Guide 7: Post-processing definitions

What every product, reduction and window means

Written for pyflightstream 0.58.2.

Source: the package page "Post-processing definitions" [1].

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

The definition of record

- `docs/post-processing-definitions.md` defines every post-processing product the package writes.
- Each definition states the requirement a product is built to.
- Where code and page disagree, the page is right and the code is a defect.

Takeaway *Read the definitions before changing a reduction, an averaging window or a product's columns. The implementation is not its own specification.*

Contents

The vocabulary
What every product states
The axes of a steady polar
The sections table
`time_average`
`per_blade`
`phase_locked`
The averaging window
The unsteady POLAR
Rotor coefficients
The quasi-steady rotor
What the pproc asks the solver for
The additional post
What the package does not judge
The token for a value that does not exist

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 1

The vocabulary

Turns, passages and windows

Term	Meaning
Revolution	One full turn of a rotor: $60/\mathrm{rpm}$ seconds, $60/(\mathrm{rpm}\,\mathrm{dt})$ solver steps
Blade passage	One revolution divided by the blade count
Azimuthal position	Where a blade is in its turn, 0 to 360 degrees
Window	An inclusive, 1-based range of solver steps that a reduction averages over

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Rotor names and reference points

Term	Meaning
SMRP	A rotor's own static moment reference point, <ALIAS>_SMRP
MRP	The global moment reference point of the aircraft
Alias	The name of a rotor, and of the integration group built from its families

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 2

What every product states

The condition block: angles and velocities

Every table the post stage composes states its condition in one block, in the order shown across these two frames.

Column	Unit	Meaning
`ALPHA, BETA`	deg	Angles the solver reports it ran at, as the row wrote them
`MACH`	-	Mach number of that point
`RE`	millions	Reynolds number the solver reports
`VINF`	m/s	Free-stream velocity the solver reports
`VREF`	m/s	Solver reference velocity, used to normalise a coefficient

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The condition block: atmosphere and references

Column	Unit	Meaning, continuing the block
`ALT`	ft	Altitude the row states; `NA` where it states none
`RHO`	kg/m^3	Air density the run resolved for that point
`TEMP`	K	Air temperature the run resolved for that point
`MU`	Pa s	Dynamic viscosity, written in scientific notation
`J`	-	Advance ratio requested; `NA` for RPM input or no rotor
`J_CLOCK`	-	Advance ratio the clock rotor ran at, $V/(nD)$; `NA` where the record or the reference does not say
`RPM_CLOCK`	rev/min	Speed the clock rotor turned at, with its hand; `NA` with no clock
`SREF`, `CREF`, `BREF`	m^2, m, m	Reference area, chord and span

Copyright (c) 2026 Geovana Neves, CC BY 4.0

A coefficient needs its normalisation

A coefficient is a force divided by

$$\frac{1}{2}\rho V^2 S.$$

- Area alone does not let a reader recover a force or compare campaigns. Density and the velocity used in the division must be stated too.
- `ALT` cannot stand in for density: a row may pin density directly.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Which velocity normalises which product

- The solver uses `VREF`. The steady polar's twenty-four coefficients are the solver's own and use `VREF`.
- The plots table, every reduction of it and the unsteady polar are rescaled to the free stream by $(\mathrm{VREF}/\mathrm{VINF})^2$ and use `VINF`.
- The rotor table recovers Newtons with `VREF`, undoing the solver's division. Its coefficients use $\rho n^2 D^4$ and carry no velocity.

Takeaway *Both velocities are stated. When they differ, post warns and names the point.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Reference area and chord must match the export

- The solver divides by the area and length in the project it opened. The loads export prints both.
- The package sets neither. It states the reference artifact's `SREF` and `CREF`, and checks them against the export.
- A difference beyond the export's printed precision is a **warning** in `post.log` naming both numbers, and every computable product is still written; `--check-frozen` refuses the simulation's products instead.

Takeaway *A reference beside a coefficient must be the reference that divided it.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Where the condition block appears

- The steady polar already carries `ALPHA`, `BETA`, `MACH` and `RE` among its twenty-four columns. It adds the rest beside them.
- `probes/<point>_plots.csv` states **no condition**. It keeps the export's own header.
- Reductions read every column of that plots table back as a plotted quantity.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 3

The axes of a steady polar

One exported force and moment, three axis systems

The loads export gives one force and one moment per surface in the geometry frame: **x aft, y right, z up**. Sum over the group's surfaces, then turn that pair.

Columns	Axes	How
`CDB, CYB, CLB, CRB25, CMB25, CNB25`	Body: forward, right, down	Half a turn about y. $\mathrm{CDB} = \mathrm{Cx}$ and $\mathrm{CLB} = \mathrm{Cz}$ of the export
`CDS ... CNS`	Stability	Body axes turned by $-\alpha_s$ about y
`CDW ... CNW`	Wind	Stability axes turned by β_w about z

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Force signs and moment normalisation

- Drag opposes $+x$ and lift opposes $+z$ in each system. Side force keeps its sign.
- Turn the moment as **one vector in one length**, then normalise.
- Divide `CR` and `CN` by the span, and `CM` by the chord.
- Under sideslip, this moves pitch into roll by the ratio of chord to span.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The rotation angles come from the flown velocity

The solver turns sideslip about body z first, then incidence. It flies

$$(u, v, w) = V(\cos a \cos b,\ \cos a \sin b,\ \sin a), \qquad \alpha_s = \operatorname{atan2}(w, u), \quad \beta_w = \arcsin(v/V).$$

- These angles equal the written `ALPHA` and `BETA` whenever either is zero. Otherwise they differ at second order.
- At `ALPHA 4, BETA 2`, they are 4.0024° and 1.9951°.
- The `ALPHA` and `BETA` columns keep what the row wrote.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Wind-axis drag and the solver's integrals

$$\mathrm{CDW} = \mathrm{CDi} + \mathrm{CDo}.$$

- Wind-axis drag from the export's own vector equals the drag the solver integrates, including under sideslip.
- Recorded exports confirm this to their printed precision.
- `CD0` and `CDI` state the solver's drag integrals. On build 26.125, `CDV` and `CDP` also state the viscous and pressure drag split when the export carries it.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Wind-axis lift comes from the exported vector

- `CLW` is the vector's wind-axis lift, not the export's `CL`.
- On 27 of 28 recorded lifting exports with lift above 0.05, the printed `CL` is 0.10 to 0.25 per cent higher. One is 0.71 per cent higher. The cause is unknown.
- Every polar axis column uses one source, keeping `CDB` and `CLB` consistent with the export's `Cx` and `Cz`.
- Older tables used the solver's `CL` directly, so their `CLS` and `CLW` are higher by those amounts.

Recorded evidence: `tests/tier1_offline/fixtures/recorded_total_rows.csv`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

An induced drag the solver did not compute is NA

- A boundary on the vorticity induced-drag list (SET_VORTICITY_DRAG_BOUNDARIES) without a defined trailing edge is not computed, and the export prints its CDi as zero [2, p.202].
- When the point's run record puts a surface on the list and its printed CDi is exactly zero, the group's CDI is NA, and so is every axis column the export's x force reaches at that point's angles: CDB, CDS, CDW; CLS and CLW off zero incidence; CYW under sideslip.
- The moments, CYB, CLB, CYS and CD0 keep their numbers; the fixed-width custom polar writes nan. post.log names the declined surfaces.
- A trailing-edged surface whose induced drag rounds to zero at the printed precision reads NA too; significant_digits narrows that band.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 4

The sections table

One export contains every declared distribution

- `sections/<point>_sections.csv` is one solver export.
- It holds every distribution the pproc declares: the wing in `XZ`, each blade in its own frame.
- The distributions follow one another with no marker between them.
- Leading columns identify which distribution each row belongs to.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The sections table: step and distribution identity

Column	Meaning
`STEP`	Unsteady: the run's last time step, from the run record. The export is written when the march ends; its own header counts inner iterations, not time steps. Steady: the solver iteration the export states. One name across every table
`FAMILY`	Geometry families of the row's distribution, joined by +
`PLANE`	Cutting plane of that distribution

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The sections table: rotor identity

Column	Meaning, after `PLANE`
`ROTOR`	Rotor whose blades those families are; `NA` for a surface no rotor owns
`AZIMUTH`	Where the block's own blade is at `STEP`, in degrees, wrapped to one turn; otherwise blade one of its rotor; `NA` without a rotor

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Each rotor has its own azimuth clock

$$\mathrm{AZIMUTH} = \left(\mathrm{blade1.azimuth_deg} + \mathrm{sense}\,\mathrm{STEP}\,\frac{360}{\mathrm{steps_per_revolution}} \right) \bmod 360.$$

- The datum, the sense of rotation and the steps per revolution all come from **that rotor**.
- Sense is the sign of the rotor's speed.
- For a block cutting only blade n, add $(n-1)360/B$ to this blade-one angle, regardless of the rotation sense; B is blade count.
- Two rotors at two speeds have two azimuths at the same step. A wing has none.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Distribution identity comes from the run record

- The run records which distribution is which. The script states surfaces by index; post cannot recover their names from that alone.
- An older record writes `NA` in all four identity fields: `FAMILY`, `PLANE`, `ROTOR`, `AZIMUTH`.
- The same rule applies when the recorded blocks do not add up to the number of exported rows.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The sections table is one instant

- At an unsteady point, the distribution is at `STEP`. It is not an average over the window.
- Its `products.json` entry states `"kind": "instant"`.
- The history is `series/<point>_sections_series.csv`, with the same identity on every row.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

One loads file and one Cp file per distribution

- Each `[[sections.distributions]]` entry gets `sections/<point>_sloads_<name>.csv` and `sections/<point>_cp_<name>.csv` when its export is available.
- The name preserves the alias or joins the declared families with `-`. Filename-invalid characters become `_`; collisions receive the entry's 1-based position suffix.
- All planes and expanded blade blocks of that entry share its file. Cp now has a table, rather than only a manifest listing.
- With per-step exports enabled, every exported step appears in ascending `STEP` order. Without them, the file holds the end-of-run export.
- The combined sections table and combined sections series remain available.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Distribution columns and recorded identity

- Rows start with `STEP`, `time_s`, `FAMILY`, `PLANE`, `ROTOR`, `AZIMUTH`, then the condition block and export columns. Cp also carries `SECTION`, the export's 1-based cross-section index.
- Missing values are `NA`. The manifest records the distribution, original families and `steps_tabled`.
- A new run records each block's distribution entry. A record can be split only when its recorded layout and pproc identify the blocks unambiguously by families, plane, frame and count.
- Missing or ambiguous identity, mismatched counts, malformed exports and unavailable steps are named skips in `products.json`. Editing today's pproc cannot supply missing recorded identity.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 5

time_average

One average of the whole point

- `time_average` is one average of the whole point over one window.
- An unsteady POLAR row is built from this average.

Takeaway *One window per point, not one per rotor: a point has one history.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 6

per_blade

One shared window, one row per blade

- `probes/<point>_per_blade_<ALIAS>.csv` averages each blade over the last part of the run: the matrix row's averaging window.
- The blades of one rotor share its window; each rotor uses its own clock. Each row states that blade's start and end azimuth.
- The rotor window follows `LAST_REVS_AVG` or `LAST_ITERS_AVG` on the matrix row.
- There is one row per blade.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The same time interval makes blades comparable

- Each blade starts at a different azimuth. Its azimuth columns state that difference explicitly.
- A separate passage window for each blade would also sample each blade at a different time, mixing azimuth and history differences.
- Averaging from step one mixes the transient with the answer.

Takeaway *The shared window compares blades of one rotor at the same times.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The leading columns of each blade row

Each row leads with these fields, in this order:

1. `REDUCTION`, `ROTOR`, `BLADE`, `FAMILY`.
2. The window: `FIRST_STEP`, `LAST_STEP`, `STEPS`.
3. `AZIMUTH_START`, `AZIMUTH_END`.
4. The condition block, then the moment point.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Blade plot columns share one heading

- A blade's columns are plots named for its family. A plot is `<parameter>_<group>`; a group cut per blade is named for the blade's family.
- Blade one's `CL_MRP_Blade1` and `FX_LOCAL_Blade1` become `CL_MRP` and `FX_LOCAL` in its row. Removing the family makes the blades line up under one heading.
- The pproc requests these with `families = "each"` or `frame = "LOCAL_AXIS"` over the rotor.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Start and end azimuth belong to that blade

At the first and last step of the shared window, use

$$\text{azimuth} = \left(\text{blade1.azimuth_deg} + \text{blade position} \, \frac{360}{\text{blades}} + \text{sense} \, \text{step} \, \frac{360}{\text{steps_per_revolution}} \right) \bmod 360.$$

- `blades` is the rotor's count. A periodic sector carrying two blades of four spaces them a quarter turn apart.
- Without the rotor's clock, the azimuths are `NA`. The averages are still written.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The reference identifies the blade families

- The rotor block's `families_blades` in the reference artifact identifies its blades.
- The run also records these families.
- A row using flat rotor keys and citing no block gets no per-blade table. `products.json` states why.

Part 7

phase_locked

A mean at a fixed azimuth across revolutions

1. Take the last `last_revolutions_avg` revolutions.
2. At each azimuth, average the samples at that same position across those revolutions.
3. Tabulate the result by azimuthal position, 0 to 360 degrees.

Takeaway *With three revolutions, three datapoints enter each azimuth's mean. The row is an azimuthal position, not a passage, revolution or blade.*

Blade and alias means in one file

- Do the fixed-azimuth average for each blade on its own, about that rotor's `SMRP`.
- Then do it for each alias, about the global `MRP`.
- Put all of it in one file with columns suffixed `_{alias or blade_name}_{SMRP or MRP}`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The azimuthal table's leading columns

`probes/<point>_phase_locked[_<ALIAS>].csv` leads with:

1. `REDUCTION`, `ROTOR`, `AZIMUTH`, `STEP`, `REVOLUTIONS`.
2. The steps the revolutions span: `FIRST_STEP`, `LAST_STEP`, `STEPS`.
3. The condition block and the moment point.
4. The plotted columns, under the names the export prints.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Rows use the rotor's last revolution

- There is one row per solver step of the rotor's last revolution, sorted by azimuth from 0 towards 360.
- `AZIMUTH` is where blade one is, using the sections-table formula.
- `STEP` is the step of the last revolution that azimuth falls on.
- `REVOLUTIONS` is the number of samples entering the mean. It equals `last_revolutions_avg` when that is a whole number.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Blade columns align by each blade's own azimuth

- A column ending in a blade family of the rotor is sampled where **that blade** is at the row's azimuth.
- The blade sits its position times $360/\mathrm{blades}$ after blade one. This offset makes two blades line up azimuth for azimuth.
- Every other column, including a rotor total or the aircraft, is tabulated by blade one's azimuth.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Azimuthal suffixes come from the plots

- A plot is <parameter>_<group>. The pproc names the group.
- The suffix _{alias or blade_name}_{SMRP or MRP} is what a group named PUSHER_SMRP or cut per blade prints.
- The package does not rename these columns.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Samples between history steps are read linearly

- One revolution earlier falls a whole number of steps earlier only when `steps_per_revolution` is whole.
- A blade offset falls on a whole step only when the steps per revolution divide by the blade count.
- Where both hold, every sample is a history row and nothing is interpolated. Between steps, the history is read linearly.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The azimuthal mean needs a rotor clock and depth

- Without blade one's datum or the rotor's signed speed, an azimuth cannot be stated. The table is a named skip.
- Declare the rotor in the reference and have the matrix row cite it.
- A depth below one revolution is a named skip too.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The pproc declares the gate and averaging depth

```
[phase_locked]
min_revolutions      = 4.0
last_revolutions_avg = 2.0
```

- `min_revolutions` is the minimum **total** revolutions required for the table to exist. Equal or greater generates it.
- Compare the revolutions the matrix row states it turns, not the length of the exported window.
- `last_revolutions_avg` is the depth of each azimuthal mean and may not exceed `min_revolutions`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The gate and depth can change at post

- The declared `[phase_locked]` table binds.
- `pyfs-matrix post` reads the current pproc again.
- Editing the gate or depth needs no solver re-run.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

A short run loses only the azimuthal reduction

- A row turning fewer total revolutions than the minimum skips this reduction.
- Its POLAR, per-blade table and every other product are written as usual.

Takeaway *Missing the threshold skips the reduction; it does not refuse the run's products.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

An undeclared table keeps the passage series

- A pproc without `[phase_locked]` still gets the averaging window cut into consecutive blade passages, one row per passage.
- This is the passage series written under this name.
- Declaring the table selects the fixed-azimuth mean instead. Leaving it absent preserves the passage series.

Takeaway *Absent is not zero: there is no minimum gate on the undeclared passage series.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 8

The averaging window

The matrix row states the averaging window

Column	Run type	Required	Unit
`LAST_REVS_AVG`	`unsteady_rotor`	Yes	Last revolutions; accepts a float
`LAST_ITERS_AVG`	`unsteady`	Yes	Last iterations

The window belongs on the matrix row beside `DELTA_TIME`, `TIME_ITERATIONS` and `RPM`. It does not belong in the pproc: these quantities define its temporal setup.

Window keys use the exact upper-case spelling

- `VAR_NAMES_VALUES` keys match the exact spelling.
- A lower-case `last_revs_avg` on a workflow row is refused as a key of no run type.
- Where that check does not run, the key is not read at all and its intended window is not applied.

Retired window keys are refused; the refusal names the key to write

- `WINDOW_STEPS`, `WINDOW_REVOLUTIONS` and `WINDOW_DEGREES` are retired.
- A row using one is refused at plan time, naming the replacement, even beside its replacement: `LAST_ITERS_AVG` for steps, `LAST_REVS_AVG` for revolutions, degrees divided by 360.
- These keys bind the `time_average` window and the passages cut from it. The unsteady POLAR is built from that time average.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

New plans and older records without current keys

- A new plan with **no window key** is refused.
- A retired `WINDOW_*` key no longer permits planning: it is refused with the key to write instead.
- An older record without the current keys is averaged over the window the run was given, with a warning naming the steps.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Retired compatibility forms are refused by name

Each retired form is refused, naming its replacement: `iteration=` (use `step=`), `run.assess_unsteady_from_plots` (use `LoadsAssessor`), the three `WINDOW_*` row keys (`LAST_ITERS_AVG` or `LAST_REVS_AVG`; degrees divided by 360) and group-member lists (one alias per group, members in the reference's `[aliases]`). The `broken_commands` reader stays as plain compatibility with no countdown; existing run records are never rewritten.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 9

The unsteady POLAR

The unsteady POLAR averages the plots history

- The source is the plots history, time-averaged over the row's window. It does not read the native coefficient export for that average.
- Columns are plot variables under the names the export prints, rather than the steady polar's fixed twenty-four columns.
- Flight-condition and reference-length columns are added from the workspace.
- The native export states only the last time step, one instant of a rotor cycle. It still ships as a health check.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

One file per simulation, one row per point

`polars/P<sim>_<name>_uns_avg.csv`

- Every file under `post/` comes from a sweep. The name states that this is an average of unsteady history.
- It carries the `P` used by every per-point product.

Each polar row states its window and moment point

1. `FIRST_STEP`, `LAST_STEP`, `STEPS` open the row and identify that point's averaging window.
2. The condition block follows.
3. Then `XMOM`, `YMOM`, `ZMOM` state the moment point.

Takeaway *The plots carry moments. A moment needs its point.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The super-file content joins the polar

After the plot columns, the table adds:

- The matrix row's cells and the run record's scalars.
- Each rotor's speed.
- The solver flags.

Takeaway *For an unsteady point, this content is part of the polar, not a second super file.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Each global plot group supplies eighteen axis columns

- The axis coefficients follow the plot columns, using the steady polar's names in this order: `CDW .. CNW25`, `CDS .. CNS25`, `CDB .. CNB25`.
- Each name takes the plot group's **whole** name as a suffix, for example `CLW_MRP_TOTAL`.
- A second global-frame group adds its own eighteen columns and renames none.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The unsteady axes use dimensional global loads

- Use `FX, FY, FZ, MX, MY, MZ` from a pproc plot group with `frame = "MRP"`, in Newtons and Newton metres.
- Average these components over the row's window. Divide by that row's $\frac{1}{2}\mathrm{RHO}\,\mathrm{VINF}^2\,\mathrm{SREF}$ and also by `CREF` for moments.
- Turn them as the steady-polar axes are turned, with the same force signs and final moment normalisation.
- A rotor's own frame cannot supply this block: its axes are not the geometry's axes.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Missing global plots and the axes block

- If the pproc has no global-frame group plotting all six components, the run adds `MRP_TOTAL` over every boundary, where the run has an `MRP` frame.
- A pproc that already plots them stays unchanged, to the byte.
- An older run without those plots has no axes block. `products.json` records this under `polars/<file>#axes`.

Takeaway *Missing source plots do not produce an axes block filled with `NA`.*

The pproc can declare output column names

The package cannot infer which plot label carries which coefficient. Undeclared names pass through exactly as the export prints them.

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

This dictionary maps an exported plot column to the name a downstream reader wants. Inventing a source label would otherwise risk an entire column of `NA`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The names dictionary changes selected headings last

- It renames the unsteady polar and averaged reductions: `time_average` and the passage series.
- `probes/<point>_plots.csv` keeps the export's names because it is the source table.
- Per-blade and azimuthal tables are not renamed: their columns are no longer the export's own names.
- Axes and `[equations]` read export names. The dictionary applies last, to the heading alone.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The whole dictionary applies, or none of it does

- A source column that no plot of the point prints invalidates the dictionary for that point.
- A destination name already carried by the table does the same.
- Every column then retains the export's name, with a warning and `products.json` entries under `polars/<file>#names` and `probes/<point>#names`.

Takeaway *A failed rename never becomes a column of `NA`.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Names that are occupied or refused

- `CL` is taken. The native export's last-step `CL`, `CDi`, `CDo`, `Cx` and the rest already appear in the polar's setup content as a health check.
- A plotted column cannot take one of those names. `CL_TOTAL` is available instead.
- Two dictionary entries giving one name, or a name that is not one word, are refused when the pproc is read.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 10

Rotor coefficients

Rotor coefficients describe one rotor

- `J`, `CT`, `CQ`, `CP`, `ETA`, `ETAW`: one table per rotor, every column suffixed with that rotor's alias.
- They are meaningful for one rotor. Summing several rotors mixes different diameters and speeds in the normalisation.

$$J = \frac{V}{|n|\,D}, \quad C_T = \frac{T}{\rho n^2 D^4}, \quad C_Q = \frac{Q}{\rho n^2 D^5}, \quad C_P = 2\pi\, C_Q \operatorname{sign}(n), \quad \eta = \frac{J\, C_T}{C_P}$$

- n is the signed speed in revolutions per **second**: its magnitude sets J, its sign multiplies C_P. These are the classical airscrew coefficients [6, 7]. [REF-36], [REF-42]

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Tip and helical Mach numbers

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

- Stated for each rotor an `unsteady_rotor` or `qsteady_rotor` row turns, for the rotor of a `steady` row that states `RPM`, and for each actuator disc a row names. R is half the rotor's diameter, or the disc's tip radius; V and a are the point's own resolved free stream and speed of sound. At $V = 0$, $M_{\mathrm{hel}} = M_{\mathrm{tip}}$.
- The rotor table includes `MTIP_<alias>` and `MHEL_<alias>`, plus the in-plane coefficients `CN`, `CS`, `CMN` and `CMS` for each alias; `plan.json` and the run record carry both under `rotor_mach`. A point whose condition does not resolve reads `NA` in both and keeps its row.
- Both are geometric: V is the free stream alone, without the rotor's induced velocity, which near hover adds to the speed the tip sees. At $M_{\mathrm{hel}} \geq 1$ the tip is sonic, beyond the linearised compressibility of a panel method, and the plan warns once, naming every such point, its rotor and its M_{hel} [1, 10]. [REF-35]

Copyright (c) 2026 Geovana Neves, CC BY 4.0

An unsteady rotor row uses the shared history window

- A steady point's row comes from the loads export.
- An unsteady point's row averages the plots history over the matrix row's window, shared with the unsteady POLAR and reductions.
- The loads export of an unsteady run is the last time step, one instant of a cycle.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The rotor history is a global group over its own families

- Use the rotor's six components `FX, FY, FZ, MX, MY, MZ` in global `MRP`, over exactly its general and blade families.
- Find them through a `[[plots.groups]]` entry with `frame = "MRP"` and those exact families, whatever its name.
- If the pproc plots none, the run adds `ROTOR_<ALIAS>`.
- A group in the rotor's own frame cannot supply this history: its forces turn with the rotor and its moments are already about the hub.
- A group that only shares the alias's name but covers other families is not a source either.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

A windowed rotor row never substitutes an instant

- A point whose history does not cover the window, or lacks the required columns, is left out and named in `products.json`.
- The unsteady POLAR applies the same exclusion rule.
- The rotor table's manifest entry states `source` and `window`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

ETAW: the full force vector in wind axes

1. Take the full rotor-frame force vector, $[F_x, F_y, F_z]_{\text{rotor}}$.
2. Carry it to the airframe body frame with the **transpose** of the rotor-to-body rotation.
3. Carry it to wind axes with the **AIAA rotation**, using both alpha and beta.
4. Take its X component, `Fx_W`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

ETAW: a dimensionless efficiency

$$\mathrm{CTW} = \frac{F_{x,W}}{\rho n^2 D^4}, \qquad \mathrm{ETAW} = \frac{J\,\mathrm{CTW}}{\mathrm{CP}}.$$

- Wind-axis force replaces thrust, normalised exactly as thrust is. `CTW` enters where `CT` enters the efficiency.
- `ETAW` reduces to `ETA` when the shaft lies along the stream.
- No wind-axis force supplied means `NA`.
- Two vector rotations preserve the off-axis force components. The cosine of a scalar angle does not.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

A static point has only a real zero advance ratio

- Every rotor coefficient is `NA` at a static point except `J`, which is `0.00000`.
- The export gives coefficients normalised by the run's dynamic pressure. At $V = 0$, that pressure is zero: the real thrust has already been divided away before the package sees it.
- No rotor coefficient can be recovered from that static export.
- A hover figure of merit cannot be offered either: it needs a force the run does not state.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 11

The quasi-steady rotor

A rotor solved steady, and what its products are

- A `qsteady_rotor` row holds the blades still and turns the free stream about the shaft at the rotor's speed; every solve is steady. It is valid only for an **isolated, axisymmetric** rotor.
- **Sector** (the row states periodic symmetry): one blade and its copies, one steady solve. **Wheel**: every blade meshed; in an axial, uniform inflow it is solved once.
- A wheel at incidence, or in a custom inflow, sees a different flow at each blade position. It is solved once per position and the positions are averaged: `PASSAGE_POSITIONS: k`, mandatory, $k \geq 1$, at $\theta_i = i\,(360^\circ/N)/k$, $i = 0 \ldots k-1$, uniform over one blade passage.
- A wheel point keeps steady products at clocking zero. Its loads at each position (`<point>_qs<i>.txt`) supply the clocking mean; sections retain all clockings. It has no unsteady time window.

Takeaway *In the single-geometry study RPT-089, two positions settled thrust and torque; in-plane forces and moments needed six or more. Check clocking convergence for your own geometry.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The validity parameter, carried by every product

- Each blade station's 1P reduced frequency,

$$k = \frac{\Omega\, c}{2\, V_{\mathrm{rel}}},$$

  compares the time the flow takes to cross the chord with the period of the once-per-revolution change. Below about 0.05 a quasi-steady answer holds; above 0.1 the lag of the circulation and the shed wake matter [10, 11, 12]. [REF-34], [REF-35], [REF-38]
- `plan` shows, per wheel point, the per cent of the span with $k > 0.1$ and k minimum, maximum and mean, and warns when that per cent is above zero.

Where it is written

- The datapoint folder: k minimum, maximum and mean; the span fractions above 0.05 and 0.1; the shares of thrust and torque from stations above 0.1.
- Every post product of the point carries them.
- The sections product carries k at each station.

With `--inflow-fft` a custom inflow's harmonics count too: $k_{\mathrm{eff}} = n_{95}\, k$, nP counted by one blade per revolution (Guide 2).

Part 12

What the pproc asks the solver for

Force plots: [plots] parameters

In the pproc	The solver's PARAMETER	Units the package requests	Plots column
CL	CL	COEFFICIENTS	CL_<group>
CDI	CDI	COEFFICIENTS	CDI_<group>
CDO	CDO	COEFFICIENTS	CDO_<group>
CD	CD	COEFFICIENTS	CD_<group>
CDV/CDP	CDV/CDP (26.125)	COEFFICIENTS	CDV/CDP_<group>
FX	FORCE_X	NEWTONS	FX_<group>
FY	FORCE_Y	NEWTONS	FY_<group>
FZ	FORCE_Z	NEWTONS	FZ_<group>
MX	MOMENT_X	NEWTONS	MX_<group>
MY	MOMENT_Y	NEWTONS	MY_<group>
MZ	MOMENT_Z	NEWTONS	MZ_<group>

- Write the short name. The pproc refuses FORCE_X. Omitted parameters selects the supported defaults. There is no CY, CM or CN plot.
- The package fixes the units shown above. Coefficient columns are rescaled by $(\mathrm{VREF}/\mathrm{VINF})^2$.
- Global MRP loads FX .. MZ feed the unsteady polar and rotor table. With MRP available, the run adds MRP_TOTAL if no MRP group plots all six, and ROTOR_<ALIAS> if none plots all six over that rotor's families.

Sources under src/pyflightstream/: cases/__init__.py, FORCE_PLOT_PARAMETERS, PlotsSpec; commands/unsteady_solver.yaml, UNSTEADY_SOLVER_NEW_FORCE_PLOT; cases/workflows.py, _pproc_plots, _plot_each_rotors_own_history; post/products.py, write_plots_table.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Probes: [[probes]] parameters

The pproc and the solver use the same spelling:

```
CP_FREE   CP_REF   MACH   VELOCITY
VX        VY       VZ     STATIC_PRESSURE_RATIO
```

- On the unsteady fluid-plot path, each parameter becomes one plot per vertex: <PARAMETER><n>, with n counting across entries. The probes table un-pivots these columns to one row per probe and step.
- On a steady run, EXPORT_PROBE_POINTS takes no variable list. Its columns are the solver's own.
- An entry with empty parameters emits nothing on either run type.

These six boundary-layer parameters are accepted on builds 26.122, 26.123 and 26.124. An unsupported parameter is refused by name and build:

```
BL_MOMENTUM_THICKNESS   BL_DISPLACEMENT_THICKNESS
BL_TOTAL_THICKNESS      BL_SHAPE_FACTOR
BL_SKIN_FRICTION        BL_TRANSITION_MARKER
```

Sources under src/pyflightstream/: cases/__init__.py, FLUID_PLOT_PARAMETERS, ProbesSpec; commands/unsteady_solver.yaml, UNSTEADY_SOLVER_NEW_FLUID_PLOT; commands/probe_points.yaml, EXPORT_PROBE_POINTS; cases/workflows.py, _emit_one_probe_table; post/products.py, write_unsteady_probes_table.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The probe source follows the run type

- On an unsteady row, `kind = "instantaneous"` uses fluid plots for drawn lines and `points_file` profiles. On 26.124 and 26.125, `kind = "default"` samples probe points once after the march; a row cannot mix the two kinds.
- Each vertex and selected parameter shares one plot counter. `probes/<point>_probes.csv` holds both forms together, one row per probe and time step, from the plots history.
- A cited profile is read at plan: a count, then `X,Y,Z,TYPE` CSV rows. Types 0 and 1 become fixed fluid-plot vertices with the entry's frame and scale. Invalid profiles are refused before solving.
- The normal-probe route retains its probe-points export. Fluid-plot probes retain the history export. Requested solver plot exports also include their selected residual, load and section-Cp kinds.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Steady probes and older unsteady records

- Steady rows keep the probe-points source for both drawn and cited probes. A nonempty `parameters` list enables the entry; an empty list disables it.
- On a steady row the list does not filter exported variables: the solver exports a fixed set. Plan warns with the entry number and frame.
- On an unsteady row the list selects the sampled fluid-plot variables.
- An older cited profile exported only as an instant has no history. Re-post skips it with its reason in `products.json`, preserving the drawn-line histories that were recorded. A new run supplies the missing history.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

What a point exports: [exports]

Key	File suffix	Solver command	Run type
simulation	.fsm	SAVEAS	Both
loads	.txt	EXPORT_SOLVER_ANALYSIS_SPREADSHEET	Both
tecplot	.dat, .vtk	EXPORT_SOLVER_ANALYSIS_VTK; the package writes the .dat	Both
sections	_cp.txt	EXPORT_ALL_SURFACE_SECTIONS	Both
sectional_loads	_sloads.txt	EXPORT_SURFACE_SECTIONAL_LOADS	Both
probes	_probes.txt	EXPORT_PROBE_POINTS	Steady only
plots	_plots.txt	UNSTEADY_SOLVER_EXPORT_PLOTS	Unsteady only
log	_log.txt	EXPORT_LOG	Both

- Keys are true or false. The original kinds default on for their run type. vtk, csv and force_distributions default off; the steady plot kinds default on (next frame).

Copyright (c) 2026 Geovana Neves, CC BY 4.0

What a point exports: [exports] (continued)

- loads is required to judge the run, and simulation is required too: every point leaves its final .fsm. Either one stated false is refused.
- tecplot exports every surface as VTK and the package writes the .dat from it, per panel, in the reference frame.
- singularity_strength = true, a top-level pproc key, off by default, adds the nodal Singularity_strength of a native auxiliary *_native_tecplot.dat, matched to the same step's VTK by geometry [4]. Off, the point makes no native export, its .dat states the strength NOT_CARRIED (so does its products.json entry), and the point is complete without it. plan says per row whether the strength is carried.

Sources under src/pyflightstream/: cases/__init__.py, EXPORT_KINDS, default_outputs, _known_export_kinds; commands/solver_export.yaml, EXPORT_SOLVER_ANALYSIS_VTK; results/surface.py.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Additional exports: plots, a volume section, force distributions

- **The solver's own plots.** `plot_residuals`, `plot_loads` (on by default) and `plot_sections_cp` (on where sections are declared) under `[exports]`: the plotted series as TEXT, one row per iteration or one x/Cp pair per section. **Never a coefficient source**: on 26.124 the plotted pitching moment is -0.897 where the exported `CMy` is -0.0247 [3]. An unsteady row saves the first two as well, once, after the march, and the section Cp too, where sections are declared. `false` turns each off.
- **A volume section.** `[volume_section]`: one plane, a rectangle (`corners_m`) or an annulus (`radii_m`), in a `plane` of a `frame`, at `offset_m`. It is sampled, not exported by the solver: probes on a steady row, fluid plots on an unsteady or rotor row, written as a vertex cloud to `post/<matrix>/fields/<point>_vsec.vtk` or `.dat` (`_step_<STEP>` per unsteady STEP). Older records keep their native export in the datapoint.
- **The force distribution.** `force_distributions = true`: per-panel pressure and viscous force coefficients, once, at the end of the run. Off by default.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

VTK and CSV surface exports

`vtk_variables` is a top-level pproc key, before any tables:

```
vtk_variables = ["X", "Y", "Z", "CP_FREESTREAM"]
[exports]
vtk = true
csv = true
```

- Both kinds default off. VTK uses `EXPORT_SOLVER_ANALYSIS_VTK` and `SET_VTK_EXPORT_VARIABLES` when the list is present. An absent list requests all variables.
- CSV uses `EXPORT_SOLVER_ANALYSIS_CSV`: `CP-FREESTREAM`, `PASCALS`, all surfaces, solver reference frame. Both exclude the wake.
- Commands and variables are checked against the selected build's database. Unsupported requests are refused at plan, naming the build.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The surface averaging window: the package averages

```
[time_averaging]
last_revs = 1.5 # OR last_iters = 54; exactly one, positive
```

- The package averages; the solver does not. The run exports the surface at every step of the window, as `EXPORT_UNSTEADY_AFTER_ITER: <first step>` would, and the post averages those exports into `surfaces/<point>_time_average.dat`: the same panel across the steps, every step the same weight, every variable. `SOLVER_TIME_AVERAGING` is never emitted: it hung 26.124, killed at 240.5 seconds with nothing written [3], and 26.123 does not carry it.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The surface window: time steps, verified on 26.122

- Bounds are inclusive, 1-based time steps ending at the run's last step. `last_iters` counts steps; `last_revs` uses the same rotor clock and rounding as `LAST_REVS_AVG`. A long window clips at step 1.
- Verified on 26.122 [3]: the solver's own average over steps 7 to 12 is the uniform mean of the run's per-step surfaces over those steps, to 1e-13 in Cp, Vx and speed, which is the package's mean. The solver keeps CF at its last instant; the package averages it.
- Steps that do not share one topology refuse the average; a step of the window that was not exported (a clock-stopped run, a continuation) skips it. Both by name, never a partial average.
- Needs 26.122 or later and `tecplot` on. The export header's inner-iteration count is not this clock.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Surface export provenance and per-step exports

- `products.json` states the averaged surface `kind: average`, with the recorded window, the steps averaged and each per-step VTK read with its sha256; the per-step exports stay `kind: instant`.
- The window is read from the run record (`surface_average_window`), never from a pproc edited since; changing it requires a new run.
- Every `.dat` states its source in `DATASETAUXDATA`: the VTK, its sha256 and a `TRANSLATION` record (cell-centred, reference frame).
- `EXPORT_UNSTEADY_AFTER_REV` or `EXPORT_UNSTEADY_AFTER_ITER` exports the VTK per step, CSV where asked; each step's `.dat` is written from its VTK.
- An older record keeps the solver's window and its meaning. The matrix's plots-reduction window remains a separate choice.
- A storage recipe's `[[prune_step_exports]]` defaults to keeping the last step. `keep_last` and `delete_steps` select other retention policies; a post needing a deleted step refuses, naming it (Guide 2).

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Surface formats without a pproc export key

- `EXPORT_SOLVER_ANALYSIS_PLOAD_BDF` still has no `[exports]` key. `EXPORT_SOLVER_ANALYSIS_FORCE_DISTRIBUTIONS` has one, `force_distributions`.
- Its parameter or payload block cannot be supplied by a setup's one-line `[[raw]]` entry.
- `EXPORT_BL_VELOCITY_PROFILE` holds an unattended script [3]: a raw line is refused at plan on 26.124.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 13

The additional post

A second pproc, extracted from the saved simulation

```
ADDITIONAL_PPROC: p002        # VAR_NAMES_VALUES of the row
pyfs-matrix post matriz.fs --workspace . --additional-pproc
```

- The row runs byte for byte as without the key; the run record never carries it.
- Afterwards, each recorded point whose final `.fsm` is on disk and hashes as its record says is copied into `datapoints/DP-<point>/additional/<pid>/` and reopened there, **with no solve**: sections cut in the frames the run created, sectional loads computed, exports written, file closed. Nothing is saved and no probe is written.
- Each extraction is recorded in `additional.json` beside `runs.json`, which is never written. The original `.fsm` is hashed again after the launch.

Source: `docs/post-processing-definitions.md` [1], **The additional post**.

What a reopened saved simulation gives back

Export	Reopened, with no solve (26.124) [3]
total loads	identical
surface solution	identical bytes
surface sections, a distribution created after reopening included	identical
sectional loads	zero until computed, identical once computed: the file stores the sections, not their loads, so the extraction computes them every time
plots history of an unsteady point	identical
probe points off the body	**not identical**: up to 4 per cent in speed, 0.094 in Cp

- Only 26.124 was measured: a row on another build is refused at plan.
- Probes, a volume section, `[plots]`, `[time_averaging]` and `base_regions` in an additional pproc are refused: nothing measured says a reopened file gives them back.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The products of an extraction

- Under `post/<matrix>/additional/<pid>/`, beside the run's own and never over them: the group polars of a steady point, one sections table per point, and on an unsteady point the plots tables and reductions.
- Every `products.json` entry for them carries:

Key	Meaning
`pproc`	the ADDITIONAL pproc id, not the one the row ran with
`additional`	always `true`; a reader of run products filters on it
`extraction`	the extractions the file holds, `<point run id>/additional/<pid>`
`derives_from`	the points those extractions were taken from

and no `runs`, which names run ids everywhere else in the index.

Which rows, which instant, and when it stops counting

- **Rows.** A reopened sections export carries the run's distributions first and the additional pproc's after them. The table keeps every row, the new blocks numbered on after the run's and marked with the pproc; `leading_sections` counts the run's rows.
- **One instant on an unsteady point.** The saved simulation is the LAST instant: the sections table's `STEP` is the run's last time step, its entry says `kind: instant`, and `post.log` says so. The plots history is the run's own.
- **Only a current extraction has products**: its point is admitted, the `.fsm` still hashes as the one it opened, and every file it wrote is on disk and hashes as recorded. A point run again archives the extraction; the stale one is skipped under `additional/<pid>/runs/<extraction id>`, never under the run's key.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 14

What the package does not judge

Last-step convergence does not judge a settled history

- For an unsteady point, the package asserts whether the numerical iterations **within the last time step** converged.
- It does not judge whether the time history has settled.
- There is no settle tolerance or convergence criterion over the history, and no point fails for one.

Takeaway *The user judges whether the history has settled afterwards, from the history.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Frozen solves and averages

- Two consecutive frozen steps make the point `FAILED_DIVERGED`. In a frozen step, inner iterations after the first print exactly zero velocity residual, and the last prints both residuals exactly zero.
- Assessment names the first frozen step and count. Post reads the native log again, including for a record previously assessed successful.
- An average whose window reaches a frozen step is written with a warning by default. `--check-frozen` refuses the affected average instead; the post log explains the decision.
- Windows wholly before the freeze, raw histories and explicitly instant products remain available. A frozen stretch requires a new solve.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The post never blocks and always writes its log

- Every `pyfs-matrix post` writes `post.log` beside `products.json`, even when nothing is wrong; the manifest names it. Read it after every post: each named skip and warning states the point, the product, the step and what settles it.
- By default a frozen point, an unread residual block, a failed status or a reference mismatch is a WARNING line, and every computable product is written. `--check-frozen` refuses the affected averages instead.
- The freeze and azimuthal guards judge the exact set of plotted samples the reducer reads, blade aliases resolved and every nonzero interpolation weight counted; the guard does no arithmetic of its own.
- `integrate = true` on a `[[sections.distributions]]` pproc entry appends `Strip_length`, `Fx_int`, `Fz_int` and `My_int` to the same sloads file: strips midpoint to midpoint, half strips at the ends, the moment about the simulation's moment reference point (not each station's quarter chord). Off by default; the file is byte-identical when off.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

One post, one log, and the log as fields

- `post.log.json` sits beside `post.log`: the same header (`version`, `workspace`, `matrix`, `time`, `check_frozen`) and one record per WARNING line, in the same order, with `point`, `product`, `message` and `remedy`. `products.json` names it under `log_json`.
- Both are written from one list, on a clean, a failed and an interrupted post, so they cannot disagree. Parse the JSON, not the text.
- A WARNING line names its own point and product: `WARNING point=camp/sim_7001/AL-020 product=available-exports: ...`.
- Each post collects the package's warnings in its own thread: two posts in one process no longer share a log, and a warning from code outside the package is not written to it [3].

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 15

The token for a value that does not exist

Every product writes `NA` for no value

- Zero is a real value. Advance ratio at rest is exactly zero, so zero cannot stand for a value that does not exist.
- A blank cannot distinguish missing data, an inapplicable column or a writer that stopped halfway.
- The token is defined once in `pyflightstream.post.products.NOT_APPLICABLE`, the public import.
- Readers still accept `-`, the token written in older files.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The physics behind the definitions, and where to read it

- **Body, stability and wind axes**, and the angles that turn one into the other, are those of flight mechanics [5].
- **The rotor coefficients** J, C_T, C_Q, C_P and η are the classical airscrew coefficients [6, 7].
- **A window average** is the time mean of a record over a finite span; its error falls as the span covers more periods of the signal [8].
- **The azimuthal (phase-locked) mean** averages the samples that share one phase of the revolution, separating the periodic part of a signal from the rest [9].
- **Tip and helical Mach numbers**, and the reduced frequency that says when a rotor's flow is quasi-steady, are rotor aerodynamics [10, 11, 12].

Copyright (c) 2026 Geovana Neves, CC BY 4.0

References

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

[2] Altair Engineering, *FlightStream User Manual*, version 26.12, 2026.

[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 "Surface exports and native singularity strength". `https://nevesgeovana.github.io/pyflightstream/latest/surface-translation/`

REF-43 B. Etkin, L. D. Reid, *Dynamics of Flight: Stability and Control*, 3rd ed., Wiley, New York, 1996.

REF-42 H. Glauert, *The Elements of Aerofoil and Airscrew Theory*, 2nd ed., Cambridge University Press, Cambridge, 1947.

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

REF-50 J. S. Bendat, A. G. Piersol, *Random Data: Analysis and Measurement Procedures*, 4th ed., Wiley, Hoboken, 2010.

REF-51 W. C. Reynolds, A. K. M. F. Hussain, "The mechanics of an organized wave in turbulent shear flow. Part 3. Theoretical models and comparisons with experiments", *Journal of Fluid Mechanics*, 54(2), pp. 263-288, 1972.

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

Copyright (c) 2026 Geovana Neves, CC BY 4.0

References (continued)

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

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.

Copyright (c) 2026 Geovana Neves, CC BY 4.0