

Hector

User Manual version 3.2.2

Machiel S. Bos

2026

TEROMOVIGO

Contents

1 Introduction **4**
- 1.1 How to cite Hector . . . 5
- 1.2 Main features . . . 5
- 1.3 A note to previous Hector users . . . 5
- 1.4 License . . . 6

2 Installation **6**
- 2.1 Virtual environments . . . 6
  - 2.1.1 Migrating from Hector v2.2 . . . 6
- 2.2 Requirements . . . 7
- 2.3 Step 1 — Install FFTW3 . . . 7
- 2.4 Step 2 — Install Hector . . . 8
  - 2.4.1 From PyPI (recommended once the package is released) . . . 8
  - 2.4.2 From source (development install) . . . 8
- 2.5 Command-line tools . . . 9
- 2.6 Updating Hector . . . 10

3 Hector Conventions **10**

4 Tutorial **13**
- 4.1 Example 1: Synthetic GNSS Time Series with Spikes and Offsets . . . 13
  - 4.1.1 Removal of Outliers . . . 14
  - 4.1.2 Estimation of the Linear Trend . . . 16
  - 4.1.3 Restricted Maximum Likelihood . . . 20
  - 4.1.4 Plotting the Power Spectral Density . . . 21
  - 4.1.5 Jupyter notebook version . . . 23
- 4.2 Example 2: The Monthly PSMSL Tide Gauge Data at Cascais . . . 23
- 4.3 Example 3: Creating Synthetic Coloured Noise . . . 24
- 4.4 Example 4: Post-Seismic Relaxation . . . 26
- 4.5 Example 5: Detecting Unknown Offset Epochs . . . 28
  - 4.5.1 Step 1 — Remove Spike Outliers from the Raw Data . . . 28
  - 4.5.2 Step 2 — Detect Offset Epochs with `findoffsets` . . . 29
  - 4.5.3 Why `FlickerGGM` and not `GGM` . . . 30
- 4.6 Example 6: Offset Detection on a Real GNSS Time Series . . . 31
  - 4.6.1 Step 1 — Convert tenv to NCF . . . 31
  - 4.6.2 Step 2 — Spike detection . . . 31
  - 4.6.3 Step 3 — Offset detection . . . 31
  - 4.6.4 Step 4 — Trend estimation . . . 32
- 4.7 Example 7: Piecewise Linear (Multi-Trend) Estimation . . . 32
  - 4.7.1 Step 0 — Generate the synthetic series . . . 32
  - 4.7.2 Step 1 — Remove spike outliers . . . 33
  - 4.7.3 Step 2 — Estimate the piecewise linear trend . . . 33
  - 4.7.4 Step 3 — Power spectral density (optional) . . . 33
- 4.8 Example 8: Toeplitz Factorisation — Levinson vs. Generalised Schur . . . 34

5 Model Specification **34**
5.1 Post-Seismic Deformation . . . 36
5.2 Multi-Trend . . . 36
5.3 Geophysical Signal . . . 37

6 Acceptable Data Format **38**
6.1 mom-format . . . 38
6.2 tenv-format (NGL) . . . 38
6.3 rlrdata-format . . . 38
6.4 ncf-format . . . 39
6.4.1 Standard channel names . . . 39
6.4.2 File structure . . . 39
6.4.3 Creating .ncf files . . . 41
6.4.4 Inspecting .ncf files . . . 41
6.4.5 Using .ncf files in Hector . . . 41
6.4.6 Batch processing with estimate_all_trends . . . 42
6.5 Sampling rates and time-axis precision . . . 42

7 Implemented Noise Models **43**
7.1 White Noise . . . 44
7.2 Power-Law Noise . . . 44
7.3 ARFIMA and ARMA . . . 44
7.4 Generalised Gauss-Markov Noise Model . . . 45
7.5 Flicker Noise and Random Walk Noise . . . 46
7.6 Comparing Noise Amplitudes Between Stations: the Modified Standard Deviation 48
7.7 Matern Noise Model . . . 49
7.8 VaryingPeriodic (Band-Pass) Noise Model . . . 49

8 The Akaike, Bayesian, and Kashyap Information Criteria **50**
8.1 Output of estimatetrend . . . 51

9 Auxiliary Python Scripts **52**
9.1 removeoutliers . . . 52
9.2 estimate_all_trends . . . 54
9.3 findoffsets . . . 55
9.4 find_all_offsets . . . 56
9.5 predicttrenderror . . . 58
9.6 simulatenoise . . . 58
9.7 convert_rlrdata2mom . . . 59
9.8 convert_tenv2netcdf . . . 59
9.9 date2mjd and mjd2date . . . 59
9.10 plot_ts . . . 60

10 Quick Reference for the Control-Files **61**
10.1 removeoutliers.ctl . . . 61
10.2 estimatetrend.ctl and findoffsets.ctl . . . 62
10.3 estimatespectrum.ctl . . . 65
10.4 simulatenoise.ctl . . . 66

11 License **67**

12 Appendix: History of Changes in Hector **68**
12.1 Version 1.1 . . . 68
12.2 Version 1.2 . . . 69
12.3 Version 1.3 . . . 69
12.4 Version 1.4 . . . 69
12.5 Version 1.5 . . . 69
12.6 Version 1.5.1 . . . 70
12.7 Version 1.5.2 . . . 70
12.8 Version 1.6 . . . 70
12.9 Version 1.7.2 . . . 70
12.10 Version 1.9 . . . 71
12.11 Version 2.0 . . . 71
12.12 Version 2.1 . . . 71
12.13 Version 2.2 . . . 71
12.14 Version 3.0 . . . 72
12.15 Version 3.1.0 . . . 73
12.16 Version 3.1.1 . . . 73
12.17 Version 3.1.2 . . . 74
12.18 Version 3.1.3 . . . 74
12.19 Version 3.1.4 . . . 75
12.20 Version 3.1.5 . . . 75
12.21 Version 3.1.6 . . . 76
12.22 Version 3.1.7 . . . 77
12.23 Version 3.1.8 . . . 77
12.24 Version 3.1.9 . . . 77
12.25 Version 3.1.10 . . . 78
12.26 Version 3.1.11 . . . 78
12.27 Version 3.1.12 . . . 78
12.28 Version 3.1.13 . . . 79
12.29 Version 3.2.0 . . . 80
12.30 Version 3.2.1 . . . 81
12.31 Version 3.2.2 . . . 81

References **82**

"The first house you build is for your enemy, the second for your friend, and the third for yourself."

1 Introduction

Hector is a software package that can be used to estimate a trend in time series with temporally correlated noise. Trend estimation is a common task in geophysical research, where one is interested in phenomena such as the increase in temperature, sea level, or GNSS-derived station position over time. The trend can be linear or a higher-degree polynomial, and in addition one can estimate periodic signals, offsets, and post-seismic deformation. Together they represent the model that is fitted to the observations.

It is well known that in most geophysical time series the noise is correlated in time (Agnew 1992; Beran 1992), and this has a significant influence on the accuracy with which the model parameters can be estimated. Therefore, the use of a computer program such as Hector is advisable.

Hector assumes that the user knows the type of temporally correlated noise present in the observations and estimates both the model parameters *and* the parameters of the chosen noise model using the Maximum Likelihood Estimation (MLE) method. Since for most observations the choice of noise model can be obtained from the literature or by inspecting the power spectral density, this is sufficient in most cases.

Another alternative is the program est_noise of (Langbein 2010), which follows a similar MLE-based approach to analyse geodetic time series. Recent versions include improvements to deal with missing data (Langbein 2017), using a different construction of the covariance matrix. In the book by (Montillet and Bos 2019) more examples on the analysis of geodetic time series with temporally correlated noise can be found.

Version 3.0 of Hector is a complete rewrite in Python. Earlier versions were written in C++, which resulted in fast programs but became difficult to maintain. Python offers a more concise notation for matrix and vector operations and improved plotting capabilities. For computationally demanding parts, Cython is used to retain high performance.

Compared to version 2.2, version 3.0 provides a speed-up of approximately 5–15× for typical GNSS time series. This gain comes primarily from the implementation of the Generalised Schur Algorithm (GSA), which reduces the computational cost of Toeplitz factorisation from $O(n^2)$ to $O(n \log^2 n)$, making the analysis of very long time series (more than 10 years of daily data) substantially faster.

In addition to the rewrite, several parts of the software have been improved and modules that were no longer in use have been removed. Version 3.0 remains compatible with previous versions for most programs. The offset detection algorithm has been significantly revised and improved, and now follows more closely the conventions used in estimatetrend and estimate_all_trends.py.

This manual starts in Section 2 by explaining how to install Hector on your computer. Section 3 provides a general description of the software, and Section 4 presents a tutorial with eight examples covering synthetic and real GNSS data, offset detection, post-seismic relaxation, multi-trend

estimation, and Toeplitz factorisation.

Section 5 describes the models that can be fitted to the observations. Sections 6 and 7 explain the accepted data formats and implemented noise models, respectively.

In Section 8 we discuss various information criteria that can be used to select the best noise model to describe the stochastic properties of the residuals. Section 9 presents auxiliary Python scripts that call Hector executables to automate various tasks. Finally, Section 10 provides a quick reference of the parameters used in the control files.

1.1 How to cite Hector

If you use **Hector version 3.0**, please cite:

> Bos, M. S. (2025). Fast noise analysis and offset detection for continuous GNSS time series. *J. Geod.*, submitted.

If you use an **earlier version of Hector** (v1.x or v2.x), please cite:

> Bos, M. S., Fernandes, R. M. S., Williams, S. D. P., and Bastos, L. (2013). Fast Error Analysis of Continuous GNSS Observations with Missing Data. *J. Geod.*, Vol. 87(4), 351–360, doi:10.1007/s00190-012-0605-0.

1.2 Main features

The main features of Hector are:

1. Correctly deals with missing data. No interpolation or zero padding of the data nor an approximation of the covariance matrix is required (as long the noise is, or has been made, stationary).
2. Allows yearly, half-yearly and other periodic signals to be included in the estimation process of the linear trend.
3. Allows the option to estimate offsets at given time epochs.
4. Includes power-law noise, ARFIMA, generalised Gauss-Markov (GGM), Matern, Varying Periodic (i.e. Bandpass) and white noise models. Any combination of these models can be made.
5. Comes with programs to remove outliers, to make power spectral density plots and to create files with synthetic coloured noise.
6. Has a program to automatically detect offsets in the time series.

1.3 A note to previous Hector users

Here are some differences:

- Hector v2.2 was distributed as compiled C++ executables that could be called from any shell or script without any setup. Hector v3.0 is a Python package. The command-line tools (`estimatetrend`, `removeoutliers`, etc.) work identically, but they must be installed into a Python environment first. See the Installation chapter for the recommended workflow and for specific advice on adapting existing bash scripts (§2.1).

- The estimation of a varying seasonal signal was never used by any user and has been removed.
- The detection of offsets has been improved significantly.

1.4 License

Hector is free for academic, research, and other non-commercial use. Commercial use requires a separate license from TeroMovigo – Earth Innovation Lda.

TeroMovigo continues to support the research community by making Hector freely available for scientific use and development.

The complete license text is reproduced in §11 of this manual and is included as the `LICENSE` file in the Hector repository.

2 Installation

Hector v3.0 is a Python/Cython package. The pure-Python parts work on any platform; the Cython extensions require a C compiler and the **FFTW3** library.

2.1 Virtual environments

Python programs are normally run inside a *virtual environment* — an isolated directory that contains its own Python interpreter and installed packages, separate from the system Python. Virtual environments prevent version conflicts when different projects require different package versions, and make it easy to reproduce a working setup on another machine. The Python documentation gives a full introduction.

The standard workflow is:

```
python -m venv env          # create the environment (once)
source env/bin/activate     # activate it (every new shell session)
pip install hector-ts       # install Hector into the active environment
```

On Windows the activation command differs:

```
env\Scripts\activate
```

Once activated, all Hector command-line tools appear on `PATH` and behave exactly as before. The shell prompt shows `(env)` as a reminder that the environment is active. To leave the environment, type `deactivate`.

2.1.1 Migrating from Hector v2.2

Hector v2.2 supplied compiled binaries that were globally available without any activation step. If you have existing bash or Python scripts that call Hector tools, choose the option below that suits your workflow.

Option 1 — pipx (closest to the v2.2 experience)

pipx installs Python command-line applications into automatically managed isolated environments and adds them to your global `PATH`. After installation, `estimatetrend`, `removeoutliers`, and every other Hector tool work from any shell or script with no activation step — exactly like the old compiled binaries. No changes to existing scripts are needed.

```
# Install pipx (once)
pip install pipx
pipx ensurepath              # adds ~/.local/bin to PATH; restart shell
# Install Hector
pipx install hector-ts
```

macOS users can also install pipx via Homebrew: `brew install pipx`.

Option 2 — Activate once in your shell profile

Add the activation command to your shell configuration file so Hector is always available in every new terminal:

```
# Append to ~/.bashrc or ~/.zshrc (adjust the path to your env)
echo 'source ~/hector/env/bin/activate' >> ~/.bashrc
```

After opening a new terminal (or running `source ~/.bashrc`) the environment is active automatically and all tools are on the path.

Option 3 — Add activation to individual scripts

For scripts that should remain self-contained, add a single line near the top:

```
#!/usr/bin/env bash
source /full/path/to/env/bin/activate   # harmless if already active
estimatetrend                           # works as before
removeoutliers
```

The activation call is cheap and idempotent: sourcing an already-active environment is safe and adds negligible overhead.

2.2 Requirements

- Python ≥ 3.10
- A C compiler (GCC, Clang, or MSVC)
- FFTW3 system library (see below)
- Python dependencies (installed automatically by pip): `numpy`, `scipy`, `matplotlib`, `pandas`, `mpmath`, `cython`, `netCDF4`

2.3 Step 1 — Install FFTW3

FFTW3 is a system library that must be present before building the Cython extensions.

macOS (Homebrew):

```
brew install fftw
```

Ubuntu / Debian:

```
sudo apt update && sudo apt install libfftw3-dev
```

Windows — try the wheel first:

On Windows, always try Step 2 (`pip install hector-ts`) first. Pre-built wheels on PyPI are compiled against a bundled FFTW3 and require no manual setup. Only follow the instructions below if pip reports a build error such as `fftw3.h: No such file or directory`.

Option A — Windows Subsystem for Linux (WSL, recommended)

WSL lets you run a full Ubuntu environment inside Windows and is the simplest path for most users. Enable it once from PowerShell:

```
wsl --install
```

Restart when prompted, then open the **Ubuntu** app that appears in the Start menu and follow the Ubuntu/Debian instructions above (`apt install libfftw3-dev`). All subsequent Hector commands are run inside the WSL Ubuntu shell. See https://learn.microsoft.com/windows/wsl/install for details.

Option B — Native Windows via MSYS2 / MinGW64

MSYS2 provides a package manager (`pacman`) and a GNU toolchain for native Windows. Download the installer from https://www.msys2.org/ and run it. Then, from the **MSYS2 MinGW64** shell, install FFTW3:

```
pacman -S mingw-w64-x86_64-fftw
```

The package page at https://packages.msys2.org/packages/mingw-w64-x86_64-fftw lists the current version and dependencies. After installation, open a standard Windows Command Prompt or PowerShell and install Hector as described in Step 2.

Alternatively, set the `FFTW_DIR` environment variable to a directory containing `fftw3.h` if FFTW3 is installed in a non-standard location.

2.4 Step 2 — Install Hector

2.4.1 From PyPI (recommended once the package is released)

```
pip install hector-ts
```

pip will download and compile the Cython extensions automatically. FFTW3 must already be installed (Step 1).

2.4.2 From source (development install)

```
git clone https://gitlab.com/machielsimonbos/hector-dev.git
cd hector-dev/code
pip install -e .
```

The `-e` flag installs in editable mode: changes to the Python source files take effect immediately. Cython (`.pyx`) files must be recompiled when changed:

```
python setup.py build_ext --inplace
```

Note — Cython extensions are required for usable performance. Hector automatically detects at start-up whether its compiled Cython extensions (`.so` / `.pyd` files) are available and falls back to pure Python if they are not. No error is raised and no configuration is needed — the fallback is silent. However, the Cython extensions provide speed-ups of 10–100× for the core Toeplitz solver, the GGM covariance, and the epoch scanner. Running without them largely defeats the purpose of Hector: a time series that takes a few seconds with the compiled extensions can take many minutes in pure Python. Always verify that the extensions loaded correctly after installation:

```
import hector.levinson, hector.ammargrag
# all should print True
print(hector.levinson._USE_CYTHON)
print(hector.ammargrag._USE_CYTHON_GAPS, hector.ammargrag._USE_CYTHON_NOGAP,
      hector.ammargrag._USE_CYTHON_GRAM)
```

If any value is `False`, the compiled extension for that component failed to load. Rerun `pip install hector-ts` (PyPI wheel) or `python setup.py build_ext --inplace` (source install) and check that the build completed without errors.

Note — one core per run. Hector runs on a single core. Its algorithms are sequential, and the only other source of threads, the linear-algebra library underneath NumPy (OpenBLAS, Accelerate or MKL), is restricted to one thread by default: more threads do not shorten a run (with eight threads, 1 % shorter on an Apple M4 and 3 % longer on an 8-core Linux machine, at up to 8.6 times the processor time). Several stations are therefore analysed fastest by running several Hector programs side by side, one per core; a run of a 40-year daily series needs less than 0.5 GB of memory. To give the linear-algebra library more threads anyway, set one of the usual variables yourself, for example `OPENBLAS_NUM_THREADS=4 estimatetrend` (or `VECLIB_MAXIMUM_THREADS` on macOS); Hector then leaves all of them alone.

2.5 Command-line tools

All executables are registered as entry points by pip and are available on `PATH` immediately after installation.

Name	Description
`estimatetrend`	Estimate trend, offsets, and periodic signals via MLE for a single station.
`findoffsets`	Iterative forward search for offset epochs in a single time series.
`estimate_all_trends`	Batch version of `removeoutliers` + `estimatetrend`: processes every `.mom` and `.ncf` file in `obs_files/`.
`find_all_offsets`	Offset detection for every station: multivariate (E+N+U) search on `.ncf` files, or `findoffsets` per `.mom` file with `--mom`.
`removeoutliers`	Remove outliers from a time series by data snooping (`IQ_factor`) or spike detection (`Spike_factor`).
`estimatespectrum`	Estimate the power spectral density via the Welch periodogram.
`simulatenoise`	Generate synthetic coloured-noise time series.
`predicttrenderror`	Predict trend uncertainty as a function of series length.

Name	Description
`convert_rlrdata2mom`	Convert PSMSL RLR sea-level files to mom format.
`convert_tenv2netcdf`	Convert NGL tenv/tenv3 GNSS position files to NCF format.
`ncfgen`	Create a `.ncf` (netCDF4) multi-channel file from ASCII data and a JSON metadata file.
`ncfdump`	Inspect or export a `.ncf` file to ASCII.
`plot_ts`	Plot a time series from a `.mom` / ASCII text file or a `.ncf` / `.nc` file.
`date2mjd`	Convert a calendar date to Modified Julian Date.
`mjd2date`	Convert a Modified Julian Date to calendar date.
`hector-examples`	Copy the examples and this manual to a directory (default `./hector-examples`).

2.6 Updating Hector

To upgrade an existing installation to the latest release on PyPI:

```
pip install --upgrade hector-ts
```

3 Hector Conventions

Hector consists of a set of simple command-line programs that help you through the steps of time series analysis: removal of outliers, estimation of a trend, and estimating the power spectral density (optionally overlaid with the spectrum of the fitted noise model). In addition, a program exists to find offsets in the data. Finally, synthetic time series with coloured noise can be created.

Each program comes with its own control-file in which parameter values needed by the program are given. For example, the program `removeoutliers` has a control-file called `removeoutliers.ctl`. All programs follow the same naming scheme for their control-files.

This control-file is a simple ASCII text file containing on each row a keyword followed by its value or a list of values. Some keywords are optional and if they are not specified, then the default value will be used. A list of all possible keywords for each control-file is given in Section 10.

Following the nomenclature of Bevis and Brown (2014), Hector estimates station trajectory models. These models can include a trend, seasonal signals, offsets and post-seismic deformation. See Section 5 for more details. In Hector you can specify the type of trend, which seasonal and other periodic signals in the control-file. In addition you can specify in the control-file if you want to estimate offsets, breaks and/or post-seismic deformation. These are a kind of *global* model parameters that you normally apply to all your time series. On the other hand, the time of an offset, break or post-seismic deformation differs from station to station and are a kind of *local* model parameters. In Hector this information is normally written in the header of the time series.

A schematic work-flow of the analysis pipeline is shown in Figure 1. The pipeline has two entry paths that converge on a shared processing chain.

Path 1 — offset epochs known. When the dates of instrumental offsets, antenna changes, or earthquake breaks are already known (e.g. from station logs), the raw time series can be placed

directly in `./obs_files` with the offset epochs annotated in the file header. No automatic offset detection is needed.

Path 2 — offset epochs unknown. When offset dates are not known in advance, a two-step pre-processing sequence is required before the shared pipeline can begin.

First, `removeoutliers` is run in **SpikeDetector mode** (`Spike_factor`) on the raw files in `./raw_files`; the cleaned output is stored in `./stage_files`. SpikeDetector is used here because it detects isolated spikes using first-difference analysis and is immune to unmodelled offsets: an offset step produces exactly one large difference with no sign reversal, so it is never flagged. This is important because at this stage no offset information is available, and using IQR/OLS data snooping on data with large undetected offsets can incorrectly remove hundreds of valid observations (the OLS fit is biased by the step, making valid points on the wrong side of it appear as outliers).

Next, `find_all_offsets` scans the spike-filtered series in `./stage_files` for statistically significant Heaviside steps using a multivariate GLR test (combining E, N and U components) and writes the detected offset epochs into the output NetCDF files in `./obs_files`.

From `./obs_files` onward both paths follow the same sequence. A second `removeoutliers` pass is run on the offset-annotated files. For `.mom` files `estimate_all_trends` uses **DataSnooping mode** (`IQ_factor 3`): because the design matrix now includes offset columns, the residuals no longer absorb the step signal, so smaller outliers that were previously hidden near an offset epoch can be detected and removed. For `.ncf` files it uses spike detection (`Spike_factor 5`). The cleaned files are stored in `./pre_files`. Finally, `estimatetrend` fits the full model (trend, periodic signals, offsets and noise) and writes the results to `./mom_files`.

The user is free to choose directory names, but the Python scripts assume the structure shown in the table below.

Step	Directory	Purpose
1	`./raw_files`	Raw observations with outliers; no offset information.
2	`./stage_files`	Large outliers removed; ready as input to `find_all_offsets`.
3	`./obs_files`	Time series with offset epochs annotated in the header.
4	`./pre_files`	Outlier-cleaned observations; ready for trend estimation.
5	`./mom_files`	Output of `estimatetrend`: observations plus fitted model.
6	`./data_figures`	Time series plots produced by `plot_ts`.
7	`./psd_figures`	Power spectral density plots produced by `estimatespectrum`.

Once `estimatetrend` has completed, the results can be visualised. The script `plot_ts` reads the output from `./mom_files` and writes observation plots to `./data_figures`. To assess how well the

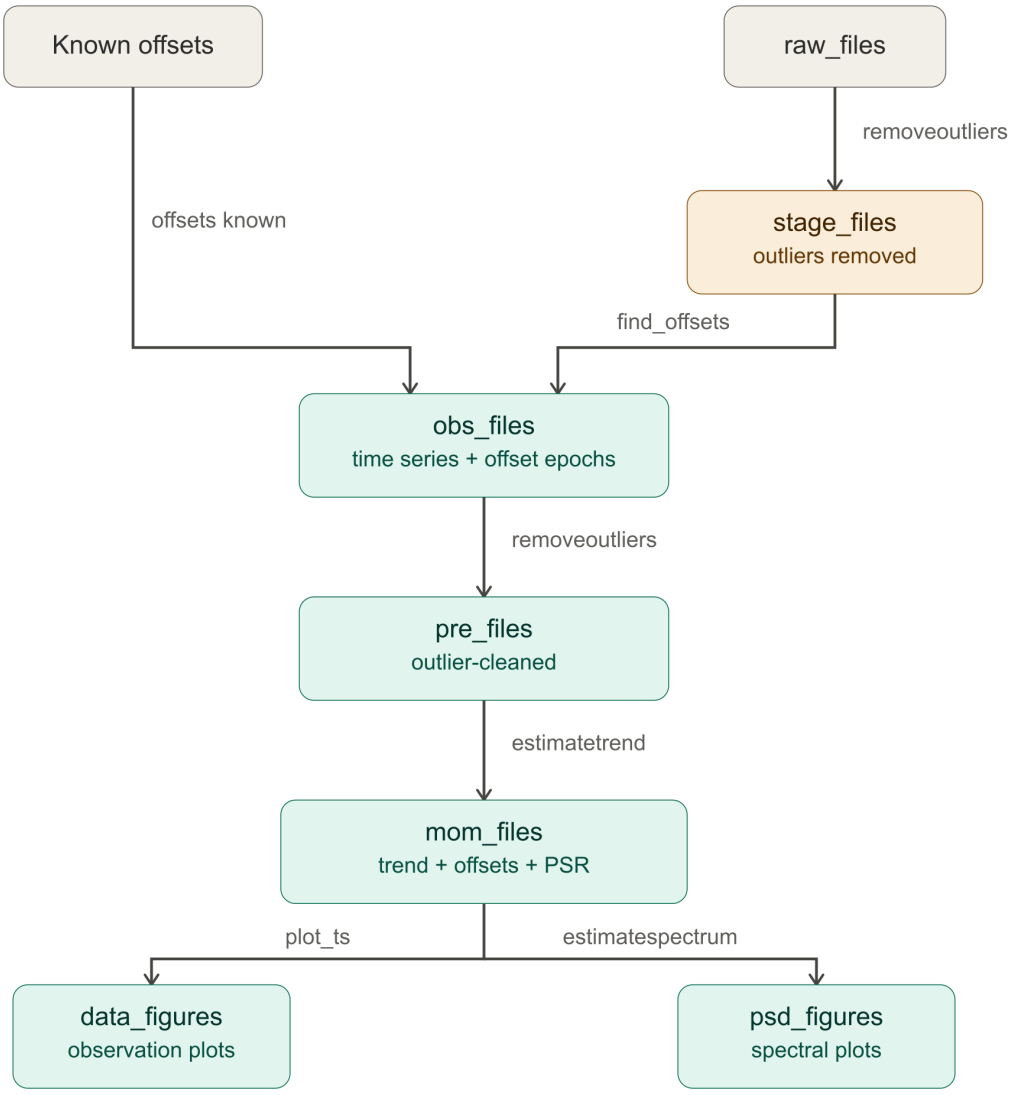


Figure 1: Schematic work-flow of the analysis pipeline. Green boxes are standard directory names; labels on arrows are Python scripts that automate each step.

noise model fits the data, `estimatespectrum` computes a power spectral density of the residuals together with the theoretical spectral shape of the estimated noise model and saves the figures to `./psd_figures`.

The auxiliary Python scripts described in Section 9 have all been updated for v3.0 and are installed alongside the command-line tools.

4 Tutorial

The eight tutorial examples are bundled with the Hector package. Copy them to a convenient working directory by running:

```
hector-examples
```

This creates a directory `hector-examples/` in the current working directory containing `ex1/` through `ex8/`. You can also specify a different destination:

```
hector-examples ~/my_hector_work
```

Since version 3.1.9 the same command also unpacks a `paper_scripts/` directory: the numerical experiments of the paper *"Faster analysis of GNSS time series"* (Journal of Geodesy), together with the raw results measured for that paper, so every figure can be re-created and every benchmark re-run. See `paper_scripts/README.md`; the driver is `run_all.py`, and `setup_tools.py` configures the optional external tools (Hector C++ v2.2, R with the gmwmx2 package) used by the timing comparisons.

By going step by step through the analysis of some example data sets the working of Hector will be explained. First, we look at some synthetic GNSS data.

4.1 Example 1: Synthetic GNSS Time Series with Spikes and Offsets

In the directory `ex1/obs_files` the file `TEST.mom` is stored. It represents some fictional observed GNSS coordinate time series. The file extension 'mom' stands for **M**odified Julian Date — **O**bservations — **M**odel. Here the last component (the fitted model) is missing. The Modified Julian Date (MJD) is a convenient format to make plots. You can use the programs `date2mjd` and `mjd2date` to convert between year/month/day/hour/minute/second and MJD values. These data are stored in a simple ASCII text file and can be inspected by any normal text editor. When doing so, one will detect a few header lines:

```
# sampling period 1.0
# offset 50284.0
# offset 50334.0
# offset 50784.0
# offset 51034.0
```

The first line just tells the program that the sampling period of the data is daily (T=1 day). The other lines tell Hector at which epochs an offset needs to be estimated. Since this information is known, the files were stored in the `obs_files` directory and not in `raw_files`. For detecting offsets, see Example 5.

An offset only makes sense if there are observations both before and after it. Hector therefore ignores an offset that lies before the first or after the last observation and, when several offsets fall between two consecutive observations (for example inside a data gap), it keeps only the first one listed in the header. Each removed offset is reported in the output. Without this check, such offsets would make the least-squares problem singular.

For the moment we assume that the information about these epochs of the offsets is given. For example, these are times when the GNSS receiver was changed as specified in the logfile. Later on we will deal with the situation when this information is missing.

After the header line, the data are listed (`TEST.mom`):

```
50084.0 -17.88951
50085.0 -16.88599
50086.0 -16.84916
...
```

The east component (column 2) can be plotted and saved to a PNG file using `plot_ts` (§9.10):

```
plot_ts -i obs_files/TEST.mom -cx 1 -cy 2 -lx Year -ly "east (mm)" -o TEST_obs.png
```

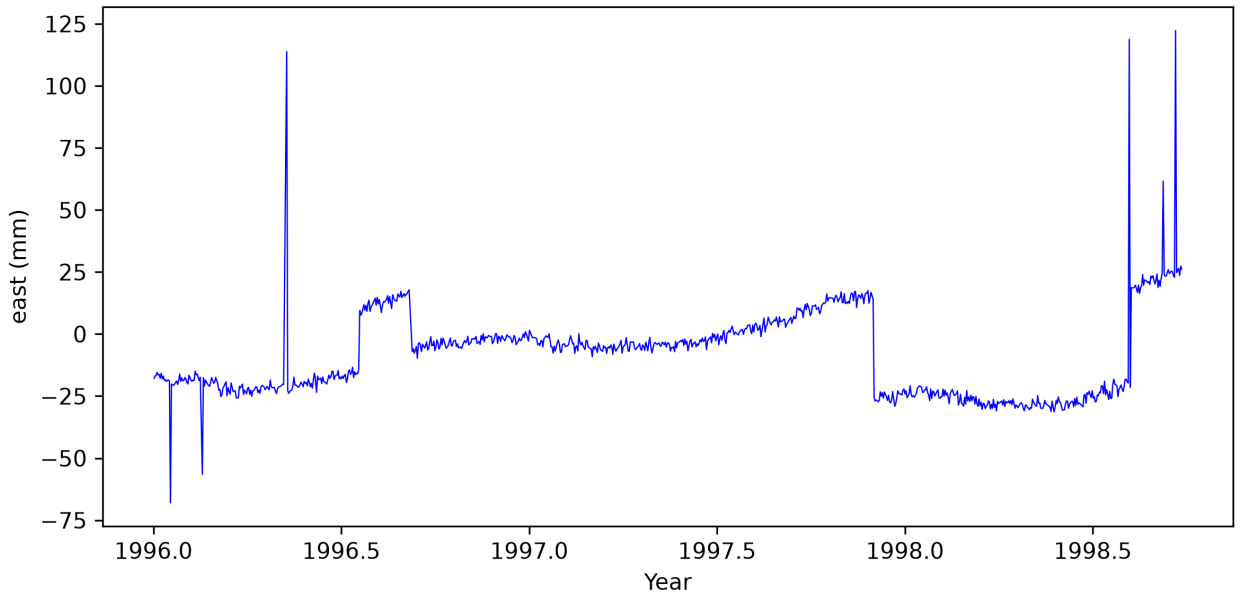


Figure 2: The observed data of `TEST.mom`.

One can detect the offsets and the presence of outliers in the resulting plot.

4.1.1 Removal of Outliers

To remove the outliers we need to run `removeoutliers` which requires a control-file called `remove-outliers.ctl`. Hector uses various control-files which are simple text files and the rows with the keywords can occur in any order. If Hector cannot find a keyword, then it will complain unless the keyword is optional. If a keyword is optional and has been omitted, then its default value will be

used. A control-file with a different name can be specified on the command line with the `-i` option. For example:

```
removeoutliers -i othercontrolfile.ctl
```

All Hector command-line tools accept the `-i` option to select the control-file; when it is omitted, each tool falls back to its default name (`removeoutliers.ctl`, `estimatetrend.ctl`, and so on) in the current directory. Note that in Hector v3.0 the control-file name must be given *after* `-i`; the bare positional form used by earlier (2.x) versions (`removeoutliers othercontrolfile.ctl`) is no longer accepted.

The contents of `removeoutliers.ctl` in the `ex1` directory is:

```
DataFile              TEST.mom
DataDirectory         obs_files
OutputFile            pre_files/TEST.mom
periodicsignals       365.25 182.625
estimateoffsets       yes
ScaleFactor           1.0
PhysicalUnit          mm
TimeUnit              days
IQ_factor             3
```

The keyword `periodicsignals` replaces the old `seasonalsignal` / `halfseasonalsignal` pair; the old keywords still work for backward compatibility. The values `365.25` and `182.625` request a yearly and a half-yearly signal.

The first line gives the name of the DataFile with the raw data which is `TEST.mom` in our case. The second line gives the directory where this file can be found and the third line contains the required name of the file with the preprocessed data (outliers removed): `pre_files/TEST.mom`. The output will always be in mom-format which stands for **M**JD, **O**bservations, **M**odel. The last column is optional and since `removeoutliers` only replaces the raw observations with the preprocessed observations, no third column will be added. See Section 6 for more details on the acceptable data format. To run `removeoutliers`, simply type `removeoutliers` on the prompt. A file called `removeoutliers.json` will be created that contains the number of epochs and the MJD values of the points that have been removed:

```
{
    "N": 1000,
    "outliers": [
        50100.0,
        50131.0,
        50213.0,
        51032.0,
        51065.0,
        51077.0
    ]
}
```

`removeoutliers` fits a linear trend to the raw data using ordinary least-squares and afterwards subtracts this linear trend from the observations to create residuals. These residuals are ordered

by size and the interquartile range is computed (this is the value of the residual at 75% of the sorted array minus the value of the residual at 25% of the sorted array). Any residual with a value less than 3 times this interquartile range below or above the median is considered to be an outlier (Langbein and Bock 2004). This factor of 3 is set by the keyword `IQ_factor` and can be changed by the user.

In this control-file one must also give the physical unit of the data. This information is not essential but reminds the user to think about the unit of the data and if some scaling is required. Such scaling is set by the keyword `ScaleFactor` which is 1.0 in this case. This keyword is optional and if omitted then a default value of 1.0 will be assumed.

The linear trend is estimated assuming a white noise model and, as can be seen from the `removeoutliers.ctl` file, a yearly and a half-yearly signal are also included in the estimation process. Offsets are also estimated. The keyword `periodicsignals` is optional and can be omitted.

4.1.2 Estimation of the Linear Trend

Now that the outliers have been removed, we can estimate the linear trend. The parameters that control this analysis are by default given in the file `estimatetrend.ctl`. As before, a different name for the control-file can be specified on the command line with the `-i` option (`estimatetrend -i othercontrolfile.ctl`). The contents of `estimatetrend.ctl` is:

```
DataFile            TEST.mom
DataDirectory       pre_files
OutputFile          mom_files/TEST.mom
interpolate         no
PhysicalUnit        mm
TimeUnit            days
ScaleFactor         1.0
periodicsignals     365.25 182.625
estimateoffsets     yes
NoiseModels         GGM White
GGM_1mphi           6.9e-06
useRMLE             yes
```

The recommended noise model for GNSS daily data is `GGM White` with `GGM_1mphi 6.9e-06`, which closely approximates power-law plus white noise while remaining strictly stationary. The `useRMLE yes` keyword activates Restricted Maximum Likelihood (RMLE), which is the recommended default in v3.0; see the dedicated subsection below for the mathematical details. The classic `Powerlaw White` combination also works and gives very similar results; the `GGM` model is preferred because it handles spectral indices down to $\kappa = -2$ without approximation.

Again, there is the keyword `DataFile` which should be given by the name of the input file which is `TEST.mom` in this case because we are now going to use the preprocessed observations. These preprocessed observations together with the estimated trend in the third column, are written to the file name given after the keyword `OutputFile`. As before, a yearly and a half-yearly signal and the offsets are estimated.

The `LikelihoodMethod` keyword has been omitted. The default method is `AmmarGrag`, which now uses the Generalised Schur Algorithm (GSA) internally for long series (more than 1000 points), giving $O(n \log^2 n)$ complexity instead of $O(n^2)$.

Note that if the `ScaleFactor` is not 1 in the file `removeoutliers.ctl`, then you probably want to set it to 1 in `estimatetrend.ctl` to avoid applying the scaling twice. Again, this keyword is optional and a default value of 1 is assumed when this keyword is not provided. The program `estimatetrend` shows the following on the screen:

```
************************************
    estimatetrend, version 3.2.2.
************************************

Filename                   : pre_files/TEST.mom
TS_format                  : mom
TimeUnit                   : days
PhysicalUnit               : mm
ScaleFactor                : 1.000000
Number of observations+gaps: 1000
Percentage of gaps         :  10.6
No Polynomial degree set, using offset + linear trend
0) GGM
1) White
Nparam : 2
useRMLE-> True
----------------
  AmmarGrag
----------------
Number of iterations : 56
min log(L)           : -1706.494080
ln_det_I             : 20.179890
ln_det_HH            : 64.521641
ln_det_C             : 113.447483
AIC                  : 3438.988161
BIC                  : 3501.332336
KIC                  : 3521.512226
driving noise        : 1.570261

Noise Models
------------
GGM:
fraction  = 0.18753
sigma     =  4.2402 mm/yr^0.31
mod. std  =  1.7207 mm (over 8 yr)
d         =  0.6204
kappa     = -1.2407
1-phi     = 6.9000e-06 (fixed)

White:
fraction  = 0.81247
sigma     =  1.4154 mm
```

```
No noise parameters to show

bias : 1.142 +/- 3.240 (at 50583.50)
trend: 16.762 +/- 1.081 mm/yr
cos  365.250 : 4.151 +/- 0.421 mm
sin  365.250 : -3.859 +/- 0.432 mm
amp  365.250 : 5.683 +/- 0.425 mm
pha  365.250 : -42.930 +/- 4.416 degrees
cos  182.625 : 0.050 +/- 0.269 mm
sin  182.625 : -0.017 +/- 0.271 mm
amp  182.625 : 0.342 +/- 0.179 mm
pha  182.625 : -4.119 +/- 95.983 degrees
offset at 50284.0000 :   24.36 +/-  0.85 mm
offset at 50334.0000 :  -24.40 +/-  0.93 mm
offset at 50784.0000 :  -40.62 +/-  0.88 mm
offset at 51034.0000 :   38.45 +/-  0.91 mm
--> mom_files/TEST.mom
---    0.527 s ---
```

The bias term corresponds to the value of the model at the reference epoch, shown in the output as a Modified Julian Date (`at 50583.50`). By default this is the midpoint of the time series. If another reference epoch is required, it can be set with the keyword `ReferenceEpoch`, followed by year, month and day. For example, a reference epoch of 1 January 2008 is given by:

```
ReferenceEpoch          2008 1 1
```

Changing the reference epoch shifts the bias value (and its formal uncertainty, which grows the further the reference epoch lies from the middle of the observed time span) but leaves the trend and all other estimated parameters unchanged.

The first lines identify the file and its format, followed by the model summary and the Nelder-Mead convergence statistics.

The quality of the chosen noise models is evaluated using the Akaike Information Criteria (AIC), Bayesian Information Criteria (BIC), and Kashyap Information Criteria (KIC). More details are given in §8.

Since we are using white and GGM noise, two noise amplitudes need to be estimated. The way how this is done in Hector has confused some users and therefore some extra explanation is in order. First, we model our GGM noise as a stationary process. This type of noise can be created by filtering white noise. In CATS, Hector and est_noise this covariance matrix is created for the case of filtering white noise with variance 1. The noise amplitude is simply a scale factor of this covariance matrix.

Secondly, we add the covariances to get the total covariance:

$$\mathbf{C} = \sigma_{wn}^2 \mathbf{I} + \sigma_{ggm}^2 \mathbf{E}(\kappa)$$

where σ_{wn}^2 and σ_{ggm}^2 are the two noise amplitudes we need to estimate. However, as (Williams 2008) explains, it is convenient to write this as:

$$\mathbf{C} = \sigma^2 \left[f_{wn}\mathbf{I} + f_{ggm}\mathbf{E}(\kappa) \right]$$

where f_{wn} and f_{ggm} are the fractions. σ^2 can directly be computed from the residuals, which means we only need to search for the fractions and the GGM spectral index d using numerical maximisation. In Section 7 more details are given.

To return to the output of `estimatetrend`, we can see both fractions and the main driving noise σ^2 which has for this case a standard deviation of 1.5703 mm. The standard deviation of the white noise is:

$$\sigma_{wn} = \sqrt{0.81247} \times 1.5703 = 1.4154 \text{ mm}$$

In Hector the covariance matrix **E** is not scaled by the factor $\Delta T^{-\kappa/2}$, where ΔT is the sampling period in years (Williams 2003). However, to facilitate comparison with amplitude values for GGM noise quoted in the literature, we divide the estimated amplitude by $\Delta T^{-\kappa/4}$:

$$\sigma_{ggm} = \frac{\sqrt{0.18753} \times 1.5703}{(1/365.25)^{(0.25\times 1.2407)}} = 4.240 \text{ mm/yr}^{0.31}$$

The second noise parameter of the GGM noise is the spectral index d which is $-1/2$ times the more often used parameter κ in other papers on GNSS time series. The values of d and the fractions need to be determined using the numerical minimisation scheme. When `GGM_1mphi` is set, the $1 - \phi$ parameter is held fixed at the given value, so it is not estimated.

For white noise there is no additional parameter to estimate so, that is why there is this line “No noise parameters to show” in the output in the white noise section. More details on the noise models are given in Section 7.

The rest of the lines show the estimated values of the model such as nominal bias (also known as intercept at t_0 and which is equal to the estimated value at t_0), linear trend and a seasonal signal. By default, the epoch t_0 is chosen at the midpoint of the time series. To explain this better, assume that we have 5 observations. The design matrix **H** looks like:

$$\mathbf{H} = \begin{pmatrix} 1 & -2 \\ 1 & -1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

The first column will estimate the nominal bias, the second the linear trend. The two columns are orthogonal since $\mathbf{H}^T\mathbf{H}$ produces a diagonal matrix. Thus, the estimation of the nominal bias is not influenced by the estimation of the linear trend which is beneficial for the accuracy of the nominal bias. It also means that the nominal bias corresponds to the value of the model at the time at row 3 (half of the time series). Hector notes this time in the output.

Also shown in the output are the values of the estimated offsets. The results of `estimatetrend` can be displayed on screen by adding `-graph` to the command, or saved as a PNG file with `-png`:

```
estimatetrend -graph
estimatetrend -png
```

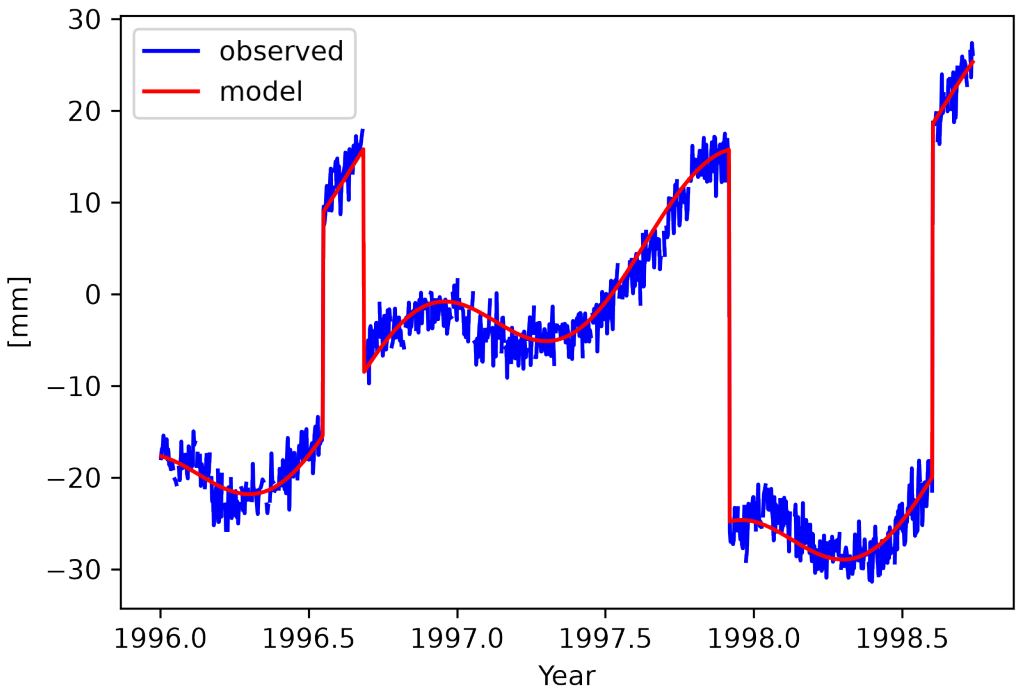


Figure 3: Raw, filtered data and estimated model of the `TEST.mom` time series.

4.1.3 Restricted Maximum Likelihood

The control file above contains the keyword `useRMLE yes`. By default (`useRMLE no`) Hector uses ordinary Maximum Likelihood Estimation (MLE), in which the noise parameters are found by maximising the log-likelihood

$$\ln L = -\frac{1}{2}\left(N \ln 2\pi + \ln |\mathbf{C}| + 2N \ln \hat{\sigma} + N\right)$$

where N is the effective number of observations (total epochs minus the number of gaps), p is the number of parameters in the trajectory model (bias, trend, periodic signals, offsets), $\mathbf{C}$ is the normalised covariance matrix of the noise, and $\hat{\sigma}$ is the estimated driving noise amplitude.

MLE is known to underestimate the noise amplitude when p is not negligible compared to N. This occurs with short time series or when many offsets are estimated simultaneously — each additional trajectory parameter 'uses up' one degree of freedom that MLE does not account for in the noise estimation.

Restricted Maximum Likelihood (RMLE, also called REML in the statistics literature) removes this bias by adding the Patterson–Thompson correction (Patterson and Thompson 1971) to the log-likelihood:

$$\ln L_{\mathrm{RMLE}} = \ln L - \frac{1}{2}\left(\ln\left|\mathbf{H}^\top\mathbf{C}^{-1}\mathbf{H}\right| - \ln\left|\mathbf{H}^\top\mathbf{H}\right|\right)$$

The additional term depends on the design matrix **H** and the current noise covariance $\mathbf{C}^{-1}$, so it changes at every Nelder-Mead iteration. Algebraically it is equivalent to replacing N by $N - p$ in the log-likelihood, which is the correct degrees-of-freedom correction for fixed effects (Harville 1974). Its effect is small when $N \gg p$ and largest when N/p is of order a few tens.

Recommendation: Following the advice of Gobron et al. (2022), use `useRMLE yes` (the v3.0 default) for all standard analyses. The Kashyap Information Criterion (KIC), printed alongside AIC and BIC in the output, uses the same Fisher-information term $\ln\left|\mathbf{H}^\top\mathbf{C}^{-1}\mathbf{H}\right|$ and is most meaningful when RMLE is active.

The `useRMLE` keyword is recognised by both `estimatetrend` and `findoffsets`.

4.1.4 Plotting the Power Spectral Density

We have used a GGM plus white noise model in our estimation process. To verify if this is correct, it is good to make a power spectral density (PSD) plot of the residuals (i.e. the difference between observations minus the estimated linear trend and additional offsets and periodic signals). This can be done using the program `estimatespectrum` which computes a Welch periodogram, stored in the file `estimatespectrum.out`. As usual, the behaviour of this program is controlled by the file `estimatespectrum.ctl`:

```
DataFile            TEST.mom
DataDirectory       mom_files
interpolate         no
PhysicalUnit        mm
TimeUnit            days
```

Two optional keywords control the periodogram: `NumberOfSegments` (default 4) and `Fraction` (default 0.1). The time series is divided into `NumberOfSegments` equal parts; with 50% overlap this gives `2×NumberOfSegments - 1` Welch periodograms to average. With the default of 4 segments this yields 7 periodograms, each spanning 25% of the data. A split-cosine-bell (Tukey) window tapers the first and last `Fraction` of each segment to zero; the default of 0.1 leaves the central 80% of each segment flat. The area underneath the (one-sided) power spectral density plot should be equal to the variance of the time series (Buttkus 2000). For now, just run:

```
estimatespectrum
```

The output printed on the screen is:

```
**************************************
    estimatespectrum, version 3.2.2.
**************************************
```

```
Filename                    : mom_files/TEST.mom
TS_format                   : mom
TimeUnit                    : days
PhysicalUnit                : mm
ScaleFactor                 : 1.000000
Number of observations+gaps: 1000
Percentage of gaps          :  10.6
DeltaT = 1.000000
```

The periodogram itself is written to `estimatespectrum.out`.

The PSD of our estimated noise model can be overlaid on the periodogram by running `estimatespectrum` with the `-model` flag:

```
estimatespectrum -model
```

This reads the noise model parameters automatically from `estimatetrend.json`, which `estimatetrend` always writes in the directory where it is run, and saves the model PSD to `modelspectrum.out`. If the JSON file has a different name, use the `-j` flag to specify it:

```
estimatespectrum -model -j mystation.json
```

The power spectral density can be displayed on screen with:

```
estimatespectrum -graph
```

or saved as a PNG file with `-png`. The plot is located in the directory `psd_figures`.

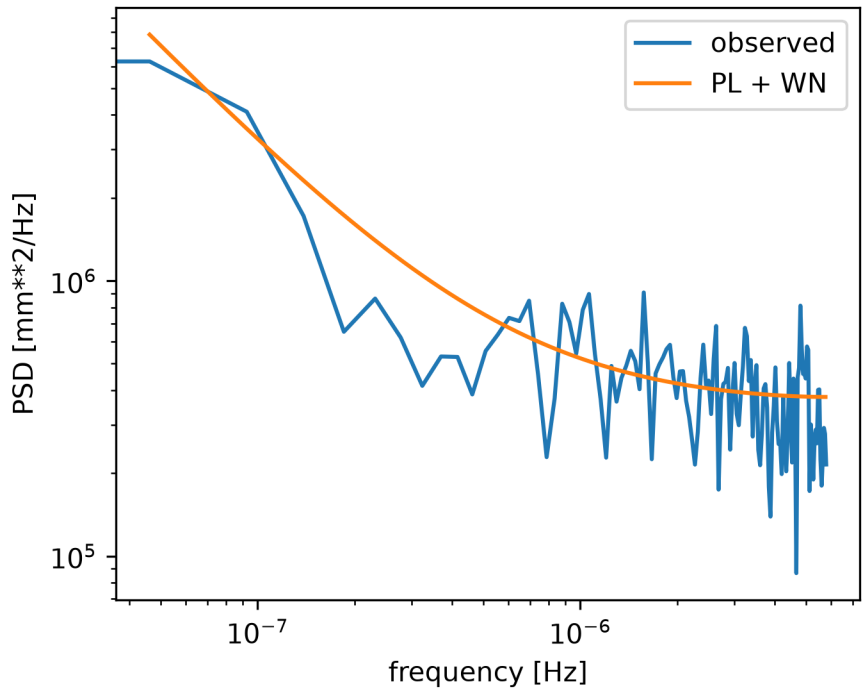


Figure 4: Power spectral density of the residuals from Example 1, with the GGM+White model overlaid.

4.1.5 Jupyter notebook version

A self-contained Jupyter notebook (`example1_jupyter.ipynb`) that covers the complete Example 1 workflow — outlier removal, trend estimation, time-series plot, residuals, and PSD — is provided in `examples/ex1/`.

To run it, install Jupyter in the same Python environment as Hector and launch it from the `ex1/` directory:

```
pip install jupyter
cd hector-examples/ex1
jupyter notebook
```

Then open `example1_jupyter.ipynb` in the browser window that appears. For help on installing and using Jupyter Notebook see https://jupyter.org/.

One thing to keep in mind: `Control`, `Observations`, `DesignMatrix`, and `Covariance` are *singleton* classes — each class has only one live instance per kernel session. Always call `SingletonMeta.clear_all()` before starting a fresh analysis in the same kernel, otherwise the second run silently reuses the objects from the first. The notebook includes this call in every analysis cell.

4.2 Example 2: The Monthly PSMSL Tide Gauge Data at Cascais

In the directory `ex2` we have stored the monthly tide gauge data of Cascais, downloaded from PSMSL (`http://www.psmsl.org/data/obtaining/stations/52.php`). This time series has no outliers so we can directly estimate the linear trend.

Since `estimatetrend` reads mom-format files, the PSMSL RLR file must first be converted:

```
convert_rlrdata2mom -i 52.rlrdata -o cascais.mom
```

The control-file `estimatetrend.ctl` is:

```
DataFile            cascais.mom
DataDirectory       ./
OutputFile          cascais_out.mom
DegreePolynomial    1
interpolate         no
periodicsignals     365.25 182.625
estimateoffsets     no
NoiseModels         ARMA
PhysicalUnit        mm
AR_p                1
MA_q                0
useRMLE             yes
```

Here we are using the ARMA noise model. To be precise, there is 1 AR coefficient (set by the `AR_p` keyword) and 0 MA coefficients (set by the `MA_q` keyword). Thus, we can shorten our notation of ARMA(1,0) to AR(1). If we now run `estimatetrend` we obtain a linear trend of 1.270 $\pm$ 0.076 mm/yr. If we now change the noise model to AR(5) (`AR_p`=5), ARFIMA with `AR_p`=1 and `MA_q`=0, and GGM, we obtain trends of 1.277 $\pm$ 0.105, 1.252 $\pm$ 0.185 and 1.246 $\pm$ 0.233 mm/yr, which all have lower BIC and AIC values than the AR(1) noise model. Since only one noise model is used each time, the

fraction is always 1. The power spectral density plot is made with `estimatespectrum`; the sampling period of these monthly data is 30.4375 days. The control-file `estimatespectrum.ctl` is:

```
DataFile            cascais_out.mom
DataDirectory       ./
interpolate         no
PhysicalUnit        mm
```

Running `estimatespectrum -model` after `estimatetrend` writes the periodogram to `estimatespectrum.out` (frequencies from 1.1317e-09 to 1.9013e-07 Hz) and the PSD of the estimated noise model, read from `estimatetrend.json`, to `modelspectrum.out`.

In sea level research one is sometimes also interested in the acceleration. It is possible to estimate this by setting the optional keyword `DegreePolynomial` to 2 in `estimatetrend.ctl`. Its default value is 1. If we do this, then we obtain, using the GGM noise model, a quadratic term of 0.003 $\pm$ 0.007 mm/yr^2. Note that this is half of the acceleration.

Next, one can include additional geophysical signals in the analysis. A simple regression coefficient will be estimated. To do so, set the keyword `estimatemultivariate yes` in the control file and specify the file with the geophysical signal, see also Section 5. A ready-made control file `estimatetrend_multivariate.ctl` is provided in the example directory. Run it with:

```
estimatetrend -i estimatetrend_multivariate.ctl
```

Here we include the monthly surface pressure provided by the Hadley Centre Sea Level Pressure dataset (HadSLP2). Using again the AR(1) noise model, the relevant part of the output is:

```
trend: 1.304 +/- 0.069 mm/yr
scale factor of hadslp2_cascais :  -11.9681 +/-   0.4459 mm
```

The last line shows the value for the regression coefficient (i.e. scale factor) for the geophysical signal, labelled with the filename stem of the multi-variate file. In this case the regression coefficient is -11.97 mm/mbar, which is close to the standard inverted barometer value.

4.3 Example 3: Creating Synthetic Coloured Noise

In order to perform Monte Carlo simulations, one must create time series with synthetic coloured noise. This task can be performed with the program `simulatenoise`. It is based on the method described by (Kasdin 1995) where an impulse response, different for each noise model, is convoluted with a white noise time series. The result is our desired synthetic noise time series. As usual, the convolution is performed using FFT. There might be some spin-up effects because implicitly it is assumed that the noise is zero before the first observation. To mitigate this problem, the keyword `TimeNoiseStart` can have a large number, normally 1000 is enough, to specify the amount of extra points before the first observations need to be modelled.

In the directory `ex3` the control-file `simulatenoise.ctl` is given:

```
SimulationDir           ./obs_files
SimulationLabel         test_base
NumberOfSimulations     10
NumberOfPoints          5000
SamplingPeriod          1
```

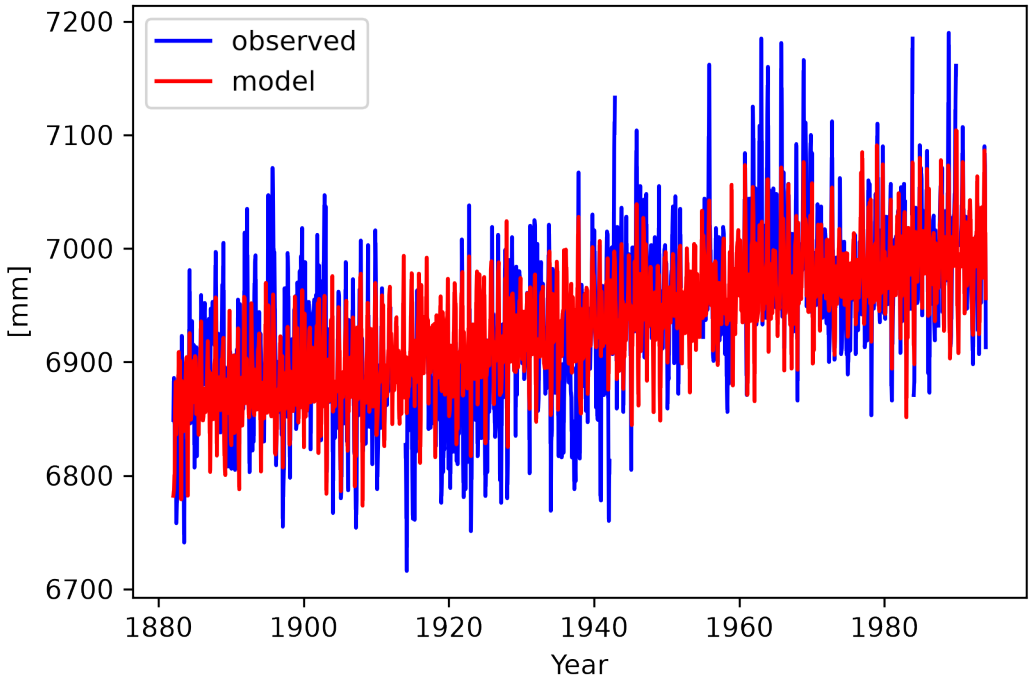


Figure 5: Observed and estimated sea level at Cascais with the surface-pressure correction included.

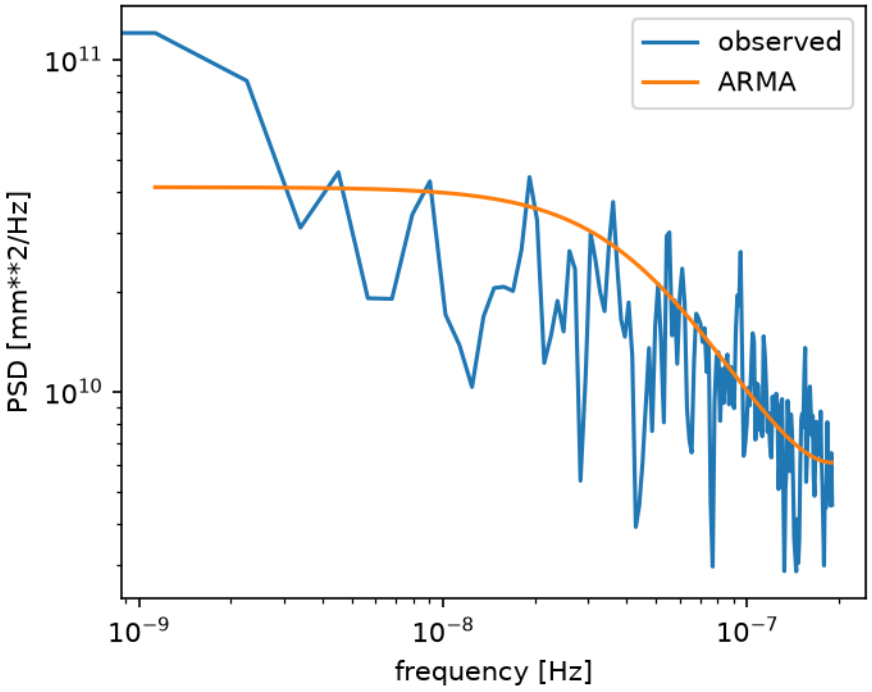


Figure 6: Power spectral density of tide gauge data at Cascais.

```
TimeNoiseStart          1000
NoiseModels             Flicker White
PhysicalUnit            mm
```

Some of these keywords are new. For example, `SimulationDir` specifies in which directory the created files should be stored. The keyword `SimulationLabel` specifies the base name of those files. The next keyword tells Hector how many simulation runs are required. The filenames will in this case be: `test_base_0.mom`, `test_base_1.mom`, ..., `test_base_9.mom`.

The keyword `NumberOfPoints` specifies the number of points in the the time series and the keyword `TimeNoiseStart` was already discussed above.

When `simulatenoise` is run, it will ask the user to manually enter the noise amplitudes for each noise model, in the order of `NoiseModels` (here a flicker noise amplitude of 10 mm/yr$^{0.25}$ and a white noise amplitude of 4 mm). The provided file `simulatenoise.inp` contains these two numbers, so the series can also be created non-interactively:

```
simulatenoise < simulatenoise.inp
```

To check that the noise properties are recovered, run

```
estimate_all_trends
```

which runs `removeoutliers` and `estimatetrend` (with `GGM White` noise) on every file in `obs_files/` and collects the results in `hector_estimatetrend.json`. Over the 10 series the median spectral index is close to $\kappa = -1$ and the median GGM and white noise amplitudes are close to 10 mm/yr$^{0.25}$ and 4 mm (in the run whose results are included: -0.92, 9.9 and 3.9). The plots of the series and of their power spectra are written to `data_figures/` and `psd_figures/`; add `-nograph` to skip them.

In some cases it is desirable to create the same set of synthetic time series each time `simulatenoise` is run. To achieve this, one can set the keyword `RepeatableNoise` to yes. It always uses the same random numbers, so two components simulated in separate runs (for example noise in one run and a seasonal signal in another) would be correlated. Use instead `RandomSeed` with a different integer for each run: every run is then reproducible and the runs are independent of each other.

4.4 Example 4: Post-Seismic Relaxation

After a large earthquake, the Earth's surface slowly returns to a new position. This post-seismic relaxation can be modelled with an exponential or logarithmic function, see Section 5. In the directory `ex4` we have created a synthetic time series with various relaxation signals. We *know* when the synthetic earthquakes occurred and therefore add offset epochs in the header of the observation file `TEST.mom` in the directory `./obs_files`. Furthermore, add information about the relaxation times (unit is days) for each event:

```
# sampling period 1.0
# offset  51994.0
# offset  53544.0
# offset  55044.0
# log  51994.0   10.0
```

```
# log  55044.0   10.0
# exp  53544.0  100.0
```

To force `estimatetrend` to model this signal, include the line `estimatepostseismic yes` in the control file:

```
DataFile              TEST.mom
DataDirectory         ./obs_files
OutputFile            ./mom_files/TEST.mom
periodicsignals       365.25 182.625
estimateoffsets       yes
estimatepostseismic   yes
useRMLE               yes

#--- Power-law noise approximated by GGM with fixed 1-phi
NoiseModels           GGM White
GGM_1mphi             6.9e-06

PhysicalUnit          mm
ScaleFactor           1.0
```

To include slow-slip events the procedure is similar. You need to include the line `estimates-lowslipevent yes` in the control file and add to the mom-file in the header `# tanh MMMM DD` where MMMM is the Modified Julian Date and DD the relaxation time in days, see Section 5.

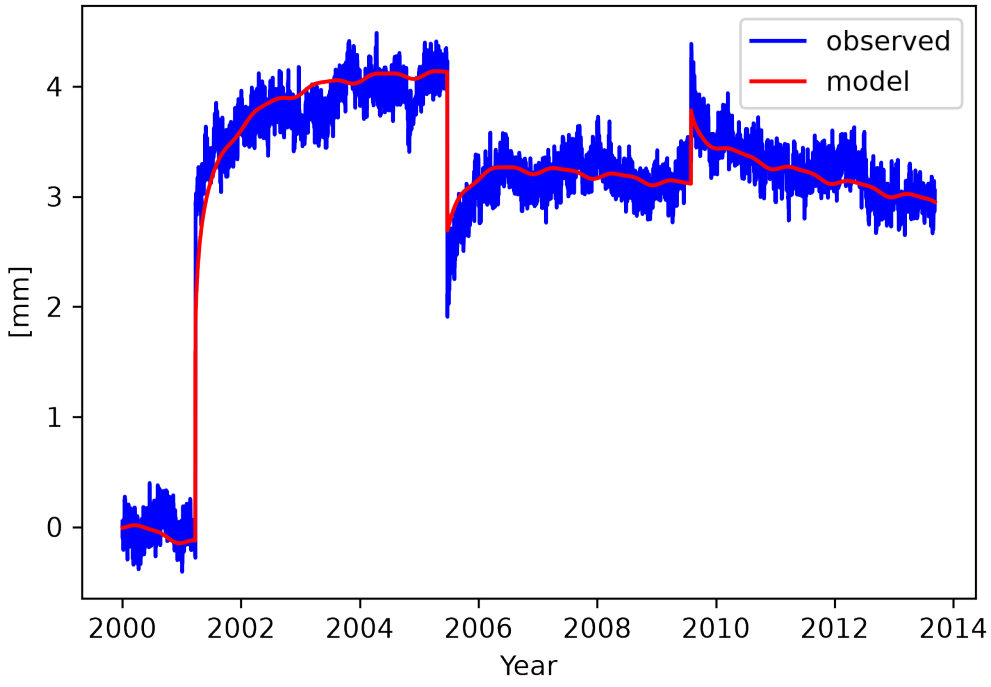


Figure 7: Observed data and fitted model with three post-seismic relaxations (two logarithmic and one exponential) from Example 4.

4.5 Example 5: Detecting Unknown Offset Epochs

Example 1 assumed that the offset epochs were already known and stored as `# offset` lines in the header of the data file in `obs_files/`. In practice, offset epochs are often unknown and must be detected automatically. This situation is modelled in `ex5`, which uses the same synthetic data as Example 1 but with all `# offset` header lines removed.

The data are stored in `raw_files/TEST.mom`:

```
# sampling period 1.0
 50084.0 -17.88951
 50085.0 -16.88599
...
```

There are no offset header lines. The file is placed in `raw_files/` rather than `obs_files/` precisely because the offset epochs have not yet been established.

The recommended workflow is:

1. Remove spike outliers from the raw data with `removeoutliers`.
2. Detect offset epochs with `findoffsets`.
3. Continue with the standard workflow using the annotated file written to `obs_files/`.

4.5.1 Step 1 — Remove Spike Outliers from the Raw Data

Before running `findoffsets`, isolated spike outliers must be removed. Unremoved spikes would be misidentified as offset candidates by the Generalised Likelihood Ratio (GLR) test.

When offset epochs are not yet known, **SpikeDetector mode** (`Spike_factor`) is the safer choice over the standard DataSnooping mode (`IQ_factor`). The spike detector works on consecutive first differences: a spike at epoch j creates two differences of opposite sign that both exceed `Spike_factor × MAD` of all differences, and the epoch is flagged. An offset *step* produces exactly one large difference (at the step epoch) with no sign reversal and is therefore never flagged — making the method immune to unmodelled offsets.

DataSnooping (`IQ_factor`) with `estimateoffsets yes` can be used instead, but it fits an OLS trajectory model whose design matrix must include offset columns. When offsets are unknown and large, the OLS fit is biased by the steps and can incorrectly flag hundreds of valid observations as outliers.

The control file reads the raw data and writes spike-filtered output to `stage_files/`:

```
DataFile              TEST.mom
DataDirectory         raw_files
OutputFile            stage_files/TEST.mom
periodicsignals       365.25 182.625
ScaleFactor           1.0
PhysicalUnit          mm
TimeUnit              days
Spike_factor          3
```

Running `removeoutliers` produces `stage_files/TEST.mom`. The JSON summary reports the detected outlier epochs:


```
{
    "N": 1000,
    "outliers": [
        50100.0,
        50242.0,
        50340.0,
        51008.0,
        51032.0,
        51033.0,
        51065.0,
        51077.0,
        50131.0,
        50213.0,
        50674.0,
        50779.0,
        51045.0
    ]
}
```


Thirteen points were removed. Six of them are the outliers that were added to this synthetic series (the same six that `IQ_factor` finds in Example 1); the other seven are valid observations where the noise happened to pass the spike test, two of them at the edge of a data gap where only one neighbour is available. Losing a few valid observations does not affect the offset detection. The filtered file retains 887 observations and can now be passed to `findoffsets`.

4.5.2 Step 2 — Detect Offset Epochs with `findoffsets`

The program `findoffsets` implements a GLR (Generalised Likelihood Ratio) forward search (Amiri-Simkooei et al. 2019): it scans every candidate epoch, computes how much the log-likelihood improves if an offset is placed there, and accepts the best candidate if the improvement exceeds `OffsetThreshold` (default 20). The search repeats until no further candidate clears the threshold.

The control file is:

```
DataFile            TEST.mom
DataDirectory       ./stage_files
OutputFile          ./obs_files/TEST.mom
PhysicalUnit        mm
ScaleFactor         1.0
periodicsignals     365.25 182.625
NoiseModels         FlickerGGM White
GGM_1mphi           6.9e-06
useRMLE             yes
```

Running `findoffsets` prints, after the usual header for each search cycle, the following summary lines:

```
0: best offset at  50784.00 (i=700) : dln=  160.744
1: best offset at  51034.00 (i=950) : dln=  172.049
2: best offset at  50284.00 (i=200) : dln=  104.895
3: best offset at  50335.00 (i=251) : dln=  133.194
4: best offset at  50807.00 (i=723) : dln=    8.800
---
Found 4 offset(s).
--> ./obs_files/TEST.mom
```

The fifth candidate (MJD 50807, $\Delta \ln L = 8.8$) falls below the threshold of 20 and is rejected. The true offset epochs were MJD 50284, 50334, 50784 and 51034; the detected epoch 50335 is off by one day because the observation at MJD 50334 happened to be among the random gaps in this dataset.

The output file `obs_files/TEST.mom` contains the detected epochs as `# offset` header lines:

```
# sampling period 1.000000000000
# offset 50784.0000
# offset 51034.0000
# offset 50284.0000
# offset 50335.0000
50084.000000    -17.889510
...
```

From this point the workflow is the same as in Example 1: `estimatetrend` reads `obs_files/TEST.mom` (the spikes have already been removed) and writes `mom_files/TEST.mom`, after which `estimate-spectrum` computes the periodogram of the residuals.

4.5.3 Why `FlickerGGM` and not `GGM`

The choice of noise model in `findoffsets.ctl` matters more than it might appear. If `NoiseModels GGM White` is used instead of `FlickerGGM White`, the spectral index d of the GGM model is a free parameter that the Nelder-Mead optimiser can adjust. When the data contain large unmodelled offsets, the optimiser finds a degenerate solution: by driving d towards 1 (random-walk behaviour), the GGM model absorbs the variance of the step discontinuities as if they were long-range correlated noise. The residual $\hat{\sigma}_\eta$ collapses to near zero, and every GLR test statistic becomes zero — so `findoffsets` reports no offsets at all.

The root cause is that a near-random-walk GGM process ($d \approx 1, 1-\phi = 6.9 \times 10^{-6}$) generates realisations with large low-frequency drifts that are visually indistinguishable from offset steps over a 3-year window. The profile likelihood is maximised by this degenerate solution rather than by correctly placing offsets in the design matrix.

`FlickerGGM` fixes $d = 0.5$ (flicker noise), which removes the degree of freedom that allows this degeneracy. In general, when running `findoffsets` on data with unknown and potentially large offsets, fixing the spectral index is safer than estimating it freely:

- Use `FlickerGGM` to fix $d = 0.5$ (appropriate for most GNSS data).
- Use `RandomWalkGGM` to fix $d = 1.0$.
- Use `GGM` with `kappa_fixed` to set a specific value of $\kappa = -2d$.

- Use `GGM` (free d) only when the data have been pre-cleaned of large offsets, so the noise model cannot exploit the degeneracy.

After the offset epochs have been identified and added to the header, re-running `estimatetrend` with the full `GGM White` model (free d) on the cleaned and annotated data is perfectly safe, as demonstrated in Example 1.

4.6 Example 6: Offset Detection on a Real GNSS Time Series

This example applies the offset-detection workflow of Example 5 to eight long European IGS stations: BOR1, GRAZ, MATE, METS, ONSA, VILL, WTZR, and ZIMM. Daily time series are downloaded from the Nevada Geodetic Laboratory (NGL, https://geodesy.unr.edu/) in tenv format and converted to NCF. The three components (E, N, U) are searched jointly with `find_all_offsets`. This example mirrors the real-data application in the companion paper.

The example directory is `ex6/`. The `.tenv` files and pre-computed `raw_files/`, `stage_files/`, and `obs_files/` are included so you can start at any step.

4.6.1 Step 1 — Convert tenv to NCF

```
convert_tenv2netcdf --start-mjd 51179
```

Converts all `.tenv` files in the current directory to multi-channel NCF files, one per station. The `--start-mjd 51179` flag discards epochs before 1 January 1999; pre-1999 data exhibit approximately ten times the spurious-detection density of later epochs due to a sparser GPS constellation and less mature processing strategies. Output: `raw_files/<station>.ncf` with channels `e`, `n`, `u`, `sigma_e/n/u` in mm.

Pre-computed `raw_files/` are included.

4.6.2 Step 2 — Spike detection

```
python3 run_stage.py
```

Runs `removeoutliers` on each raw NCF file using the SpikeDetector (`Spike_factor 3`). The spike detector flags an epoch only when both adjacent first differences exceed 3 × MAD of all differences *and* have opposite sign — a criterion immune to unmodelled offsets, since a genuine step produces one large first difference without a sign reversal. This makes the filter safe to apply before offset epochs are known. Output: `stage_files/<station>.ncf`.

Pre-computed `stage_files/` are included.

4.6.3 Step 3 — Offset detection

```
find_all_offsets
```

Runs the multivariate (E+N+U) forward offset search on every `stage_files/<station>.ncf`. At each iteration the three-component test statistic $T_3 = \sum_k 2\,\Delta \ln \bar{L}_k$ is compared to the $\chi^2(3, \alpha = 0.001)$ threshold of 16.27. Key defaults (override on the command line):

Flag	Default	Meaning
`-n`	`PLWN`	GGM + White noise (free spectral index)
`-t`	`16.27`	GLR threshold $\chi^2(3, \alpha=0.001)$
`--min_gap`	`30`	Minimum days between accepted offsets
`-maxoffsets`	`50`	Maximum offsets per station

With the default threshold many offsets are found that are not in the NGL catalogue; for real GNSS series a threshold between 30 and 70 (e.g. `find_all_offsets -t 30`) is advised, see the section on `find_all_offsets`.

Output: `obs_files/<station>.ncf` with detected offset epochs embedded, plus `find_all_offsets_ncf.json` with per-station results.

Pre-computed `obs_files/` are included.

4.6.4 Step 4 — Trend estimation

```
estimate_all_trends
```

Processes every NCF file in `obs_files/` using `removeoutliers.ctl` (Spike_factor 5, post-detection spike cleanup) followed by `estimatetrend.ctl` (GGM + White noise model, useRMLE yes). Results are written to `pre_files/` and `mom_files/`.

4.7 Example 7: Piecewise Linear (Multi-Trend) Estimation

This example demonstrates `estimatemultitrend` on a synthetic 9-year daily up-component time series with three distinct trend regimes. Flicker plus white noise is added with amplitudes representative of the ONSA up component from Example 6. The example directory is `ex7/`.

The synthetic signal has:

Segment	Epoch range	True slope
1	year 0–3 (MJD 51544–52639)	0 mm/yr
2	year 3–6 (MJD 52639–53734)	3 mm/yr
3	year 6–9 (MJD 53734–54829)	1 mm/yr

Annual amplitude 6.4 mm and semi-annual amplitude 1.4 mm are included (ONSA up values), and noise is GGM ($d \approx 0.5$, $\sigma = 5$ mm) plus white ($\sigma = 1$ mm).

4.7.1 Step 0 — Generate the synthetic series

```
python3 create_signal.py
```

This writes `obs_files/MULTITREND.mom`. The break epochs are embedded in the file header:

```
# sampling period 1.0
# break  52639.0
# break  53734.0
```

4.7.2 Step 1 — Remove spike outliers

```
removeoutliers
```

Uses `Spike_factor 3` in `removeoutliers.ctl`. Expected: about 25 spikes flagged. Because the data are synthetic and contain no injected outliers, this step is not strictly necessary here — it is included to mirror the standard workflow and because real data always requires it.

4.7.3 Step 2 — Estimate the piecewise linear trend

The `estimatetrend.ctl` for this example adds one keyword to the standard setup:

```
DataFile              MULTITREND.mom
DataDirectory         pre_files
OutputFile            mom_files/MULTITREND.mom
PhysicalUnit          mm
TimeUnit              days
ScaleFactor           1.0
periodicsignals       365.25 182.625
estimatemultitrend    yes
NoiseModels           GGM White
GGM_1mphi             6.9e-06
useRMLE               yes
JSON                  yes
```

Run:

```
estimatetrend -png
```

The `-png` flag saves a plot of the observations and fitted piecewise model to `data_figures/MULTITREND.png`; an example of such a plot is shown in Figure 8 (§5.2).

Expected output:

```
Piecewise trend segments:
  segment 1 (before 52639.0) :   0.169 mm/yr
  slope change at 52639.0  :  +3.317 +/- 0.846 mm/yr
  segment 2 (52639.0 - 53734.0) :   3.486 mm/yr
  slope change at 53734.0  :  -2.473 +/- 0.847 mm/yr
  segment 3 (53734.0 - end) :   1.013 mm/yr
```

The recovered slopes (0.17, 3.49, 1.01 mm/yr) agree well with the true values (0, 3, 1 mm/yr). The per-segment uncertainty of about ±0.85 mm/yr reflects the correlated nature of flicker noise: because the noise is correlated over timescales comparable to the segment length (3 yr), the effective number of independent observations per segment is much smaller than the nominal count.

4.7.4 Step 3 — Power spectral density (optional)

```
estimatespectrum -model
```

Computes the Welch periodogram of the residuals from `mom_files/MULTITREND.mom` and overlays the estimated GGM+White noise model. The residuals should follow the expected flicker-dominated PSD (slope ≈ −1 in log–log space at low frequencies).

4.8 Example 8: Toeplitz Factorisation — Levinson vs. Generalised Schur

This example is a pure Jupyter notebook that shows the mathematical heart of Hector's speed advantage.

Every GGM covariance matrix is a symmetric positive-definite Toeplitz matrix — constant along each diagonal and therefore fully described by a single vector of length n rather than n^2 entries. Hector exploits this structure to factorize the matrix without ever forming it explicitly. Two algorithms are compared:

Algorithm	Complexity	Hector class
Durbin–Levinson	O(n^2)	`Levinson`
Generalised Schur (GSA)	O($n \log^2 n$)	`Schur`

The notebook builds a GGM flicker-noise covariance vector of length 10 000, runs both algorithms, checks the result against the Gohberg–Semencul formula (C^{-1} reconstructed from `l1`, `l2`, δ), and produces timing curves over a range of series lengths. No data files are needed — everything is computed from the GGM parameters.

To run it, change to the `ex8/` directory and start Jupyter:

```
cd hector-examples/ex8
jupyter notebook
```

Then open `toeplitz_factorisation.ipynb`.

5 Model Specification

We assume that the observations are the sum of a deterministic model and stochastic noise. This deterministic model is (Bevis and Brown 2014):

$$
\begin{aligned}
\mathbf{x} = & \sum_{i=0}^{n_P} \mathbf{p}_i (t - t_R)^i + \sum_{i=1}^{n_J} \mathbf{b}_j H(t - t_j) \\
& + \sum_{i=1}^{n_F} \mathbf{s}_i \sin(\omega_i (t - t_0)) + \mathbf{c}_i \cos(\omega_i (t - t_0)) \\
& + \sum_{i=1}^{n_T} \mathbf{e}_i (1 - \exp(-(t - t_i)/T_i)) \\
& + \sum_{j=1}^{n_L} \mathbf{a}_j \log(1 + (t - t_j)/T_j) \\
& + \sum_{k=1}^{n_T} \frac{\mathbf{u}_k}{2} \left[\tanh\left((t - t_k)/T_k\right) - 1\right] \\
& + \sum_{l=1}^{n_l} a_l \times f_l(t)
\end{aligned}
$$

where $\mathbf{p}_i$ are the coefficients of a n_P degree polynomial. By default $n_P = 1$: a linear trend. t_R is normally the time that is exactly in the middle of start and end point of the time series. Following convention, we give the rate in unit per year, where a year is defined as 365.25 days. Note that acceleration is normally defined to be twice the quadratic term. Thus, we fit $a(t - t_R)^2$ where a is estimated.

$H(t)$ is the Heaviside step function and used to model offsets with amplitudes $\mathbf{b}_j$.

An annual and semiannual signal are commonly included in the model. In the equation their angular velocities are represented by ω_i. These are most conveniently specified using the `periodicsignals` keyword, followed by a list of their periods in days. For example, `periodicsignals 365.25 182.625` requests annual and semi-annual signals. The reference epoch t_0 of these signals is 1 January 2000 (MJD 51544), as in Hector 2.2. The estimated phase therefore refers to the calendar year: the same annual signal gives the same phase, whatever the first epoch of the time series. For files in the generic `gen` format, which have no MJD time axis, t_0 is the first epoch of the series. The old keywords `seasonalsignal` and `halfseasonalsignal` still work for backward compatibility.

It is often desirable to speak about the amplitude of the annual and semi-annual period. To facilitate this, Hector shows also this parameter. It has been computed by first assuming a mean uncertainty for s_i and c_i which we call σ. In this case the amplitude follows a Rice distribution with a mean of $\sigma\sqrt{\pi/2}\, L_{1/2}(-\nu^2/(2\sigma^2))$ where $\nu = \sqrt{c_i^2 + s_i^2}$ and $L_{1/2}(x) = {}_1F_1(-1/2, 1, x)$.

The phase is $\phi_i = \arctan(s_i/c_i)$ (computed with `atan2`), so that the signal is $A_i \cos(\omega_i(t-t_0) - \phi_i)$; for the annual signal the maximum is reached $\phi_i/360 \times 365.25$ days after 1 January (add 360° to a negative phase). The probability distribution function of the phase is more complicated. Therefore, simply a Monte Carlo simulation is performed using 10000 samples of c_i and s_i, each with its own uncertainty, to get the mean and standard deviation of the phase. As in Hector 2.2, the random generator starts from a fixed seed, so the same data always give the same phase.

5.1 Post-Seismic Deformation

Since version 1.6 one can also include post-seismic deformation in the model by extending the header of the data file by adding lines that contain the time when the relaxation starts (in MJD) and the relaxation period in days. An example is:

```
# log  51994.0   10.0
# log  55044.0   10.0
# exp  53544.0  100.0
```

One can use an exponential or logarithmic function, see the equation above. To estimate these post-seismic relaxations, one must set the keyword `estimatepostseismic` to yes in `estimatetrend.ctl` (the default is no). The relaxation terms are read from the header of `.mom` files only; the post-seismic variables of `.ncf` files are not used yet.

In most cases, one should estimate an offset (due to the earthquake) and the post-seismic deformation. Therefore, one normally has both in the header. An example is:

```
# offset 53544.0
# exp    53544.0  100.0
```

Since version 1.7 one can in addition model slow slip events using a hyperbolic tangent function (Larson et al. 2004). The two parameters t_k and T_k, representing the date of the mid-point of the slow slip event and its width respectively, must also be set in the header as follows:

```
# tanh 51994.0   10.0
```

One must add the keyword `estimateslowslipevent yes` in `estimatetrend.ctl` (the default is no). Note that one can consider this function as some kind of offset. Thus, no additional offset should be estimated. Of course this is just a general indication.

5.2 Multi-Trend

So far we have assumed that there is only one linear trend in the time series. However, it has become clear that some GNSS time series show a different linear trend after a major earthquake. In those cases a piecewise linear trend is more appropriate. To estimate the linear trend in each segment, the piecewise linear function is written as a sum of special functions that grow linearly in a particular segment and afterwards remain constant.

The time of a break in the linear trend is given in the header of the time series (`.mom` files only), for example:

```
# break  51994.0
```

To estimate multi-trends, one must set the keyword `estimatemultitrend` to `yes` in `estimatetrend.ctl`. Hector reads the `# break` epochs from the data file header and adds a ramp function $\max(0, t - t_{\text{break}})$ for each break epoch to the design matrix, estimating the slope change at each break. The ramps are added on top of the polynomial set by `DegreePolynomial`. The piecewise segment slopes and their uncertainties are reported in the output and written to `estimatetrend.json` under the keys `trend_segments` and `break_trend_changes`. A worked tutorial example is given in §4.7 (Example 7). The figure below shows a typical multi-trend result.

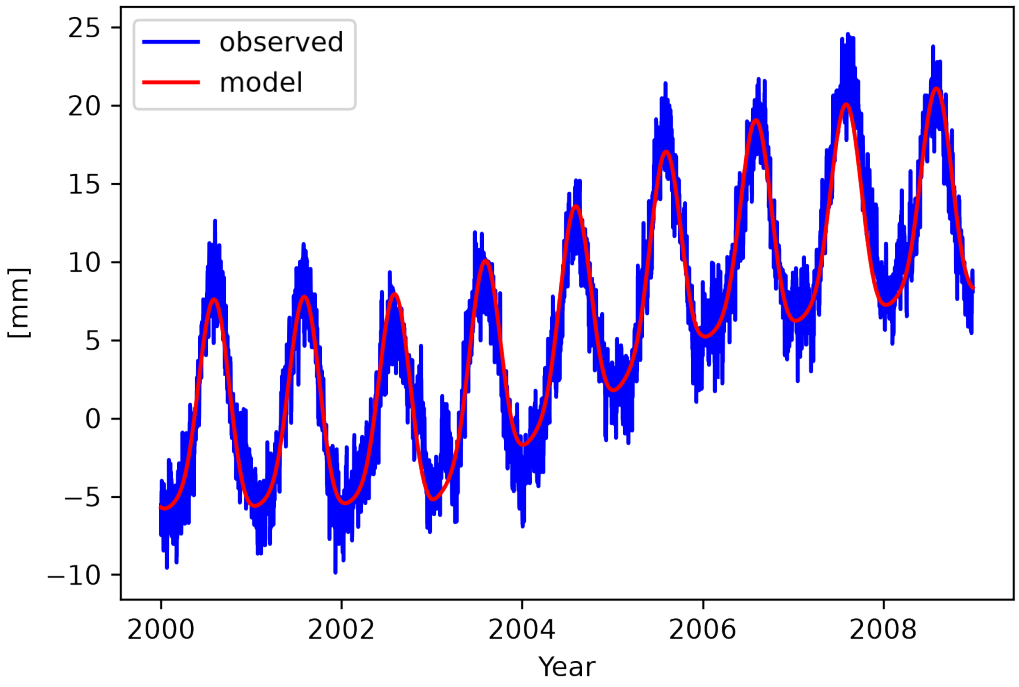


Figure 8: Example of multi-trend estimation.

5.3 Geophysical Signal

The last term in the trajectory model equation represents a scale factor a_l (also known as regression coefficient) times a function f_l. This can represent a geophysical signal. In Example 2, we showed how surface pressure can be included in the analysis and how we obtained a scale value close to the inverted barometer value. Another example is the local pressure admittance in tidal gravity time series (Dam and Francis 1998). By specifying in the control file:

```
estimatemultivariate        yes
MultiVariateFile            XXXX.mom
```

this can be achieved.

Mom-file format (`MultiVariateFile` points to a `.mom` file): File `XXXX.mom` is in the standard mom format — MJD in the first column, geophysical signal values in the second column. Exactly one signal is read. The file must start at or before the first observation epoch and end at or after the last; linear interpolation is used to evaluate the signal at the observation epochs, so the signal and observation grids do not need to match. The regression coefficient in the output is labelled with the filename stem (e.g. `hadslp2_cascais`).

NetCDF format (`MultiVariateFile` points to a `.ncf` file): When the signal file is in netCDF format, the keyword `MultiVariateSignals` followed by one or more channel names selects which channels to include — each becomes a separate column in the design matrix with its own regression coefficient. The same spanning and interpolation rules apply. Example:

```
estimatemultivariate        yes
```

```
MultiVariateFile          era5_grid.ncf
MultiVariateDir           ./signals
MultiVariateSignals       pressure IVT
```

The optional `MultiVariateDir` keyword sets the directory where the signal file is located; it defaults to `DataDirectory` if not given.

6 Acceptable Data Format

Hector reads three formats: `mom` (the default), `gen` (a generic ASCII format with the same layout whose time axis need not be MJD) and `ncf` (netCDF4). The format is set with the keyword `TS_format`; files whose name ends in `.ncf` or `.nc` are recognised automatically. In the ASCII formats the time should always be increasing and the time step constant (missing epochs are allowed), and the number of digits and the spacing between the columns are free.

For the mom and gen formats the sampling period, in days, must be given in the header:

```
# sampling period 1.0
```

A file without this line is refused with an error message. For `.ncf` files the global attribute `sampling_period` is used, or else the first time step.

6.1 mom-format

This format expects 2 or 3 columns. The first column contains the time in MJD, the second the Observations. The third column is optional and should contain the estimated Model. Missing data are allowed.

6.2 tenv-format (NGL)

The Nevada Geodetic Laboratory (NGL) distributes daily GNSS positions in two ASCII formats: `tenv` (17 columns) and `tenv3` (23 columns). Hector does not read these files directly. Use `convert_tenv2netcdf` (§9.8) to convert one or more stations to NCF format, then pass the resulting `.ncf` files to `removeoutliers`, `estimatetrend`, and `findoffsets`.

```
# convert all *.tenv3 / *.tenv files in the current directory
convert_tenv2netcdf

# convert a single station
convert_tenv2netcdf -s KOSG
```

NGL files can be downloaded from https://geodesy.unr.edu/.

6.3 rlrdata-format

PSMSL distributes monthly sea-level data in the RLR format (http://www.psmsl.org/). Hector v3.0 cannot read this format directly. Use `convert_rlrdata2mom` (§9.7) to convert the file to mom format first, then process the resulting `.mom` file with `estimatetrend` as usual (see Example 2).

For reference, the PSMSL epoch convention used internally by `convert_rlrdata2mom` is:

$$MJD = 30.4375\,(12(\text{year} - 1859) + (\text{month} - 1)) + 59$$

6.4 ncf-format

The `.ncf` format is a netCDF4 file designed for multi-channel time series — the primary use case being three-component GNSS displacement (east, north, up) where all components share the same time axis, discontinuity events, and post-seismic relaxation metadata. In this version Hector uses the offset epochs of a `.ncf` file; the post-seismic relaxation variables are stored but not yet used in the trajectory model. The format also suits any other application that requires several named channels in a single self-describing file.

Hector recognises both `.ncf` and the more common `.nc` extension as netCDF4 files; both are treated identically.

Unlike the mom format, which stores a single channel per file as plain text, a `.ncf` / `.nc` file can hold any number of named channels and carries all event metadata as structured arrays inside the file itself.

6.4.1 Standard channel names

For three-component GNSS data Hector uses the following lowercase names (as written by `convert_tenv2netcdf`):

Channel	Meaning
`e`	East displacement
`n`	North displacement
`u`	Vertical (up) displacement

The batch scripts `estimate_all_trends` and `find_all_offsets`, and `removeoutliers` without `ColumnName`, process exactly these three channels. Custom names are allowed for other applications (e.g. `pressure`, `temperature`, `acc_x`); select them with the keyword `ColumnName` and run the single-station programs.

Names ending in `_model` or `_residual` are reserved for Hector's own output — do not use these suffixes for raw observation channels.

6.4.2 File structure

```
Dimensions:
    time      = UNLIMITED          # number of observations
    n_offset  = <N>                # number of discontinuity events (absent if none)
```

Time variable

```
double time(time)
    time:units     = "days since 1858-11-17 00:00:00"
    time:long_name = "Modified Julian Date"
```

Values are plain MJD. The epoch `1858-11-17` is the standard MJD origin, so no conversion is needed when working with MJD values from other sources. A `double` (float64) has sufficient precision for any practically relevant sampling rate: at MJD ≈ 60000 the resolution is better than 1 microsecond, well below the 5 ms step of a 200 Hz IMU.

Channel variables

```
float <name>(time)              # raw observations, e.g. e, n, u
float <name>_model(time)        # trajectory model written by estimatetrend
float <name>_residual(time)     # observations minus model
```

Missing values use `NaN` as the fill value (`_FillValue = NaN`). There are two sources of `NaN` in a channel:

- **Data gaps**: epochs that are simply absent from the record. Rather than storing a placeholder, such gaps are implicit from the time axis; however if a specific epoch is present in `time` but has no valid observation, store `NaN` for that sample.
- **Outliers**: `removeoutliers` detects outliers (data snooping or spike detection) and writes the channel to `OutputFile` with the offending samples set to `NaN`. Subsequent tools — `estimatetrend`, `estimatespectrum` — treat any `NaN` value as a missing observation and exclude it from the analysis.

When preparing a `.ncf` file for Hector, use `NaN` for any sample that should be excluded from the analysis (sensor malfunction, known bad epoch, etc.). The `_model` and `_residual` channels are written by Hector and inherit `NaN` at all positions where the observation was missing or removed.

Offset event variables (on the `n_offset` dimension)

```
double offset_time(n_offset)
    units = "days since 1858-11-17 00:00:00"

int32  offset_type(n_offset)
    flag_values   = "0, 1, 2"
    flag_meanings = "unknown equipment_change earthquake"

float  offset_amp_<name>(n_offset)      # amplitude per channel; NaN if not applicable
float  psr_amp_<name>(n_offset)           # post-seismic amplitude; NaN if none
float  psr_tau_<name>(n_offset)           # relaxation time in days; NaN if none
int8   psr_log_or_exp_<name>(n_offset)    # 0=logarithmic 1=exponential; -1 if none
    flag_values   = "0, 1"
    flag_meanings = "logarithmic exponential"
```

All discontinuity events across the file share a single `n_offset` dimension. Per-channel amplitudes and post-seismic relaxation (PSR) parameters are stored in separate variables named `offset_amp_<channel>`, `psr_amp_<channel>`, etc. A `NaN` value means that event does not affect that channel; if a channel has no `offset_amp_<channel>` variable, all events apply to it. Offsets found by `findoffsets` in a `.ncf` file are added as events of type 0 for the analysed channel only, with an amplitude entry of 0 (the amplitude is not stored).

Global attributes

```
sampling_period : <float>    # sampling interval in days (e.g. 1.0 for daily,
```

```
                              # 5.787e-8 for 200 Hz)
station          : <string>   # optional station identifier
component        : <string>   # optional component label
```

6.4.3 Creating .ncf files

The `ncfgen` tool creates a `.ncf` file from an ASCII data file and a JSON metadata file:

```
ncfgen -m meta.json -d data.txt -o output.ncf
```

`data.txt` is a whitespace-separated table with MJD in the first column followed by one column per channel. Lines starting with `#` are skipped. `meta.json` specifies channel names, global attributes, and (optionally) the offset event metadata. Use `null` in JSON arrays to represent a `NaN` (not-applicable) value:

```
{
  "sampling_period": 1.0,
  "station": "REYK",
  "component": "u",
  "channels": ["e", "n", "u"],
  "offsets": {
    "offset_time":  [59000.0, 59500.0],
    "offset_type":  [1, 2],
    "offset_amp_e":         [null, 3.2],
    "offset_amp_n":         [1.1,  null],
    "psr_amp_u":            [null, 1.5],
    "psr_tau_u":            [null, 30.0],
    "psr_log_or_exp_u":     [null, 0]
  }
}
```

6.4.4 Inspecting .ncf files

`ncfdump` prints a human-readable summary of a `.ncf` file and can export one channel to ASCII:

```
ncfdump -i file.ncf                     # summary: attributes, channels, offset table
ncfdump -i file.ncf -c e -o data.txt    # export channel e (+ _model, _residual if present)
```

The exported text file has columns `MJD <channel> [<channel>_model <channel>_residual]` and omits rows where the observation is `NaN`.

6.4.5 Using .ncf files in Hector

A file name ending in `.ncf` or `.nc` selects the ncf format automatically; it can also be set explicitly with `TS_format ncf`. The keyword `ColumnName` selects which channel to analyse. The time unit is always days (MJD), identical to the mom format.

```
DataFile            REYK.ncf
DataDirectory       ./obs_files
OutputFile          mom_files/REYK.ncf
```

```
TS_format           ncf
ColumnName          n
```

`estimatetrend` writes the `<channel>_model` and `<channel>_residual` variables to `OutputFile`: if that file does not exist yet it is first created as a copy of the input file, otherwise the variables are added to it (no other data in the file is touched). `removeoutliers` writes the cleaned channel to its `OutputFile` in the same way, with the detected outliers set to `NaN` so that `estimatetrend` treats them as missing data. Without `ColumnName`, `removeoutliers` cleans the channels `e`, `n` and `u`.

6.4.6 Batch processing with estimate_all_trends

`estimate_all_trends` handles `.ncf` files automatically. Place the `.ncf` file(s) in `obs_files/` and run:

```
estimate_all_trends -n GGMWN -useRMLE
```

For every `.ncf` file in `obs_files/`, `removeoutliers` is run once on the channels `e`, `n` and `u` (spike detection, `Spike_factor 5`), writing `pre_files/<station>.ncf`. Then `estimatetrend` is run for each of the three channels, writing the model and residuals to `mom_files/<station>.ncf`, and, unless `-nograph` is given, `estimatespectrum -model` computes the spectrum of the `<channel>_residual` channel. The summary `hector_estimatetrend.json` uses a nested structure:

```
{
  "REYK": {
    "e": { "trend": 0.8, "trend_sigma": 0.1, ... },
    "n": { "trend": 1.5, "trend_sigma": 0.1, ... },
    "u": { "trend": 2.1, "trend_sigma": 0.3, ... }
  }
}
```

6.5 Sampling rates and time-axis precision

Hector has no hard-coded limit on the sampling frequency, but the time axis is stored as double-precision (64-bit) floating-point numbers, and at high sampling rates the *absolute value* of the epochs decides how precisely a sample can be placed on the regular grid. A double carries about 16 significant digits, so the resolution near an epoch t is roughly $t \times 10^{-16}$. On an absolute MJD axis ($t \approx 60000$) that resolution is 7×10^{-12} days $\approx 0.6\,\mu$s.

To decide which samples are missing, Hector requires every epoch to lie within 1% of a sampling period of the regular grid. On an absolute MJD axis this is satisfied up to roughly **30 kHz**; beyond that, epochs can no longer be written precisely enough in MJD, and Hector stops with a clear error message rather than guessing which samples are gaps. (Above $□$800 kHz consecutive MJD epochs are not even representable as distinct doubles.)

For higher rates the solution is a **relative time axis**: let the first epoch be 0.0 instead of an absolute MJD. Near $t = 0$ the double-precision resolution is essentially unlimited, and a series sampled at, say, 1.23 MHz is then processed without difficulty — the mom format does not require the first column to be a true MJD, only that the header sampling period uses the same unit (days). Offsets and other header epochs must of course use the same relative convention, and plots will not show calendar dates. As a rule of thumb, on a relative axis the grid stays exact as long as the total series

duration satisfies $T \times 10^{-16} < 0.01 \times \Delta t$, which for a week-long record still allows several hundred kHz, and for an hour-long record several GHz.

Three practical notes for high-rate data:

- The `# sampling period` header is written in scientific notation when the period is below 10^{-4} days, so that all its significant digits survive a write/read round trip; older files whose header carries fewer digits are still read correctly, because Hector re-derives the sampling period from the file's own time span.
- Gaps are classified by counting whole sampling periods between *consecutive* epochs (never by stepping through the file one sample at a time), so rounding errors do not accumulate over long gaps — a series with millions of samples and long outages is classified exactly.
- The practical ceiling is usually the series **length**, not the rate: maximum-likelihood estimation is comfortable up to a few million observations. For longer high-rate records, decimate (with a proper anti-aliasing filter) or analyse segments.

7 Implemented Noise Models

Hector can use various types of noise models and, in addition, accepts combinations of them, with GGM plus white noise being the most popular and recommended choice for GNSS time series. The combined covariance matrix for n noise models is:

$$\mathbf{C} = \sigma^2 \left(f_1 \mathbf{E}_1 + f_2 \mathbf{E}_2 + ... + f_n \mathbf{E}_n \right)$$

where $\mathbf{E}_i$ is the unit covariance matrix of noise model i (normalised so that its first element equals 1) and f_i is the *fraction* of noise model i. The fractions must satisfy $f_i \geq 0$ and $\sum_{i=1}^{n} f_i = 1$. The overall driving noise amplitude σ appears as a separate factor.

For the common two-model combination (e.g. GGM plus white noise), (Williams 2008) introduced the parameterisation $f_1 = \cos^2 \phi$, $f_2 = \sin^2 \phi$ with $\phi \in [0, \pi/2]$, which automatically satisfies the sum-to-one constraint. Hector v3.0 replaces this with **direct fraction parameters**. For n noise models, $n - 1$ fractions $f_1, \ldots, f_{n-1}$ are the free optimisation parameters; the last fraction is computed as:

$$f_n = 1 - \sum_{i=1}^{n-1} f_i$$

Each fraction, and the sum of the free fractions, is constrained to $[0, 1]$ by penalty terms added to the log-likelihood during Nelder-Mead optimisation. This avoids the singularities that occur in the angle-based parameterisation at $\phi = 0$ and $\phi = \pi/2$ and gives the optimizer a simpler, linear parameter space. The starting value for each fraction is $1/n$ (equal weight).

As was noted in Section 1, only stationary noise is accepted. This creates a Toeplitz covariance matrix and only the first column of the covariance matrix needs to be stored. This column vector will be denoted by γ.

7.1 White Noise

For white noise the covariance matrix is just the unit matrix. The first column of the covariance matrix **C**, with $\sigma = 1$, is:

$$\begin{aligned} \gamma_i &= 1 \quad \text{for } i = 0 \\ &= 0 \qquad i \neq 0 \end{aligned}$$

Its one-sided power spectral density is:

$$S(f) = 2\frac{1}{f_s}$$

where f_s is the sampling frequency in Hz. If you integrate this from zero frequency to the Nyquist frequency, you get the variance that is observed in the time series, as it should be.

7.2 Power-Law Noise

For power-law noise the first column of the covariance matrix is:

$$\gamma_i = \frac{\Gamma(d+i)\Gamma(1-2d)}{\Gamma(d)\Gamma(1+i-d)\Gamma(1-d)}$$

Its one-sided power spectral density, with $\sigma = 1$, is:

$$S(f) = 2\frac{1}{f_s}\frac{1}{(2\sin(\pi f/f_s))^{2d}}$$

This expression for γ_i requires stationary noise, $d < 0.5$: the `Powerlaw` model keeps $\kappa = -2d$ between -0.998 and 1.998. Flicker noise ($\kappa = -1$) and steeper spectra are therefore modelled with the GGM model (see below).

7.3 ARFIMA and ARMA

The definition of the ARFIMA noise model is (Sowell 1992):

$$\Phi(L)(1-L)^d z_t = \Theta(L)\epsilon_t$$

L is the backshift operator ($Lx_i = x_{i-1}$), z_t is the residual at time t (observation minus modelled signal) and ϵ is a white noise signal. In other words, this equation says that the residuals in the observations can be produced by applying some transformations on a white noise process. The operators Φ and Θ are defined as (Hosking 1981):

$$\Phi(L) = 1 - \phi_1 L - \phi_2 L^2 - ... - \phi_p L^p$$
$$\Theta(L) = 1 + \theta_1 L + \theta_2 L^2 + ... + \theta_q L^q$$

This definition is implemented in Hector but note that the definition of the signs before the ϕ coefficients in Φ are positive in (Sowell 1992). To complicate matters further, the coefficients of Θ are negative in the formulae of (Zinde-Wash1988).

The value of the integers p and q are set by the keywords `AR_p` and `MA_q` respectively in `estimatetrend.ctl`; at least one of them must be larger than zero (for pure power-law noise use `Powerlaw` or `GGM`). As in Hector 2.2, the search starts with all coefficients equal to 0.1; during the search the AR roots are kept inside the unit circle (stationary noise) and d between -0.999 and 0.499. We have implemented the method of (Doornik and Ooms 2003) to compute the first column of the covariance matrix of `ARFIMA`, for which d is always estimated. For the special case $d = 0$, we use the equations of (Zinde-Wash1988); this model is selected with the name `ARMA` after the keyword `NoiseModels`. For sea level research the first order auto-regressive noise model is a popular choice: ARMA(1,0). For pure ARMA noise models faster Maximum Likelihood Methods exist, see for example (Brockwell and Davis 2002), but Hector will give the same result. To specify AR(1) in `estimatetrend.ctl` one must write:

```
NoiseModels          ARMA
AR_p                 1
MA_q                 0
```

The same AR(1) model is also available on its own as `NoiseModels AR1`. Its coefficient ϕ is estimated between 0 and 0.99999, or held fixed with the keyword `phi_ar1_fixed`.

7.4 Generalised Gauss-Markov Noise Model

(Langbein 2004) took the first order Gauss Markov noise model depending on the parameter ϕ and modified with an additional parameter, d, to create power-law noise with a slope of $2d$ in the power density spectrum which flattens to white noise at the very low and very high frequencies. The analytical expression for the autocovariance vector (with $\sigma = 1$) for this noise model is:

$$\gamma_i = \frac{\Gamma(d+i)(1-\phi)^i}{\Gamma(d)\Gamma(1+i)} \, {}_2F_1(d, d+i; 1+i; (1-\phi)^2)$$

This noise model can be used using the name `GGM` after the keyword `NoiseModels` in `estimatetrend.ctl`. (Bos et al. 2014) provide some additional formulae.

The $1 - \phi$ parameter can be held fixed a priori by adding to the control-file:

```
GGM_1mphi            XXXX
```

where XXXX is the value you want to give this parameter. If it is small enough, 6.9e-06 is a good value, then GGM approximates a pure power-law model, see also next sub-section.

> **Recommended default for GNSS daily data (v3.0):** Use `NoiseModels GGM White` together with `GGM_1mphi 6.9e-06`. This combination closely approximates power-law

plus white noise while remaining strictly stationary and numerically well-behaved. The $1-\phi = 6.9 \times 10^{-6}$ value places the low-frequency corner well below the Nyquist frequency of a 30-year daily series, so GGM behaves like pure power-law noise across the entire observable frequency range.

GGM makes use of the hypergeometric ${}_2F_1$ function, and some care is required with its parameters. In exact arithmetic the GGM covariance matrix is positive definite for *every* $d > 0$ as long as $1-\phi > 0$, so there is no fundamental restriction. The limit is purely numerical: the covariance is built from a double-precision backward recursion of ${}_2F_1$, and for a large d combined with a very small $1-\phi$ (a steep spectrum with a very low corner frequency) that recursion loses accuracy and the resulting Toeplitz matrix becomes indefinite. This is a loss of positive-definiteness, not an overflow of ${}_2F_1$ (which stays far below the double-precision range for any realistic parameters). Empirically the cliff sits close to $\log_{10}(1-\phi) = 9d - 19.6$ and is essentially independent of the series length. Hector therefore restricts the estimate to $\kappa \geq -3$ ($d \leq 1.5$) and, for $d > 1$, to $\log_{10}(1-\phi) \geq 9d - 18.5$, which keeps a comfortable margin from the cliff (see the figure below). If an estimate is pushed onto this boundary, Hector clamps it there and prints a warning; to explore a steeper index, increase `GGM_1mphi`.

A second, independent numerical subtlety concerns the two *seed values* of that backward recursion, ${}_2F_1$ evaluated at the longest lags. Their argument is $z = (1-\phi)^2$, and the `mpmath` library evaluates them with a direct series only for $z \leq 0.8$, i.e. $1-\phi \geq 0.106$; below that it switches to a $z \to 1-z$ transformation whose two terms cancel each other more and more strongly as $m\,(1-z) \approx 2m(1-\phi)$ grows. For a long series with a *moderate* $1-\phi$ (roughly $50/m < 1-\phi < 0.106$, the orange strip in the figure) each evaluation then takes seconds, hangs, or aborts — and because the Nelder-Mead search starts at $1-\phi = 0.1$, exactly inside this strip, estimating GGM with a free $1-\phi$ on a long series used to crash there (versions up to 3.1.3). Since version 3.1.4 Hector computes the seeds in this strip — and everywhere above it, the whole hatched region — with the plain Gauss series, whose terms are all positive (no cancellation) and which needs only about $39/(2(1-\phi))$ terms — cheap exactly where `mpmath` is fragile (above the strip, for $1-\phi \geq 0.106$, `mpmath` would be fine as well; the series is simply the same algorithm without the overhead). Conversely, for a very small $1-\phi$ (below the band, e.g. the recommended 6.9×10^{-6}) the `mpmath` transformation converges instantly while the plain series would need millions of terms, so there `mpmath` is kept. The two methods agree to $\sim 10^{-14}$ where both work, and the choice is made automatically; no user action is required.

7.5 Flicker Noise and Random Walk Noise

Flicker noise and Random Walk noise are simply two types of power-law noise where the spectral index d has the fixed value of 0.5 and 1.0 respectively. However, as was noted in the introduction, Hector can only deal with stationary noise because that results in Toeplitz covariance matrices that allow fast inversion techniques. This can be achieved by using the Generalised Gauss Markov noise model with a small value for the $1-\phi$ parameter. Example:

```
NoiseModels         FlickerGGM
GGM_1mphi           6.9e-06
```

`FlickerGGM` and `RandomWalkGGM` fix d at 0.5 and 1.0. Without `GGM_1mphi`, the parameter $1-\phi$ is estimated (between 10^{-6} and 0.999); the output marks with "(fixed)" which of d and $1-\phi$ is held fixed. Note that the names `Flicker` and `RandomWalk` are only known to `simulatenoise`.

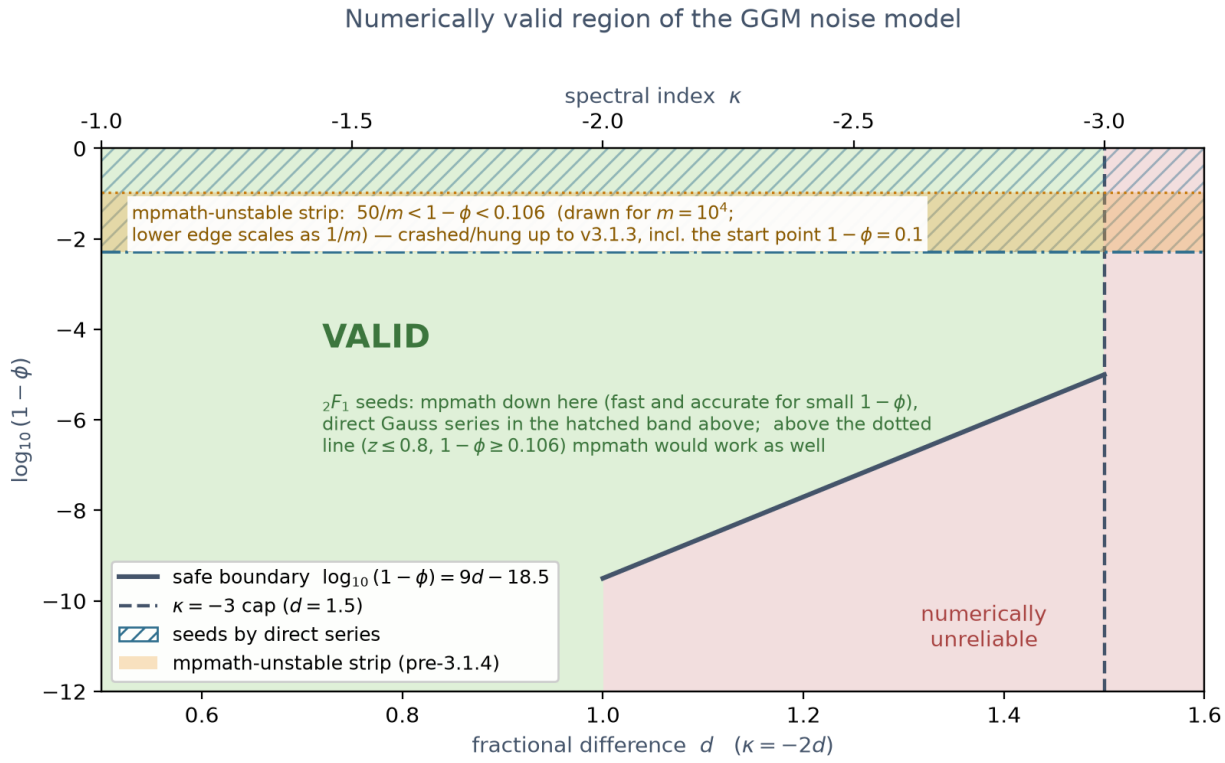


Figure 9: Numerically valid region of the GGM noise model. The covariance is positive definite everywhere in exact arithmetic; the shaded boundary marks where the double-precision ${}_2F_1$ recursion loses accuracy (large d with a very small $1-\phi$). Hector keeps a ~1 decade margin from the empirical cliff. In the hatched region at the top the two ${}_2F_1$ seed values are computed with a direct Gauss series; `mpmath` is used below it. Within the hatch, `mpmath` itself is actually unstable only in the orange strip $50/m < 1-\phi < 0.106$ (its lower edge scales as $1/m$; drawn for $m = 10^4$) — above the strip it would work too, the series is simply cheaper.

The value of 6.9×10^{-6} looks strange to many people. It is simply the smallest value that did not give numerical problems in previous versions of Hector. Since version 3.0 the Gaussian hypergeometric function is evaluated with a stable subroutine (`mpmath`, complemented since 3.1.4 by a direct Gauss series where `mpmath` is fragile, see §7.4) and 6.9×10^{-6} remains a good value for most cases. The valid parameter region for the underlying GGM model is shown in the figure in §7.4.

7.6 Comparing Noise Amplitudes Between Stations: the Modified Standard Deviation

Hector reports power-law and GGM noise amplitudes in the customary scaling of (Williams 2003), with units of mm/yr$^{-\kappa/4}$. Because the unit depends on the estimated spectral index, amplitudes of two stations with different $\hat{\kappa}$ are expressed in *different units* and cannot be compared or differenced directly. To overcome this, (Gobron et al. 2021) introduced the *modified standard deviation* $\hat{\sigma}'_{pl}$: the expected sample standard deviation (about the sample mean) of a realization of the estimated noise process over a fixed reference span, expressed in the physical unit (e.g. mm). This number has the same unit for every station and every spectral index, which makes it the right quantity for cross-station comparisons and maps.

Since version 3.1.5 Hector prints this value for the power-law and GGM noise models (including `FlickerGGM` and `RandomWalkGGM`) as an extra output line,

```
sigma     =  4.2402 mm/yr^0.31
mod. std  =  1.7207 mm (over 8 yr)
```

and stores it in the JSON output as `modified_std` (with `reference_span`). It is computed from the estimated model's own covariance: with t the first row of the unit-driving Toeplitz covariance evaluated at m' samples,

$$\hat{\sigma}' = \hat{\sigma}\sqrt{\; t_0 \;-\; \frac{1}{m'^2}\Big(m' t_0 + 2\sum_{k=1}^{m'-1}(m'-k)\,t_k\Big)\;},$$

which equals $\sqrt{\mathrm{E}\{\text{sample variance}\}}$ of an m'-long realization, Eq. (5) of (Gobron et al. 2021). The reference span is 8 years by default (the median span in that study) and can be changed with the control-file keyword `ReferenceSpan` (in years). Because processes with $\kappa \le -1$ are non-stationary — their sample scatter grows with the observed span — the reference span must be kept the same for all stations that are being compared; do not compare `modified_std` values obtained with different `ReferenceSpan` settings.

One technical note: (Gobron et al. 2021) define $\hat{\sigma}'_{pl}$ with the finite-past pure power-law covariance of (Williams 2003), whereas Hector evaluates the same statistic with the stationary covariance of the model it actually estimated (GGM with a small $1-\phi$ in the recommended setup). Since the statistic depends only on the correlation at lags shorter than the reference span, the two conventions agree closely as long as the GGM memory $1/(1-\phi)$ is much longer than the reference span: over the spectral-index range $-1.6 \le \kappa \le -0.4$ the difference is between 0.02% and 1.2% (largest for the steepest indices), negligible compared to the scatter of station-to-station comparisons.

7.7 Matern Noise Model

The Matern noise model is well explained by (Lilly et al. 2016). Its power spectrum is $S(\omega) \propto 1/(\omega^2 + \lambda^2)^\alpha$: a power law $\omega^{-2\alpha}$ at high frequencies that flattens below the corner frequency λ (in radians per sampling interval). It is the continuous-time counterpart of the Generalised Gauss Markov model: for frequencies well below Nyquist the GGM spectrum has the same form with $\alpha = d$ and $\lambda = 1 - \phi$. Hector reports the slope as $\kappa = -2\alpha$, the same convention as for the other power-law models. A finite variance requires $\alpha > 1/2$, so the Matern model only covers spectra steeper than flicker noise: Hector restricts $-3 < \kappa < -1$. One or both parameters can be kept fixed to specific values in the control file `estimatetrend.ctl`. For example:

```
NoiseModels         Matern
kappa_fixed         -1.6
lambda_fixed        0.01
```

Hector versions up to 3.1.9 (and the C++ version 2.2) printed and read $\kappa = -\alpha$ instead of -2α, i.e. half the true slope; multiply old Matern κ values by two to obtain the actual spectral index.

(Lilly et al. 2016) describe how one can simulate synthetic noise time series by performing a Cholesky decomposition and multiplying the decomposed matrix **U** with the vector **w** which contains white noise (Gaussian random variables). In Hector synthetic noise is created by convolving white noise with an impulse function (Kasdin 1995).

These two approaches differ in their treatment of the lack of values before the first observation. The Cholesky approach, to ensure the desired variance and co-variance values are produced, adjusts the impulse function coefficients (the rows in matrix **U**). The impulse function approach keeps its coefficients constant and thus will not produce the desired variance and co-variance values at the start of the time series. After some time, the two methods converge to the same result. A consequence of this is that the decomposed upper triangular **U** becomes more and more Toeplitz (values on each diagonal are the same) and these become identical with the impulse response coefficients, see also (Bos et al. 2013). We performed a Cholesky decomposition of the covariance matrix and take the last row of the matrix **U** as our impulse response coefficients.

7.8 VaryingPeriodic (Band-Pass) Noise Model

The amplitudes of the annual and semi-annual signals are mostly not exactly constant over time but vary a little in a random manner. Amplitude modulation of a periodic signal causes that the pure peak in the power-spectrum plot widens a bit. This can be viewed as a separate type of noise which can be added to the other noise models in the analysis.

(Langbein 2004) introduced a band-pass spectrum that captures this widening effect in the power-spectrum. From this spectrum one can compute the impulse response coefficients h_i from which one can again compute the autocovariance function. However, to maintain a Toeplitz covariance matrix that will ensure that we can continue to use all our numerical tricks to speed up the computations, we use a variant. We use the results of (Klos et al. 2019) where the amplitude modulation of the annual signal is modelled by a first order autoregressive model, AR(1). The corresponding autocovariance function is:

$$\gamma_k = \sigma^2 \frac{\phi^k}{2(1-\phi^2)} \cos(\omega_0 k)$$

Besides an annoying overloading of the symbol ϕ for yet another noise model, this is just the autocovariance of the AR(1) process multiplied by a cosine and divided by a factor 2. ω_0 is the angular frequency of the seasonal signal: $2\pi/365.25$ rad/day for `VaryingAnnual` and $2\pi/182.625$ rad/day for `VaryingSemiAnnual`; the lag k is converted to days with the sampling period. These models need a time axis in MJD (`mom` or `ncf` format). Varying annual and semi-annual signals are specified as follows:

```
NoiseModels         VaryingAnnual VaryingSemiAnnual White
phi_varying_fixed   0.999
```

Either model can also be used alone. As usual, one can estimate the value of ϕ (between 0 and 0.99999) or set it fixed by including `phi_varying_fixed` in the control file; this value is used by both models. Estimating both varying signals together with coloured noise is difficult: the likelihood can have more than one maximum, and a small semi-annual amplitude may not be resolvable. Fixing ϕ helps. Using the fact that $\cos(a)\cos(b) = [\cos(a+b) + \cos(a-b)]/2$, we have for the one-sided spectrum, with $\omega = 2\pi f/f_s$:

$$S(f) = \frac{\sigma^2}{2f_s}\left[\frac{1}{1-2\phi\cos(\omega+\omega_0)+\phi^2} + \frac{1}{1-2\phi\cos(\omega-\omega_0)+\phi^2}\right]$$

8 The Akaike, Bayesian, and Kashyap Information Criteria

Hector allows the estimation of linear trend or a higher order polynomial, seasonal and other periodic signals and a variety of noise models which can be combined. To choose the best model one can make use of information criteria (Akaike 1974; Schwarz 1978; Kashyap 1982). All three use the log-likelihood as their starting point but add penalties for adding parameters in order to avoid overfitting.

The definition of the log-likelihood is:

$$\ln(L) = -\frac{1}{2}\left[N\ln(2\pi) + \ln\det(\mathbf{C}) + \mathbf{r}^T\mathbf{C}^{-1}\mathbf{r}\right]$$

where N is the actual number of observations (gaps do not count). The covariance matrix $\mathbf{C}$ is decomposed as:

$$\mathbf{C} = \sigma^2\bar{\mathbf{C}}$$

where $\bar{\mathbf{C}}$ is the sum of various noise models and σ the standard deviation of the 'driving' white noise process as was explained in Section 7. σ is estimated from the residuals:

$$\sigma = \sqrt{\frac{\mathbf{r}^T\bar{\mathbf{C}}^{-1}\mathbf{r}}{N}}$$

Using this relation and the fact that $\det c\mathbf{A} = c^N \det \mathbf{A}$, the following formulation for the likelihood is implemented:

$$\ln(L) = -\frac{1}{2}\left[N\ln(2\pi) + \ln\det(\bar{\mathbf{C}}) + 2N\ln(\sigma) + N\right]$$

The number of parameters k is the sum of parameters in the Design matrix **H** and the noise models and the variance of the driving white noise process. For example, for estimating a linear trend using power-law + white noise models 5 parameters are involved: nominal bias, linear trend, distribution of variances between power-law and white noise, spectral index of the power-law and the variance of the driving white noise process ($k = 2 + 2 + 1 = 5$).

The Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC), and Kashyap Information Criterion (KIC) are defined as:

$$\begin{aligned}
\text{AIC} &= 2k - 2\ln(L) \\
\text{BIC} &= k\ln(N) - 2\ln(L) \\
\text{KIC} &= k\ln(N) - 2\ln(L) + \ln\left|\mathbf{H}^\top\mathbf{C}^{-1}\mathbf{H}\right|
\end{aligned}$$

The preferred model is the one with the minimum value. Note that these are relative measures between various choices, not absolute criteria. KIC adds the log-determinant of the Fisher information matrix $\ln\left|\mathbf{H}^\top\mathbf{C}^{-1}\mathbf{H}\right|$ to BIC, penalising models that are poorly identified by the data (Kashyap 1982). This term is the same correction used by RMLE (§4.1.4), so KIC is most meaningful when `useRMLE yes` is active.

8.1 Output of estimatetrend

After optimisation, `estimatetrend` prints the building blocks of the information criteria alongside the final AIC, BIC, and KIC values (see, for example, the output in §4.1):

```
min log(L)          : -1706.494080
ln_det_I            : 20.179890
ln_det_HH           : 64.521641
ln_det_C            : 113.447483
AIC                 : 3438.988161
BIC                 : 3501.332336
KIC                 : 3521.512226
```

`min log(L)` is the maximised log-likelihood ln(L) at the optimal noise model parameters. It is labelled "min" because the optimiser internally minimises $-\ln(L)$, but the value printed is the log-likelihood itself. With `useRMLE yes` it is the restricted log-likelihood, which includes the term $-\frac{1}{2}(\ln|\mathbf{I}| - \ln|\mathbf{H}^\top\mathbf{H}|)$; AIC, BIC and KIC are then computed from this value.

`ln_det_C` is $\ln|\bar{\mathbf{C}}|$, the log-determinant of the normalised covariance matrix $\bar{\mathbf{C}}$. It is one of the three additive terms in the log-likelihood formula above, alongside $N\ln(2\pi)$ and $2N\ln(\sigma)$. Comparing `ln_det_C` across noise model fits is therefore a direct way to see how much the covariance structure contributes to the likelihood.

`ln_det_HH` is $\ln \left|\mathbf{H}^\top \mathbf{H}\right|$, the log-determinant of the unweighted Gram matrix of the design matrix $\mathbf{H}$. It is a geometric property of the experiment design — the choice of trend, periodic signals, and offset epochs — and does not change with the noise model. It enters the RMLE correction together with `ln_det_I`.

`ln_det_I` is $\ln \left|\mathbf{H}^\top \mathbf{C}^{-1} \mathbf{H}\right|$, the log-determinant of the Fisher information matrix for the design parameters. When noise is large, $\mathbf{C}^{-1}$ shrinks and `ln_det_I` falls relative to `ln_det_HH`, reflecting that the data constrain the design parameters less tightly. It appears directly in the KIC formula: KIC $=$ BIC $+$ `ln_det_I`.

9 Auxiliary Python Scripts

The tools described in this section are installed alongside the core command-line programs when you install Hector (see the Installation chapter). Single-station tools (`estimatetrend`, `findoffsets`) and their batch counterparts (`estimate_all_trends`, `find_all_offsets`) form the two tiers of the v3.0 workflow.

In all these scripts, the same abbreviations for the noise models are used which are listed in the table below. Combinations of noise model abbreviations are also allowed in most cases. For example, GGMWN is the recommended combination for the analysis of GNSS time series.

Abbreviation	Description	Written to the control file as
WN	White noise	`White`
FN	Flicker noise	`FlickerGGM` with `GGM_1mphi 6.9e-06`
PL	Power-law noise	`GGM` with `GGM_1mphi 6.9e-06`
RW	Random Walk noise	`RandomWalkGGM` with `GGM_1mphi 6.9e-06`
GGM	Generalised Gauss-Markov noise model	`GGM` (with `GGM_1mphi` from `-phi` if given)
AR1	First-order autoregressive noise model	`AR1`
MT	Matern noise model	`Matern`
VA	Varying Annual (bandpass type noise; `estimate_all_trends` only)	`VaryingAnnual`
VSA	Varying Semi-Annual (bandpass type noise; `estimate_all_trends` only)	`VaryingSemiAnnual`

9.1 removeoutliers

`removeoutliers` detects and removes outliers from a single time series. Two detection modes are available, selected by the control file:

DataSnooping mode (`IQ_factor`): fits a preliminary trajectory model (trend, periodic signals, offsets) using ordinary least squares and flags observations whose residual deviates from the median

residual by more than `IQ_factor` × IQR (interquartile range); this is repeated until no new outliers are found. Use for the second outlier-removal pass, after offset epochs are known and included in the design matrix.

SpikeDetector mode (`Spike_factor`): detects isolated spikes using first-difference analysis. A spike at index j produces two consecutive differences of opposite sign that both exceed `Spike_factor` × MAD of all differences. Because an offset step produces only one large difference (no sign reversal), this mode is immune to unmodelled offsets. Use for the first outlier-removal pass, before offset detection, when offsets are unknown.

The mode is selected automatically: if `Spike_factor` is present in the control file, `SpikeDetector` is used (also when `IQ_factor` is present); otherwise `DataSnooping` is used, which requires `IQ_factor`. Without either keyword `removeoutliers` stops with an error message.

DataSnooping control file:

```
DataFile            TEST.mom
DataDirectory       ./obs_files
OutputFile          ./pre_files/TEST.mom
periodicsignals     365.25 182.625
estimateoffsets     yes
IQ_factor           3
PhysicalUnit        mm
```

`IQ_factor` scales the interquartile range to set the rejection threshold (required in this mode, 3 is a typical value; higher values are more conservative).

SpikeDetector control file:

```
DataFile            TEST.ncf
DataDirectory       ./raw_files
OutputFile          ./stage_files/TEST.ncf
periodicsignals     365.25 182.625
estimateoffsets     yes
Spike_factor        5
PhysicalUnit        mm
TimeUnit            days
```

`Spike_factor` scales MAD(first differences) to set the rejection threshold (required in this mode, 3–5 are typical values; higher values are more conservative).

CLI flags:

Flag	Meaning
`-i FILE`	Control file name (default: `removeoutliers.ctl`)
`-graph`	Show result plot on screen
`-png`	Save result plot to `data_figures/<PlotName>.png`
`-eps`	Save result plot to `data_figures/<PlotName>.eps`

`PlotName` is an optional keyword; its default is the name of the data file without extension.

For NCF (NetCDF) input each processed channel gets its own figure, saved as `data_figures/<station>_<column>` (or `.eps`) — e.g. `<station>_e.png`, `<station>_n.png`, `<station>_u.png` for a standard GNSS file, or `<station>_<ColumnName>.png` when `ColumnName` selects a single channel. The observed series is drawn in blue and the cleaned (filtered) series in red, so the effect of the chosen `IQ_factor` or `Spike_factor` can be judged at a glance.

Output: - `OutputFile` — the cleaned series in the format of the input: for `.mom` files the outliers (and gaps) are left out, for `.ncf` files they are set to `NaN` - `removeoutliers.json` — always written; for a `.mom` file it contains the number of epochs (including gaps) and the MJD of every removed point (Example 1):

```
{
    "N": 1000,
    "outliers": [50100.0, 50131.0, 50213.0, 51032.0, 51065.0, 51077.0]
}
```

For a `.ncf` file the same information is given per channel, under the key `channels`, together with the names of the input and output files.

All keywords accepted by `removeoutliers.ctl` are listed in the Quick Reference chapter.

9.2 estimate_all_trends

`estimate_all_trends` is the batch counterpart of `estimatetrend`. It reads every `.mom` and `.ncf` file in `./obs_files/`, runs `removeoutliers` followed by `estimatetrend` for each station, and collects all results into a single JSON file. `.ncf` files are processed as described in the section on the ncf format (spike detection, channels `e`, `n` and `u`); the steps below are those for `.mom` files.

Usage:

```
estimate_all_trends [-n NOISEMODEL] [-phi VALUE] [-s STATION]
                    [-useRMLE] [-nograph] [-noseasonal]
```

Flag	Default	Meaning
`-n`	`PLWN`	Noise model code (same abbreviations as in §9)
`-phi`	`0.0`	`GGM_1mphi` for a free GGM (code `GGM`, e.g. `GGMWN`); ignored for PL/FN/RW, which always use `6.9e-6`
`-s`	*(all)*	Process a single named station instead of all (its `.ncf` file if both exist)
`-useRMLE`	off	Use Restricted Maximum Likelihood
`-nograph`	off	Skip generating PNG plots
`-noseasonal`	off	Omit annual and semi-annual signals from the model

For each station the script:

1. Writes a temporary `removeoutliers.ctl` (DataSnooping, `IQ_factor 3`) and runs `removeoutliers`, saving the cleaned file to `./pre_files/`.
2. Writes a temporary `estimatetrend.ctl` and runs `estimatetrend -png` (or without `-png` if `-nograph` is set), saving the fitted series to `./mom_files/`.
3. Writes a temporary `estimatespectrum.ctl` and runs `estimatespectrum -model -png` to produce a PSD plot (skipped with `-nograph`).

On completion it writes **`hector_estimatetrend.json`** with the combined `estimatetrend.json` output for every station:

```
{
    "test_base_0": { "trend": -0.4933, "trend_sigma": 0.275344, ... },
    "test_base_1": { "trend":  0.0592, "trend_sigma": 0.222823, ... }
}
```

(from Example 3).

9.3 findoffsets

`findoffsets` is the v3.0 command-line tool for detecting the epochs of offsets in a single time series. It uses an iterative *forward search*:

1. Estimate the noise parameters via MLE (or RMLE if `useRMLE yes`).
2. For each candidate epoch, compute the improvement in log-likelihood $\Delta \ln L$ when a Heaviside offset column is added at that epoch. This is done with a fast scan that keeps the noise parameters fixed, so no extra MLE calls are needed (with `AmmarGrag` and no gaps it costs O(m) operations per epoch; with gaps or `FullCov` the scan is slower).
3. Accept the epoch with the largest $\Delta \ln L$ if it exceeds `OffsetThreshold`. Add it to the design matrix and repeat from step 1.
4. Stop when no remaining candidate exceeds the threshold or when `MaxOffsets` offsets have been found.

The control file is `findoffsets.ctl` (or any name passed with `-i`). It uses the same keywords as `estimatetrend.ctl` with two additions:

```
DataFile            TEST.mom
DataDirectory       ./pre_files
OutputFile          ./pre_files/TEST.mom
periodicsignals     365.25 182.625
estimateoffsets     yes
NoiseModels         GGM White
GGM_1mphi           6.9e-06
useRMLE             yes
PhysicalUnit        mm
OffsetThreshold     20.0
MaxOffsets          50
```

`OffsetThreshold` is the minimum $\Delta \ln L$ of the analysed component required to accept an offset (default 20.0). Under the null hypothesis of no offset $2\Delta \ln L$ follows a $\chi^2(1)$ distribution, so the

default corresponds to $2\Delta \ln L > 40$, a very strict test; lower values detect more offsets but also more false ones. This threshold is set in the control file of `findoffsets`; for the batch tool `find_all_offsets` it is set on the command line (see below). `MaxOffsets` (default 50) is a safety cap.

Running `findoffsets` prints the iterative search progress; for Example 5 the summary lines are:

```
0: best offset at  50784.00 (i=700) : dln=  160.744
1: best offset at  51034.00 (i=950) : dln=  172.049
2: best offset at  50284.00 (i=200) : dln=  104.895
3: best offset at  50335.00 (i=251) : dln=  133.194
4: best offset at  50807.00 (i=723) : dln=    8.800
---
Found 4 offset(s).
```

The search stops at step 4 because $\Delta \ln L = 8.8 < 20.0$. Two output files are written:

- **`findoffsets.json`** — machine-readable list of detected offset MJDs:

  ```
  {
      "offsets": [50784.0, 51034.0, 50284.0, 50335.0]
  }
  ```

- **`OutputFile`** (here `./pre_files/TEST.mom`) — the input `.mom` file with `# offset MJD` header lines added for every detected offset. This file can be passed directly to `estimatetrend`. For a `.ncf` file the detected epochs are added as offset events of the analysed channel (see the section on the ncf format).

`findoffsets` is the single-station counterpart of `find_all_offsets`, in the same way that `estimatetrend` is the single-station counterpart of `estimate_all_trends`.

9.4 find_all_offsets

`find_all_offsets` detects offsets for every station in a directory. It has two modes.

Default mode (`.ncf`, multivariate). For every `.ncf` file in `./stage_files/` (spike-filtered data, see the Conventions chapter) a forward search is run on the three components `e`, `n` and `u` together: at each step the statistic $T_3 = 2(\Delta \ln L_E + \Delta \ln L_N + \Delta \ln L_U)$ is computed for every epoch and the epoch with the largest T_3 is accepted if it exceeds the threshold (default 16.27, the $\chi^2(3)$ value for $\alpha = 0.001$). Epochs within `--min_gap` days of an accepted offset are not considered again. The detected epochs are added, as offset events of type 0 that apply to all channels, to a copy of the file in `./obs_files/<station>.ncf` (existing offset events and their per-channel variables are kept), and all results to **`find_all_offsets_ncf.json`**:

```
{
    "STATION": {
        "offsets": [57012.0, 58340.0],
        "details": [
            {"mjd": 57012.0, "T3": 170.4, "delta_lnL": 85.2, "iteration": 1},
            {"mjd": 58340.0, "T3": 47.8, "delta_lnL": 23.9, "iteration": 2}
        ]
```

```
    }
}
```

`--mom` mode (univariate). For every `.mom` file in `./obs_files/` the script runs `removeoutliers` (DataSnooping, `IQ_factor 3`, writing to `./pre_files/`) and then `findoffsets`, which adds the detected offsets to the file in `./pre_files/`. The results are collected in **`find_all_offsets.json`**:

```
{
    "TEST":  {"offsets": [50784.0, 51034.0, 50284.0, 50335.0]},
    "SITE2": {"offsets": []}
}
```

Usage:

```
find_all_offsets [-n NOISEMODEL] [-phi VALUE] [-s STATION]
                 [-t THRESHOLD] [-maxoffsets N] [-useRMLE]
                 [--min_gap DAYS] [--mom]
```

Flag	Default	Meaning
`-n`	`PLWN`	Noise model code (same abbreviations as in §9)
`-phi`	`0.0`	`GGM_1mphi` for a free GGM (code `GGM`); PL/FN/RW always use `6.9e-6`
`-s`	*(all)*	Process a single named station instead of all
`-t`	`16.27` (`--mom`: `20.0`)	Threshold on T_3 (default mode) or on $\Delta \ln L$ (`--mom`)
`-maxoffsets`	`50`	Maximum number of offsets per station
`-useRMLE`	off	Use Restricted Maximum Likelihood
`--min_gap`	`30`	Minimum separation in days between two detected offsets (default mode)
`--mom`	off	Use the univariate `.mom` mode

The threshold of `find_all_offsets` is set with `-t` on the command line: the control files are written by the script for every station, so a `findoffsets.ctl` prepared by the user is overwritten (with `--mom`, the value of `-t` is written into it as `OffsetThreshold`).

Choosing the threshold. The default 16.27 is the statistically correct value for $\alpha = 0.001$ when the trajectory model and the noise model describe the data. Real GNSS time series contain signals that the model of trend, seasonal terms and offsets does not capture, and the best of thousands of candidate epochs is tested at every step, so with the default many epochs are detected that are not related to equipment changes or earthquakes: for the eight stations of Example 6 the default gives about 200 offsets against 42 entries in the NGL catalogue. To avoid detecting too many false offsets in real GNSS series, a threshold `-t` between 30 and 70 is advised. Offset detection in real

time series remains difficult; treat the detected epochs as candidates and compare them with the station log where possible. Offset detection uses `AmmarGrag`, except for stations with more than 50 % missing data, for which `FullCov` is used automatically.

9.5 predicttrenderror

`predicttrenderror` uses the noise parameters from a completed `estimatetrend` run to predict how the trend uncertainty would change with series length. It is useful for planning campaigns or for understanding the effect of coloured noise on velocity precision.

Usage:

```
predicttrenderror [-i JSON] [-dt DAYS] [-t0 DAYS] [-t1 DAYS] [-seasonal] [-graph] [-eps] [-png]
```

Flag	Default	Meaning
`-i`	`estimatetrend.json`	Input JSON with estimated noise parameters
`-dt`	`1`	Prediction step (days)
`-t0`	`730`	Start of prediction range (days, ≈ 2 years)
`-t1`	`7300`	End of prediction range (days, ≈ 20 years)
`-seasonal`	off	Include annual signal in the trajectory model
`-graph`	off	Show the plot on screen
`-eps`	off	Save plot to `data_figures/trend_sigma.eps`
`-png`	off	Save plot to `data_figures/trend_sigma.png`

The tool writes `trend_sigma.out` and optionally plots trend uncertainty (in the same physical unit as the input, per year) versus series length for the estimated noise model. All noise models supported by `estimatetrend` are handled, including ARFIMA and ARMA, except `Matern`. The `estimatetrend` run must have had `TimeUnit days` (or `seconds`) in its control file, because the time unit is read from the JSON file.

9.6 simulatenoise

`simulatenoise` generates synthetic time series with prescribed noise properties. It is fully documented in Example 3 (§4.3) and the Quick Reference (`simulatenoise.ctl`). Run `simulatenoise` from a directory containing a `simulatenoise.ctl` control file, or give the control file with `-i FILE`. The noise amplitudes (and κ, $1-\phi$ or ϕ when they are not set in the control file) are asked for on the screen; they can also be fed from a file, as in Example 3.

9.7 convert_rlrdata2mom

`convert_rlrdata2mom` converts PSMSL RLR (Revised Local Reference) annual or monthly sea-level files to the mom format accepted by Hector.

Usage:

```
convert_rlrdata2mom -i INPUT_FILE -o OUTPUT_FILE
```

The script auto-detects yearly (`# sampling period 365.25`) or monthly (`# sampling period 30.4375`) sampling from consecutive time stamps. Observations flagged as missing (value −99999) or with a non-zero quality flag are excluded. See Example 2 (§4.2) for a worked example using PSMSL tide-gauge data.

9.8 convert_tenv2netcdf

`convert_tenv2netcdf` converts Nevada Geodetic Laboratory (NGL) daily GNSS position files in tenv (17-column) or tenv3 (23-column) format to the NCF format accepted by Hector.

Usage:

```
# Convert all *.tenv3 / *.tenv files in the current directory
convert_tenv2netcdf

# Convert a single station
convert_tenv2netcdf -s KOSG

# Write to a custom output directory
convert_tenv2netcdf --outdir raw_files

# Discard noisy data before a given MJD (e.g., before 1 Jan 1999)
convert_tenv2netcdf --start-mjd 51179
```

Output: one `.ncf` file per station in `raw_files/` (or the directory given by `--outdir`). Each file contains six channels: `e`, `n`, `u` (East, North, Up displacement relative to the first epoch, mm) and `sigma_e`, `sigma_n`, `sigma_u` (formal uncertainties, mm).

Positions are stored as displacements relative to the first valid epoch so that float32 precision is sufficient (absolute E/N/U positions reach 10^2–10^6 m, causing unacceptable rounding when converted to mm without referencing).

After conversion, the standard workflow applies:

```
removeoutliers -i removeoutliers.ctl   # reads raw_files/<STATION>.ncf
estimatetrend  -i estimatetrend.ctl
estimatespectrum
```

NGL tenv/tenv3 files can be downloaded from https://geodesy.unr.edu/.

9.9 date2mjd and mjd2date

Two small utilities for date conversions between calendar date and Modified Julian Date (MJD):

```
date2mjd  year month day hour minute second
mjd2date  MJD
```

Both print all six date components and the MJD to standard output:

```
$ date2mjd 2000 1 1 0 0 0
year   : 2000
month  :    1
day    :    1
hour   :    0
minute :    0
second : 0.000000
MJD    : 51544.000000
```

`date2mjd` requires all six arguments. `mjd2date` takes a single MJD value and prints the corresponding calendar date.

9.10 plot_ts

`plot_ts` is a fast time-series viewer located in `src/hector/plot_ts.py`. It supports both plain-text files (mom, gen, ASCII) and netCDF4 files (`.ncf` or `.nc`).

```
plot_ts -i FILE -cy CHANNEL [options]
```

Flag	Description
`-i FILE`	input file (`.mom` / ASCII text, or `.ncf` / `.nc`)
`-cx`	x-axis: **column number** (1-based integer; required for text files) or **channel name** (`.ncf`/`.nc`); defaults to `time` for netCDF4
`-cy`	y-axis: column number (text) or channel name (`.ncf`/`.nc`) — **required**
`-cy2`	overlay a second channel on the same axes, or the second vector component for `-m` (column number for text files, channel name for `.ncf`/`.nc`)
`-cy3`	overlay a third channel on the same axes, or the third vector component for `-m` (column number for text files, channel name for `.ncf`/`.nc`)
`-lx LABEL`	x-axis label
`-ly LABEL`	y-axis label
`-title TEXT`	plot title
`-o FILE`	save to file (`plot.png`, `plot.pdf`, …) instead of opening a window
`-m`	plot vector magnitude $\sqrt{y^2 + y_2^2 + y_3^2}$, where y, y_2, y_3 are the columns/channels given by `-cy`, `-cy2`, `-cy3` (all three required)

Text files use 1-based integer column indices:

```
plot_ts -i obs/TEST.mom -cx 1 -cy 2 -lx MJD -ly "east (mm)"
```

`.ncf` files use channel names. The x-axis defaults to the MJD time axis, displayed as decimal year. Use `ncfdump -i FILE` to list available channel names:

```
plot_ts -i flight.ncf -cy acc_z -ly "acc_z (m/s²)"
plot_ts -i flight.ncf -cy east -cy2 east_model -ly "east (mm)"
plot_ts -i flight.ncf -cy gyro_x -o gyro_x.png
plot_ts -i flight.ncf -cy acc_x -cy2 acc_y -cy3 acc_z -ly "acc (m/s²)"
plot_ts -i flight.ncf -cy acc_x -cy2 acc_y -cy3 acc_z -m -ly "|acc| (m/s²)"
```

Without `-m`, giving `-cy2` and/or `-cy3` simply overlays up to three series (`-cy`, `-cy2`, `-cy3`) on the same axes, each with its own colour and a legend. With `-m`, the three series collapse into a single magnitude line instead — `-cy2` and `-cy3` are then both required to identify the other two vector components.

If a channel name is not found the script exits with an error message that lists all channels present in the file. `-m` requires `-cy2` and `-cy3` to identify the other two vector components explicitly — it no longer assumes they are the next two columns after `-cy`.

10 Quick Reference for the Control-Files

The keyword `interpolate`, which still appears in some example control files, is ignored: gaps are always treated as missing data.

10.1 removeoutliers.ctl

This file is read by `removeoutliers`.

Keyword	Value(s)
`DataFile`	name of file with observations (required)
`DataDirectory`	directory where file with observations is stored (optional, default `.`)
`OutputFile`	name of the file with the cleaned observations, in the format of the input (required)
`TS_format`	`mom` (default) | `gen` | `ncf` — selects the file format; `.ncf`/`.nc` files are recognised automatically
`ColumnName`	channel to clean in a `.ncf` file (optional; default: the channels `e`, `n` and `u`)
`TimeUnit`	`days` (optional; for mom and ncf files only `days` is accepted)
`DegreePolynomial`	degree of polynomial: 0–12 (optional, default=1)
`estimatemultitrend`	yes|no (optional, default=no). If yes, a ramp is added for every `# break` epoch, on top of the polynomial
`estimatepostseismic`	yes|no (optional, default=no)
`estimateslowslipevent`	yes|no (optional, default=no)
`seasonalsignal`	yes|no (kept for backward compatibility; prefer `periodicsignals`)
`halfseasonalsignal`	yes|no (kept for backward compatibility; prefer `periodicsignals`)

Keyword	Value(s)
periodicsignals	a sequence of numbers representing the period in days (optional). Example: 365.25 182.625
estimateoffsets	yes|no (optional, default=yes)
ScaleFactor	a number to scale the observations (optional, default=1)
PhysicalUnit	the physical unit of the observations
IQ_factor	the number used to scale the interquartile range (DataSnooping mode)
Spike_factor	the number used to scale MAD(first differences) (SpikeDetector mode; used when present)
estimatemultivariate	yes|no (optional, default=no)
MultiVariateFile	Name of the geophysical signal file (.mom for one signal, .ncf for named channels)
MultiVariateDir	Directory of the signal file (optional, defaults to DataDirectory)
MultiVariateSignals	Space-separated channel names to read from a .ncf signal file
ReferenceEpoch	year month day — reference epoch for the nominal bias (optional)
PlotName	name of the figure in data_figures/ (optional, default: name of the data file)
Verbose	yes|no (optional, default=yes)

Exactly one of IQ_factor and Spike_factor is needed. The file removeoutliers.json with the removed epochs is always written.

10.2 estimatetrend.ctl and findoffsets.ctl

These files are read by estimatetrend and findoffsets respectively. They share all keywords listed below; findoffsets.ctl additionally accepts OffsetThreshold and MaxOffsets. DataFile, OutputFile, PhysicalUnit and NoiseModels are required.

Keyword	Value(s)
DataFile	name of file with observations
DataDirectory	directory where file with observations is stored (optional, default .)
OutputFile	name of file with observations *and* estimated model, in the format of the input
TS_format	mom (default) | gen | ncf — selects the file format; .ncf/.nc files are recognised automatically
ColumnName	name of the channel to read (required for .ncf files)

Keyword	Value(s)
`UseResiduals`	`yes`|`no` (optional, default=`no`). When `yes`, subtract `<channel>_model` from the observations before analysis (ncf only)
`TimeUnit`	`days` (optional; for mom and ncf files only `days` is accepted; needed by `predicttrenderror`)
`DegreePolynomial`	degree of polynomial: 0–12 (optional, default=1)
`estimatemultitrend`	yes|no (optional, default=no). If yes, a ramp is added for every `# break` epoch, on top of the polynomial
`estimatepostseismic`	yes|no (optional, default=no)
`estimateslowslipevent`	yes|no (optional, default=no)
`seasonalsignal`	yes|no (kept for backward compatibility; prefer `periodicsignals`)
`halfseasonalsignal`	yes|no (kept for backward compatibility; prefer `periodicsignals`)
`periodicsignals`	a sequence of numbers, separated by spaces, representing the period of the periodic signals in days (optional). Example: `365.25 182.625`
`estimateoffsets`	yes|no (optional, default=yes)
`ScaleFactor`	a number to scale the observations (optional, default=1)
`PhysicalUnit`	the physical unit of the observations
`useRMLE`	yes|no (optional, default=no). Use Restricted Maximum Likelihood. **Recommended: yes**.
`NoiseModels`	choose any set from: `White`, `Powerlaw`, `GGM`, `FlickerGGM`, `RandomWalkGGM`, `Matern`, `ARFIMA`, `ARMA`, `AR1`, `VaryingAnnual` and `VaryingSemiAnnual`. Recommended for GNSS: `GGM White`
`LikelihoodMethod`	`AmmarGrag` or `FullCov` (optional, not case-sensitive). Default: ordinary least squares when `White` is the only noise model; otherwise `AmmarGrag` if at most 50% of the epochs are missing, else `FullCov`. `AmmarGrag` uses the GSA for long series.
`AR_p`	number of AR coefficients (only for ARFIMA or ARMA)
`MA_q`	number of MA coefficients (only for ARFIMA or ARMA)
`GGM_1mphi`	value of $1 - \phi$ (optional, for `GGM`, `FlickerGGM` and `RandomWalkGGM`; estimated when absent). Recommended: `6.9e-06`
`kappa_fixed`	keep value of spectral index fixed to given value (optional, only for `GGM` and `Matern`)

Keyword	Value(s)
`lambda_fixed`	keep value of lambda fixed (optional, only for Matern)
`phi_ar1_fixed`	keep ϕ of the `AR1` model fixed (optional)
`phi_varying_fixed`	keep ϕ of the `VaryingAnnual` and `VaryingSemiAnnual` models fixed (optional)
`RandomiseFirstGuess`	yes|no (optional, default=no)
`Tolerance`	Nelder-Mead convergence tolerance (optional). Without it, `fatol` = 10^{-4} max$(1, \lvert \ln L_0 \rvert)$ and `xatol` = 10^{-6}; when given, `fatol` = `Tolerance` $\times$ max$(1, \lvert \ln L_0 \rvert)$ and `xatol` = `Tolerance`, where $\ln L_0$ is the log-likelihood at the starting point and the parameters are searched in units of their starting values. So it means the same number of significant digits whatever the length of the series and the scale of each parameter. Example: `Tolerance 1e-8`
`MaxIterations`	maximum number of Nelder-Mead iterations (optional, default=10000)
`ReferenceSpan`	span in years over which the modified standard deviation of the power-law/GGM noise is computed (optional, default=8.0). See the "Comparing Noise Amplitudes Between Stations" section. Use the same value for all stations being compared. Example: `ReferenceSpan 8.0`
`estimatemultivariate`	yes|no (optional, default=no)
`MultiVariateFile`	Name of the geophysical signal file (`.mom` for one signal, `.ncf` for named channels)
`MultiVariateDir`	Directory of the signal file (optional, defaults to `DataDirectory`)
`MultiVariateSignals`	Space-separated channel names to read from a `.ncf` signal file
`ReferenceEpoch`	`year month day` — reference epoch for the nominal bias (optional, default: midpoint of the time series). Example: `ReferenceEpoch 2008 1 1`
`PlotName`	name of the figure in `data_figures/` (optional, default: name of the data file)
`OffsetEpochsFile`	text file with lines `STATION MJD`; these epochs are drawn as vertical lines in the figure (optional)
`Verbose`	yes|no (optional, default=yes)
`OffsetThreshold`	minimum $\Delta \ln L$ to accept an offset in `findoffsets` (optional, default=20.0)
`MaxOffsets`	maximum number of offsets to find in `findoffsets` (optional, default=50)

`estimatetrend.json` and `findoffsets.json` are always written.

Command-line flags of `estimatetrend`:

Flag	Description
`-i FILE`	control file (default `estimatetrend.ctl`)
`-graph`	display the observations and the fitted model on screen
`-png`	save this figure as `data_figures/<PlotName>.png`
`-eps`	save this figure as `data_figures/<PlotName>.eps`

`findoffsets` has only the flag `-i FILE` (default `findoffsets.ctl`).

10.3 estimatespectrum.ctl

This file is read by `estimatespectrum`, which writes `estimatespectrum.out`. When the input has a model column (or model channel), the spectrum of the observations minus the model is computed. When called with the `-model` flag, `estimatespectrum` also reads the noise model parameters from `estimatetrend.json` and writes the model PSD to `modelspectrum.out`. This replaces the separate `modelspectrum` program from Hector C++ v2.2.

Keyword	Value(s)
`DataFile`	name of file with observations (required)
`DataDirectory`	directory where file with observations is stored (optional, default `.`)
`TS_format`	`mom` (default) | `gen` | `ncf`
`ColumnName`	name of the channel to read (required for `.ncf` files)
`PhysicalUnit`	the physical unit of the observations (required)
`TimeUnit`	`days` (optional; for mom and ncf files only `days` is accepted)
`ScaleFactor`	a number to scale the observations (optional, default=1)
`NumberOfSegments`	number of equal-length segments to divide the time series into (optional, default=4)
`Fraction`	fraction of each segment tapered to zero at both ends by a split-cosine-bell (Tukey) window (optional, default=0.1)
`PlotName`	name of the figure in `psd_figures/` (optional, default: name of the data file)
`Verbose`	yes|no (optional, default=yes)

Overlap is fixed at 50%, so `NumberOfSegments = 4` produces 7 Welch periodograms.

Command-line flags:

Flag	Description
`-i FILE`	control file (default `estimatespectrum.ctl`)
`-model`	overlay the theoretical noise model PSD; reads `estimatetrend.json` by default
`-j FILE`	use `FILE` instead of `estimatetrend.json` when `-model` is active
`-graph`	display the PSD plot on screen
`-png`	save the plot as a PNG in `psd_figures/`
`-eps`	save the plot as an EPS in `psd_figures/`

10.4 simulatenoise.ctl

This file is read by `simulatenoise`. The files are written as `<SimulationDir>/<SimulationLabel>_<k>.mom`; the control file must not contain `DataFile`.

Keyword	Value(s)
`SimulationDir`	directory where created files will be stored
`SimulationLabel`	base name of the created files
`NumberOfSimulations`	number of simulations
`NumberOfPoints`	length of each simulation
`SamplingPeriod`	specifies the time step in days
`NoiseModels`	choose any set from: `White`, `Powerlaw`, `Flicker`, `RandomWalk`, `GGM`, `Matern`, `AR1`, `VaryingAnnual` and `VaryingSemiAnnual`
`GGM_1mphi`	value of $1 - \phi$ for `GGM` (optional; asked for when absent)
`kappa_fixed`	spectral index for `Matern` (optional; asked for when absent)
`lambda_fixed`	λ for `Matern` (optional; asked for when absent)
`phi_varying_fixed`	ϕ for `VaryingAnnual` and `VaryingSemiAnnual`, the same keyword as in `estimatetrend.ctl` (optional; the older `phi_fixed` is still accepted; asked for when both are absent)
`TimeNoiseStart`	number of extra points simulated before the first observation and discarded, to avoid spin-up effects (optional, default=0). A value of 1000 is recommended for coloured noise models.
`Trend`, `NominalBias`	linear trend (unit per year) and bias added to the noise (optional, both together, default 0)
`AnnualSignal`	amplitude of an annual cosine added to the noise (optional, default 0)
`RepeatableNoise`	yes|no. Each time `simulatenoise` is run, the same synthetic time series are created. Default is no.

Keyword	Value(s)
`RandomSeed`	integer seed of the random-number generator (optional). Runs with the same seed give the same series, runs with different seeds independent ones. Takes precedence over `RepeatableNoise`.

The noise amplitudes are asked for on the screen, in the order of `NoiseModels`.

11 License

HECTOR SOFTWARE LICENSE AGREEMENT

Version 1.0

Copyright © 2026 TeroMovigo – Earth Innovation Lda (“Licensor”)

---

1. Definitions

Software means the Hector software package, including source code, documentation, and any associated materials provided by the Licensor.

Non-Commercial Use means use solely for research, academic, educational, or other purposes that are not primarily intended for or directed toward commercial advantage or monetary compensation.

Commercial Use means any use that does not qualify as Non-Commercial Use, including use by for-profit entities, use in connection with revenue-generating activities, or use in proprietary products or services.

2. Grant of License

Subject to the terms of this Agreement, the Licensor hereby grants a non-exclusive, non-transferable, royalty-free license to use, reproduce, modify, and distribute the Software for Non-Commercial Use.

3. Commercial Use

Commercial Use of the Software is not permitted under this license. Any Commercial Use requires a separate written agreement with the Licensor.

Commercial entities or individuals intending to use the Software for Commercial Use must contact:

TeroMovigo – Earth Innovation Lda
info@teromovigo.com — https://teromovigo.com

4. Restrictions

Except as expressly permitted under this Agreement, you may not:

(a) use the Software for Commercial Use;

(b) sublicense, sell, lease, or otherwise distribute the Software for Commercial Use;

(c) remove or alter any copyright or attribution notices.

5. Attribution

Any redistribution of the Software, in whole or in part, must retain this license text and provide appropriate attribution to the Licensor.

6. Ownership

The Software is licensed, not sold. All intellectual property rights, including copyright, remain with the Licensor.

7. No Warranty

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT.

8. Limitation of Liability

IN NO EVENT SHALL THE LICENSOR BE LIABLE FOR ANY CLAIM, DAMAGES, OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT, OR OTHERWISE, ARISING FROM OR IN CONNECTION WITH THE SOFTWARE OR ITS USE.

9. Termination

This license terminates automatically if you breach any of its terms. Upon termination, you must cease all use of the Software.

10. Governing Law

This Agreement shall be governed by and construed in accordance with the laws of Portugal, without regard to conflict-of-law principles.

12 Appendix: History of Changes in Hector

12.1 Version 1.1

1. The programs `estimatetrend`, `removeoutliers`, `estimatespectrum` and `modelspectrum` now accept the name of the control-file on the command line. In Hector v3.0 this is done with the `-i` option, for example:

   ```
   estimatetrend -i mycontrol.ctl
   ```

2. The epoch of the nominal bias is shown in the output as a Modified Julian Date. By default it is the midpoint of the time series. The keyword `ReferenceEpoch` from Hector C++ v2.2, followed by year, month and day, can be used to set another reference epoch, e.g. `ReferenceEpoch 2008 1 1`.

3. The ability to leave out a keyword also made it possible to define other default parameters. Now it is no longer necessary to provide the `ScaleFactor` keyword if this is not different from 1 and the `LikelihoodMethod` keyword is optional. If it is missing, then the `AmmarGrag` method

will be used when the amount of missing data is less than 50% of the whole time series. Otherwise the `FullCov` method is used.

4. The ARFIMA model had two bugs, one due to a sign error and one due to accessing arrays outside their range, which have been resolved.
5. The keyword `MinimizingMethod` has been removed because the Nelder-Mead Simplex method is the only method available.
6. It is now also possible to estimate a quadratic polynomial by setting the keyword `QuadraticTerm` to yes.
7. The program `simulatenoise` was added.
8. Implemented the `dd-mm-year NaN NaN NaN` offset format for external files.

12.2 Version 1.2

1. `simulatenoise` has now correct power.
2. The ARMA and ARFIMA noise model now also accept first-difference (removed from version 1.5 onwards).
3. All programs now accept ridiculously long names.

12.3 Version 1.3

1. Removed a bug which caused Hector to crash on some computers. The reason was that `Nnumbers` was not set to zero explicitly.
2. The parser of the control-files had trouble when there was a space after the last label. For example `NoiseModels PowerlawApprox White` where there is a space behind the last word. This has now hopefully been corrected.
3. Using the Generalised Gauss Markov noise model did not converge in rare situations but they did occur. Now the maximum value of ϕ has been lowered from 0.9999 to 0.999. (Has been solved in version 1.6 and now the limit is closer to 0.999999.)
4. The binaries are now stored in `/usr/local/bin` instead of `/usr/bin` which we hope is more in line with the Linux standard.

12.4 Version 1.4

1. Better command line parsing for `estimatespectrum`.
2. Added `# sampling period 1.0` in `.enu` file created by `convert_sol_files.tcl`.
3. Removed root-message in ARFIMA.
4. Removed trailing space bug in Control parser (again).
5. Improved C++ correctness (`fp.getline(..)!=NULL`) in Observations parser.

12.5 Version 1.5

1. Removed the option to apply first difference to data. This option was never used and it makes the source code unnecessarily complicated.

2. Added Hann window to `estimatespectrum`. You can now choose between the Hann and Parzen window function and select the percentage of data to which this window will be applied at both ends of the segment.
3. Implemented a Taylor expansion of the Hypergeometric function in the GGM noise model and now ϕ can be closer to 1: 0.9999. This helps to simulate pure power-law noise.
4. Changed the output of `estimatetrend` by eliminating parameters d, driving noise and fractions. The noise amplitudes now follow CATS.
5. The Plate Boundary Observatory changed their time series file format. It is easier to write a small script to convert the new format to mom-format than to update Observations parsing.
6. Removed subsection "Some additional tests" since it was not read or created confusion for those who did.

12.6 Version 1.5.1

1. Removed a small bug in the Minimizer that left the first parameter 0.001 larger than the optimal value (residual of computing Fisher information matrix). It had a large effect if "Random Walk" was the first chosen noise model.
2. Added to the manual that you can set the $1 - \phi$ value a priori in the control-file.

12.7 Version 1.5.2

1. Still a problem of having spaces in list of items in the control-file parser which now hopefully has been solved.

12.8 Version 1.6

1. Introduced logarithmic and exponential post-seismic relaxation in the design matrix.
2. Generalised linear and quadratic trend to N degree polynomial. To keep maintenance simple, the `QuadraticTerm` keyword was removed.
3. Number of allowed periodical signals in the estimation process has been increased from 20 to 40.
4. One can select to estimate multi linear trends instead of a single polynomial.
5. `estimatespectrum` and `modelspectrum` now allow the keyword `OutputFile` to redirect their output to another file. Note that the default control-file for `modelspectrum` is `modelspectrum.ctl` to be consistent.
6. Removed memory leaks.
7. Added program `findoffset`.

12.9 Version 1.7.2

1. Added the program `findoffset`.
2. Added Python scripts: `analyse_timeseries.py` and `analyse_and_plot.py`.
3. Use generalised spheres to compute fractions which solves the problem in older versions of Hector when all fraction values went haywire when one of the fraction values approached zero.
4. Added hyperbolic tangent to model slow slip events.

5. The MLE performs a numerical minimisation search. The initial guess of the noise parameter values is fixed. In this version the starting values are allowed to vary a bit randomly each run when the keyword `RandomiseFirstGuess yes` is set in the control file. So far, we have not encountered any benefit of this but rumours have it that CATS, and perhaps Hector, sometimes do not converge (it would help us if people send us their problematic time series). This extra randomisation helps to prove, by running Hector several times and always finding the same answer, that a unique solution has been found.
6. `removeoutliers` now produces another file, `removeoutliers.out`, that contains the epoch of the found outliers.

12.10 Version 1.9

1. There is no 1.8 because that was a disaster. The reason was that ATLAS was no longer maintained by MacPorts and did not install cleanly. On CentOS it still installed but GSL conflicted with it because of slightly different CBLAS headers.
2. The source code was adapted to make use of the OpenBlas library instead of ATLAS. Furthermore, the Boost library was used to generate random numbers and compute hypergeometric functions ${}_1F_1$ and ${}_2F_1$.
3. Added the Matern noise model.
4. `modelspectrum` can now perform a Monte Carlo simulation to get an idea of how well one can estimate the spectrum.
5. More Python scripts are included. These scripts can be used to convert the output of Hector into a JSON file which should facilitate its parsing into other programs.
6. Created Dockerfiles in order to quickly cross-compile Hector to other Linux distributions.

12.11 Version 2.0

1. Bugs in the multivariate subroutine were corrected. The problem occurred when the multivariate file ended on the same date as the observations which caused a crash. If no crash occurred, then values were correct.
2. Multi-trend now has proper piecewise linear segments without the need to add offsets on the date when the trend changes.
3. The phase of estimated periodic signals now has a standard deviation.
4. `simulatenoise` now always creates different synthetic time series when it is run. To create the same synthetic time series, one must set the keyword `RepeatableNoise` to yes.

12.12 Version 2.1

1. OpenBlas changed how to call LAPACK subroutines directly. The `dgels`, `dpotrf` and `dpotri` subroutines now have another parameter at the end which indicates the length of the string used for the first parameter. The changes to these subroutines were necessary to compile Hector on a MacBook Pro and for Ubuntu 22.04.

12.13 Version 2.2

1. Better information on the screen in case RMLE is used or not.

2. Better checks when `estimatevaryingseasonal` is used because one should not use `seasonalsignal` and `halfseasonalsignal` keywords in this case.

12.14 Version 3.0

1. **Complete Python/Cython rewrite.** Hector v3.0 is a full rewrite in Python with Cython extension modules for the performance-critical inner loops. All command-line executables install via `pip install hector-ts` as entry points. The FFTW3 system library is required for the GSA and gap-correction extensions.
2. **Generalised Schur Algorithm (GSA).** The `AmmarGrag` likelihood method now uses the GSA for series longer than approximately 2000 points. This reduces the Toeplitz factorisation from $O(n^2)$ to $O(n \log^2 n)$, delivering a **5–15× speedup** over v2.2 for typical GNSS time series (5–15× at 30 years, growing further for very long series).
3. **FFT-accelerated gap correction.** The Bos (2013) gap correction has been rewritten to avoid redundant FFT calls on unit vectors. Complexity is now $O(m \log m + k^2)$ instead of $O(k^2 m)$, where k is the number of gaps and m the total series length.
4. **New keyword `useRMLE`.** Setting `useRMLE yes` in `estimatetrend.ctl` activates Restricted Maximum Likelihood estimation. This is the **recommended default** for v3.0.
5. **New keyword `periodicsignals`.** The old `seasonalsignal` / `halfseasonalsignal` keyword pair is replaced by `periodicsignals 365.25 182.625`. The old keywords remain for backward compatibility.
6. **GGM+White recommended default.** `NoiseModels GGM White` with `GGM_1mphi 6.9e-06` is the recommended noise model combination for GNSS daily data. It approximates power-law plus white noise and is strictly stationary.
7. **Automated offset detection.** The new `findoffsets` program replaces `findoffset`. It uses an iterative GLR forward search: noise parameters are estimated once per iteration, after which the epoch scan reuses cached Schur factors and FFT-whitened columns. The companion `find_all_offsets` processes an entire network in one command.
8. **Python API and Jupyter support.** All functionality is accessible via a Python API (`from hector.mle import MLE`, etc.). Singletons are reset with `SingletonMeta.clear_all()`. Worked Jupyter notebooks are included in `examples/ex1/` (full workflow) and `examples/ex8/` (Toeplitz factorisation benchmark); see §4.1 and §4.8.
9. **Installation via pip.** `pip install hector-ts` installs all command-line tools as entry points. The package requires FFTW3 (system library) and Cython; see the Installation chapter.
10. **Spurious matmul warnings silenced (Apple Silicon).** On some Apple Silicon Macs, Apple's Accelerate BLAS raises bogus floating-point flags during large matrix multiplies, which NumPy reports as `RuntimeWarning: divide by zero / overflow / invalid value encountered in matmul`. The computed results are unaffected. Hector now suppresses these spurious warnings around its linear-algebra routines (see NumPy issue #28687); they were never a sign of a problem with your data or installation.
11. **`LikelihoodMethod` keyword honoured; `FullCov` sped up.** The `LikelihoodMethod` keyword (`AmmarGrag` or `FullCov`) is now read and overrides the automatic choice. The `FullCov` method

was rewritten with a vectorised covariance assembly and triangular solves (no explicit matrix inverse), making it roughly 20× faster than before with identical results.

12. **Optimizer tuning keywords.** `Tolerance` and `MaxIterations` expose the Nelder-Mead convergence tolerance and iteration cap; the defaults reproduce the previous behaviour. Tightening `Tolerance` is useful when a run appears to stop short of the maximum likelihood.

12.15 Version 3.1.0

1. **Correction of the gapped-data trend uncertainty and log-likelihood (reported by John Langbein).** John Langbein discovered a serious flaw in the software which underestimated the trend uncertainty and changed the log-likelihood value. The effect only occurs in time series with gaps and is more significant in long time series with high spectral indices and many gaps. For example, in a time series of 40 years with random-walk noise and 50 % gaps, the trend uncertainty was underestimated by about 80 % (a factor of five) and the log-determinant of the covariance matrix was too large by roughly 6. The cause was the Chan circulant matrix being used as a direct approximation of the inverse gap matrix. It is now used only as a preconditioner in a conjugate-gradient solver that recovers the exact inverse, and the log-determinant correction is obtained by stochastic Lanczos quadrature. Anyone analysing gapped time series — especially long records with red noise — should upgrade to version 3.1.0.
2. **FFTW-accelerated gap matvec.** The conjugate-gradient solver and the log-determinant estimator apply the gap matrix through a batched, multi-threaded FFTW3 extension whose plans are created once and reused across all iterations. This keeps the exact correction fast: the residual cost over the previous (approximate) version is only about 1.3× per evaluation for typical gapped series, and gap-free series are unchanged.

12.16 Version 3.1.1

1. **Fix for the Matérn noise model with gaps (steep index and long correlation).** When the Matérn model was combined with a small value of the frequency parameter λ and a steep spectral index (for example $\kappa = -1.6$), the gap-correction step could abort with a "matrix is not positive definite" error. The cause was the Chan circulant preconditioner, which loses positive definiteness when the autocovariance has not decayed within the length of the series. Hector now detects this case and falls back to the exact gap Gram matrix, so the trend, its uncertainty and the log-likelihood remain exact (identical to the full-covariance method). The fall-back is somewhat slower but only triggers for these extreme long-memory cases.
2. **More digits for the regression scale factor (reported by John Langbein).** The scale factor of a multivariate covariate (for example atmospheric pressure) is now printed with four decimals instead of two. The JSON output already contained the full-precision value.
3. **Single-threaded option for parallel runs.** Setting the environment variable `HECTOR_FFTW_THREADS=1` forces the FFTW gap matrix-vector product to use a single thread. This avoids CPU oversubscription when many Hector processes run in parallel (for example a Monte-Carlo study using a process pool); the default remains multi-threaded for single-process use.

12.17 Version 3.1.2

Clearer error messages when input files are malformed (reported by John Langbein, who hit an obscure crash on a control file with a stray line). Hector now tells you what is wrong and on which line, instead of stopping with a Python traceback, and always exits with a non-zero status on error so that scripts detect the failure.

1. **Control-file parsing.** A keyword without a value (a common cause is a left-over `EOF` line from a here-document) now reports, for example, `Error in control file 'estimatetrend.ctl', line 17: keyword 'EOF' has no value.` A missing required keyword such as `OutputFile` is named explicitly rather than raising a `KeyError`.
2. **Data-file (`.mom`) reading.** Every malformed input is reported with the file and line number: a missing `# sampling period` header (which previously could hang the program), a non-numeric value, a `# exp`/`# log`/`# tanh` header without its time constant, epochs that are out of order or duplicated, a wrong number of columns, and a file with no data rows. A missing `DataDirectory` now defaults to the current directory instead of silently producing an empty run.
3. **Regression tests.** A new error-handling test suite (`tests/test_input_errors.py`) checks that each of these mistakes produces a clear message and a non-zero exit code, and it runs as part of the release quality-control gate.

12.18 Version 3.1.3

Fixes for the Generalized Gauss-Markov noise model and the optimiser, plus support for very long (high-rate) time series. Reported by John Langbein.

1. **GGM spectral index no longer depends on `GGM_1mphi`.** For a strongly non-stationary series (power-law index ~2.3) the estimated index used to track the fixed `GGM_1mphi` value (e.g. −1.4 instead of −2.3). The cause was an over-conservative "danger line" in the GGM penalty that cut deep into the numerically valid region. The GGM covariance is in fact positive definite for every `d>0` in exact arithmetic; the real limit is the double-precision `2F1` recursion losing accuracy only for a large `d` combined with a very small `1-phi`. The valid region and the boundary Hector enforces are documented in the noise-model section (see the "Numerically valid region" figure).
2. **The minimiser keeps its best solution instead of aborting.** When the Nelder-Mead search cannot meet a very tight `Tolerance` — which can be below the numerical precision of the log-likelihood for gapped data — Hector now reports the best solution found with a warning, matching Hector 2, rather than stopping with "Minimisation failed". (Internally, the penalty no longer mutates the optimiser's own working array.)
3. **Very long / high-rate series.** The fast GSA solver allocated its FFTW plans eagerly up to a fixed size, which capped analyses at ~32,768 samples. Plans are now built lazily on first use, so short series stay light while series up to ~8.4 million samples (high-rate GNSS) are supported.
4. **Regression test.** `tests/test_ggm_index.py` runs John Langbein's PL2.3 series and checks that the estimated index is ~−2.3 for every `GGM_1mphi`.

12.19 Version 3.1.4

Fix for the Generalized Gauss-Markov noise model with a *free* `1-phi` parameter on long time series. Reported by John Langbein.

1. **GGM with free `1-phi` no longer crashes on long series.** Estimating GGM noise with both the spectral index and `1-phi` free used to abort on long series (e.g. 30 years of daily data) with an `mpmath` error (`hypercomb() failed to converge ...`), or to spend seconds per likelihood evaluation, while fixing `GGM_1mphi` to a small value worked fine. The cause was not the parameter region but the evaluation of the two `2F1` seed values: for a moderate `1-phi` (roughly between `50/m` and 0.106, which includes the optimiser's starting point 0.1) `mpmath`'s internal transformation suffers a cancellation that grows with the series length. In that band Hector now computes the seeds with a direct Gauss series (all-positive terms, no cancellation), keeping `mpmath` for small `1-phi` where it converges instantly; both agree to ~1e-14 where they overlap. See the expanded discussion and the updated "Numerically valid region" figure in the GGM noise-model section.
2. **Regression tests.** `tests/test_ggm_float.py` reproduces the crash setup (a 11,333-point strongly non-stationary series with `NoiseModels GGM White` and no `GGM_1mphi` keyword) and checks that the run completes and recovers the correct index. `tests/test_ggm_band.py` additionally sweeps the GGM covariance computation across the entire fragile `1-phi` band (under a watchdog, so a reintroduced hang fails instead of freezing the suite) and cross-checks the direct Gauss series against `mpmath` where both are reliable.

12.20 Version 3.1.5

New feature, suggested by Janusz Bogusz.

1. **Modified standard deviation.** For the power-law and GGM noise models (including `FlickerGGM`/`RandomWalkGGM`), `estimatetrend` and the other tools now also report the *modified standard deviation* of Gobron et al. (2021, Eq. 5): the expected sample standard deviation of the noise component over a fixed reference span (new keyword `ReferenceSpan`, default 8 years), expressed in the physical unit. Unlike the customary mm/yr^(-kappa/4) amplitude, this value has the same unit for every spectral index, so it can be compared — and differenced — across stations. Printed as `mod. std = ... mm (over 8 yr)` and stored in the JSON as `modified_std` (with `reference_span`). See the new section "Comparing Noise Amplitudes Between Stations" for the definition and for the (small, quantified) difference with the finite-past convention of the original paper.
2. **Regression test.** `tests/test_modified_std.py` checks the value against an exact white-noise result, dense-matrix linear algebra, a Monte Carlo simulation of the process definition, the finite-past convention of Gobron et al. (agreement within 1.2%), and an end-to-end estimatetrend run with the `ReferenceSpan` keyword.

This release also contains the results of a code-quality round: the most complex routines were restructured and the test suite was extended to the parts of Hector that had no coverage yet, which uncovered two real defects.

3. **`Powerlaw` no longer collapses silently on red noise.** With `NoiseModels Powerlaw White`, the spectral index started at the generic value +0.1 (blue noise); on time series with negative

spectral index the optimiser then reduced the power-law fraction to zero before the index could become negative, silently returning a pure white-noise fit with far too small trend uncertainties — even for perfectly valid stationary data. The search now starts at flicker noise (kappa=-1), like GGM. (Users who followed the manual's advice to use GGM with a small `GGM_1mphi` instead of `Powerlaw` were never affected.)

4. **Pure-Python GGM fallback fixed for pure power-law.** On installations without the compiled extensions, the pure power-law case (`GGM_1mphi 0`) of the Python fallback returned an all-zero covariance (a code-porting error that the compiled path never hit). Installations with the compiled extensions — all wheels from PyPI — were never affected.

5. **estimatespectrum.** A random-walk (`RandomWalkGGM`) component was missing from the plot legend due to a spelling error in the label table; internally the program was restructured around a single noise-model dispatch table.

6. **Test suite extended from 12 to 16 sections.** New golden/truth tests for `estimatespectrum` (Welch and model PSD), `predicttrenderror` (against dense linear algebra), the rarely used noise models (AR1, Powerlaw, Matern — verified against the textbook Bessel-K correlation — and VaryingAnnual), and the pure-Python GGM covariance fallback.

12.21 Version 3.1.6

1. **`Tolerance` is now relative to the log-likelihood.** It was passed to the minimiser as an *absolute* bound on ln(L). Since ln(L) grows with the number of observations, the same keyword meant something different for every series: on a 30-year daily record with ln(L) ~ 26783, the documented `Tolerance 1e-8` asks for thirteen significant digits, while the calculation on gapped data is accurate to nearer 1e-2. That target can never be met, so convergence came down to chance – if the simplex happened to collapse until every vertex returned an identical value the run stopped (~100 iterations), otherwise it ground on to `MaxIterations` (10000) and warned. Changing a *fixed* GGM `1-phi` by 1.4 per cent was enough to flip between the two. `fatol` is now `Tolerance` x |ln L|, so the keyword means "this many significant digits" whatever the length of the series. Reported by John Langbein; reproduced on a synthetic 11333-point series, which went from over ten minutes without converging to 75 iterations in five seconds, returning the same spectral index. All examples are unchanged.

2. **The parameter tolerance is relative as well.** `xatol` was compared against one absolute number for parameters living on very different scales. In a typical GGM+White fit the three parameters end near 0.78, -2.29 and 1.5e-3, so the default 1e-6 meant 1.3e-6, 4.4e-7 and 6.8e-4 relative – the GGM `1-phi` was pinned some 1500 times more loosely than the spectral index, and at `1-phi` ~ 7e-6 it was not constrained at all. Unlike the log-likelihood case this did not stop a run; it stopped one *too early*, and reported an imprecise `1-phi` without saying so. The search now runs in units of the starting values, so one tolerance means the same relative thing for every parameter. The search path is unchanged – the initial simplex was already built from 5 per cent relative steps – and all examples reproduce exactly. A parameter that ranges over decades, as `1-phi` does, would be better searched in log space; that remains future work.

3. **`1-phi` is printed in scientific notation.** At the values actually used (e.g. 6.9e-6) the previous fixed-point format printed `0.0000`. John Langbein's suggestion.

12.22 Version 3.1.7

1. **`removeoutliers` now plots NCF (NetCDF) files.** The `-graph`, `-png` and `-eps` flags were silently ignored for `.ncf`/`.nc` input: the multi-channel code path returned before the plotting code was reached, so no figure appeared and no message explained why. Each processed channel now gets its own figure – observed in blue, cleaned in red, with a calendar date axis – saved as `data_figures/<station>_<column>.png` (or `.eps`) so that the channels of a multi-channel file do not overwrite each other. Reported by Francisco Neves.

12.23 Version 3.1.8

1. **The dense missing-data matrix F is built lazily.** Only the AmmarGrag likelihood method reads F (size $m \times n_{\text{gaps}}$); ordinary least-squares (pure white noise) and FullCov never touch it. F was nevertheless allocated whenever a file was read, so a long series with many gaps – e.g. 220,000 samples of which 44% are flagged, for which F would need 155 GB – could not be analysed at all, even with a white noise model that never needs the matrix. F is now materialised only when AmmarGrag asks for it, with a preemptive check against available memory that explains the alternatives (`LikelihoodMethod FullCov`, or a pure `White` noise model) instead of letting the operating system kill the process.
2. **Gap classification is exact at high sampling rates.** Reading a time series inserted NaNs by stepping through each gap one sampling period at a time; at 800 Hz (sampling period 1.4×10^{-8} days) the NCF reader's absolute tolerance exceeded the sampling period, so high-rate NetCDF files could not be read at all, and the mom reader drifted off the grid across long gaps because the `# sampling period` header kept only ~5 significant digits. All readers now count whole sampling periods between consecutive epochs (no accumulation), snap the sampling period to the file's own time span, and `momwrite` stores sub-10^{-4}-day periods in scientific notation. See the new section *Sampling rates and time-axis precision*.
3. **`removeoutliers` and `estimatetrend` report out-of-memory conditions with a clear message and exit code 1** instead of a Python traceback (or a silent kill on Linux).

12.24 Version 3.1.9

1. **Numerical-stability guard in the Generalised Schur Algorithm.** At every step of the recursion the reflection coefficient is checked to satisfy $|\gamma_k| < 1$, the defining property of a numerically positive-definite Toeplitz matrix. On violation (which can only occur at extreme ill-conditioning, near condition numbers of 10^{15}) Hector no longer produces silent NaNs but falls back automatically to the Durbin-Levinson recursion, and ultimately to the dense `FullCov` path. On healthy data the results are bit-identical to version 3.1.8.
2. **The paper scripts ship with the package.** `hector-examples` now also unpacks `paper_scripts/`, containing the benchmark drivers, the synthetic series generators, and the raw results of the paper *"Faster analysis of GNSS time series"* (Journal of Geodesy): `run_all.py` re-creates every figure of that paper from the shipped results (about 5 minutes) or regenerates them from scratch (`--full`), and `setup_tools.py` locates or asks for the optional external tools (Hector C++ v2.2, R with the gmwmx2 package) used by the timing comparisons, which are skipped gracefully when absent. The same material is archived at https://doi.org/10.5281/zenodo.22889030.

12.25 Version 3.1.10

1. **Matern spectral index corrected — old Matern κ values must be doubled.** Up to version 3.1.9 (and in the C++ version 2.2) the Matern model used $\alpha = 2d$ while it reported $\kappa = -2d$, so the printed and stored κ was only half of the true spectral slope -2α of (Lilly et al. 2016). Now $\alpha = d$, so κ is the true slope, with the same meaning as for the other power-law models. This also applies to `kappa_fixed` in `estimatetrend.ctl` and `simulatenoise.ctl` and to the Matern spectrum drawn by `estimatespectrum`. **Multiply Matern κ values obtained with older versions by two** to get the actual spectral index. Because a Matern process needs $\alpha > 1/2$ for a finite variance, κ is restricted to $-3 < \kappa < -1$: the model cannot represent noise flatter than flicker noise (see the section on the Matern noise model). A `kappa_fixed` of -1 or higher is now refused with an error message instead of silently producing a meaningless covariance.
2. **Keeping only one of the two Matern parameters fixed no longer crashes.** Setting just `kappa_fixed` or just `lambda_fixed` stopped with an `AttributeError`.
3. **Combinations of three or more noise models are now estimated correctly — older fits with three or more models, two of them with free parameters, should be rerun.** Two errors in the combination of noise models have been fixed:
   - each noise model now checks its own parameters against its own bounds. Before, every model checked the parameters of the *first* model, which gave wrong estimates whenever at least two models had free parameters (for example `Powerlaw AR1 White` returned $\kappa = 0$ for data with $\kappa = -0.8$, and `GGM Matern White` did not converge). **Results of such combinations obtained with versions 3.0.0–3.1.9 should be recomputed.**
   - the fraction of the last noise model is now one minus the sum of all other fractions, so the reported fractions add up to one. For combinations in which at most one model has free parameters, such as `FlickerGGM RandomWalkGGM White`, the estimated noise amplitudes, trend and uncertainties were already correct and are unchanged; only the reported fractions were wrong.

   Combinations of two noise models, such as the recommended `GGM White`, are not affected by any of these changes.

12.26 Version 3.1.11

1. **Offsets without observations between them no longer crash `estimatetrend`.** Two offsets inside the same data gap, or an offset before the first or after the last observation, gave the design matrix two identical columns (or a copy of the intercept or a column of zeros). The program then stopped with “Singular matrix” or “Matrix is not positive definite”, or, worse, returned a meaningless trend without any warning. As in the C++ version 2.2, such offsets are now removed: offsets outside the observation span are ignored and of several offsets between two consecutive observations only the last one is kept. Only real observations count, so this also applies to gaps created by `removeoutliers`. Each removed offset is reported in the output. Reported by Janusz Bogusz.

12.27 Version 3.1.12

1. **Redundant offsets are now removed exactly as in Hector 2.2.** When several offsets fall between two consecutive observations, the first one listed in the header is kept (version 3.1.11

kept the last one). The estimated trend, offsets and other parameters do not depend on this choice, but the log-likelihood, AIC and BIC do slightly, because the gap epochs enter the determinant of $H^T H$. With this change all of them agree with Hector 2.2.

12.28 Version 3.1.13

1. **The phase of periodic signals now refers to 1 January 2000, as in Hector 2.2.** Up to version 3.1.12 the cosine and sine terms were computed from the first epoch of the time series, so the estimated phase (and the separate cosine and sine coefficients) depended on the day of the year on which the series started. Now t_0 is 1 January 2000 (MJD 51544), so the annual phase is the phase within the calendar year and can be compared between stations and with Hector 2.2. The fitted model, trend, offsets and noise parameters are not affected. The amplitude can change slightly when the uncertainties of the cosine and sine terms differ, because its computation uses their mean. **Cosine, sine and phase values obtained with versions 3.0.0–3.1.12 differ from those of this version.**
2. **The phase uncertainty now follows Hector 2.2.** The Monte Carlo estimate of the phase uses the separate uncertainties of the cosine and sine terms instead of their mean, and a fixed random seed, so that the printed phase no longer changes slightly from run to run.
3. **`ARMA` with more than one AR coefficient and `ARFIMA` work again.** Since version 3.0 they stopped at the first likelihood evaluation (`IndexError` or "Matrix is not positive definite"). The search now starts, as in Hector 2.2, with all coefficients equal to 0.1, and the AR roots are always computed correctly. The results agree with Hector 2.2 (Example 2: AR(5) and ARFIMA(1,0)).
4. **Post-seismic and slow-slip terms.** `# log` headers were written to `OutputFile` as `# exp`, and any file with a `# tanh` header made `estimatetrend` stop while writing `OutputFile`. The slow-slip term tanh is now applied at all epochs, as in the model equation and in Hector 2.2 (it was set to zero before t_k). The keywords `estimatepostseismic` and `estimateslowslipevent` again default to no, as in Hector 2.2 and as documented; since version 3.0 they defaulted to yes. **Fits of files with `# exp`, `# log` or `# tanh` headers made with versions 3.0.0–3.1.12 should be redone.**
5. **`UseResiduals yes` now analyses the residuals.** It subtracted the `<channel>_residual` channel, so the model was analysed instead.
6. **`TimeNoiseStart` works.** `simulatenoise` simulated the extra points but did not discard them, so the spin-up was always included; now, as in Hector 2.2, the first `TimeNoiseStart` points are skipped. The keyword is optional (default 0). Example 3 has been regenerated; its input file `simulatenoise.inp` contained an extra number, so the series were simulated with the wrong noise amplitudes.
7. **`phi_ar1_fixed` works** (it made the `AR1` model crash or was ignored).
8. **`findoffsets` on `.ncf` files stores the detected offsets** in `OutputFile`, as offset events of the analysed channel, and `find_all_offsets` no longer drops the existing per-channel offset and post-seismic variables when it writes its `.ncf` files.
9. **`find_all_offsets`: the multivariate test statistic is now $T_3 = 2\sum \Delta \ln L$, as documented.** Up to version 3.1.12 the plain sum $\sum \Delta \ln L$ was compared with the threshold, so the default 16.27 ($\chi^2(3)$, $\alpha = 0.001$) acted as $T_3 > 32.5$ ($\alpha \approx 4 \times 10^{-7}$). With this version more offsets are detected; `find_all_offsets_ncf.json` gives both `T3` and the summed `delta_lnL`. The default stays 16.27, but for real GNSS series a threshold between 30 and 70 is advised (section `find_all_offsets`); `-t 32.54` reproduces the results of earlier versions.

The pre-computed `obs_files/` of Example 6 have been regenerated.

10. **`removeoutliers` on `.ncf` files uses all keywords** of the control file (e.g. `DegreePolynomial`, post-seismic terms), not only a fixed subset, and stops with a clear message when neither `IQ_factor` nor `Spike_factor` is given.
11. Smaller fixes: the GGM output marks the fixed one of d and $1 - \phi$; `estimate_all_trends -phi` writes the value with full precision; leftover debug output was removed from `removeoutliers`, `estimatespectrum` and `estimatetrend`; `estimatetrend.json` no longer contains the misspelt key `PhyiscalUnit`; the non-existing program `modelspectrum` is no longer installed (use `estimatespectrum -model`); `hector-examples` describes the examples correctly.
12. **Manual.** The worked examples, keyword tables, defaults and command-line flags were checked against the code and corrected throughout.

12.29 Version 3.2.0

1. **Exact gap correction without approximations.** For time series with missing data, the log-likelihood needs the gap matrix $\mathbf{M} = \mathbf{F}^T\mathbf{C}^{-1}\mathbf{F}$, the block of the inverse covariance matrix that belongs to the missing epochs. Versions 3.1.x approximated it by a circulant (Chan) matrix, used as a preconditioner of a conjugate-gradient solver, and estimated its log-determinant with stochastic Lanczos quadrature (eight random probes, fixed seed). The trend and its uncertainty were correct, but the log-likelihood carried an error of a few hundredths (up to 0.074 at 40 % missing data), which entered likelihood-ratio tests and information criteria. Version 3.2.0 builds **M** exactly from the two Gohberg–Semencul vectors that the Toeplitz solver already provides, with one running sum per gap-epoch separation, and factorises it with a Cholesky decomposition. The log-likelihood now agrees with the full-covariance method to about 10^{-9}, nothing depends on random numbers, and a run is 1.1–1.4 times faster than with version 3.1.13 on one core (10 % missing data). The Matérn fall-back of version 3.1.1 is no longer needed, because the exact matrix is always used. **Log-likelihood, AIC and BIC values of gapped series change in the second or third decimal** compared with versions 3.1.0–3.1.13; the estimated parameters change negligibly.
2. **Single-threaded by default.** Hector restricts the linear-algebra library underneath NumPy to one thread (see the note in the installation chapter), so that several Hector programs can run side by side without competing for the cores. Setting any of `OMP_NUM_THREADS`, `OPENBLAS_NUM_THREADS`, `MKL_NUM_THREADS`, `VECLIB_MAXIMUM_THREADS` or `NUMEXPR_NUM_THREADS` yourself restores multi-threading. The variable `HECTOR_FFTW_THREADS` of version 3.1.1 no longer exists, because the FFTW gap matrix-vector product it controlled has been removed.
3. **No crash when a factorisation fails.** If the noise parameters tried by the optimiser give a covariance or gap matrix that cannot be factorised in double precision, that trial is now rejected and the search continues, instead of `estimatetrend` stopping with “Matrix is not positive definite”.
4. **Paper scripts.** `hector-examples` unpacks the scripts and raw results of the revised paper *“Faster analysis of GNSS time series”*: besides the figures, `run_all.py` now also re-creates Table 4 (the comparison with Hector 2.2, including the number of likelihood evaluations under both stopping rules) and the single-core measurements (threads, memory, parallel runs). The scripts can be started from any folder, also one whose path contains spaces. The same material is archived at https://doi.org/10.5281/zenodo.23617991.

12.30 Version 3.2.1

1. **`simulatenoise` reads ϕ of `VaryingAnnual` from `phi_varying_fixed`** (reported by Matt King), the keyword that `estimatetrend` uses. It read only `phi_fixed`, so with a control file written for `estimatetrend` it asked for ϕ; when the answers were given in an input file, the noise amplitude was taken as ϕ and the simulated series were far too small. `phi_fixed` is still accepted.
2. **New keyword `RandomSeed` in `simulatenoise`** (reported by Matt King). `RepeatableNoise yes` always uses seed 0, so components simulated in separate runs were correlated. `RandomSeed` makes runs reproducible and, with different seeds, independent.
3. **Durbin–Levinson recursion** (reported by Matt King). The prediction-error variance is now updated as $\delta_{k+1} = \delta_k(1-\gamma_k^2)$, which avoids cancellation for strongly correlated noise, and a covariance matrix that is not positive definite is rejected (the trial is skipped, as in version 3.2.0) instead of stopping the pure-Python version with `math domain error`. The recursion is used for series of up to 1000 points and when the generalised Schur algorithm fails.
4. **Warning when the compiled Toeplitz solver is missing.** If the compiled generalised Schur extension cannot be loaded, Hector now says so once instead of silently using the slower Durbin–Levinson recursion.
5. `VaryingAnnual` with an unsupported time-series format stops with its intended message and a non-zero exit code, instead of a `NameError`.
6. **The `VaryingSemiAnnual` noise model is back** (requested by Matt King). As in Hector 2.2 it is the `VaryingAnnual` model at a period of 182.625 days, and both use `phi_varying_fixed`. It is available in `estimatetrend`, `simulatenoise`, `estimatespectrum`, `predicttrenderror` and, with the abbreviation `VSA`, in `estimate_all_trends`.
7. **Model spectrum of `VaryingAnnual`.** The modelled power spectrum that `estimatespectrum` draws for `VaryingAnnual` was $4/\pi$ times too large; it now integrates to the variance of the model. The formula in the manual was a factor 2 too large. The estimated noise parameters are not affected.
8. **`predicttrenderror` with `VaryingAnnual`.** The sampling period was taken as zero, so the cosine of the model was always one and the model acted as an AR(1) process. It now uses the sampling period given with `-dt`.
9. **Faster with missing data and with the varying seasonal models** (after a patch by Matt King). The positions of the missing data are now found once per likelihood evaluation from the data instead of by a search through every column of the gap matrix: with 10 % missing data an `estimatetrend` run takes about a quarter less time, with identical results. The covariance of `VaryingAnnual` and `VaryingSemiAnnual` is computed without Python loops, which makes a run with one of these models about 15 % faster; the results change only in the last digits.

12.31 Version 3.2.2

1. **A trajectory model that cannot be estimated is reported, not computed.** When some terms of the model are (nearly) indistinguishable in the data, for instance periodic signals whose periods are nearly identical over the length of the series, `estimatetrend` stops with a message that names the pairs of periods concerned and how far they drift apart over the series, instead of a Python traceback (“Matrix is not positive definite”). This also catches models that Hector 2.2 accepted without an error but for which it printed meaningless numbers. A period that is longer than the series, or one given twice, is reported as well.

2. **Warnings for Hector 2.2 keywords that are no longer used.** Hector 3 ignores `estimatevaryingseasonal`, `subtractsignal`, `interpolate`, `WindowFunction`, `OffsetFile`, `Kashyap`, `BIC_c_ExtraPenalty`, `beta_size`, `beta_spacing`, `MonteCarloConfidence` and `TrueParams`. They are still ignored, but Hector now says so when they would have changed the result in Hector 2.2 (for example `estimatevaryingseasonal yes` or `WindowFunction Parzen`). Both issues were found with a control file sent by Janusz Bogusz.
3. The paper scripts ship with the results of Hector 3.2.1 (comparison runs, gap-fraction sweep with single-threaded linear algebra, and the single-core measurements).

References

Agnew, Duncan Carr. 1992. "The time-domain behaviour of power-law noises." *Geophys. Res. Letters* 19 (4): 333–36.

Akaike, Hirotugo. 1974. "A new look at the statistical model identification." *IEEE Transactions on Automatic Control* 19 (6): 716–23.

Amiri-Simkooei, Ali Reza, Mehdi Hosseini-Asl, Jamal Asgari, and Fazileh Zangeneh-Nejad. 2019. "Offset Detection in GPS Position Time Series Using Multivariate Analysis." *GPS Solutions* 23 (1): 13. https://doi.org/10.1007/s10291-018-0805-z.

Beran, J. 1992. "Statistical methods for data with long-range dependence." *Statistical Science* 7 (4): 404–16.

Bevis, M., and A. Brown. 2014. "Trajectory models and reference frames for crustal motion geodesy." *Journal of Geodesy* 88 (March): 283–311. https://doi.org/10.1007/s00190-013-0685-5.

Bos, M. S., R. M. S. Fernandes, S. D. P. Williams, and L. Bastos. 2013. "Fast Error Analysis of Continuous GNSS Observations with Missing Data." *J. Geod.* 87 (4): 351–60. https://doi.org/10.1007/s00190-012-0605-0.

Bos, M. S., S. D. P. Williams, I. B. Araújo, and L. Bastos. 2014. "The effect of temporal correlated noise on the sea level rate and acceleration uncertainty." *Geophysical Journal International* 196 (March): 1423–30. https://doi.org/10.1093/gji/ggt481.

Brockwell, P. J., and R. A. Davis. 2002. *Introduction to Time Series and Forecasting*. Second edition. Springer-Verlag, New York.

Buttkus, B. 2000. *Spectral Analysis and Filter Theory in Applied Geophysics*. Springer-Verlag Berlin Heidelberg.

Dam, Tonie M van, and Olivier Francis. 1998. "Two Years of Continuous Measurements of Tidal and Nontidal Variations of Gravity in Boulder, Colorado." *Geophysical Research Letters* 25 (3): 393–96.

Doornik, J. A., and M. Ooms. 2003. "Computational Aspects of Maximum Likelihood Estimation of Autoregressive Fractionally Integrated Moving Average Models." *Computational Statistics and Data Analysis* 42: 333–48.

Gobron, Kevin, Paul Rebischung, Michel Van Camp, Alain Demoulin, and Olivier de Viron. 2021. "Influence of Aperiodic Non-Tidal Atmospheric and Oceanic Loading Deformations on the Stochastic Properties of Global GNSS Vertical Land Motion Time Series." *Journal of Geophysical Research: Solid Earth* 126 (9): e2021JB022370. https://doi.org/10.1029/2021JB022370.

Gobron, Kevin, Paul Rebischung, Michel Van Camp, Alain Demoulin, and Olivier de Viron. 2022. "Impact of Offsets on Assessing the Low-Frequency Stochastic Properties of Geodetic Time Series." *Journal of Geophysical Research: Solid Earth* 127 (1): e2021JB022419. https://doi.org/10.1029/2021JB022419.

Harville, David A. 1974. "Bayesian Inference for Variance Components Using Only Error Contrasts." *Biometrika* 61 (2): 383–85. https://doi.org/10.1093/biomet/61.2.383.

Hosking, J. R. M. 1981. "Fractional differencing." *Biometrika* 68: 165–76.

Kasdin, N. Jeremy. 1995. "Discrete simulation of colored noise and stochastic processes and $1/f^{\alpha}$ power-law noise generation." *Proc. IEEE* 83 (5): 802–27.

Kashyap, Rangasami L. 1982. "Optimal Choice of AR and MA Parts in Autoregressive Moving Average Models." *IEEE Trans. Pattern Anal. Mach. Intell.* 4 (2): 99–104. https://doi.org/10.1109/TPAMI.1982.4767213.

Klos, Anna, Machiel S Bos, Rui MS Fernandes, and Janusz Bogusz. 2019. "Noise-Dependent Adaption of the Wiener Filter for the GPS Position Time Series." *Mathematical Geosciences* 51 (1): 53–73.

Langbein, J., and Y. Bock. 2004. "High-rate real-time GPS network at Parkfield: Utility for detecting fault slip and seismic displacements." *Geophys. Res. Letters* 31 (June): 15. https://doi.org/10.1029/2003GL019408.

Langbein, John. 2004. "Noise in two-color electronic distance meter measurements revisited." *J. Geophys. Res.* 109 (B04406). https://doi.org/10.1029/2003JB002819.

Langbein, John. 2010. "Computer Algorithm for Analyzing and Processing Borehole Strainmeter Data." *Computers & Geosciences* 36 (5): 611–19. https://doi.org/10.1016/j.cageo.2009.08.011.

Langbein, John. 2017. "Improved Efficiency of Maximum Likelihood Analysis of Time Series with Temporally Correlated Errors." *Journal of Geodesy* 91 (8): 985–94. https://doi.org/10.1007/s00190-017-1002-5.

Larson, Kristine M., A. R. Lowry, Vladimir Kostoglodov, et al. 2004. "Crustal Deformation Measurements in Guerrero, Mexico." *Journal of Geophysical Research: Solid Earth* 109 (B4). https://doi.org/10.1029/2003JB002843.

Lilly, Jonathan M, Adam M Sykulski, Jeffrey J Early, and Sofia C Olhede. 2016. “Fractional Brownian Motion, the Matérn Process, and Stochastic Modeling of Turbulent Dispersion.” *arXiv Preprint arXiv:1605.01684*.

Montillet, Jean-Philippe, and Machiel S Bos. 2019. *Geodetic Time Series Analysis in Earth Sciences*. Springer. https://doi.org/10.1007/978-3-030-21718-1.

Patterson, H. D., and R. Thompson. 1971. “Recovery of Inter-Block Information When Block Sizes Are Unequal.” *Biometrika* 58: 545–54. https://doi.org/10.1093/biomet/58.3.545.

Schwarz, Gideon. 1978. “Estimating the Dimension of a Model.” *The Annals of Statistics* 6 (2): 461–64.

Sowell, F. 1992. “Maximum likelihood estimation of stationary univariate fractionally integrated time series models.” *J. Econom.* 53: 165–88.

Williams, S. D. P. 2003. “The effect of coloured noise on the uncertainties of rates from geodetic time series.” *J. Geod.* 76 (9-10): 483–94. https://doi.org/10.1007/s00190-002-0283-4.

Williams, S. D. P. 2008. “CATS : GPS coordinate time series analysis software.” *GPS Solutions* 12 (2): 147–53. https://doi.org/10.1007/s10291-007-0086-4.