lst-tools

Linear Stability Theory Utilities

Christoph Hader

Table of contents

1. lst-tools

`lst-tools` is a Python toolkit for pre- and post-processing of Linear Stability Theory (LST) analyses of high-speed boundary layers.

1.1 Features

- **CLI**: Streamlined workflow from the terminal
- **API**: Python interface for scripting and automation
- **HPC**: Job script generation for cluster environments
- **Post-processing**: Ridge-line maxima extraction and 3-D tracking volume assembly

1.2 Quick Start

Download PDF Documentation

1.3 PDF Downloads

- Latest PDF (stable URL): https://uahypersonics.github.io/lst-tools/pdf/lst-tools-documentation.pdf
- Versioned PDF (tag builds): https://uahypersonics.github.io/lst-tools/pdf/lst-tools-vX.Y.Z.pdf

For example, tag `v0.1.3` publishes: `https://uahypersonics.github.io/lst-tools/pdf/lst-tools-v0.1.3.pdf`

Install the package (see Installation for detailed instructions):

```
pip install lst-tools
```

```
# initialize a config file
lst-tools init

# convert base flow to LASTRAC format
lst-tools lastrac

# set up and run parsing sweep
lst-tools setup parsing --auto-fill

# set up and run tracking
lst-tools setup tracking --auto-fill

# post-process tracking results (maxima + volume)
lst-tools process tracking --interpolate

# optional: set up and process spectra
lst-tools setup spectra
lst-tools process spectra

# optional: clean generated artifacts
lst-tools clean tracking --force
```

`lst-tools` can also be used as a Python library. See the API Reference for details.

1.4 License

BSD-3-Clause. See LICENSE for details.

2. Installation

2.1 Requirements

- Python 3.10 or later

2.2 From PyPI

```
pip install lst-tools
```

2.3 From Source

Clone the repository and install in editable mode:

```
git clone https://github.com/uahypersonics/lst-tools.git
cd lst-tools
pip install -e .
```

2.4 Development Installation

For development, install with the `dev` extras:

```
pip install -e ".[dev]"
```

This includes:

- pytest and pytest-cov for testing
- ruff for linting

2.5 Verify Installation

```
import lst_tools
print(lst_tools.__version__)
```

Or from the command line:

```
lst-tools --version
```

3. User Guide

3.1 User Guide

This section covers how to use `lst_tools` in practice.

- Basic Usage: End-to-end workflow from config initialization to post-processing
- CLI: Complete command reference for setup, processing, cleaning, and utilities
- Config Files: Structure and key fields of the `lst.cfg` configuration file

3.2 Basic Usage

This page walks through a practical end-to-end workflow.

3.2.1 1. Initialize Configuration

Create a baseline config file:

```
lst-tools init
```

Common variants:

```
# pre-populate geometry fields
lst-tools init --geometry cone

# seed from flow_conditions.dat if available
lst-tools init --flow flow_conditions.dat

```

3.2.2 2. Convert Meanflow for LASTRAC

Convert the HDF5 base flow in `input_file` to `meanflow.bin`:

```
lst-tools lastrac
```

If you want extra diagnostics:

```
lst-tools --debug lastrac
```

This also writes a Tecplot debug snapshot to `./debug/`.

3.2.3 3. Set Up Parsing

Generate the parsing input deck:

```
lst-tools setup parsing
```

If key sweep values are unset, auto-fill from `meanflow.bin`:

```
lst-tools setup parsing --auto-fill
```

3.2.4 4. Run the External LST Solver

`lst-tools` prepares input decks and scripts; solver execution is external.

- For a single parsing deck, run your local `lst.x` process as needed.
- For cluster usage, generate a scheduler script with:

```
lst-tools hpc
```

3.2.5 5. Set Up Tracking and Process Results

Set up tracking cases:

```
lst-tools setup tracking --auto-fill
```

After running tracking cases, process results:

```
# maxima + volume
lst-tools process tracking

# enable sub-grid peak interpolation
lst-tools process tracking --interpolate

```

Target selected case directories only:

```
lst-tools process tracking --dir kc_10pt00 --dir kc_20pt00
```

3.2.6 6. Set Up and Process Spectra

Set up spectra cases:

```
lst-tools setup spectra
```

After running spectra jobs, process output:

```
lst-tools process spectra
```

3.2.7 7. Clean Generated Artifacts

```
lst-tools clean parsing --force
lst-tools clean tracking --force
lst-tools clean spectra --force
```

3.2.8 8. Inspect Meanflow Quickly

```
lst-tools info meanflow.bin
```

This prints summary metadata, coordinate range, and reference quantities.

3.3 CLI

`lst-tools` provides the `lst-tools` command-line interface for the full LST workflow.

3.3.1 General Usage

```
lst-tools [options] <subcommand> [<args>]
```

Global options:

```
lst-tools --version
lst-tools --verbose
lst-tools --debug
```

Show version:

```
lst-tools --version
```

Get help for any subcommand:

```
lst-tools <subcommand> --help
```

3.3.2 Subcommands

`init`

Generate a default configuration file:

```
# create default lst.cfg
lst-tools init

# pre-populate geometry defaults
lst-tools init --geometry cone

# specify output path
lst-tools init --out myconfig.cfg

# merge with flow_conditions.dat
lst-tools init --flow flow_conditions.dat

# overwrite existing config
lst-tools init --force

```

`lastrac`

Convert the HDF5 base flow to LASTRAC format:

```
lst-tools lastrac

# explicit config path
lst-tools lastrac --cfg myconfig.cfg

```

`setup parsing`

Generate input decks for the LST parsing (initial frequency/wavenumber sweep):

```
lst-tools setup parsing

# auto-fill parameters (x_s, x_e, i_step, f_min, f_max, d_f, etc.) from the
meanflow
lst-tools setup parsing --auto-fill

# auto-fill and overwrite existing values
lst-tools setup parsing --auto-fill --force

# write input deck into another directory/name
lst-tools setup parsing --out runs --name lst_input.dat

```

`setup tracking`

Set up tracking calculations (directory structure and input decks):

```
lst-tools setup tracking

# auto-fill parameters from the meanflow
lst-tools setup tracking --auto-fill

# auto-fill and overwrite existing values
lst-tools setup tracking --auto-fill --force

# set a fixed initialization frequency
lst-tools setup tracking --finit 120000.0


```

`setup spectra`

Generate input decks for spectral analysis at multiple streamwise locations:

```
lst-tools setup spectra

# explicit config path
lst-tools setup spectra --cfg myconfig.cfg

```

`process tracking`

Post-process tracking calculation results (ridge maxima + optional 3-D volume):

```
lst-tools process tracking

# only maxima extraction
lst-tools process tracking --maxima

# only volume assembly
lst-tools process tracking --volume

# process selected kc directories (repeat --dir)
lst-tools process tracking --dir kc_10pt00 --dir kc_20pt00

# enable/disable sub-grid interpolation explicitly
lst-tools process tracking --interpolate
lst-tools process tracking --no-interpolate

```

`process spectra`

Post-process spectra calculation results:

```
lst-tools process spectra
```

`visualize parsing`

Render parsing contours via the `cfd-viz` LST renderer:

```
# use default parsing file and defaults
lst-tools visualize parsing

# explicit input/output paths
lst-tools visualize parsing \
    --input growth_rate_with_nfact_amps.dat \
    --out alpi_contours_parsing

# render one selected k-plane only
lst-tools visualize parsing --single-k --k-index 1

# choose contour policy and level count
lst-tools visualize parsing \
    --levels-policy positive-rounded \
    --levels-count 60

```

`visualize tracking`

Render tracking volume contours via the same `cfd-viz` backend:

```
# use default tracking volume file and defaults
lst-tools visualize tracking

# explicit input/output paths
lst-tools visualize tracking \
    --input lst_vol.dat \
    --out viz_tracking

# custom field and variable mapping
lst-tools visualize tracking \
    --field "-im(alpha)" \
    --xvar s \
    --yvar "freq,freq." \
    --kvar beta


```

Tracking fallback behavior:

1. If `lst_vol.dat` exists, `lst-tools visualize tracking` renders directly from it.
2. If `lst_vol.dat` is missing, it discovers `kc_*` directories and reads `growth_rate_with_nfact_amps.dat` from each completed case.
3. In fallback mode, output PNGs are written to `alpi_contours_tracking/` in the current working directory with a shared contour scale across all discovered cases.

> **Note**
>
> ```
> The `visualize` wrappers call `cfd-viz` internally. Install `cfd-viz`
> in the same Python environment as `lst-tools`.
> ```

`hpc`

Generate run scripts for HPC systems:

```
lst-tools hpc

# explicit config path
lst-tools hpc --cfg myconfig.cfg
```

`info`

Inspect a LASTRAC meanflow binary:

```
lst-tools info meanflow.bin
```

`clean parsing`

Remove parsing-generated artifacts from one directory:

```
lst-tools clean parsing --dir .
lst-tools clean parsing --dir . --force
```

`clean tracking`

Remove solver artifacts from tracking case directories:

```
# clean all kc_* directories in current directory
lst-tools clean tracking --force

# clean selected directories only
lst-tools clean tracking --dir kc_10pt00 --dir kc_20pt00 --force

```

`clean spectra`

Remove spectra setup outputs:

```
lst-tools clean spectra --dir . --force
```

3.3.3 Typical Workflow

```
# 1. initialize config
lst-tools init --geometry cone

# 2. convert base flow to LASTRAC format
lst-tools lastrac

# 3. run parsing sweep
lst-tools setup parsing --auto-fill

# 4. set up and run tracking
lst-tools setup tracking --auto-fill

# 5. post-process tracking results
lst-tools process tracking --interpolate

# 6. set up and run spectra
lst-tools setup spectra

# 7. post-process spectra results
lst-tools process spectra


```

3.4 Config Files

`lst_tools` uses a TOML-style configuration file (`lst.cfg`) to control the LST workflow.

3.4.1 Creating a Config File

Generate a default configuration:

```
lst-tools init
```

Geometry presets are available:

```
lst-tools init --geometry cone
```

3.4.2 Configuration Sections

Top-Level

Key	Type	Default	Description
`input_file`	`str`	`"base_flow.hdf5"`	Path to the HDF5 base flow file
`lst_exe`	`str`	`"lst.x"`	Path to the LST solver executable

`[flow_conditions]`

Flow conditions for the LST analysis:

Key	Type	Default	Description
`mach`	`float`	—	Freestream Mach number
`re1`	`float`	—	Unit Reynolds number [1/m]
`gamma`	`float`	`1.4`	Ratio of specific heats
`cp`	`float`	`1005.025`	Specific heat at constant pressure [J/(kg K)]
`cv`	`float`	`717.875`	Specific heat at constant volume [J/(kg K)]
`rgas`	`float`	`287.15`	Specific gas constant [J/(kg K)]
`pres_0`	`float`	—	Stagnation pressure [Pa]
`temp_0`	`float`	—	Stagnation temperature [K]
`pres_inf`	`float`	—	Freestream pressure [Pa]
`temp_inf`	`float`	—	Freestream temperature [K]
`dens_inf`	`float`	—	Freestream density [kg/m^3]
`uvel_inf`	`float`	—	Freestream velocity [m/s]
`visc_law`	`int`	`0`	Viscosity law index

`[geometry]`

Geometry description:

Key	Type	Default	Description
`type`	`str`	—	Geometry kind (e.g., cone, flat plate)
`theta_deg`	`float`	—	Half-angle [deg]
`r_nose`	`float`	—	Nose radius [m]
`l_ref`	`float`	`1`	Reference length [m]
`is_body_fitted`	`bool`	`false`	Whether the grid is body-fitted

`[meanflow_conversion]`

Controls for base flow conversion:

Key	Type	Default	Description
`i_s`	`int`	`0`	Start index
`i_e`	`int`	—	End index
`d_i`	`int`	`1`	Index stride
`set_v_zero`	`bool`	`true`	Zero out wall-normal velocity

`[lst.solver]`

LST solver settings:

Key	Type	Default	Description
`type`	`int`	`1`	Solver type
`is_simplified`	`bool`	`false`	Use simplified equations
`alpha_i_threshold`	`float`	`-100.0`	Threshold used in alpha_i filtering
`generalized`	`int`	`0`	Generalized eigenproblem switch
`spatial_temporal`	`int`	`1`	Spatial (1) or temporal (0) analysis
`energy_formulation`	`int`	`1`	Energy equation formulation

`[lst.options]`

Additional solver options:

Key	Type	Default	Description
`geometry_switch`	`int`	—	Geometry mode switch
`longitudinal_curvature`	`int`	`0`	Enable longitudinal curvature terms

`[lst.params]`

LST calculation parameters:

Key	Type	Default	Description
`ny`	`int`	`150`	Number of wall-normal grid points
`yl_in`	`float`	`0.0`	Initial wall-normal location
`tol_lst`	`float`	`1e-5`	Convergence tolerance
`max_iter`	`int`	`15`	Maximum iterations
`x_s`	`float`	—	Start streamwise location
`x_e`	`float`	—	End streamwise location
`i_step`	`int`	—	Streamwise index stride
`tracking_dir`	`int`	`1`	Tracking direction flag
`f_min`	`float`	—	Minimum frequency [Hz]
`f_max`	`float`	—	Maximum frequency [Hz]
`d_f`	`float`	—	Frequency step [Hz]
`f_init`	`float`	`0.0`	Tracking initialization frequency [Hz]
`beta_s`	`float`	—	Start spanwise wavenumber
`beta_e`	`float`	—	End spanwise wavenumber
`d_beta`	`float`	—	Spanwise wavenumber step
`beta_init`	`float`	`0.0`	Tracking initialization spanwise wavenumber
`alpha_0`	`complex`	`(0, 0)`	Initial complex alpha guess

`[lst.io]`

LST input/output file paths:

Key	Type	Default	Description
`baseflow_input`	`str`	`"meanflow.bin"`	Converted base flow file
`solution_output`	`str`	`"growth_rate.dat"`	LST output file

`[hpc]`

HPC job settings:

Key	Type	Default	Description
`account`	`str`	—	Account/allocation name
`nodes`	`int`	—	Number of nodes
`time`	`str`	—	Wall time
`partition`	`str`	—	Partition/queue name
`extra_env`	`table`	—	Extra environment variables injected into run scripts

`[processing]`

Tracking post-processing controls:

Key	Type	Default	Description
`interpolate`	`bool`	`false`	Enable parabolic peak interpolation
`gate_tol`	`float`	`0.10`	Ridge-tracking gating tolerance
`min_valid`	`int`	`40`	Minimum valid points for ridge acceptance
`peak_order`	`int`	`1`	Local peak search order

3.4.3 Programmatic Access

```python
from lst_tools.config import read_config, write_config

# read as typed dataclass
config = read_config("lst.cfg")

# modify values
config.lst.params.ny = 200

# write to TOML
write_config("lst_modified.cfg", overwrite=True, cfg_data=config.to_dict())

```

4. Theory

4.1 Theory

Under Construction

This page is incomplete.

5. API Reference

5.1 API Reference

Technical reference for the `lst_tools` package.

5.1.1 Modules

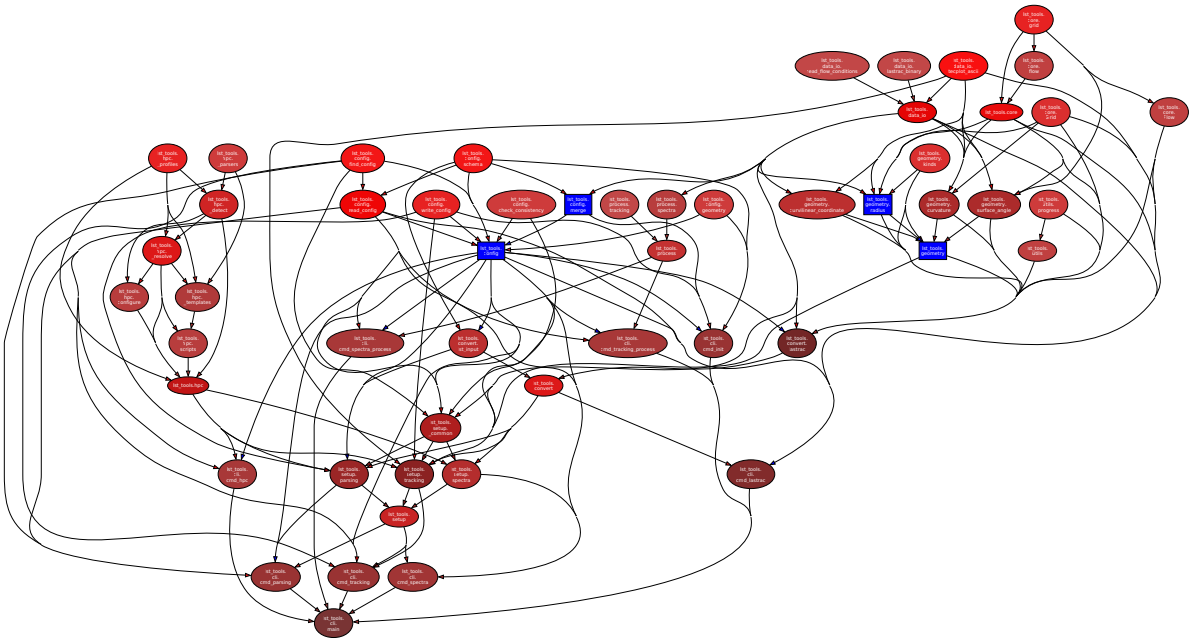
Autogenerated from source docstrings.

- Grid & Flow: Core data structures for grid and flow data
- Config: Configuration management and schema
- Convert: Format conversion (HDF5 to LASTRAC)
- Setup: LST calculation setup (parsing, tracking, spectra)
- Processing: Post-processing of LST results
- Data I/O: File readers and writers
- Geometry: Surface geometry computations
- HPC: HPC job script generation

5.2 Package Architecture

Visual overview of the `lst_tools` module structure and data flow.

5.2.1 Module Dependency Graph



Download architecture (SVG) View architecture (SVG)

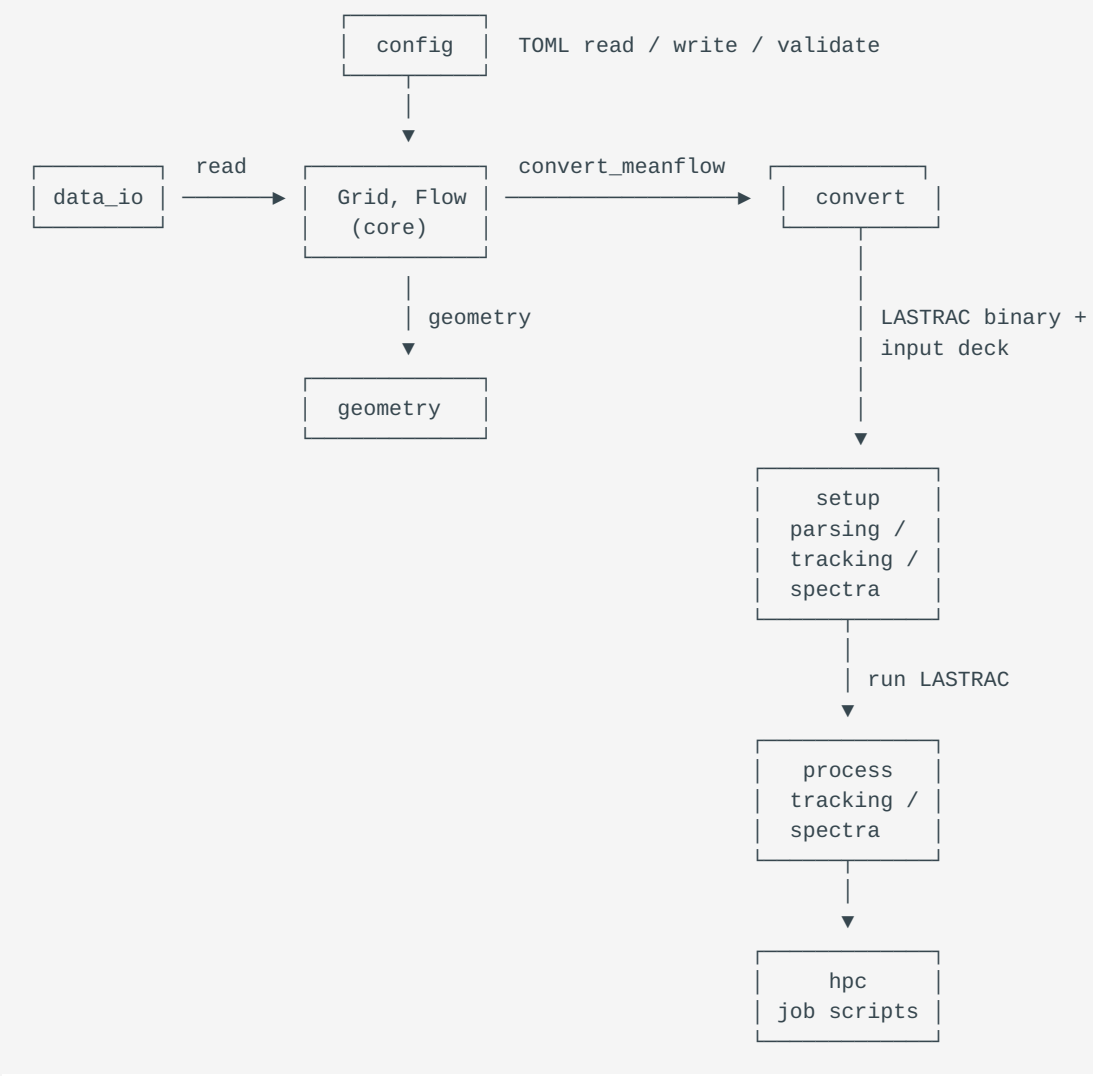
Regenerate

```
pydeps src/lst_tools --noshow --max-bacon=4 --cluster -o docs/assets/
architecture.svg
```

5.2.2 Module Overview

Subpackage	Responsibility
`cli`	Typer CLI — thin wrappers that dispatch to library functions
`config`	Read, write, validate TOML config; consistency checks
`core`	`Grid` and `Flow` dataclasses — central data containers
`convert`	HDF5 mean-flow → LASTRAC binary; input-deck generation
`data_io`	File I/O: Fortran binary, LASTRAC binary, Tecplot ASCII, flow conditions
`geometry`	Surface geometry: curvature, curvilinear coordinate, surface angle, radius
`setup`	Case setup: parsing, tracking, and spectra configuration
`process`	Post-processing: tracking and spectra results
`hpc`	HPC environment detection, job script generation
`utils`	Shared helpers (progress bars)

5.2.3 Data Flow



5.2.4 CLI Command Hierarchy

The CLI uses Typer with two sub-groups (`setup`, `process`) to mirror the LST workflow:

```
lst-tools
├── init              initialise a working directory with template config
├── lastrac           run the LASTRAC solver
├── setup
│   ├── parsing       set up LST parsing calculations
│   ├── tracking      set up LST tracking calculations
│   └── spectra       set up LST frequency-spectra calculations
├── process
│   ├── tracking      post-process tracking results
│   └── spectra       post-process spectra results
└── hpc               generate HPC job scripts

```

5.2.5 External Dependencies

Dependency	Usage
`cfd-io`	`FortranBinaryReader` / `FortranBinaryWriter` (re-exported via `data_io`)
`numpy`	Array computation throughout
`scipy`	Interpolation in geometry and setup modules
`typer`	CLI framework
`tomli` / `tomli-w`	TOML config read / write

5.3 Grid & Flow

Core data structures for representing base flow data.

5.3.1 `Grid`

```
from lst_tools import Grid
```

The `Grid` class holds the computational grid coordinates.

::: lst_tools.core.Grid

5.3.2 `Flow`

```
from lst_tools import Flow
```

The `Flow` class holds the flow field variables on the grid.

::: lst_tools.core.Flow

5.4 Config

Configuration management for `lst_tools`.

5.4.1 Functions

`read_config`

```
from lst_tools import read_config

config = read_config("lst.cfg")
```

Read a TOML configuration file and return a typed `Config` dataclass.

::: lst_tools.config.read_config

`write_config`

```
from lst_tools import write_config

write_config("lst.cfg", overwrite=True, cfg_data=config.to_dict())
```

Write configuration data to a TOML file.

::: lst_tools.config.write_config

`Config`

```
from lst_tools.config import Config

cfg = Config.from_toml("lst.cfg")
```

Typed configuration schema and validation.

::: lst_tools.config.Config

`find_config`

```
from lst_tools import find_config

path = find_config(".")
```

Search for a configuration file in the current directory and parent directories.

::: lst_tools.config.find_config

`check_consistency`

```
from lst_tools import check_consistency, format_report

report = check_consistency(config)
print(format_report(report))
```

Check configuration for internal consistency and return a diagnostic report.

::: lst_tools.config.check_consistency

::: lst_tools.config.format_report

5.5 Data I/O

File readers and writers for various scientific data formats.

5.5.1 Fortran Binary

`FortranBinaryReader`

```
from lst_tools import FortranBinaryReader

reader = FortranBinaryReader("meanflow.bin")
```

::: lst_tools.data_io.FortranBinaryReader

`FortranBinaryWriter`

```
from lst_tools import FortranBinaryWriter

writer = FortranBinaryWriter("output.bin")
```

::: lst_tools.data_io.FortranBinaryWriter

5.5.2 Flow Conditions

`read_flow_conditions`

```
from lst_tools import read_flow_conditions

fc = read_flow_conditions("flow_conditions.dat")
```

::: lst_tools.data_io.read_flow_conditions

5.5.3 Tecplot ASCII

`read_tecplot_ascii`

```
from lst_tools import read_tecplot_ascii

data = read_tecplot_ascii("profile.dat")
```

::: lst_tools.data_io.read_tecplot_ascii

5.6 Geometry

Surface geometry computations from grid data.

5.6.1 Functions

`curvature`

```
from lst_tools import curvature

kappa = curvature(grid)
```

Compute the surface curvature distribution.

::: lst_tools.geometry.curvature

`curvilinear_coordinate`

```
from lst_tools import curvilinear_coordinate

s = curvilinear_coordinate(grid)
```

Compute the curvilinear coordinate along the surface.

::: lst_tools.geometry.curvilinear_coordinate

`surface_angle`

```
from lst_tools import surface_angle

theta = surface_angle(grid)
```

Compute the surface angle distribution.

::: lst_tools.geometry.surface_angle

`radius`

```
from lst_tools import radius

r = radius(grid)
```

Compute the local body radius.

::: lst_tools.geometry.radius

5.6.2 Geometry Kinds

`GeometryKind`

```
from lst_tools import GeometryKind, list_geometry_kinds

kinds = list_geometry_kinds()
```

Enumeration of supported geometry types.

::: lst_tools.geometry.GeometryKind

::: lst_tools.geometry.list_geometry_kinds

5.7 Convert

Format conversion utilities for base flow data.

5.7.1 Functions

`convert_meanflow`

```
from lst_tools import convert_meanflow

convert_meanflow(grid, flow, config)
```

Convert an HDF5 base flow to the Fortran binary format used by the LST solver.

::: lst_tools.convert.convert_meanflow

`generate_lst_input_deck`

```
from lst_tools import generate_lst_input_deck

generate_lst_input_deck(config)
```

Generate an input deck for the LST solver.

::: lst_tools.convert.generate_lst_input_deck

5.8 Setup

LST calculation setup functions for parsing, tracking, and spectra.

5.8.1 Functions

`parsing_setup`

```
from lst_tools import parsing_setup

parsing_setup(config)
```

Set up the parsing step (initial frequency/wavenumber sweep).

::: lst_tools.setup.parsing_setup

`tracking_setup`

```
from lst_tools import tracking_setup

tracking_setup(config)
```

Set up the tracking step (spatial marching along disturbance trajectories).

::: lst_tools.setup.tracking_setup

`spectra_setup`

```
from lst_tools import spectra_setup

spectra_setup(config)
```

Set up spectra calculations at multiple streamwise locations.

::: lst_tools.setup.spectra_setup

5.9 Processing

Post-processing of LST calculation results.

5.9.1 Functions

`tracking_process`

```
from lst_tools import tracking_process

tracking_process(config)
```

Post-process tracking calculation results (N-factors, growth rates).

::: lst_tools.process.tracking_process

`spectra_process`

```
from lst_tools import spectra_process

spectra_process(config)
```

Post-process spectra calculation results.

::: lst_tools.process.spectra_process

5.10 HPC

HPC job script generation for cluster environments.

5.10.1 Functions

`hpc_configure`

```
from lst_tools import hpc_configure

hpc_cfg = hpc_configure(config, set_defaults=True)
```

Build an HPC configuration object from the project config.

::: lst_tools.hpc.hpc_configure

`script_build`

```
from lst_tools import script_build

script_build(hpc_cfg)
```

Generate job submission scripts from the HPC configuration.

::: lst_tools.hpc.script_build

`ResolvedJob`

```
from lst_tools import ResolvedJob
```

Resolved scheduler-specific job settings used by script generation.

::: lst_tools.hpc.ResolvedJob

`detect`

```
from lst_tools.hpc import detect

env = detect()
```

Detect scheduler/environment information (for example SLURM or PBS context).

::: lst_tools.hpc.detect

6. Examples

Representative command-line examples for common workflows.

6.1 End-to-End Tracking Workflow

```
# 1) initialize config
lst-tools init --geometry cone

# 2) convert HDF5 meanflow to meanflow.bin
lst-tools lastrac

# 3) set up parsing and run solver externally
lst-tools setup parsing --auto-fill

# 4) set up tracking and run solver externally in kc_* folders
lst-tools setup tracking --auto-fill

# 5) process tracking outputs
lst-tools process tracking --interpolate

```

6.2 Spectra Workflow

```
lst-tools setup spectra
lst-tools process spectra
```

6.3 HPC Script Generation

```
lst-tools hpc
```

6.4 Utilities

```
# inspect meanflow metadata
lst-tools info meanflow.bin

# clean generated artifacts
lst-tools clean parsing --force
lst-tools clean tracking --force
lst-tools clean spectra --force

```