Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

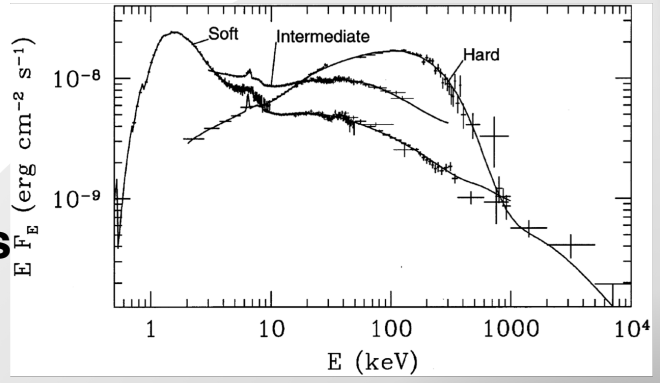
Stingray: Next-Generation Spectral Timing
Optimising Stingray

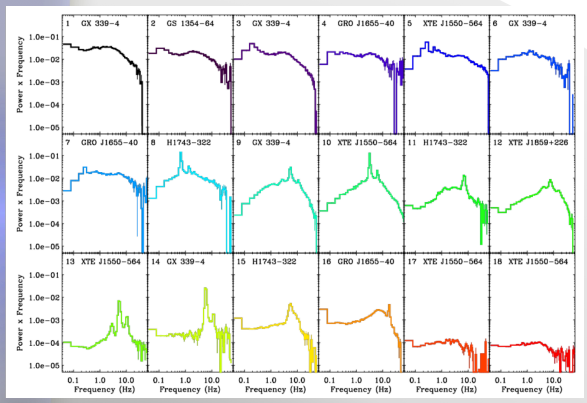
Eleonora Veronica Lai, Matteo Bachetti, Maura Pilia,
+ Daniela Huppenkothen and Stingray developers

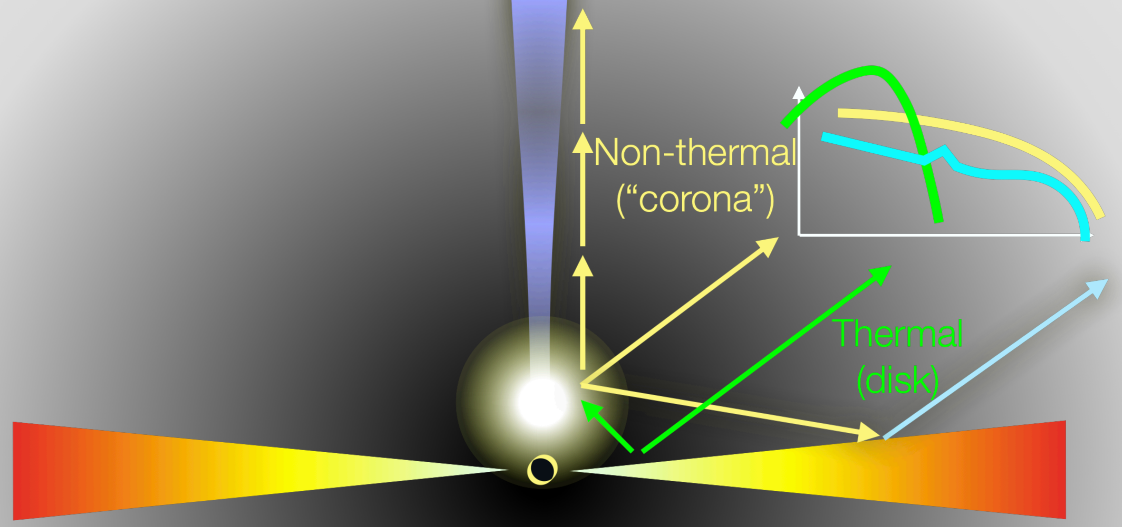
Spoke 3 General Meeting, Elba 5 - 9 / 05, 2024

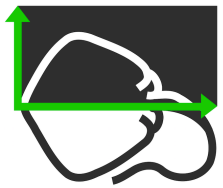
Scientific Rationale

- Some observe spectra, some observe variability. Is possible to use the full information?
- Example: a variable accretion flow that **propagates** through an atmosphere (corona), that **illuminates** the accretion disk and gets **reflected**. Can we disentangle the emission regions?
- Stingray: ease the learning curve for advanced spectral-timing techniques, with a correct statistical framework







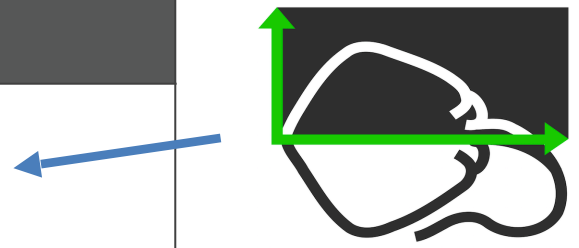


Huppenkothen et al. (2019)

History

- **2016**, Leiden workshop „The X-ray Spectral-Timing Revolution": **Daniela Huppenkothen, Matteo Bachetti, Abigail Stevens, Simone Migliari, Paul Balm** decide the fusion of three existing packages for spectral timing, creating the **Stingray** library and two packages for interactive analysis based on it: **HENDRICS** (batch scripts) and **DAVE** (GUI)
- At the time, various official software packages for X-ray spectral fitting (e.g. XSPEC, ISIS, …), but **no open, well-maintained software for timing or spectral timing**

Spectral analysis	Timing analysis (+ lags)	Spectral timing
•Xspec •Sherpa •ISIS •(…)	•(XRONOS) POWSPEC •SITAR, lsisscripts.sl	X



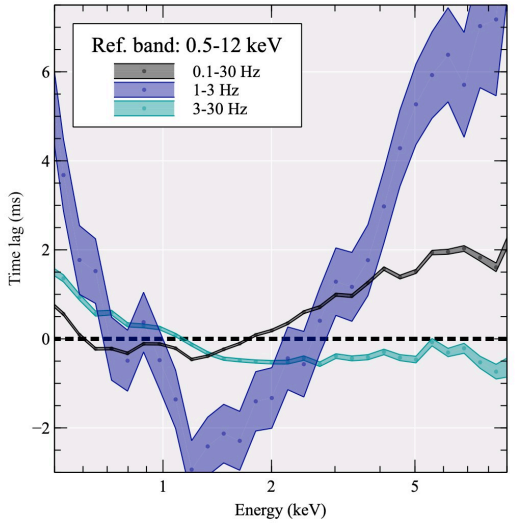
- Today: Stingray is widely used in the X-ray community… even for things **that were not originally planned** (e.g. radio and optical data!)
- Significant code contributions from the community and Google Summer of Code students.

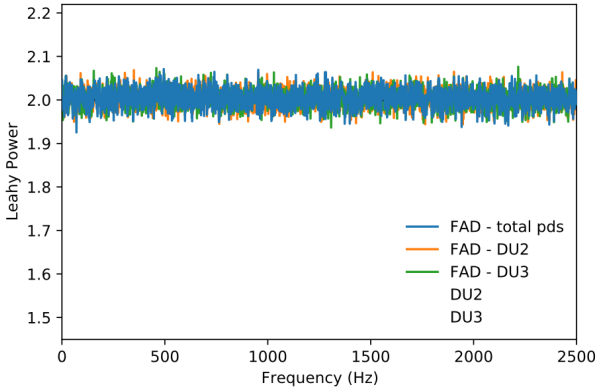
Technical Objectives, Methodologies, and Solutions: what we want

- **"Timing" analysis**
  - Pulsation searches and timing
  - Aperiodic variability, periodogram modelling (ML, Bayesian)
- **Spectral analysis -> connect to Xspec, Sherpa**
  - Continuum modeling
  - Broad lines (e.g. Fe complex, cyclotron lines)
- **Polarimetry (be creative!)**
- **All mixed together! E.g.**
  - Time lags
  - Spectral covariance, spectral polarimetry
  - Phase/Time-resolved spectroscopy and polarimetry
  - Time-resolved-energy-resolved polarimetry (Whatev')

… all with instrument awareness

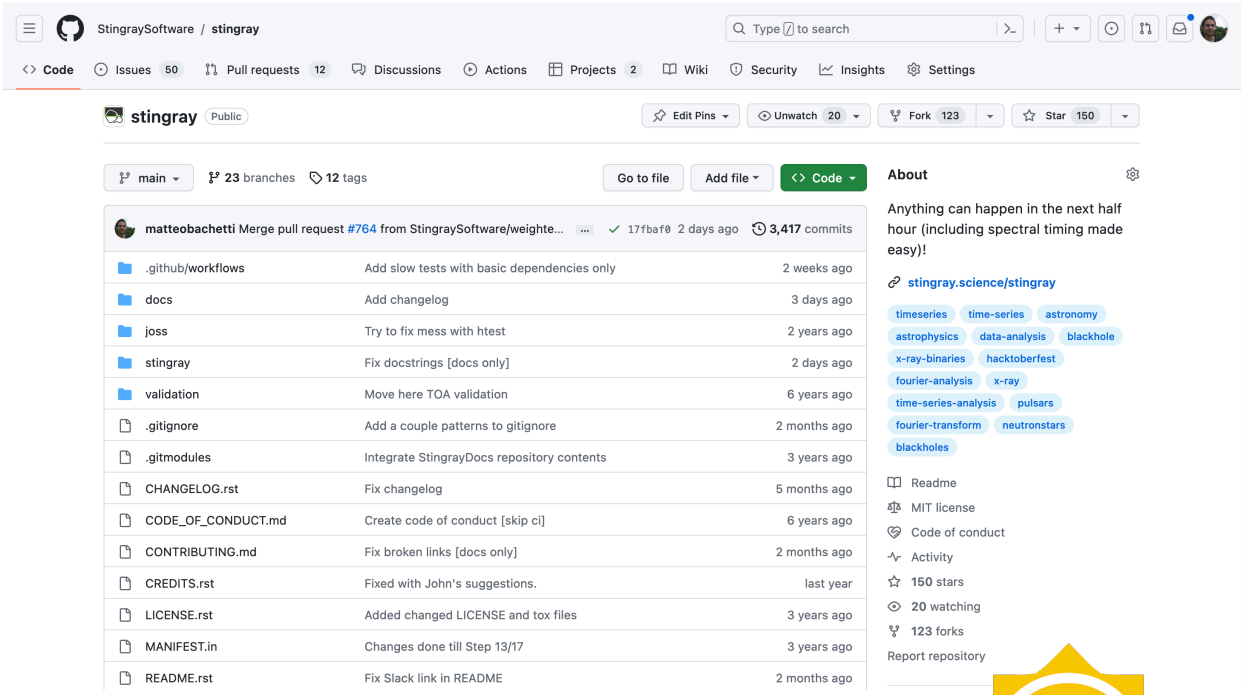
- Be aware of instrumental systematics: dead time, frame time, good time intervals, etc.
- Mission support





Technical Objectives, Methodologies, and Solutions: an open development model

- **Github-based** workflow:
  - Issue tracking
  - Assignments
  - Pull Requests
  - Automatic BM
- Community outreach:
  - Public **Slack** channel
  - Talks
  - Hackatons/Tutorials
  - OpenAstronomy involvement
  - Astropy affiliated package
- Developers:
  - Astronomers
  - Google Summer of Code students



UNIAM
PRIN MIUR 2017

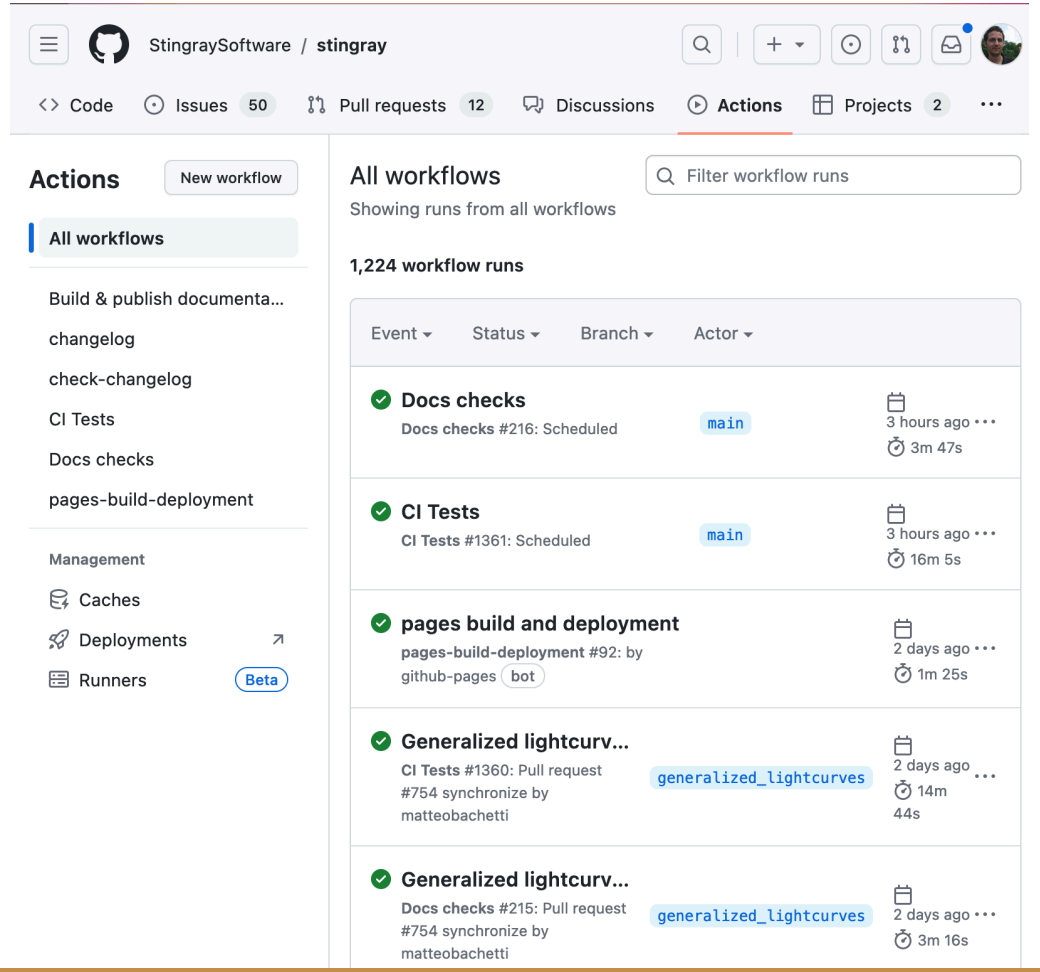
SpecTemPolar!
PRIN INAF 2019

INAF
ISTITUTO NAZIONALE DI ASTROFISICA

Google Summer of Code 2016, 2017, 2018, 2020, 2021, 2022, 2023, 2024

Technical Objectives, Methodologies, and Solutions: reliability and performance testing

- Code correctness
  - Test-based development
  - Literature reproduction
- Regression testing: continuous integration with **Github Actions** and **tox**
  - **Unit tests**
  - **Integration tests**
- Performance
  - Profiling: %**time**, **cProfile**, **line_profiler**, **memray**, etc.
  - Small-dataset testing (< RAM): verify "acceptable" execution times
  - Scalability for larger-than-RAM datasets
- Documentation
  - Use **Sphinx** + **Github Actions** for automatic docs building
  - **Linkcheck** for periodic link checking in the docs



Timescale, Milestones and KPIs

M7 (September 2023 - February 2024)

- Performance and robustness tests on a dataset smaller than the RAM (< 8GB) from accreting black hole. Comparison made with competing (proprietary) code for the main spectral-timing libraries as a function of frequency.
- PR - GitHub: Unevenly sampled: #737 (Lomb-Scargle), #739 (GP); Optimization: #757, #764; Cleanup: #755; Docs: #769, #770, #771
- Formation via USC VIII courses:
  - Theoretical and hands-on training on Containerisation concepts and orchestration (Dec 2023);
  - Fundamentals on Data Management Plan usage and how to structure scientific data collections (Jan 2024).

Timescale, Milestones and KPIs

M8 (March - June 2024)

- New release of the code Stingray v2.0 comprising a list of new tools and libraries (e.g. Power Colors à la Heil 2016, etc.)
- Automatic Benchmarks
- Performance tests on a simulated dataset (up to 90 GB). Comparison made between new release (i.e. v2.0) and previous release as of Oct 2023 (i.e. v1.1.2.4) code for the main timing libraries.
- New X-ray mission in input: data handling for the bulk analysis of the Rossi X-ray Timing Explorer (RXTE) archive.
- Formation via USC VIII courses:
  - Computing and High Performance Computing in Astronomy and Astrophysics (Bologna, June - July 2024)

Accomplished Work, Results

M7 (September 2023 - February 2024)

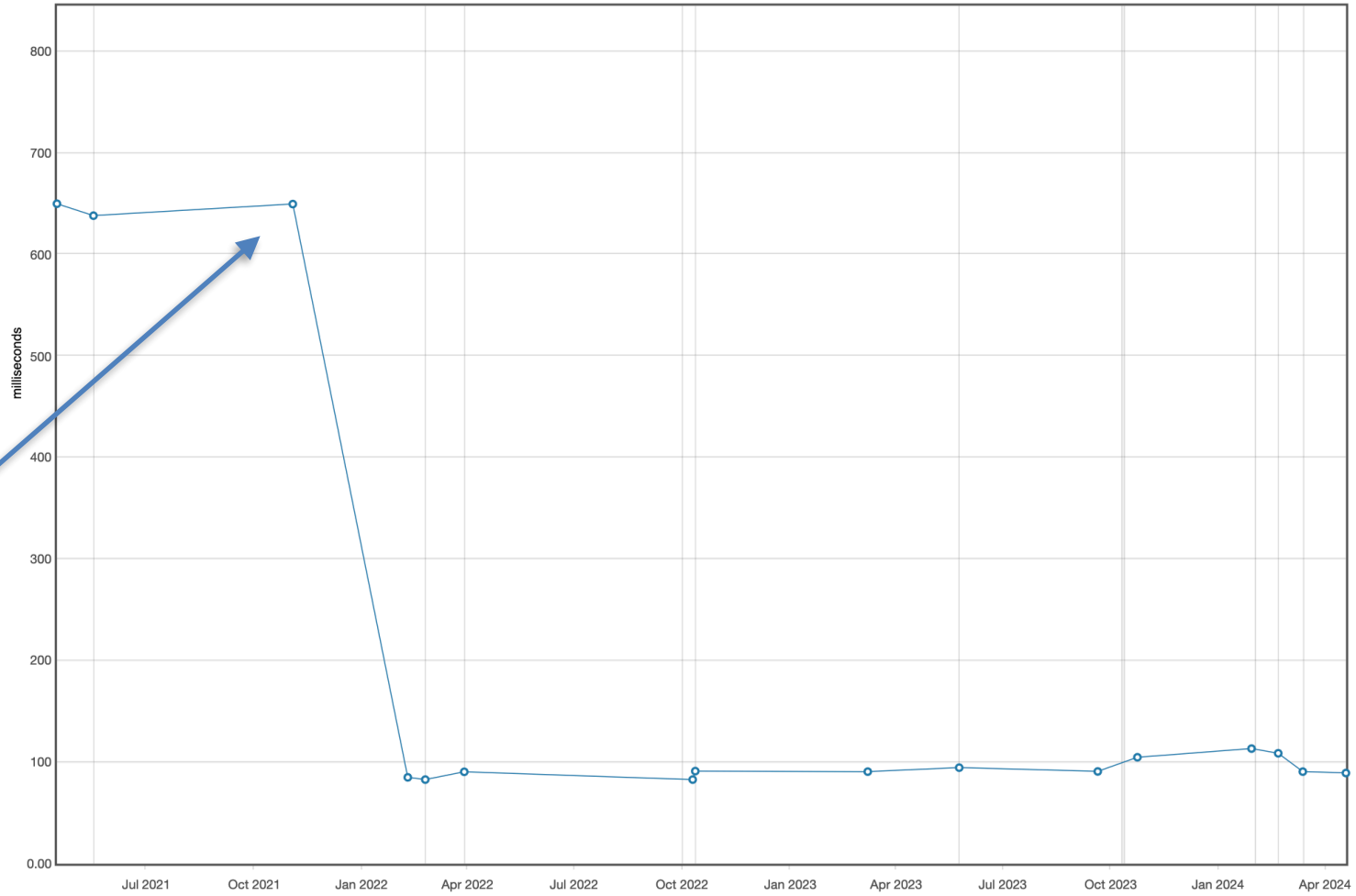
- Performance and robustness tests on a dataset smaller than the RAM (< 8GB) from accreting black hole. Comparison made with competing (proprietary) code for the main spectral-timing libraries as a function of frequency.
  - **Performance**: speed test of many (spectra) timing products: lightcurves, (averaged, single) **periodograms** and cross spectra, lags, coherence (intrinsic, raw)
  - **Robustness**: are results compatible?
- Results:
  - **Performance:** excellent, code generally faster than competitor
  - **Robustness**: good, found one possible bug in the computation of intrinsic coherence.

Accomplished Work, Results

M8 (March - June 2024)

Simple benchmarks to track Stingray's performance comparing the different releases through the years to today!

Also me :)
after FOXT conference

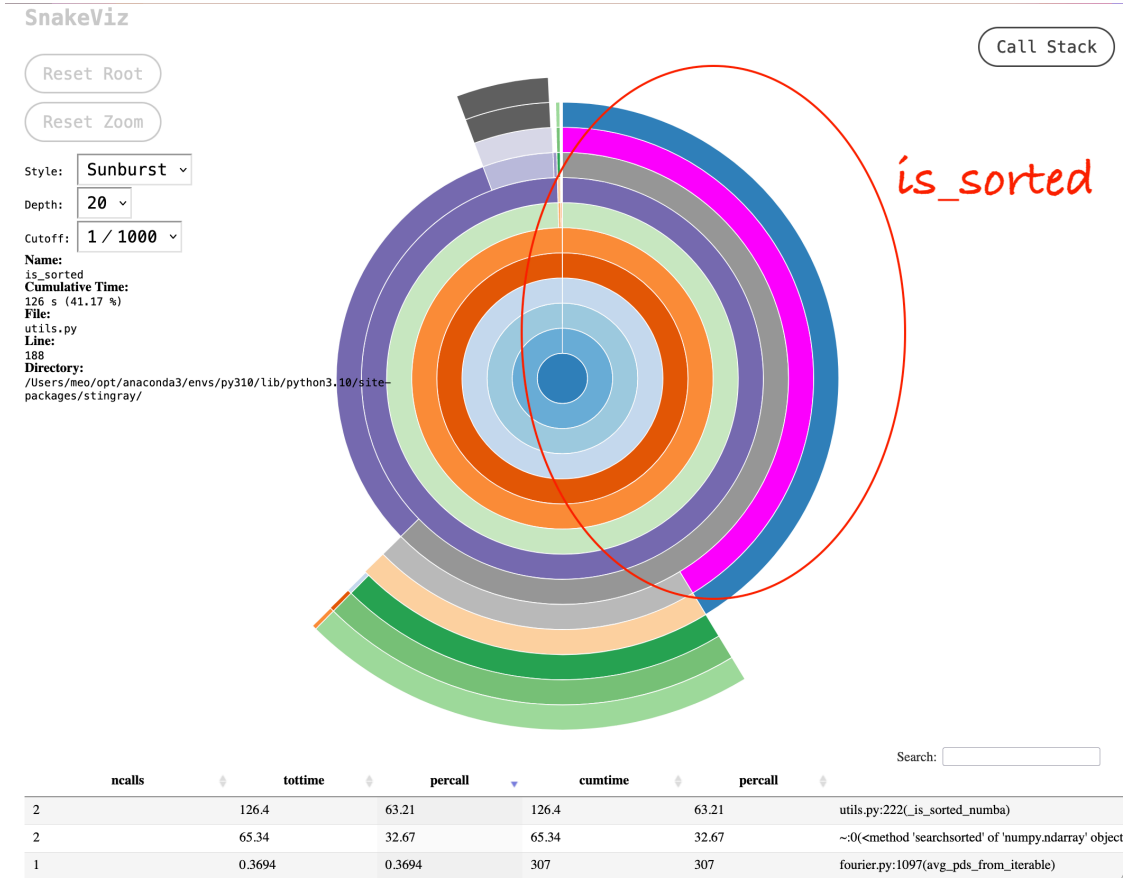


Accomplished Work, Results

M8 (March - June 2024)

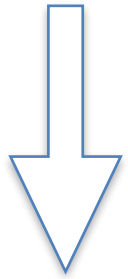
- Performance tests on a simulated dataset (up to 90 GB).
  Comparison made between the latest release (i.e. v2.0) and previous release as of Oct 2023 (i.e. v1.1.2.4) of the main libraries associated with the timing products (averaged and single power spectrum)

Accomplished Work, Results - M8



ncalls	tottime	percall	cumtime	percall	
2	126.4	63.21	126.4	63.21	utils.py:222(_is_sorted_numba)
2	65.34	32.67	65.34	32.67	~:0(<method 'searchsorted' of 'numpy.ndarray' object
1	0.3694	0.3694	307	307	fourier.py:1097(avg_pds_from_iterable)

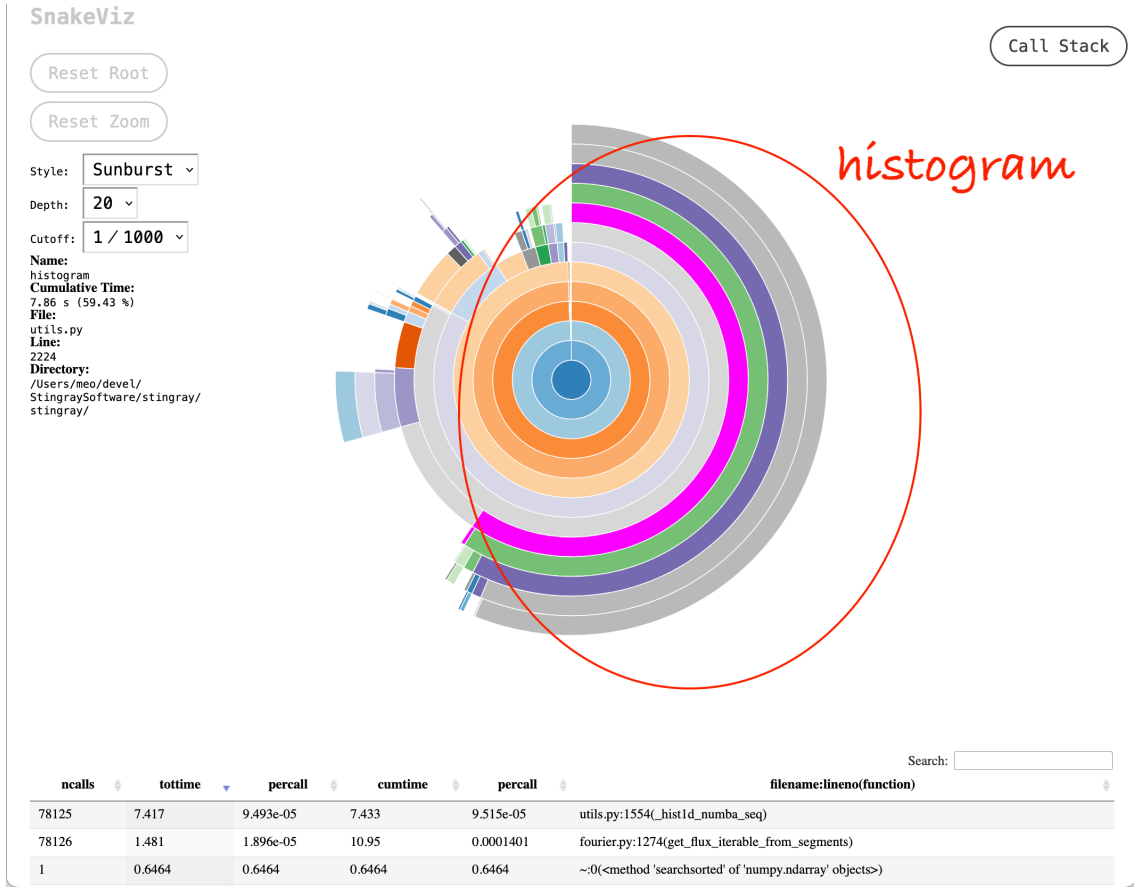
In v1.0, we found that is_sorted is the library dominating the execution time



In v2.0, is_sorted is eliminated

Accomplished Work, Results - M8

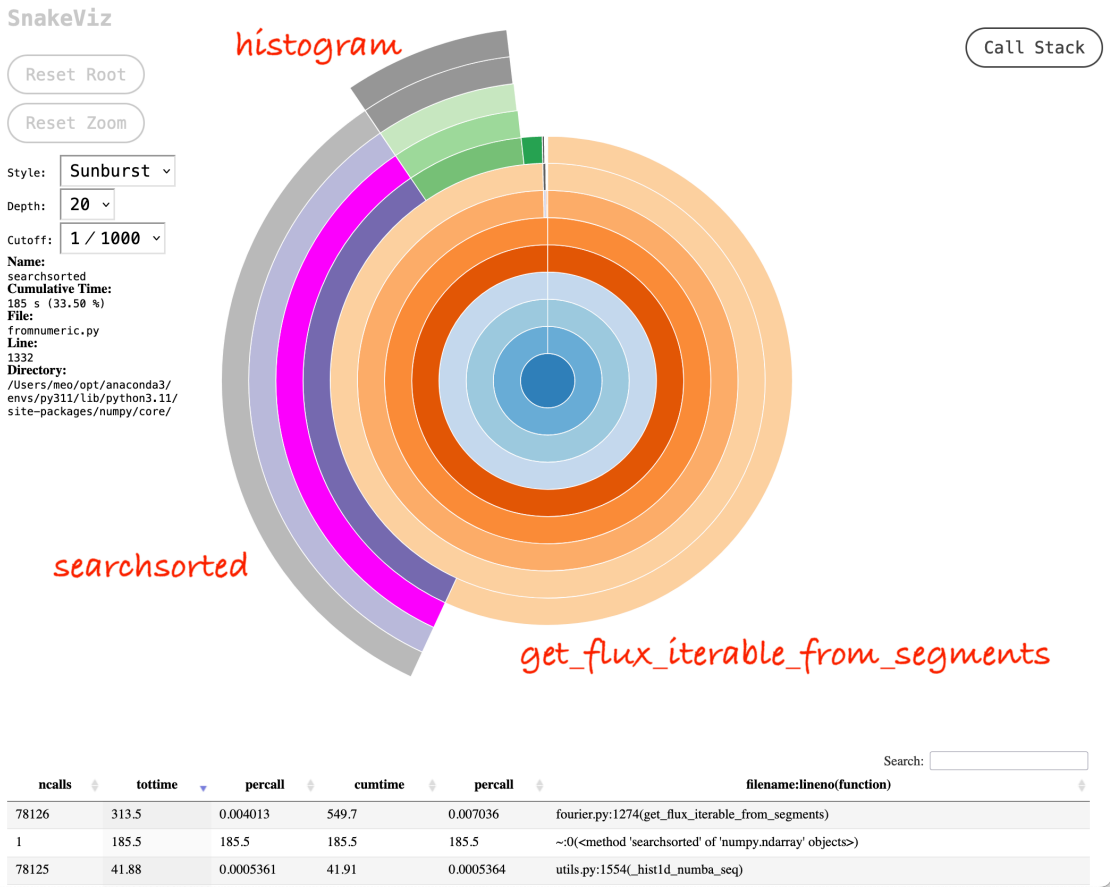
Smaller than RAM (< 32 GB)



Search:

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
78125	7.417	9.493e-05	7.433	9.515e-05	utils.py:1554(_hist1d_numba_seq)
78126	1.481	1.896e-05	10.95	0.0001401	fourier.py:1274(get_flux_iterable_from_segments)
1	0.6464	0.6464	0.6464	0.6464	~:0(<method 'searchsorted' of 'numpy.ndarray' objects>)

Larger than RAM (> 32 GB)

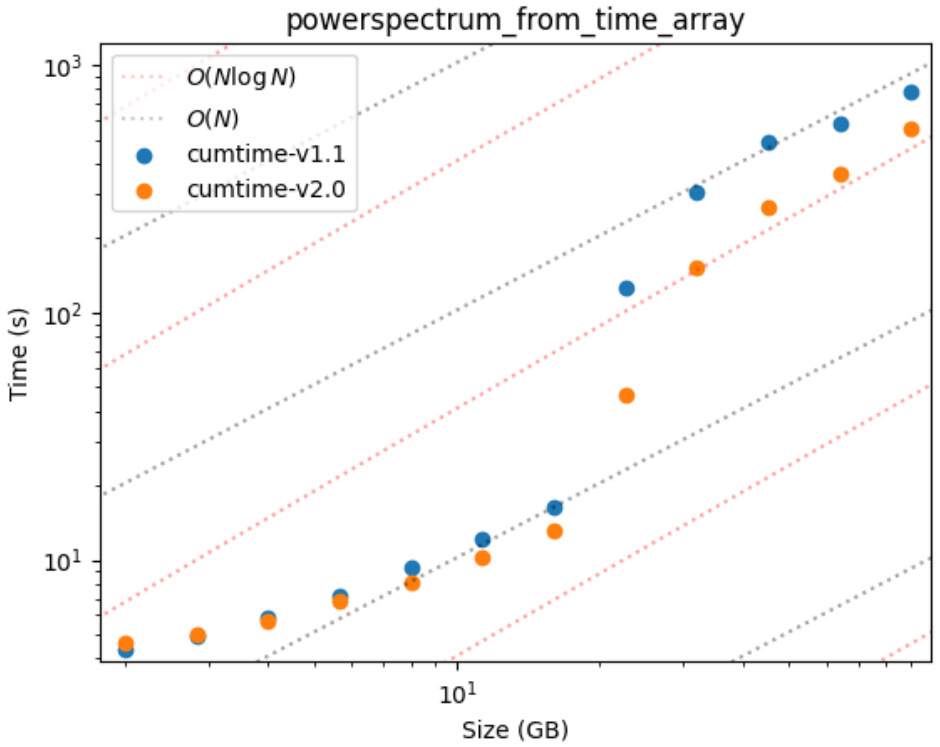


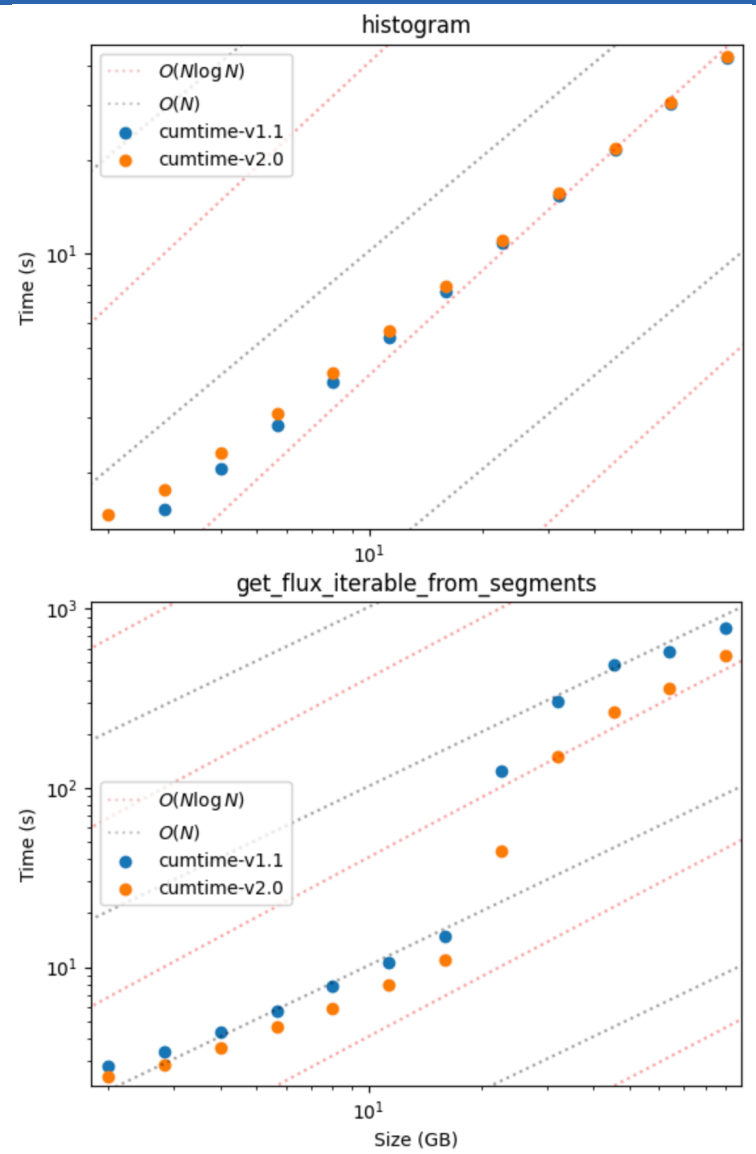
Search:

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
78126	313.5	0.004013	549.7	0.007036	fourier.py:1274(get_flux_iterable_from_segments)
1	185.5	185.5	185.5	185.5	~:0(<method 'searchsorted' of 'numpy.ndarray' objects>)
78125	41.88	0.0005361	41.91	0.0005364	utils.py:1554(_hist1d_numba_seq)

Accomplished Work, Results - M8

Scalability test: these libraries scale linearly or sub-linearly





Next Steps and Expected Results

Next steps:

- Extend performance tests for simulated data > 90 GB, making use of the cluster at OACa (CED)
- histogram, search sorted and get_flux_iterable_from_segments are the most time consuming libraries, thus we plan to parallelise them or to port them on GPUs

Expected results:

- We need to make sure that Stingray v2.0 scales correctly to very large datasets (with a linear or close to linear scalability as obtained up to now for smaller dataset)
- From the developers, in particular from me, I expect to learn how to program on GPUs and in parallel in order to port the libraries that slow down the most the code