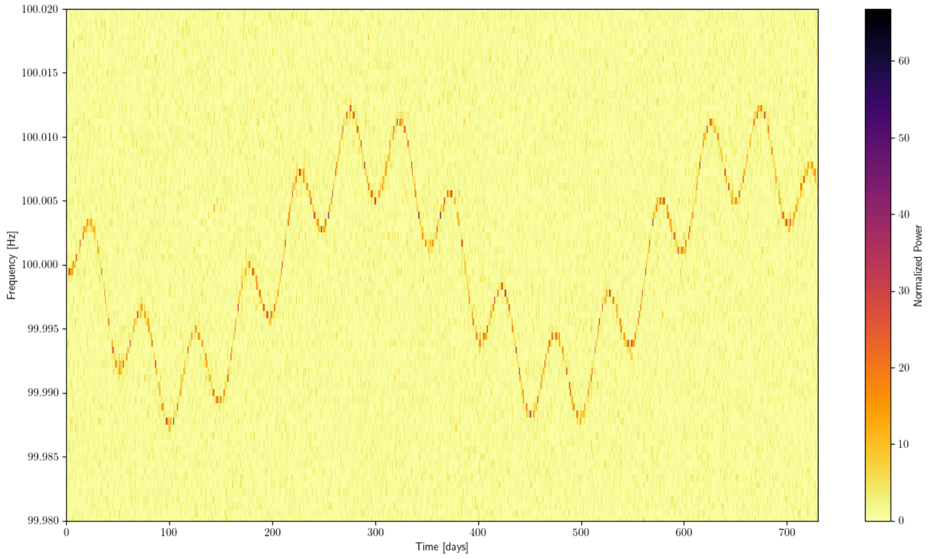
PyFstat tutorials – MMCW workshop, Nikhef

Solutions – David Keitel & Rodrigo Tenorio, 2023/07/13

Example 1: generating data and spectrograms

QUIZ:

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

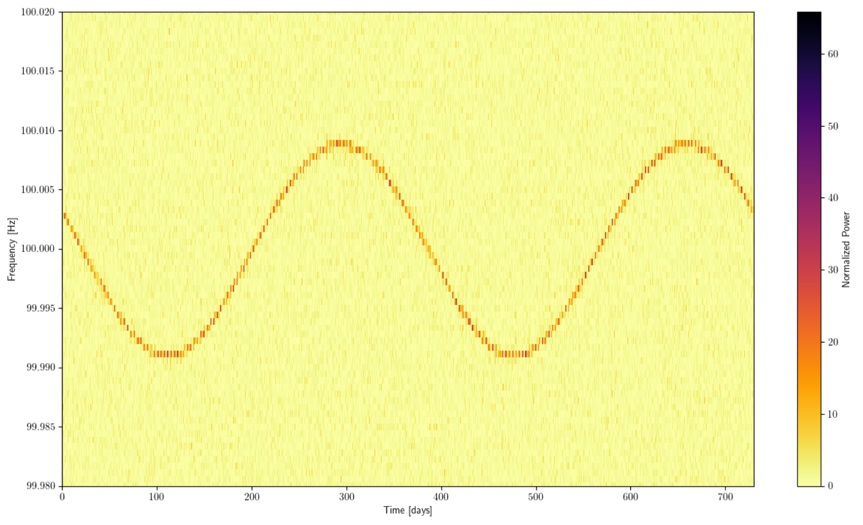


ANSWER: One can see by eye the expected yearly modulation from Earth's orbit, but *not* the daily one from Earth's rotation (see task 1 below). The medium-duration modulation seen here comes from the binary orbital parameters of the injected signal, note especially this one: `"period": 50 * 86400`

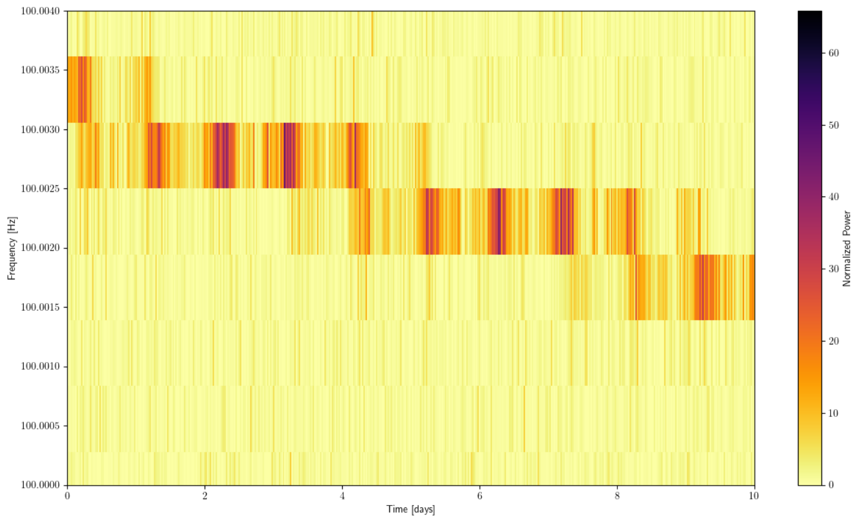
TASKS:

(now look inside the notebook/script, box 5 / L53 will answer the quiz question: we've been calling a special `BinaryModulatedWriter` class that knows how to inject a signal with orbital modulation for a CW signal from a NS in a binary system.)

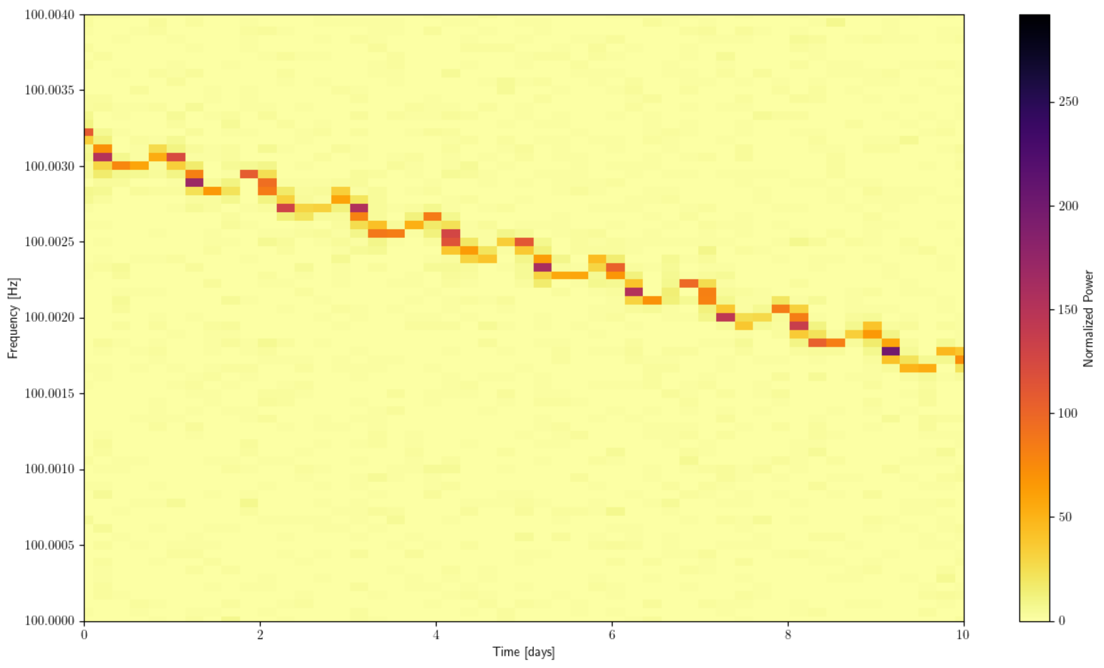
1. Can you make one of the modulations disappear by changing the "`signal_parameters`"?
   ANSWER: Commenting out the three lines for binary parameters `"tp", "asini"` and `"period"`, we get the following plot:



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?
   ANSWER: Yes we should be able to see the daily modulation from Earth's rotation, but we don't really even if we zoom in both horizontally and vertically:



The reason is that the natural resolution of a SFT-based spectrogram is 1/Tsft, i.e. for the 1800s used here we only have ~5mHz resolution. (This may seem a very fine resolution to CBCers, but it would be very coarse for radio astronomers!) So we should be able to make it more visible by using longer SFTs. And indeed, if we change to `"Tsft": 18000`, we start seeing the daily modulation clearly:



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

QUIZ:

1. Have a look over the command-line output that the script generates:
   a. How many templates does the grid cover, and how does this simple example compare to production CBC searches in that regard? How much time did a single-template calculation take on your system?
   ANSWER: Early in the output, there should be a line
   `“INFO    : Running search over a total of 100200 grid points…”`
   Right before it goes through a progress bar for a few seconds. So that's quite a few templates for such a short time! On my laptop, it was showing ~10kit/s, i.e. 10 thousand iterations per seconds, or less than a millisecond per template. This is just 1 month of simulated data, but the main cost factor of real searches comes not from the individual evaluations being costly, but from the *huge* template banks: millions still count as “narrowband”, and all-sky searches easily have 10^{14} to 10^{17} templates.
   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)
   ANSWER: The output file `MCMCSearchF0F1_samples.dat` has 20k lines (minus header comments), corresponding to the number of output samples. Where does this number come from? The MCMC setup (box X / L228-232 of the script) is
   ```
       ntemps = 2
       log10beta_min = -1
       nwalkers = 100
       nsteps = [200, 200]  # [burnin,production]
   ```
   This gives us:
   `nwalkers*nstepsprod = nsamples = 100*200 = 20000`
   As the acceptance fraction I got [0.69,0.65] in the burnin and production steps, respectively, so in total this run should have sampled
   `ntemps*nwalkers*(nstepsburn/accburn+nstepsprod/accprod)= 2*100*(200/0.68+200/0.65) = 120362` which is at a similar level as the grid search.
   Of course in practice the setups of both approaches need to be tuned to the application at hand, and there's no general rule about the relative runtimes for a “good coverage” of a region. As found by Ashton&Prix2018 and Tenorio+2021, especially in multi-stage hierarchical setups, the MCMC approach can get away with quite few evaluations (short runtimes) to zoom into the interesting parts of limited parameter-space regions (candidate followup scenario).

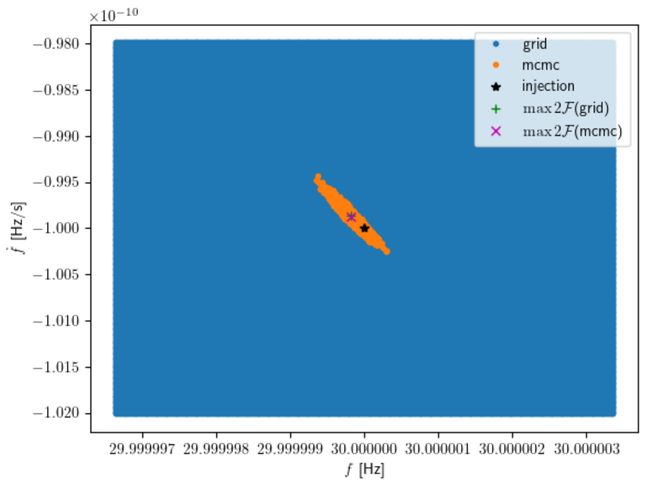
2. Have a look over all of the plots produced and see if you can make sense of them. (Note: some are a bit redundant, and sorry for not all being exactly publication quality.)
    a. You should notice some offset between the injection point and the maximum points recovered by grid and MCMC searches. Given the observation time and the approximate expected scaling of the metric, is this offset significant?
    ANSWER: The offsets are best seen in box X / the file `grid_vs_mcmc_F0F1_zoom.png` and should be on the order of half a grid spacing for the grid search and less for the MCMC, or in absolute terms ~1e-8 in frequency and ~1e-14 in f1dot. Looking at box X / L131f we see that the grid spacing here was defined in terms of the natural metric scaling for a very fine mismatch:

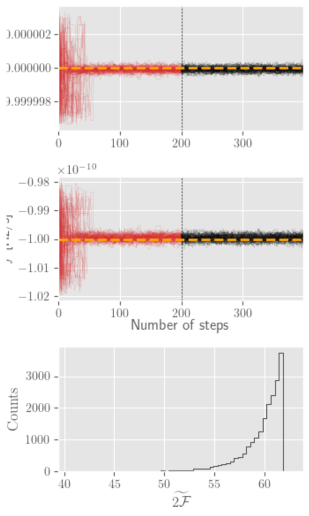
```
m = 0.001
dF0 = np.sqrt(12 * m) / (np.pi * duration)
dF1 = np.sqrt(180 * m) / (np.pi * duration ** 2)
```

    Hence, it's fair to say that the signal has been recovered extremely accurately by both methods – however, that's no wonder given that it is very strong, with a recovered 2F~5000!

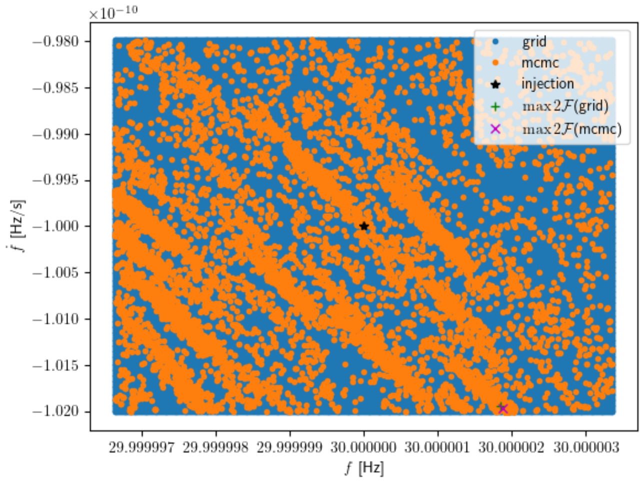
TASKS:

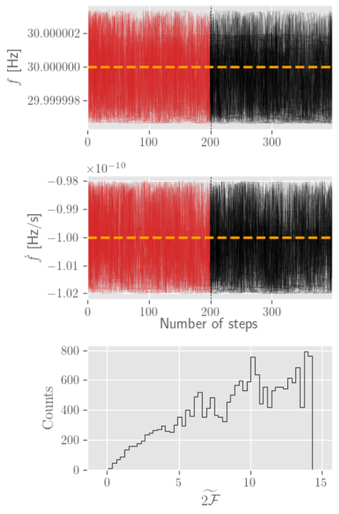
1. Try what happens with a weaker signal (lower injection `h0` in box 5 / L41).
ANSWER: lowering to `"h0": 0.005 * sqrtSX` (corresponding to a depth of 200), both grid and MCMC search no longer recover the injection point quite as accurately, but still well.



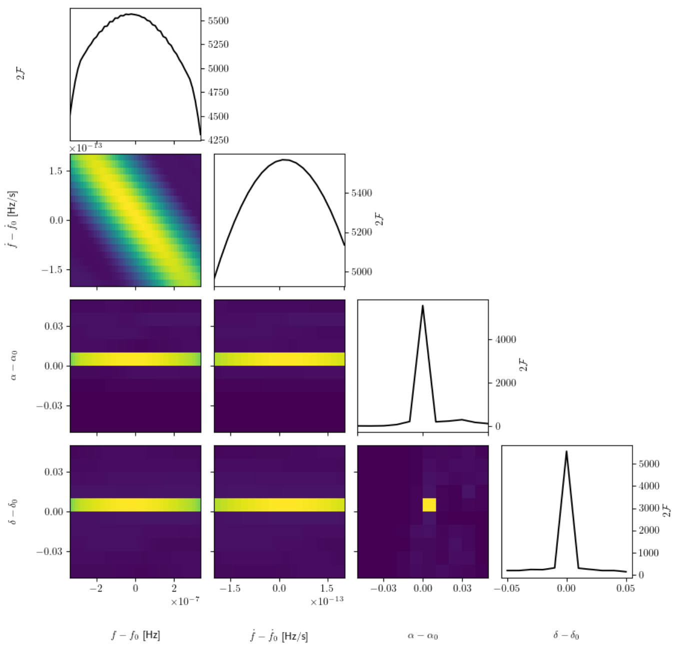


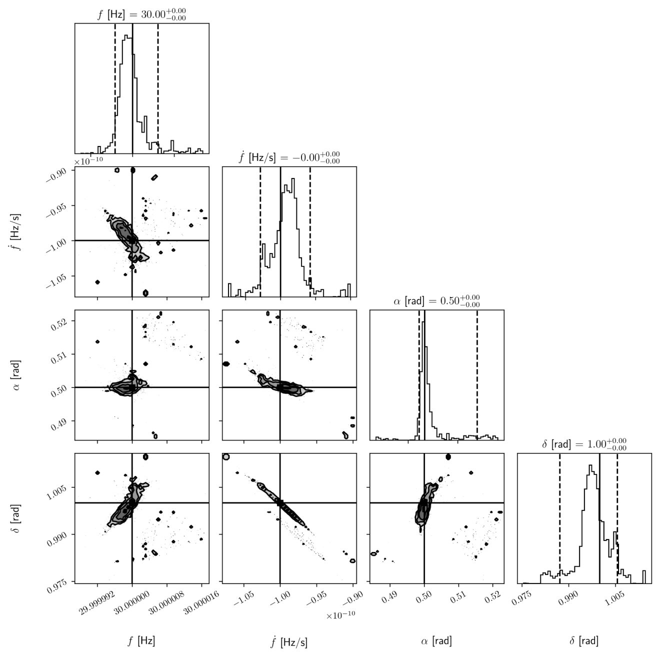
On the other hand, educing h0 by another factor 10 (corresponding to a depth of 2000), we only recover random noise:



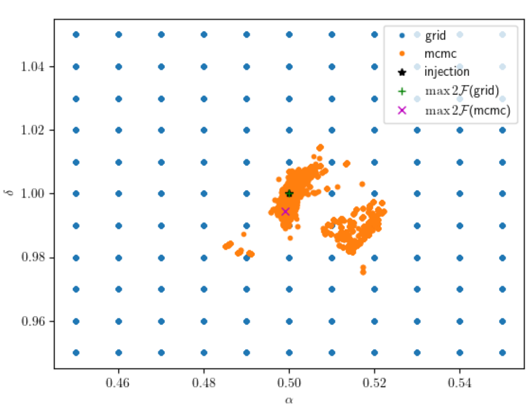


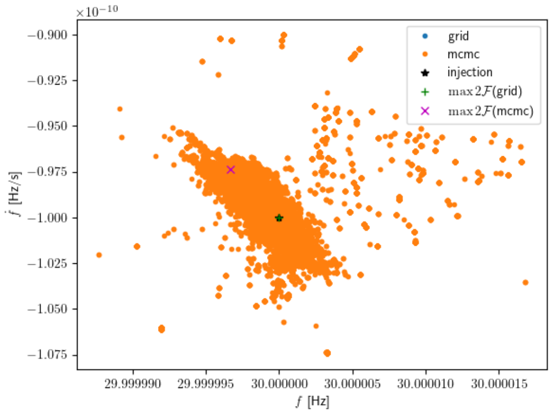
2. Try "`sky = True`" in box 3 / L18. (Note that results will land in a separate directory.)
<u>ANSWER:</u> After running for a few minutes this time, corner plots for example should look like this:





As we can also see from the direct comparison plots, our grid search setup now obviously had not gotten ideal spacings in each dimension, as the MCMC samples explore them quite differently!

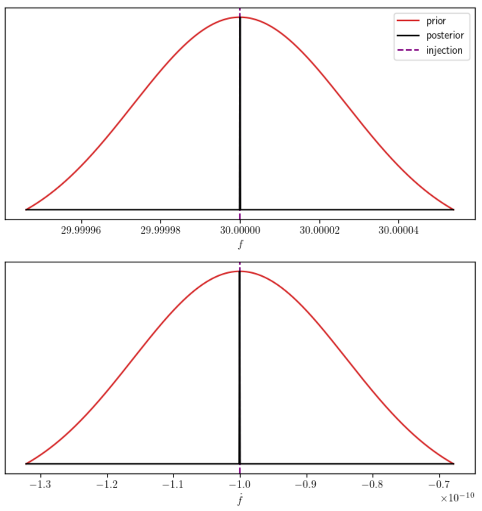


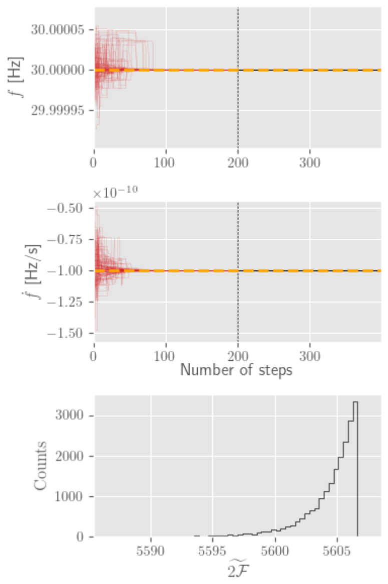


3. Try to change the MCMC priors to Gaussians in box 12 / L199ff – see documentation here, in short you need `"type": "norm"` and instead of the parameters `"lower"` and `"upper"` you need the pair of `"loc"` for location (mean) and `"scale"` (stdev). Observe what happens as you make them wider, or mismatched from the injection.
4. **ANSWER:** We see that the injection is strong enough to zoom in from a quite wide prior, for example with

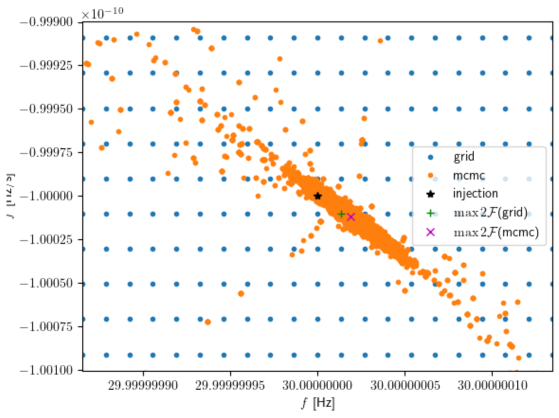
```
theta_prior = {
"F0": {
        "type": "norm",
        "loc": inj["F0"],
        "scale": 4*DeltaF0,
},
"F1": {
        "type": "norm",
        "loc": inj["F1"],
        "scale": 4*DeltaF1,
},
"F2": inj["F2"],
}
```

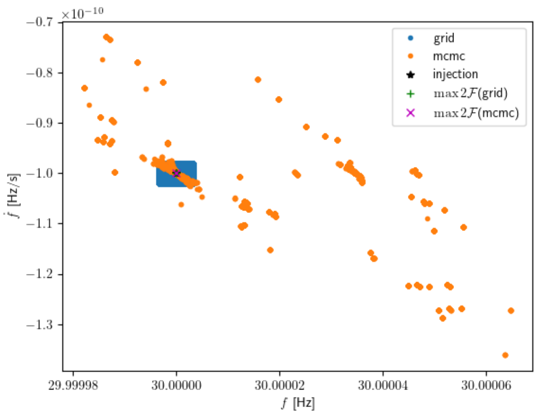
we still get very similar corner and comparison plots, while the prior_posterior and walker plots clearly show the zoom-in:





Increasing the widths further to `100*DeltaFk` we see that the peak is no longer covered as well, but in principle it still worked:





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

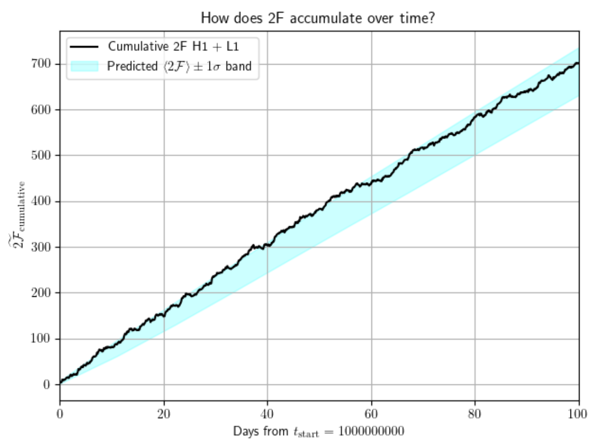
You should easily be able to modify these to try out different signal injection parameters, dimensionalities of the search parameter space (varying the sky position, higher spindowns) and MCMC algorithmic parameters.

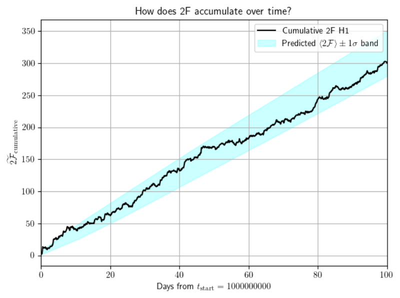
Example 5 (optional): cumulative 2*F*

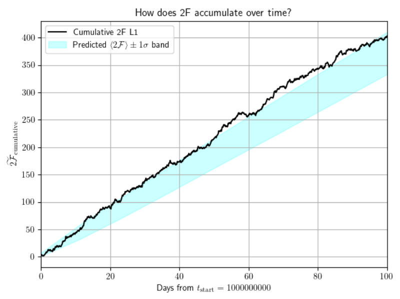
QUIZ:

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?
   ANSWER: The expectation value of 2*F* in noise+signal data is 4 + SNR^2 where the SNR goes as h0 * sqrt(duration / SX) where SX is the noise *P*SD (its sqrt is the *A*SD, called `sqrtSX` in the codes). So we expect:
   ```
   >>> (amplitude_parameters["h0"]/gw_data["sqrtSX"])**2*gw_data["duration"]
   864.0
   ```
   and the script gives us “`Predicted twoF value: 670.48`” or 78% of the simplified calculation. This is within the expected range from different sky-position dependent antenna pattern functions of the detectors, which our simplified calculation ignored.

2. Based on the plots, does this signal behave as expected for a true, persistent CW signal? And does it also pass the “detector consistency veto” briefly discussed in the lecture?
ANSWER: The plots look like this:







As we can see, there is some variation in how the $2F$ accumulates over the 100 days of observation, due to both random noise fluctuations and the different orientation of the detectors (note that `lalapps_predictFstat` does take sky location into account, not just the rough SNR scaling we tested above!). But there is *no significant deviation* from the expectation: H1L1 and H1 separately stay within 1sigma, and L1 barely leaves it.
Also, we have a total $2F$ of ~700 for the H1L1 combined result and ~300 and ~400 for the individual detectors respectively, so *this passes the detector consistency veto*. (Note that the H1L1 result is *not* simply the sum of H1 and L1 F-statistics, which are power statistics and hence would be an *incoherent* sum, but computed as the *coherent* matched filter over the combined data, with the right phase offsets corresponding to the detector locations and orientations.)

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 L55, 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"]
```

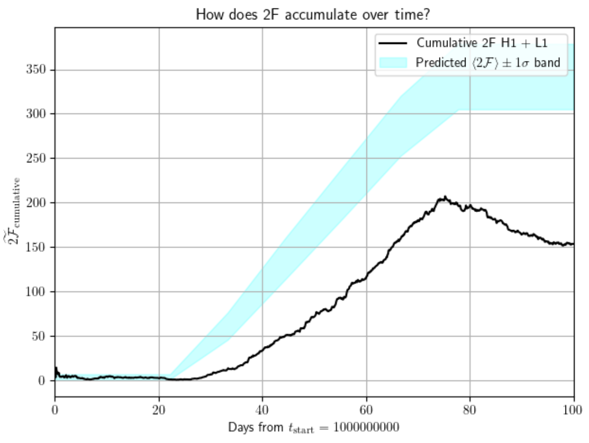
```
        PFS_input["transientWindowType"] =
transient_parameters["transientWindowType"]
```

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

ANSWER: Defining the transient parameters in L49 as:

```
        transient_parameters = {
                "transientStartTime": gw_data["tstart"]+0.25*gw_data["duration"],
                "transientTau": 0.5*gw_data["duration"],
                "transientWindowType": "rect",
        }
```

and adding the above `PFS_input` keys and also `**transient_parameters` to the `BinaryModulatedWriter` call, we get the following H1L1 plot (and similar for H1 and L1 individually):



As we can see, this now doesn't behave as predicted at all: no signal is accumulated at the start, then it grows linearly as expected while the injection is "on", and at the end it actually *goes down* again, because the *coherent matched filter* that is the *F*-statistic now penalises the added data for having mismatched phase evolution for those final weeks.

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.