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 5

The input files

The row names things, the files define them



Takeaway *By name, never by path:* `REF`, `SET` *and* `PPROC` *carry a file stem,* `GEOMETRY` *names the mesh. That is what lets one matrix drive two workspaces.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

matriz.fs: nineteen columns

Group	Columns
identity: does it run at all	POL, HIDDEN, RUN
what it is	AIRCRAFT, CONFIGURATION, DESCRIPTION
what it flies	FLIGHT_CONDITION, SWEEP_VALUES
what it cites	GEOMETRY, REF, SET, PPROC
how it solves	SYMMETRY, SYMMETRY_LOADS, NCPUS, WALLTIME, FS_BUILD
what builds it	WORKFLOW, VAR_NAMES_VALUES

- A key **every row** must answer is a column; a key only some rows need goes in the free cell, VAR_NAMES_VALUES.
- - means *stated nothing*, so it cannot be confused with a truncated line.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

One POL, once, in the whole workspace

- `POL` names `sims/sim_<POL>` and the run ids, so it appears **once across every matrix** of the workspace, `RUN = 0` rows included.
- `plan` refuses a repeated POL and lists where it appears.

```
pyfs-matrix plan matriz.fs --workspace . --fs-version 26.123 --update-ids
```

- `--update-ids` gives each repeated row of *this* matrix the next free POL above every POL, run record and `sims/` folder **it can see**.

It writes the matrix before planning

The renumbering stays written even if the plan then refuses. Commit first, so the change is one diff you can read.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The free cell: move, turn, raw

```
TRANSLATE: {DISTANCE: 0.05 / AXIS: PUSHER_SMRP-X / ALIAS: PUSHER}
ROTATE:    {ANGLE: 3 / AXIS: PUSHER_SMRP-Y / ALIAS: PUSHER}
RAW:       {COMMAND: SOLVER_SET_ITERATIONS 350 / BEFORE: init}
```

- `TRANSLATE`: metres, along **one** axis; a diagonal is two records.
- `ROTATE`: degrees, about one axis.
- **Every translation comes before every rotation**, whatever the order written.
- One row is one position: neither is swept.

What to know before using it

- The alias's frames move with it; `<ALIAS>_SMRP_ORIGINAL` keeps the hub where it was.
- A moved part **comes away** from what it touches: the junction is open.
- The distance is read in metres; keep the meshes in metres.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The flight condition: constraints on one flow state

```
FLIGHT_CONDITION: MACH:0.20, REmi:5.5
FLIGHT_CONDITION: TASmps:68.08, ALTFT:10000,
     dISA:5
```

Key	Unit	Constrains
`MACH`	none	velocity, by the state's speed of sound
`TASmps`	m/s	velocity, directly
`REmi`	**millions**	density
`ALTFT`	**feet**	pressure and temperature
`dISA`	**Celsius, a delta**	temperature

- The keys decide what is solved for: the first line solves the **density**, the second is an atmosphere point whose Reynolds number follows [5, 7, 8]. [REF-01], [REF-02]
- Exactly one of `MACH` or `TASmps`. `REmi` needs the row's `REF`: a Reynolds number has no meaning without a length.
- Five keys **pin** the fluid (`RHOkgm3`, `MUPas`, `ASMPS`, `TK`, `PPA`), on the row or in the setup's `[flight_condition]`.
- A rotor: $V = J\,(\mathrm{RPM}/60)\,D$; state two of the three.
- The set is closed: an unknown key is refused, never ignored.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

A matrix from an older layout

- A matrix written under an earlier layout is recognised by its header and **refused rather than misread**. No converter ships: edit the rows by hand to the current layout, then plan them with `pyfs-matrix plan`.
- When editing by hand, a sweep code becomes the word `sweep` where the value would be: `AL` becomes `ALPHA:sweep`. A row that varies both angles is refused by name: it is one row per sideslip, each with its own POL.
- **The values move across verbatim; the result may not.** A Reynolds number that was only recorded becomes a constraint that solves for density, so the migrated row emits a fluid state it never emitted before. To keep the old behaviour, drop `REmi` and let the Reynolds number follow.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

r001.toml: what things are called (Guide 5)

```
area_m2 = 19.502
chord_m =  1.322
span_m  = 15.059
[moment_point]
x_m = 0.0

[aliases]
airframe = ["Spinner","Nacelle"]

[PUSHER]
kind = "rotor"
axis = "X"
diameter_m = 1.80
blade1 = { azimuth_deg = 0, zero = "Y" }
families_blades = ["Blade1", "Blade2"]
```

The template's synthetic numbers; a study replaces every one.

- Reference lengths and the moment point divide every coefficient.
- An **alias** is a word rows, groups and plots cite.
- A rotor block creates <NAME>_SMRP, <NAME>_RMRP and <NAME>_RMRP<k>.
- The blade count is the length of families_blades.

The failure nothing refuses

A wrong moment point gives plausible, wrong coefficients. Get it right by reading.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Setup, post-processing and profiles

s001.toml

How hard to solve and when to stop. Multiplatform. A command every row needs goes in its `[[raw]]` table.

p001.toml

What the run exports. The log is declared **last**: its presence says the run finished. The `.fsm` of every point is always saved; `simulation = false` is refused.

inputs/profiles/

Probe surveys a `[[probes]]` entry cites by `points_file`, and the radial profiles a row's `PROFILE` names for an actuator disc. Resolved to an absolute path; a missing file is refused at plan.

```
[[probes]]
frame = "MRP"
parameters = ["VELOCITY"]
points_file = "disk_survey.txt"      # inputs/profiles/disk_survey.txt
```

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The solver path and the cluster profile

Two files, one sentence

`executables.toml` is about the **study** and is versioned. `executables.local.toml` is about the **machine** and is not.

One profile per cluster

One profile in `inputs/hpc/` needs no option. With several, `--hpc NAME` on `plan`, `run` and `collect` selects `inputs/hpc/<NAME>.toml`; none selected is refused, never guessed.

```toml
# inputs/hpc/h001.toml
max_walltime = "48:00:00"   # longest job
     the queue takes

[descriptor.fields]
version = "{fs_build_alias}"

[builds]
"26.123" = "26.1"
```

- The row keeps the build, the profile gives the scheduler's word.
- A declaration, not a check: the build number in the collected log is the proof.
- `max_walltime` (`HH:MM:SS`, hours may pass 24) caps a grouped job; without the key there is no limit.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

inputs/costs/: what a point costs on this machine

```
# inputs/costs/c001.toml
[reference]     # one run measured here
mesh_faces = 10000
ncpus = 4
wall_time_s = 1000.0
steps = 100

[parallel]      # speedup per NCPUS
ncpus = [1, 2, 4, 8]
speedup = [1.0, 1.9, 3.6, 6.5]

[mesh]
exponent_b = 1.0
[steps]
exponent = 1.0
[flags]         # one multiplier per flag
viscous_coupling = 2.0
```

- One file per machine, read by `plan --cost` and by the estimate of a grouped plan. Several: `--cost-file NAME`.
- Time scales from the anchor run: the speedup, linear between two stated counts; mesh faces to the power b; steps to their exponent; each flag the row sets.
- A processor count outside the curve is **never extrapolated**: the estimate is withheld. An unknown key is refused.
- The values are yours: the package ships only a synthetic `examples/costs/c000.toml`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The row states its window

- `unsteady_rotor`: `LAST_REVS_AVG`, in revolutions; fractions are accepted.
- `unsteady`: `LAST_ITERS_AVG`, in solver steps.
- With no window key, `plan` refuses the row. `WINDOW_*` keys are refused, naming the key to write.

Temporal fragments for the free cell, beside the other required keys:

```
DELTA_THETA: 5 / REVOLUTIONS: 4 / LAST_REVS_AVG: 1
DELTA_TIME: 0.01 / TIME_ITERATIONS: 100 / LAST_ITERS_AVG: 20
```

One declaration of each attitude

`ALPHA` or `BETA` in both `FLIGHT_CONDITION` and `VAR_NAMES_VALUES` is refused, even if the values agree.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Groups and optional pproc tables

```
[groups]
PUSHER = "PUSHER"
AIRFRAME = "airframe"
TOTAL = "all"

[phase_locked]
min_revolutions = 4.0
last_revolutions_avg = 2.0

[products]
superfile_format = "legacy_polar"
```

- One alias as a string. A list is refused with the one-alias line to write; several members go in the reference's `[aliases]`.
- `[phase_locked]`: one row per azimuth across revolutions; no declaration keeps the passage series.
- `[equations]` and `[glossary]` bind too: derived columns at post, named `<NAME>_<alias>`; equations may chain.
- Fixed-width super file: sixteen characters per field; `csv` remains the default.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Names and plot-group declarations

```
[names]
CL_MRP_TOTAL = "CL_TOTAL"
```

- Requires that plotted source column. The whole dictionary applies or none does; missing sources and occupied destinations produce a `#names` skip. `CL` is already the native last-step health check.
- Applied after axes and equations to the unsteady polar, time average and passage series. Plots, per-blade and azimuthal headings retain their names.
- Plot-group names: letters, digits, underscores and bare `{family}` only. No spaces, format spec or conversion.
- Ambiguous names cannot supply rotor histories or global axis blocks.
- `VARIABLES.md` and `WRITING-EQUATIONS.md` are generated in `inputs/pproc/` by init, plan and post; commit or ignore them.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

A mesh file, OBJ or STL

```
# wing/wing.obj.boundaries.toml, beside wing.obj
fs_build = "26.125"
boundaries = ["Wing"]        # OBJ: from its groups
[import]
units = "METER"              # the FILE's unit
[trailing_edges.wing]        # one named entry per set
file = "wing.te.txt"         # points file
type = "STANDARD"
```

Export the mesh in metres

On 26.124 a wing OBJ in millimetres, its trailing edge detected, solved as the same wing in metres to the print [3]: `IMPORT` converts. The file route in millimetres, and every other build, are not measured.

- `pyfs-mesh mark-boundaries wing.obj` previews; `--apply` (licensed) marks once and saves `wing.fsm`.
- The row names `GEOMETRY`: `wing.fsm`; an OBJ there is refused, except on an FSI row.
- `pyfs-mesh inventory` writes the sidecar from the groups. **By hand**: `fs_build`, the unit, the trailing-edge entries, and an STL's names.
- `[[import.operations]]`: scale, rename, mirror, translate, rotate, in order, at marking.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The trailing edge, by file

```
METER                                   # the unit of the points
0.216872113,-0.002159825,0.223199974    # one MID-POINT of an edge per line
```

- **The default route.** Every point is checked against the mesh when marking previews; the first one off an edge is refused by line and position.
- The marking pass writes the node file in metres, imports it with `IMPORT_WAKE_EDGES_FROM_FILE`, and checks the saved trailing edges before it publishes the `.fsm`. Runs never import them.
- VERIFIED on 26.124 and 26.125.
- Detection is the second route, written in full: `detect = "auto"`, or by surface with a sweep angle, with `type = "STANDARD"`.
- `BASE_REGIONS` names the **base**, not the body [3].

Copyright (c) 2026 Geovana Neves, CC BY 4.0

A public airframe to practise on

- The SMI airframe geometry [9] is published as IGES components: the fuselage, the nacelles with their pylons, the lifting surfaces, and the wing-body geometry they share. Each component is whole, so intermediate configurations can be built from it.
- It carries **no propeller**. Any rotor is added by the user and placed by a rotor block of the reference (Guide 5), from any propeller geometry the user holds.
- Mesh it with the site's mesher, export OBJ **in metres**, and write its `<stem>.obj.boundaries.toml` beside it, then mark it: the route of the two previous frames.

Its licence

CC BY-NC 4.0: credit the record, no commercial use. Cite it as [9] wherever a result uses it.

Takeaway *A first workspace on a known public geometry makes a mistake visible before it costs a campaign.*

Every key in one generated page, and an optional workbook

inputs/INPUTS.md and inputs/input_template.md

- `INPUTS.md` lists every key an input may state: the row by run type, the setup, the pproc, the reference, the sidecar. Each says what it sets, its unit or values and the command it reaches.
- `input_template.md` shows every input **file**, one complete example per kind, each one a file the package reads as it stands.
- Both are **generated** by `init`, `plan` and `post`; never edit them.

An Excel workbook, if you want one [6]

An ordinary `.xlsx`, no macros. Python creates it and synchronises saved files in both directions, always as a preview first, then an apply:

```
pip install "pyflightstream[excel]"
python -m pyflightstream.workspace.excel preview
   <book>.xlsx --workspace . --direction read
   --batch read-preview.json
python -m pyflightstream.workspace.excel apply
   read-preview.json
```

The ASCII matrix stays the record; the workbook is an editor.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 6

Site folders

ansa/, builds/, docs/

ansa/
The mesh scripts. The script is versioned, the mesh is not.

builds/
One folder per solver build number.

docs/
What a second person needs to repeat the first one's work: these guides, the install notes.

Adding a build, in order

1 copy into its own folder **2** register in `executables.toml` **3** point your local file at it, and add its `[builds]` line on the cluster **4** plan one row and read the build number in its log.
Keep the old build.

Part 7

Choosing a workflow

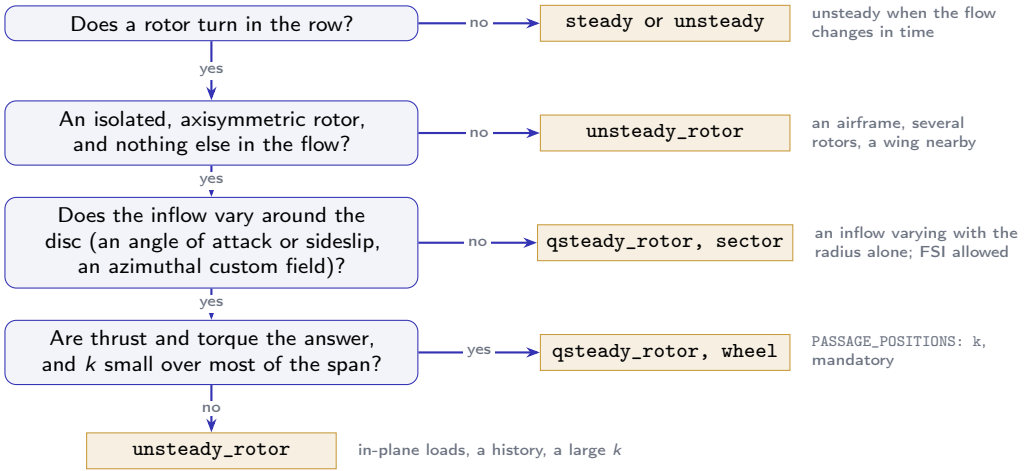
Four workflows, five ways to solve a row

WORKFLOW	What is solved	When to use it	Cost
steady	one converged steady solve per point; a sweep is one job	a body in a steady flow: polars, sweeps, a fixed wing (FSI with its own weight)	seconds per point
unsteady	a march in physical time, nothing turning	a flow that changes in time without a rotor: a manoeuvre rate, a start-up, a fixed wing coupled per step	minutes
unsteady_rotor	the blades turn, the wake is shed in time	the reference for any rotor: installed, interacting, several rotors, a time history, the in-plane loads	minutes to hours
qsteady_rotor, sector	one blade held still, periodic copies, the free stream turning	an isolated, axisymmetric rotor in an inflow the same at every azimuth; blade FSI with centrifugal loads	seconds
qsteady_rotor, wheel	every blade held still; one steady solve per clocking, averaged	the same rotor at an angle or in a custom inflow, when thrust and torque are what matter	seconds per clocking

Every row names its workflow in the WORKFLOW column; a key another workflow registers is refused by name at plan. Costs are orders of magnitude on one workstation, measured on a six-blade research propeller [14].

Copyright (c) 2026 Geovana Neves, CC BY 4.0

A decision chart



k is the blade's 1P reduced frequency (Part "the quasi-steady rotor"); `plan` prints it for every wheel point.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

steady and unsteady

steady

One converged solve per point; an `ALPHA:sweep` or `BETA:sweep` row is one job, every point starting cold unless `COLD_START: false`. The polar, the super file and the sections are instants of the last iteration. A custom inflow (`FREESTREAM`) or a free-stream rate (`roll_rate`, `pitch_rate`, `yaw_rate`) is still one steady solve.

unsteady

A march of `TIME_ITERATIONS` steps of `DELTA_TIME`, and the window it is averaged over, `LAST_ITERS_AVG`, which the row must state. Nothing turns but the free stream; a rotor is `unsteady_rotor`'s. The products are averages over the window, and the history is kept.

Takeaway *The window belongs to the row, not to the pproc: changing it is a new* `post`*, never a new run (Guide 7).*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

unsteady_rotor: the reference, and its price

- The blades turn at RPM (or from ADVANCE_RATIO), about the rotor block's axis, with its hand; the step is DELTA_THETA degrees or DELTA_TIME, the length REVOLUTIONS or TIME_ITERATIONS, and LAST_REVS_AVG is mandatory.
- The only workflow for a rotor on an airframe, for several rotors (MOTIONS), for the time history and phase-locked products, and the one to trust for the in-plane loads.
- FSI is refused on it in this release (Guide 8).

A few revolutions are not a developed wake

On a six-blade research propeller the thrust still rose about 0.5 % per revolution at revolution 6 [14]. A window over the third of three revolutions averages a wake still growing: run more revolutions before the window when the level matters, and compare points at the same revolution count when the difference matters.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 8

The quasi-steady rotor

qsteady_rotor: a rotor solved steady

- **One workflow, for an isolated, axisymmetric rotor only.** The blades stay still; the free stream turns about the shaft at the rotor's speed (SET_FREESTREAM ROTATION, the hub frame, the shaft, the signed speed); every solve is steady.
- The reference declares exactly one rotor block; the row states RPM or ADVANCE_RATIO. A second rotor, an actuator disc, a body rate, or a boundary that is none of the rotor's is refused.
- **Sector**: SYMMETRY periodic, with PERIODIC_COPIES: the meshed blade once. **Wheel**: no symmetry, every blade meshed. MIRROR is refused.

Why it stands for the turning blade

One blade held still in the rotating free stream against the same blade turning, after one revolution at the same azimuth: axial force within 0.35 %, in-plane force within 2.5 % [14].

> ***Takeaway*** *It is callable without FSI. FSI is its optional right-hand variable, on the sector only (Guide 8).*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Sector or wheel: what the inflow allows

Sector

Stands for the wheel only in an inflow the **same at every azimuth**. An angle of attack or of sideslip is refused. A custom inflow (`FREESTREAM`) is accepted only where it varies with the radius alone: rows at one distance from the shaft state one axial, radial and swirl velocity. FSI allowed: the structural solver applies the centrifugal loads at the free-stream rotation's rpm.

Wheel

In an axial, uniform inflow it is steady in the rotating frame: one solve. At an angle, or in a custom inflow that varies around the disc, each blade meets another flow: `PASSAGE_POSITIONS: k` is **mandatory**. A custom inflow file is the **total** velocity of the air, in the global frame; the package writes $v - \boldsymbol{\Omega} \times (p - p_{\mathrm{hub}})$ into a new file and never touches yours. FSI refused.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

PASSAGE_POSITIONS: clockings inside one passage

- k steady solves, the wheel clocked by

$$\theta_i = \frac{i\,(360^\circ/N)}{k}, \quad i = 0 \ldots k-1,$$

  uniform inside ONE blade passage: a wheel of N alike blades repeats every $360^\circ/N$.
- Clocking 0 is solved last with every export; clocking i leaves `<point>_qs<i>.txt`. The post averages them: the products are those of an unsteady point.
- **Guidance**: 2 for thrust and torque; 6 or more for the in-plane loads.

k	F_x [N]	F_y [N]	F_z [N]	M_z [N m]
1	−2210.56	−23.37	258.84	−347.98
2	−2206.12	−23.81	257.17	−346.05
3	−2207.84	−27.53	260.58	−348.87
6	−2204.64	−27.09	257.75	−345.45
12	−2203.95	−27.46	258.21	−345.95
24	−2203.82	−27.57	258.23	−345.95

A six-blade research propeller at 5°, 26.124 [14]: F_x and M_x within 0.2 % from $k = 2$; at $k = 6$ every component within 1.8 % of $k = 24$. About 1.7 s per clocking.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Is a quasi-steady answer valid? The reduced frequency

- A wheel point at an angle sees its inflow change once per revolution. Each blade station's **1P reduced frequency**,

$$k = \frac{\Omega\, c}{2\, V_{\mathrm{rel}}} = \frac{\pi c / 2R}{\sqrt{J^2 + \pi^2 (r/R)^2}},$$

  compares the time the flow takes to cross half a chord with the period of that change: below about 0.05 the lag of the circulation and the shed wake is small; above 0.1 it is not [10, 11, 12]. [REF-34], [REF-35], [REF-38]
- It depends on J and the chord alone, never on the speed: the same J at another Mach number gives the same k.

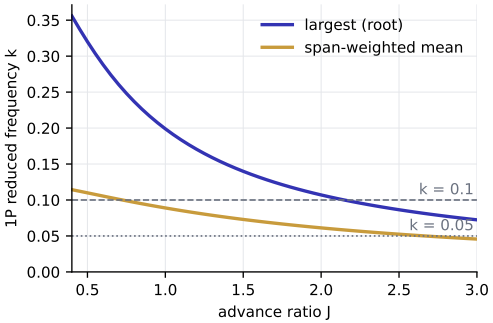
What the package states

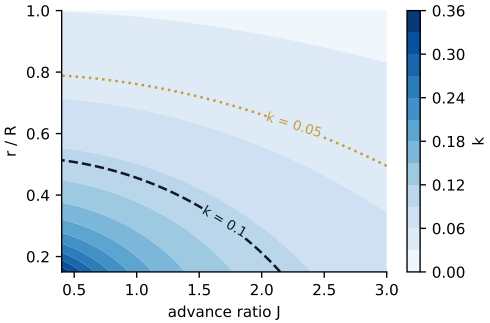
`plan`: four values per wheel point, the per cent of the span with $k > 0.1$ and k minimum, maximum and mean; a **warning** when that per cent is above zero, never a refusal.

`<point>_qsteady_validity.json`: the post's k statistics and shares of thrust and torque from stations above 0.1; every product of the point carries them, the sections product k per station.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The reduced frequency of the package's test blade





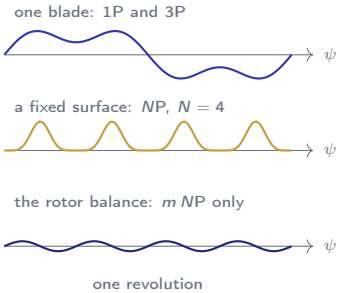
J	0.6	1.0	1.4	1.7	2.2	3.0
k largest (root)	0.288	0.199	0.149	0.125	0.098	0.072
span with $k > 0.1$ [%]	41.5	36.4	28.0	19.5	0	0

The tier-3 blade of the repository (NACA 4409, $R = 1.8288$ m, chord over R from 0.14 at the hub to 0.06 at the tip), from its chord law; drawn by `figures/qsteady_reduced_frequency.py`. The root is the worst station: short radius, wide chord.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

What nP counts: the blade's own count

- **nP is counted by ONE blade**: how many times that blade meets the same perturbation in one revolution. An angle of attack is 1P; a field with four lobes around the disc is 4P, for every blade alike.
- It is **not** what a fixed surface near the rotor feels: a wing behind it is struck each time any of the N blades passes, so it counts NP (blade passing).
- It is **not** what a balance under the whole rotor measures: summing N alike blades cancels every harmonic but the multiples of N, so only $m\,N$P survive there [12]. [REF-35]



Copyright (c) 2026 Geovana Neves, CC BY 4.0

`--inflow-fft`: how many clockings a custom inflow needs

- A plan option, for `qsteady_rotor` wheel rows with a custom inflow: at each radius, the Fourier series in azimuth of the angle of attack the **blade** meets, harmonic n being nP in the blade's count.
- n_{95}: the harmonics holding 95 % of that perturbation; $k_{\mathrm{eff}} = n_{95}\, k_{1\mathrm{P}}$, reported as minimum, maximum, mean and the span with $k_{\mathrm{eff}} > 0.1$. With the option the validity warning reads k_{eff}.
- The suggested `PASSAGE_POSITIONS` $\geq n_{\max}/N + 1$, and a warning when the row states fewer.

Why $n_{\max}/N + 1$

The wheel's total load holds only $m\,N$P. k uniform clockings inside one passage average away every $m\,N$P with m not a multiple of k; the first one that survives is $k\,N$P. So the average is clean when $k\,N > n_{\max}$: a blade harmonic above that aliases into the mean.

What to expect against `unsteady_rotor`

Component	qsteady against unsteady	Why, and what moves it
F_x, M_x (thrust, torque)	**larger in magnitude**: 7 to 9 % at 10° steps and 3 to 6 revolutions	the steady wake is the developed one; the unsteady run is still approaching it, so the gap **shrinks** with finer time steps and more revolutions
F_z, M_z (normal force, yawing moment)	**smaller in magnitude**, by roughly 8 to 20 %	the phase and level of the 1P blade load on the downgoing half of the disc: the lag a steady solve does not have
F_y, M_y (side force, pitching moment)	**unreliable**: tens of N against a clocking noise of about 13 N, and they may flip sign	small resultants of large blade loads; more clockings do not close the gap

A six-blade research propeller at 5° on 26.124 [14]. At 0° the same axial gap appears (−7.5 % against 6 revolutions), so it is the wake, not the angle. Use `qsteady_rotor` for thrust, torque and trends; use `unsteady_rotor` for the in-plane loads.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The tests behind it

	U, last rev	Q, 6 clockings	$(Q-U)/\lvert U\rvert$
F_x [N]	−2022.76	−2204.64	−8.99 %
F_y [N]	55.36	−27.09	opposite sign
F_z [N]	320.52	257.75	−19.6 %
M_x [N m]	−2274.67	−2433.44	−6.98 %
M_y [N m]	−54.83	44.68	opposite sign
M_z [N m]	−410.73	−345.45	+15.9 %
C_T, C_P	0.2903, 0.5623	0.3164, 0.6015	+9.0, +7.0 %
wall time	102.7 s	10.8 s	

U: all six blades turning, 10° per step, three revolutions, the last averaged. Q: blades still, six clockings. RPT-089 [14].

The time step closes the axial gap

Same propeller, the time-step study in RPT-089, section 5: at 0° and 3 revolutions the axial gap is 8.4 % at 10° per step, 6.6 % at 5°, 4.0 % at 2.5°; 3.9 % at 5° and 6 revolutions. At 5° incidence, 5° steps, 6 revolutions: F_x 3.6 %, F_z 11.7 %, M_z 8.2 % apart, the side force and pitching moment still of opposite sign. No step size levelled off.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Cost, side by side

The same rotor point	Wall time, one launch	What it buys
`qsteady_rotor` wheel, 6 clockings	10.8 s	thrust and torque; in-plane loads as estimates
`qsteady_rotor` wheel, 24 clockings	about 42 s	the same, the clocking noise below 0.5 %
`unsteady_rotor`, 10°, 3 revolutions	102.7 s	the history, every component, a wake still developing
`unsteady_rotor`, 5°, 6 revolutions	about 18 min	the axial gap to the steady wake down to about 4 %

Takeaway *Seconds against minutes: sweep J, the angle and the geometry with* `qsteady_rotor`*, then confirm the points that matter, and every in-plane load, with* `unsteady_rotor`*.*

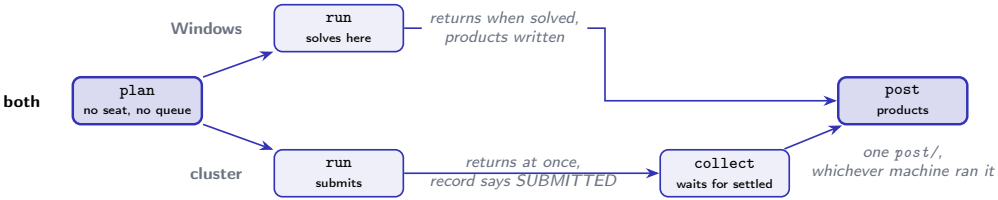
Six-blade research propeller, 26.124, licence checkout included [14]; the 18 min is the time-step study of the previous frame.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 9

Running

Four stages



Stage	Windows	Cluster
`plan`	required, no seat	required, no queue
`run`	solves, then returns	submits, returns at once, `SUBMITTED`
`collect`	not needed	waits for settled outputs, judges them
`post`	rebuilds the products	the same

Copyright (c) 2026 Geovana Neves, CC BY 4.0

One matrix, any build

	25.000	25.100, 26.000	26.100	26.101 and later
`steady`	refused	runs	runs	runs
`unsteady`	refused	single march	single march	single march, or actions
`unsteady_rotor`	refused	Euclidean rotor	unmarked, rev/min	rotary, or actions

- Only `FS_BUILD` changes. Actions exist from 26.122.
- Without actions: `WALLTIME`, the snapshot thresholds and `FINISH_PENDING`/`ADDITIONAL_REVS` are **refused by name** at plan.
- `march_strategy` in `plan.json`, the record and the superfile.
- A preset or pproc file can still block an older build: **plan first**.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Tip and helical Mach, on every rotor point

$$\Omega = \frac{2\pi\,\mathrm{RPM}}{60}, \qquad M_{\mathrm{tip}} = \frac{\Omega R}{a}, \qquad M_{\mathrm{hel}} = \frac{\sqrt{V^2 + (\Omega R)^2}}{a}$$

- For each rotor an `unsteady_rotor` or `qsteady_rotor` row turns, a `steady` row that states `RPM`, and each actuator disc a row names. R is the rotor's half diameter or the disc's tip radius; V and a are the point's own resolved free stream and speed of sound.
- `plan` prints both per rotor and per point, and **warns**, naming the point, when $M_{\mathrm{hel}} \geq 1$: the tip is sonic or supersonic, where the linearised compressibility of a panel method stops holding [4, 12, 13]. A warning, never a refusal. [REF-31], [REF-35]
- The rotor table includes `MTIP_<alias>` and `MHEL_<alias>`. Both are geometric: V is the free stream alone, without the rotor's own induced velocity.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

`run`, on each machine

Windows

- A steady angle sweep is **one job over all its points**; each point starts cold. `COLD_START: false` starts each point from the previous one.
- `WALLTIME` arms a clock; a run it stopped is `WALLTIME_REACHED`, **not a failure**.

Cluster

- An unsteady point is **its own job**, in its own datapoint folder with its descriptor, action program and clock.
- A steady angle sweep is one job; a flow-variable sweep is one per point.
- A point already queued is refused a second submission. Collect it first.
- `--batch N` or `--polar-sweep` puts several polars' points in one job instead (Part 10).

Copyright (c) 2026 Geovana Neves, CC BY 4.0

`collect` watches the workspace, not the scheduler

```
pyfs-matrix collect --workspace .            # one sweep, what a cron wants
pyfs-matrix collect --workspace . --watch    # loop until nothing is outstanding
*/15 * * * *  cd <study> && $FTS/py-venv/pfs0410/bin/pyfs-matrix collect --workspace .
```

Settled is two signals

Size and time stable across two looks, **and** the log present, because the log is declared last.

Exit	Meaning
0	everything settled and collected
1	a point it could not complete
3	still outstanding

Copyright (c) 2026 Geovana Neves, CC BY 4.0

RESTART: continuing a march that stopped

Write in the free cell	When
`RESTART: {FINISH_PENDING}`	a run the *clock* stopped
`RESTART: {ADDITIONAL_ITERS=n}`	any continuable run, *n* more steps
`RESTART: {ADDITIONAL_REVS=n}`	any continuable run, *n* more revolutions

- You write `RESTART` and nothing else; the manifest says which run and how many steps are left. The count is a **remainder**.
- It continues the point's **most recent** run. A finished point is refused; a failed continuation is not retried.
- What it replaces is archived first, under a day and hour stamp.

Takeaway *On the cluster this is the ordinary case: the queue enforces the wall clock, so `{FINISH_PENDING}` is the form you will use most.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Points already run, and a Linux run kept local

```
pyfs-matrix run matriz.fs --workspace . --resume
pyfs-matrix run matriz.fs --workspace . --force-rerun '<point name or run id>'
pyfs-matrix run matriz.fs --workspace . --local
```

- A recorded point is refused before anything runs; `plan` marks it `ALREADY_RECORDED`.
- `--resume`: only the new points of a matrix that grew.
- `--force-rerun`: redo a named point; its old record and outputs move to `archive/`, nothing is deleted. Plan again first. Not together with `--resume`.
- `--local`: on Linux with a profile, run here instead of submitting; the record says `forced_local`. Elsewhere it changes nothing.

Redo on purpose

A recorded point is never solved again by `--resume`. A point whose inputs were wrong is redone by name with `--force-rerun`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The additional post

```
ADDITIONAL_PPROC: p002                          # free cell; the run is unchanged
pyfs-matrix post matriz.fs --workspace . --additional-pproc
```

Reopened `.fsm`, no solve	**Measured [3]**
total loads, surface, sections	identical
sectional loads	identical once computed
unsteady plots history	identical
probe points off the body	not identical

- Opens a copy, cuts the sections, exports, closes. Never solves or saves.
- p002 may ask sections, their loads and the surface only.
- 26.124 only. Unsteady: the last instant.
- On the cluster: submitted, recorded `SUBMITTED`; `collect` records `EXTRACTED`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 10

Grouped runs

A grouped run: several points, one solver

```
pyfs-matrix plan matriz.fs --workspace . --batch 4     # decide the split
pyfs-matrix run  matriz.fs --workspace . --batch 4     # run the receipt's jobs
pyfs-matrix collect --workspace . --watch              # finish every point
```

What it is

A grouped run solves several points of a polar, or several polars, in **one solver instance**, in order.

- `--polar-sweep`: one job per polar; `--batch N`: N jobs.
- Steady (`steady`, `qsteady_rotor`) and unsteady polars join, never in the same job.
- Each acoustic polar is a job of its own.

The plan decides, the run obeys

`plan` groups the polars by processors, build, kind and user actions, and cuts batches of whole polars, the longest as short as it can be. It prints one line per job and records the split in `plan.json`.

The receipt gates the run

Another mode, N or selection, no job, or job folders that hold files: `run` refuses. Plan again.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

What a grouped plan leaves out

The polar	Why it stays out
a LEGACY row	grouped modes run the package's steady and unsteady run types only
a coupled row on `steady` or `qsteady_rotor`	its coupling loop starts only after the script ends, and on 26.124 the next point of the job crashed the instance
a steady row stating `COLD_START` false	a warm sweep; a grouped job re-initializes the solver before every point
a `RESTART` row	it opens a datapoint's saved simulation
an unsteady row on a build without the action command	the build does not document the unsteady solver action command
a polar whose points do not splice into one instance	for one, a steady point that initializes the solver twice
a polar whose points are all recorded	nothing is left to run

Takeaway *Named, never refused, in the plan and in `plan.json`: run those rows point by point. A geometry carrying saved solver actions blocks instead: `pyfs-mesh inventory <file> --clean`.*

Where a grouped job runs

```
sims/sim_<POL>/FULL-POLAR.txt                          # --polar-sweep: the polar's job
sims/batch/<matrix>_b<ID>/BATCH-<first>-<last>.txt     # --batch: the batch's job
sims/sim_<POL>/datapoints/DP-<point>/                  # each point's outputs, as always
```

- The batch ID is assigned by the package, never typed. While a batch runs, its simulations live in its folder, and every save and export names an absolute path.
- The job's folder is the working directory of the whole job: a relative `SCRIPT` file or a relative path inside a `COMMAND_LINE` action resolves there, and the plan warns, naming the folder.
- The job's clock replaces each point's: the job asks for a walltime of its own (next slide).
- `run` does not post. Every point is recorded `SUBMITTED`, also when the job runs on this machine (Windows, or `--local`): one instance, one license, one job at a time.
- `--force-rerun` and `--force-rerun-all` are refused beside a grouped option.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The walltime of a grouped job

The sum of its points

The `WALLTIME` cell (`4h`) is the budget of **one** datapoint of its row. A job asks for the sum over its points, capped at the profile's `max_walltime` with a warning that suggests a larger N. A row whose own cell is above `max_walltime` is refused.

BEST: the package prices it

The job's estimate times 1.25, plus the setup's `walltime_margin_s`, rounded up to the minute. In a mixed job each `BEST` row adds its priced estimate to the other rows' cells. An unknown `BEST` point takes `max_walltime`, with a warning, and is refused without one. A plan with neither `--batch` nor `--polar-sweep` refuses `BEST`, naming the row.

Takeaway *A point the job's clock stopped is `WALLTIME_REACHED`, not a failure. To run it again from the start in the next grouped plan, `pyfs-matrix collect --discard-walltime` marks it `FAILED_MARKED`, its outputs kept.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Collecting a grouped job

```
pyfs-matrix collect --workspace . --watch
pyfs-matrix collect --workspace . --discard-walltime   # clock-stopped: run again
```

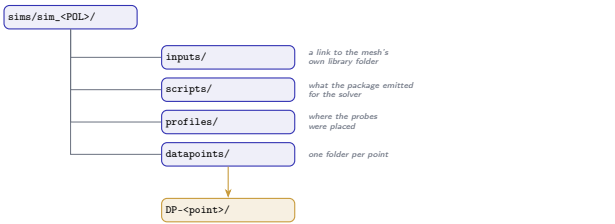
- `collect` reads the job's descriptor, solver log and end files in the job's folder, and completes each point as soon as its files settle, while the job still runs.
- Until a batch ends it **copies** each finished simulation to `sims/` and leaves the batch folder alone; once it ends, it **moves** them.
- It cuts the job's log into one log per point.
- A point that failed without stopping the instance does not stop the job. The points a stopped job never started are named, and the next grouped run takes them, and every failed point, again.
- `--hpc NAME` selects the profile when the workspace holds several, as on `plan` and `run`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 11

What a run leaves

A simulation folder



- `inputs/`: a link to the mesh's library folder.
- `scripts/`: what the solver was handed. The fastest answer to *what did this run really do*.
- `profiles/`: where the probes were placed.
- `datapoints/DP-<tag>/`: one folder per point.

Takeaway *Nothing here is edited. Everything here is reproducible from the inputs.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Inside a datapoint: what can be deleted

File	Holds	Read afterwards by
`<stem>.fsm`	saved simulation	`RESTART`. Delete it and the point cannot continue
`<stem>.txt`	loads	`polars/`, `campaign_sweep.csv`, the judge
`<stem>_sloads.txt`	sectional loads	`sections/`
`<stem>_probes.txt`	probe samples	`probes/`
`<stem>_plots.txt`	per-step plots (unsteady)	polar, rotor averages, reductions
`<stem>_cp.txt`	section pressures	per-distribution Cp tables
`<stem>.dat`	Tecplot surface	a viewer
`<stem>_log.txt`	solver log	`collect`; the first file to open

The stem is P<POL>-<point name>, for example P6112-M200AL+020BE+000; the name follows the declared condition keys.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Read the recorded outcome, not just the iteration count

Not a failure	
`CONVERGED`	reached its criterion
`COMPLETED_MAX_ITER`	ended without a convergence claim
`MAX_ITER_REACHED`	steady cap reached, residual unsettled
`RAN_MISSING_LOG`	ran, log evidence missing
`WALLTIME_REACHED`	clock stopped it
`SUBMITTED`	in the queue

Failure	
`FAILED_DIVERGED`	diverged
`FAILED_EXECUTION`	solver did not run
`FAILED_SCRIPT`	script rejected
`FAILED_INCOMPLETE_OUTPUT`	outputs missing
`FAILED_MARKED`	marked failed with a reason

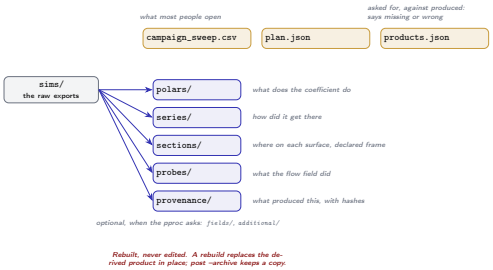
Takeaway *`WALLTIME_REACHED` and `COMPLETED_MAX_ITER` carry real numbers, and are read wrongly more often than any real failure.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 12

What post-processing leaves

Who writes the products



`run`	at its end, unless it submitted; no flag turns it off
`collect`	after a sweep that collected something, and only then; `--no-post`
`post`	when you ask; no seat, no solver

All three overwrite products in place without an archive by default (FR-397). Use `post --archive` or `collect --archive` to keep replaced products, including `series/`, under `archive/<day and hour>/`. `--force-overwrite` explicitly selects no archive and asks for confirmation.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The five product folders

Folder	Answers	Plot against
`polars/`	what the coefficients do as the condition changes; one file per group, body, stability and wind axes	`ALPHA` or `BETA`
`series/`	how it got there: loads, sections and probes per step	`azimuth_deg`
`sections/`	sectional loads, in each declared frame	`Offset`
`probes/`	the flow samples, and time-average, phase-locked, per-blade reductions	`STEP`, `AZIMUTH`, `BLADE`
`provenance/`	inputs, script, outputs with sha256, argv, build	makes a plot defensible

- A rotor row reduces **per rotor**: `<point>_per_blade_<ALIAS>.csv`.
- In `campaign_sweep.csv`, check `fs_build` first: the build the solver reported.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

additional/, and what the products hold

- post/<matrix>/additional/<pid>/: the additional post's polars, sections and, unsteady, plots reductions. Every products.json entry says "additional": true, pproc, extraction, derives_from; filter on it.
- Its sections table keeps the run's rows first, then the new ones; leading_sections counts the run's.
- An induced drag nobody computed is NA, with the axis columns it reaches. A steady probe line is exported once.
- Unsteady series/<point>_loads_series.csv: moments about the moment point.
- post.log.json beside post.log. Post a workspace with the package version that ran it.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Vector loads and window averages

- Steady polar axes use the exported force and moment vectors. `CDB = Cx`, `CLB = Cz`; sideslip points get rows.
- `CLS`/`CLW` fall by 0.10 to 0.25 per cent on 27 of 28 recorded lifting exports, including zero sideslip; one differs by 0.71 per cent. The cause of the export's lift discrepancy is unknown.
- Unsteady rotor tables are window averages of all six global `MRP` components over exactly the rotor's families. Missing history is a skip, never a last-step fallback. A run adds `ROTOR_<ALIAS>` if needed.
- `RPM_<alias>` and `DIAMETER_<alias>` join the rotor header; `J_<alias>` uses the free stream. $\mathrm{ETAW} = J\,\mathrm{CTW}/\mathrm{CP}$ uses the whole force in wind axes and equals `ETA` along the stream.
- `ETAW` and the shaft angle follow the stream direction at incidence or sideslip; a re-post needs no solver.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Read columns by name

- Condition: `ALPHA`, `BETA`, `MACH`, `RE`, `VINF`, `VREF`, `ALT`, `RHO`, `TEMP`, `MU`, `J`, `SREF`, `CREF`, `BREF`. Plots keep the export's own headings.
- Unsteady polar: `P<sim>_<name>_uns_avg.csv`, beginning with `FIRST_STEP`, `LAST_STEP`, `STEPS`; adds moment point, group-suffixed axes, equations and super-file content.
- Axes need all six global `MRP` plots. A new run supplies `MRP_TOTAL` if absent; a history without them has a `#axes` skip.
- Reductions carry `ROTOR` and `XMOM`, `YMOM`, `ZMOM`, not `Time-step` and `Time (sec)`.
- Sections: `STEP`, `FAMILY`, `PLANE`, `ROTOR`, `AZIMUTH`; still one instant. Azimuth is blade one of that row's rotor, or `NA` without one.
- Series use `STEP`, add the condition block; probes add `PROBE`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

One row per blade, or per azimuth

- `per_blade`: one row per blade, each blade's own rotor clock and matrix window, `BLADE, FAMILY, AZIMUTH_START, AZIMUTH_END`. A blade's family suffix is removed from its own plot columns so all blades share headings.
- Declared `[phase_locked]`: one row per azimuth of the last revolution, averaged across the requested revolutions. Blade columns align by each blade's own azimuth; other columns use blade one.
- The row's total revolutions must be *at least* `min_revolutions`. A shorter run loses only that table. Without the pproc declaration, the file remains a passage series.
- Editing the matrix window moves the reductions together at post; editing the pproc gate or depth also needs no new solver run. Records stay as written.
- Read `products.json`: `complete`, `interrupted`, sources, windows and skips. `#axes`, `#equations`, `#names` count for `post --strict` even if the rest of the file was written.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Collection and manifest limits

- An early export at an iteration-limit run, confirmed by the solver log, and a velocity mismatch are failures and leave the products.
- A continuation's `continues` field excludes its predecessor from products and the sweep table.
- Stamped series exports are found in both simulation and datapoint folders; a missing kind is a skip, not a header-only product.

Manifest ownership

A live writer renews its owned lock every 5 seconds. Takeover requires a dead local owner or a heartbeat older than 300 seconds. Finish every older writer before a newer one uses the same workspace.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Reading a result honestly

- For unsteady, `CONVERGED` checks the last time step's inner iterations. Whether the history has settled is the user's judgement. It does not mean the mesh was fine, the reference right, or the physics the one you meant.
- A curve that stops early at `WALLTIME_REACHED` is real up to that step. Continue it, do not repeat it.
- One diverged point in a swept row is judged on its own.

Limitation

The workspace makes a study reproducible and traceable. That is not the same as correct, and no amount of green status is evidence of the second.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Re-posting a recorded campaign

- Run `pyfs-matrix post --workspace <root>` without a solver. Records remain unchanged; each product is written or skipped with its reason.
- Nothing blocks by default: frozen-solve averages are WARNED in `post.log` and written; `--check-frozen` withholds them. The unsteady polar's `ADVANCE_RATIO` states the point's `J`.
- Every unsteady probe uses fluid plots. A cited profile recorded only as an instant needs a new run for its missing history.
- Each distribution gains `_sloads_<name>.csv` and `_cp_<name>.csv`, with every available exported `STEP`.
- New VTK/CSV exports and the pproc surface-averaging window need a new solver run. Guides 6 and 7 have the keys [4].
- `integrate = true` on a sections entry adds the strip force and moment columns on re-post, without a solver run.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 13

Windows and an HPC cluster: two workspaces and sync

Two workspaces, one of them the main

The main workspace, on Windows

- `C:/<work-folder>/fts-workspace`: **the reference**. Its `runs.json`, its `post/` and its matrices are the record everybody reads.
- It is the one under version control.
- Runs can be done here: planning, one point, a steady sweep.

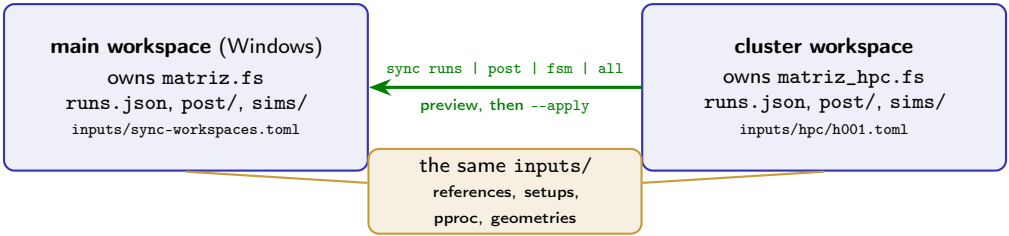
The cluster workspace

- Where runs are **submitted** to the queue and collected: `<cluster-root>/fts-unix/fts-workspace`.
- The same `inputs/` library as the main: references, setups, pproc files, geometries.
- Its own matrices, its own `sims/` and `runs.json`.

Takeaway *Runs are done in either workspace.* `pyfs-workspace sync`*, run on the main, brings the cluster's runs and results home.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

How they relate



- **Every matrix is owned by exactly one workspace.** The owner edits it; the other only receives it.
- sync runs **on the main** and pulls. It never writes into the cluster workspace.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

inputs/sync-workspaces.toml

```
main = "windows"

[workspaces.windows]
path = "."
matrices = ["matriz"]

[workspaces.cluster]
path = "//<server>/<share>/fts-unix/fts-workspace"
matrices = ["matriz_hpc"]
```

- `main` names the entry you stand in; `sync` is run there.
- One `[workspaces.<name>]` table per workspace: its `path` (as the main sees it) and the `matrices` it **owns**, by stem.
- A stem declared by two workspaces refuses the sync, and so does a matrix file that no workspace declares.
- `--from cluster` pulls from one workspace; without it, from every other one.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Four levels, each one the previous plus more

```
pyfs-workspace sync post --workspace .            # preview: changes nothing
pyfs-workspace sync post --workspace . --apply    # then do it
```

Level	Brings
`runs`	the records in `runs.json`, and the provenance each needs to be believed: every simulation's `scripts/` and its datapoint logs
`post`	the above, plus the `post/` folder, its products and their provenance
`fsm`	the above, plus the saved simulation of every datapoint
`all`	the above, plus everything else under `sims/`

Takeaway *Every call previews by default and changes files only with `--apply`. Read the preview first.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

How a conflict is settled

Run records and files

- Only in the other workspace: added. Identical: left alone.
- Different: if main's record is still `SUBMITTED`, the other's wins; otherwise **main wins** unless `--prefer-other`.
- A file in both with different content: main's copy stays unless `--overwrite`, which archives it to `archive/sync-<stamp>/` first.

Matrices: ownership decides

Any difference between main's copy and the source's is reported as a **MERGE CONFLICT**, every time, and **the owner's copy wins**. When the source owns it, main's copy is archived and replaced; when main owns it, main's copy stays. `--prefer-other` and `--overwrite` do not apply to matrices.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

What sync never does

- **It never copies a mesh.** A synced simulation's `inputs` is linked again into the main's own `inputs/geometries/`, to the same geometry folder. A geometry the main library lacks is not linked, and the sync says so.
- **A run still `SUBMITTED` there syncs its record only**: nothing finished is worth copying, and a job may still be writing.
- It deletes nothing in the main and never touches `inputs/`, except a declared matrix.
- It never brings back a run id that `delete-sims` retired in the main.
- `runs.json` is archived before it is rewritten. A lock in the main (a run in progress) refuses the sync; a lock in a source skips that source, with the reason.

Takeaway *Every call, one entry per source workspace, is recorded in `storage_management.json` at the main's root.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

A routine that works

1. On the main: edit and plan the matrices it owns; commit them.
2. Bring the cluster's `inputs/` up to date from the main (the site's copy procedure), before planning there.
3. On the cluster: plan, run (it submits), `collect` until nothing is outstanding. Edit only the matrices the cluster owns.
4. On the main, in Windows:

```
pyfs-workspace sync post --workspace .          # read the preview
pyfs-workspace sync post --workspace . --apply
```

5. Commit the synced matrices on the main, with the reason.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 14

Space on disk

Three commands that keep a workspace small

```
pyfs-workspace space-in-use --workspace .                # sizes; changes nothing
pyfs-workspace free-space m001 --workspace . --apply     # inputs/management/m001.toml
pyfs-workspace delete-sims --sims 4001 2009 --workspace . --apply
```

- `space-in-use`: sizes by top folder, by `sims/sim_*` and by extension (`--top N`).
- `free-space`: prunes per-step exports, compacts simulation folders into `sims/sim_<id>.zip`, deletes files of named extensions, compacts or deletes `post/<matrix>/archive/`. A compacted folder is restored, hash checked, the moment `post`, `collect` or a continuation needs it.
- `delete-sims`: a simulation, its own products and its records; a product shared with other points needs `--matrix-products`.
- The `.fsm`, `scripts/`, logs and every file a record names are never deleted; neither is the mesh, whose link is undone first. Every call is recorded in `storage_management.json`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Per-step exports: keep the last step

- An unsteady point that exports along its march (the surface of an averaging window, the stamped series) leaves one file per step. They are the largest thing a campaign keeps.
- The recipe table `[[prune_step_exports]]` of `inputs/management/m<id>.toml` deletes them and **keeps only the last step's** export of each unsteady point. Like every storage call it previews first and deletes only with `--apply`, and the call is recorded.
- Products already written stay as they are.

A later post that needs a deleted step

is **refused**, naming the point and the step that is gone. Nothing is averaged over a window with holes in it. Post everything a campaign needs before freeing its steps, or run the point again.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 15

Version control

What version control carries

In	Why
the matrices	the study; written, not produced
`inputs/`	every file a row cites
`ansa/`	the script encodes the decision
`docs/`	a second person needs it
`builds/`	a site decision: a checkout is complete

Out	Why
`executables.local.toml`	true of one machine only
the meshes	large, derived from `ansa/`
`sims/`, `post/`, `archive/`	produced
`pfs-wheels/`, `py-venv/`	binaries, absolute paths

Takeaway *Version inputs, not run products. The generated `VARIABLES.md` and `WRITING-EQUATIONS.md` in pproc may be committed or ignored.*

Version control lives with the main workspace

- **The main workspace is the checkout.** The cluster workspace holds a copy of its `inputs/`, not a checkout.
- Every matrix of the study is in version control, whichever workspace owns it: `sync` brings the cluster's matrices to the main, where they are committed.
- `plan` checks every POL against **every** `.fs` of the workspace it runs in.

Why a stale copy lets a collision through

`--update-ids` picks the next free POL from the matrices, run records and `sims/` **of its own side**. A POL used only on the cluster is invisible on the main until a sync brings its matrix, and the cluster sees a new main row only once its inputs are refreshed.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The ritual that keeps `plan` honest

1. **Update** the main checkout, and **sync** it from the cluster, before touching any matrix.
2. **Edit and plan the main's matrices on the main**, where every matrix sits side by side, so a repeated POL is refused there.
3. **Commit** at once, with the reason in the message.
4. **Refresh the cluster's inputs** from the main before planning or running there.
5. **Edit a cluster-owned matrix only on the cluster**, and bring it home with the next sync; commit it on the main before its results are used.

Takeaway *The main workspace is the one place every matrix meets. A plan run anywhere else is only as right as the last refresh.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The daily cycle, and the two commits that hurt everyone

Every day

- **Update and sync before you plan.**
- **Commit an input change with its reason**: which row, and what changed about the physics.
- **Never commit while a run is writing.**

Never commit

- `sims/` after a campaign: hundreds of megabytes in every colleague's next update.
- `executables.local.toml`: it silently redirects everybody who updates.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 16

Reference

Refusals you will meet first

Message contains	Do this
`POL(s) are stated more than once across the matrices`	renumber, or `plan ... --update-ids`; then commit and refresh
`cites the probe survey ... holds no such file`	put the survey in `inputs/profiles/`
`is already in a scheduler's queue as`	`pyfs-matrix collect` first
`maps no alias for build(s)`	add the build's line under `[builds]`
`and nothing says which cluster this machine is`	select one: `--hpc <NAME>` on plan, run and collect
`the record it continues does not say where it stopped`	use `{ADDITIONAL_ITERS=n}`
`which is not a run that STOPPED with more to do`	drop `RESTART`; nothing to continue
`TRANSLATE DISTANCE is ..., which is not a finite number`	write metres, as `0.05`
`is ignored: it is either a non-existing path`	the wheels folder is not there (Guide 9)
`bad interpreter`	the environment was moved; rebuild it in place

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Refusals to resolve before using results

- No unsteady window: add `LAST_REVS_AVG` or `LAST_ITERS_AVG`. `WINDOW_*` keys are refused.
- Duplicate attitude: keep `ALPHA/BETA` in one cell. Duplicate stabilisation: keep one preset interface.
- Linked file destination or names differing only by case: use ordinary, distinct staging destinations and update references.
- Unknown export-frame rotation: export loads in global `MRP`.
- Export area/chord mismatch: reconcile the project and reference `SREF/CREF`; the simulation otherwise gets no product.
- Invalid plot name: letters, digits, underscores and bare `{family}` only. Ambiguous names skip rotor histories or axes blocks.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Forms that are refused, and what to write

Each is refused naming its replacement:

- `iteration=`: write `step=`.
- `run.assess_unsteady_from_plots`: use `LoadsAssessor`.
- The three `WINDOW_*` row keys: `LAST_ITERS_AVG` or `LAST_REVS_AVG` (degrees divided by 360).
- Group-member lists: one alias per group, members in the reference's `[aliases]`.

The `broken_commands` reader stays as plain compatibility. Existing run records are never rewritten.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Cheatsheet

Step	Windows (main)	Cluster
get in	`.venv\Scripts\activate`	`source $FTS/bin/pfs0410-env`
get out	`deactivate`	new shell, or `exit`
update	update the checkout	refresh the inputs from the main
plan	`pyfs-matrix plan <matrix>.fs --workspace . --fs-version 26.123`	same
POL fix	`... --update-ids`	same, then sync and commit on the main
run	`pyfs-matrix run <matrix>.fs --workspace . --fs-version 26.123`	same; submits
collect	not needed	`pyfs-matrix collect --workspace .`
sync	`pyfs-workspace sync post --workspace . --apply`	not run here
post	`pyfs-matrix post <matrix>.fs --workspace .`	same
continue	`RESTART: {FINISH_PENDING}` in the free cell	same
space	`pyfs-workspace space-in-use --workspace .`	same

`$FTS` is `<cluster-root>/fts-unix`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Where to go next

- **The other guides**: 1, overview; 3, GUI to pyfs; 4, cheatsheet; 5, references; 6, solver setup; 7, post-processing definitions; 8, FSI; 9, Python on an offline machine.
- **The package**: its own documentation [1, 2].

Cite the version

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`.

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

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

[5] Geovana Neves, *pyflightstream documentation*, page "Flight conditions". https://nevesgeovana.github.io/pyflightstream/latest/flight-conditions/

[6] Geovana Neves, *pyflightstream documentation*, page "Optional Excel matrix workbook". https://nevesgeovana.github.io/pyflightstream/latest/excel-matrices/

REF-01 International Organization for Standardization, *ISO 2533:1975, Standard Atmosphere*, ISO, Geneva, 1975.

REF-02 F. M. White, *Viscous Fluid Flow*, 3rd ed., McGraw-Hill, New York, 2006. Sutherland's law for the viscosity of air.

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

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

Copyright (c) 2026 Geovana Neves, CC BY 4.0

References (continued)

REF-38 W. R. Sears, "Some aspects of non-stationary airfoil theory and its practical application", *Journal of the Aeronautical Sciences*, 8(3), pp. 104-108, 1941.

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

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.

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

Copyright (c) 2026 Geovana Neves, CC BY 4.0