PyFstat tutorials – MMCW workshop, Nikhef

David Keitel & Rodrigo Tenorio, 2023/07/13

Basic background and instructions

PyFstat is a CW data analysis toolkit/interface *built on top of LALSuite*, using the SWIGLAL wrappings (by Karl Wette) to call functions of the lalpulsar library, and some of its executables via the command-line (using the subprocess module).
PyFstat does *not* constitute a full replacement for all LALSuite CW functionality. Rather it is...

- An easier way to access some standard functionality (e.g. generating and working with SFTs) without leaving a python interface.
- A consolidated interface to the low-level F-statistic computation tools from lalpulsar, which allows easier development of new search algorithms on top of it (instead of having to write in C and rebuild all of LALSuite, or figuring out the - in parts quite obscure - “raw” SWIG-wrapped python interface yourself).
- A collection of ready-to-use search classes for simple (square) grid searches (no fancy metric setups like with e.g. lalpulsar_ComputeFstatistic_v2 and lalpulsar_Weave), and particularly for MCMC analyses that can be useful in candidate follow-up applications.

See here for a bibliography of works to cite when using PyFstat for papers; the package was originally developed and introduced by Ashton&Prix2018 with a focus on the MCMC followup application; and the current reference for the package in general is this JOSS paper (Keitel+2021).

See here for installation instructions for your local machine (or any cluster you have access on, as long as you can use conda/mamba or venvs). Also see the "Troubleshooting" section below.
Alternatively, you can use Google Colab, but note that this can be quite slow. The examples for today are prepared in notebook format in this drive. You have to be logged into any Google account to be able to launch them interactively.

For offline running, if you’ve managed to install successfully, the full set of examples is available for file-by-file download via the documentation page, via github, or by cloning the whole repository:

```
      >$ git clone https://github.com/PyFstat/PyFstat.git
```

Having downloaded one of those .py files, you generally have to run them like

```
      >$ python PyFstat_example.py
```

(They don’t include a shebang for compatibility reasons with readthedocs and binder.)

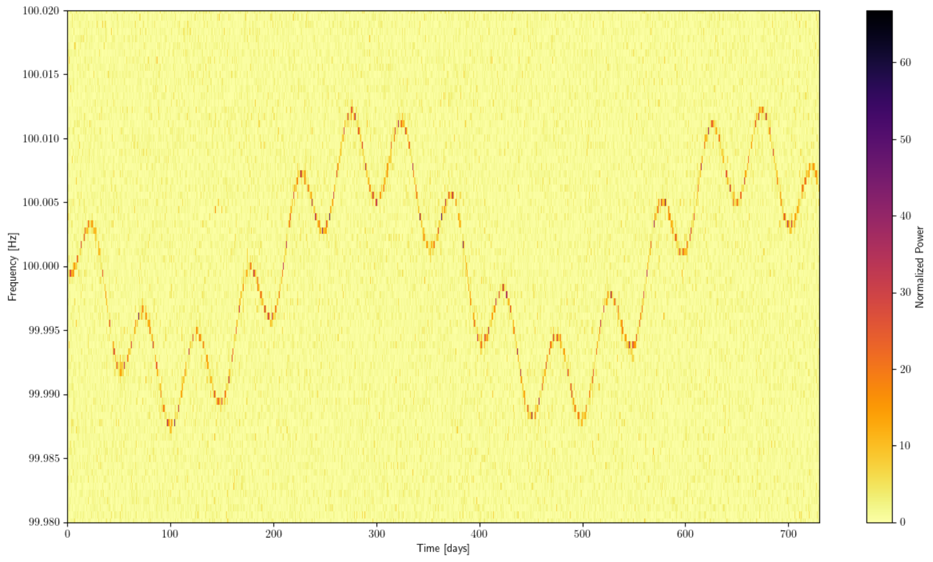
Example 1: generating data and spectrograms

Ideally without reading the notebook/script in detail first (to be unbiased for the first quiz question below), you can do either of the following:

- run this Colab notebook
- from a local git clone, run
  `>$ python examples/other_examples/PyFstat_example_spectrogram.py`
- download script from here and run
  `>$ python PyFstat_example_spectrogram.py`

In a newly generated directory `PyFstat_example_data/PyFstatExampleSpectrogram/` under your current working path, you should see these files (or if you run as a Colab notebook, the plot should show up inline):

- `H-35040_H1_1800SFT_PyFstatExampleSpectrogram-1000000000-63072000.sft`
  The generated data in the SFT format discussed before.
- `PyFstatExampleSpectrogram.cff` Parameters of a simulated CW signal.
- `PyFstatExampleSpectrogram.log` Copy of logging from while running.
- `PyFstatExampleSpectrogram.png`
  The following plot of an extremely strong signal's time-frequency track over a much weaker noise background (real signals would never stand out visually like this):



QUIZ:

1. How many kinds of modulations can you identify here, and what are their causes?

TASKS:

(now look inside the notebook/script, box 5 / L53 will answer the quiz question)

1. Can you make one of the modulations disappear by changing the "`signal_parameters`"?
2. Shouldn't there be another one visible (short period)? Does it show up if you just zoom in (changing axis ranges)? What parameter of the signal or data could you change to make it better visible?

Example 2: *F*-stat grid and MCMC searches

You can do either of the following:

- run this Colab notebook
- from a local git clone, run
  ```
  >$ python examples/mcmc_vs_grid_simple_example/PyFstat_example_simple_mcmc_vs_grid_comparison.py
  ```
- download script from here and run
  ```
  >$ python PyFstat_example_simple_mcmc_vs_grid_comparison.py
  ```

This time don't be shy about sneaking a peek into the details of the notebook/script in advance. Just ignore the plotting helper functions defined in box 7 of the notebook / L56-111 of the script, the interesting parts are only those above and below that.
Here we will generate a shared set of SFT data, then run first a grid-based and then an MCMC-based analysis on it, and make some comparison plots. Results will be in this directory (except for plots that in the notebook will be inline):
`PyFstat_example_data/PyFstatExampleSimpleMCMCvsGridComparison/`

QUIZ:

1. Have a look over the command-line output that the script generates:
    a. How many templates does the grid cover? How much time did a single-template calculation take on your system?
    b. By comparison, how many individual per-template *F*-statistics has the MCMC search approximately evaluated? (taking into account the final number of samples, the number of chains, the burnin/production split, and the acceptance fraction)
2. Have a look over all of the plots produced and see if you can make sense of them. (Note: some are a bit redundant, and sorry for not all being exactly publication quality.)
    a. You should notice some offset between the injection point and the maximum points recovered by grid and MCMC searches. Given the observation time and the approximate expected scaling of the metric, is this offset significant?

TASKS:

1. Try what happens with a weaker signal (lower injection `h0` in box 5 / L41).
2. Try "`sky = True`" in box 3 / L18. (Note that results will land in a separate directory.)
3. Try to change the MCMC priors to Gaussians in box 12 / L199ff – see documentation here, in short you need `"type": "norm"` and instead of the parameters `"lower"` and `"upper"` you need the pair of `"loc"` for location (mean) and `"scale"` (stdev). Observe what happens as you make them wider, or mismatched from the injection.

Examples 3/4 (optional): more MCMCs

If you have time left or would like to play around with things further after the workshop, cleaner starting points than the slightly messy grid-vs-MCMC comparison script above are

- PyFstat_example_fully_coherent_MCMC_search.py
- PyFstat_example_semi_coherent_MCMC_search.py

You should easily be able to modify these to try out different signal injection parameters, dimensionalities of the search parameter space (varying the sky position, higher spindowns) and MCMC algorithmic parameters.

Example 5 (optional): cumulative 2*F*

This one gives a good intuition for how the *F*-statistic actually behaves. Download the script from here and execute. (Sorry, we didn't prepare a Colab version of this one.)

- This one again generates SFT data with a CW signal in it, for 2 detectors H1 and L1.
- Then in L69 it uses another LALSuite tool `lalpulsar_predictFstat` to *predict* how large the total 2*F* from such a signal should be, given the known analytical distribution of 2*F*, at a perfectly matched template, and as an average over noise realisations.
- Then it computes the actual value, but not only over the full duration, but *cumulatively* in the sense that it does searches over [tstart,T1], [tstart,T2], …, [tstart,tend]. Also for H1, L1 and the coherent combination H1L1.
- And it compares these to partial predictions for each corresponding duration; see the plots generated in `PyFstat_example_data/PyFstatExampleTwoFCumulative/`

QUIZ:

1. Does the total predicted 2*F* value (line "`Predicted twoF value`" in the command-line output) roughly match what you'd expect for the given signal amplitude, noise spectral density, and data duration?
2. Based on the plots, does this signal behave as expected for a true, persistent CW signal? And does it also pass the traditional "detector consistency veto" of single-detector values not exceeding the combined multi-detector statistic?

TASKS:

1. Try to modify this for a CW-like long-duration transient signal (as could be produced e.g. by a newborn NS or a glitching pulsar): you'll need to add three injection parameters "`transientStartTime`", "`transientTau`" (duration) and "`transientWindowType`" (an amplitude modulation window, choose "rect" for a simple turn-on, stay-constant, turn-off rectangular window). For StartTime and Tau, choose anything that falls within the original data duration.
   NOTE: You could in principle add these new parameters to the `get_predict_fstat_parameters_from_dict()` call in L52, but while preparing this tutorial, I noticed a small key translation bug with that one. Instead,

assuming you have defined a new dict `transient_parameters` for adding to the injection, manually add these three lines after it:

```
        PFS_input["transientStartTime"] = transient_parameters["transientStartTime"]
        PFS_input["transientTau"] = transient_parameters["transientTau"]
        PFS_input["transientWindowType"] =
transient_parameters["transientWindowType"]
```

Now what do the accumulation plots look like? Can you make sense of the different slopes of the predicted and measured curves?

Troubleshooting

- If you can't use conda for some reason, a standard python venv and using pip should work as well on most systems, if your system-installed python is recent enough (>=3.8). Note you may have to do an extra self-upgrade step for pip because with too old versions, no recent enough LALSuite will be available:

```
python -m venv pyfstat-venv
source pyfstat-venv/bin/activate
pip install --upgrade pip setuptools
pip install pyfstat
```

.

- To verify your installation works:

1. Run `lalpulsar_version`
   This should output several package versions, ending with
   `LALPulsar: 6.0.1 (CLEAN 9a0927aac5e23cd7c93ada22a24aef9e3a12a2c7)`

2. Download the file
   https://pyfstat.readthedocs.io/en/stable/_downloads/8c78c80844a39a078813a0c3fd82afdb/PyFstat_example_grid_search_F0.py

3. Run, from the right directory: `python PyFstat_example_grid_search_F0.py`
   This should run through without errors, mention that it is running PyFstat 2.0.0, and produce some output in a directory `PyFstat_example_data/PyFstatExampleGridSearchF0/` created under your current working directory. PyFstat generally tries to reuse data on disk if possible. Normally it is quite smart about this, but for example when you change signal injection parameters, sometimes it will not delete previous SFT files and there will be conflicting files in the same directory that it tries to read together. A typical error from that starts with "`data gap or overlap`". In such a case, just clean out the output directory.

- If you get too creative with modifying signal or search parameters, it's possible that your search ends up trying to read SFT data outside the narrow band that was generated. In such a case, you may get the same error message as above, or another quite cryptic one, always referring to some number of bins. In such a case, try increasing the "`Band`" parameter of the `Writer` class.
- If you get LaTeX errors related to plot labels, on Debian/Ubuntu-based systems the following may help, or for other OSs you may find similar solutions online:
  ```
  sudo apt-get install dvipng texlive-latex-extra texlive-fonts-recommended
  cm-super
  ```
- Ask us f2f, via mail, or LVK mattermost (if you're a member) for other things that go wrong.

Acknowledgments

David Keitel is supported by the Spanish Ministerio de Ciencia, Innovación y Universidades (ref. BEAGAL 18/00148) and cofinanced by the Universitat de les Illes Balears. Rodrigo Tenorio is supported by the Spanish Ministerio de Universidades (ref. FPU 18/00694). This work was supported by the Universitat de les Illes Balears (UIB); the Spanish Ministry of Science and Innovation (MCIN) and the Spanish Agencia Estatal de Investigación (AEI) grants PID2019-106416GB-I00/MCIN/AEI/10.13039/501100011033, RED2022-134204-E, RED2022-134411-T; the MCIN with funding from the European Union NextGenerationEU (PRTR-C17.I1); the FEDER Operational Program 2021-2027 of the Balearic Islands; the Comunitat Autònoma de les Illes Balears through the Direcció General de Política Universitaria i Recerca with funds from the Tourist Stay Tax Law ITS 2017-006 (PRD2018/23, PDR2020/11); the Conselleria de Fons Europeus, Universitat i Cultura del Govern de les Illes Balears; and EU COST Action CA18108. This material is based upon work supported by NSF's LIGO Laboratory which is a major facility fully funded by the National Science Foundation.