Brute force correlation of drifting lines

Bas Swinkels (Nikhef)

Detchar meeting, 22/06/2018

Advanced Virgo

Nikhef

Motivation

- Current motivation: some annoying lines that drift in frequency, would like to correlate these to all possible channels
- Similar idea as Matlab code by Soumen Koley used to correlate aliased lines to mirror temperature just before O2
- Need to correlate other things as well: line frequency, brms, range with channels ...
- Old idea of making a flexible pipeline of tools, see VIR-0503A-17
- Ideally use code that can also work at LIGO

Step 1: speeding up spectrograms

- Making a spectrogram over several days costs hours, and if you used the wrong NFFT you have to start over
- Typically interested only in a small frequency band (so daily and weekly plots on VIM/MoniWeb are still very useful to find them, but typically you want to ‘zoom in’)
- **Use demodulation to reduce data by factor 100-1000!** Same trick used for online drum-mode tracker, see lb #38844
- Download raw data in small chunks, multiply with complex sine, decimate in several steps, save complex time series to hdf5 file
- Have to do this only once, can redo the next step many times
- Speed: ~15 min to demodulate full day at 10kHz
- Todo: use multiprocessing to speed things up, needs way of avoiding filter transients of decimation filter

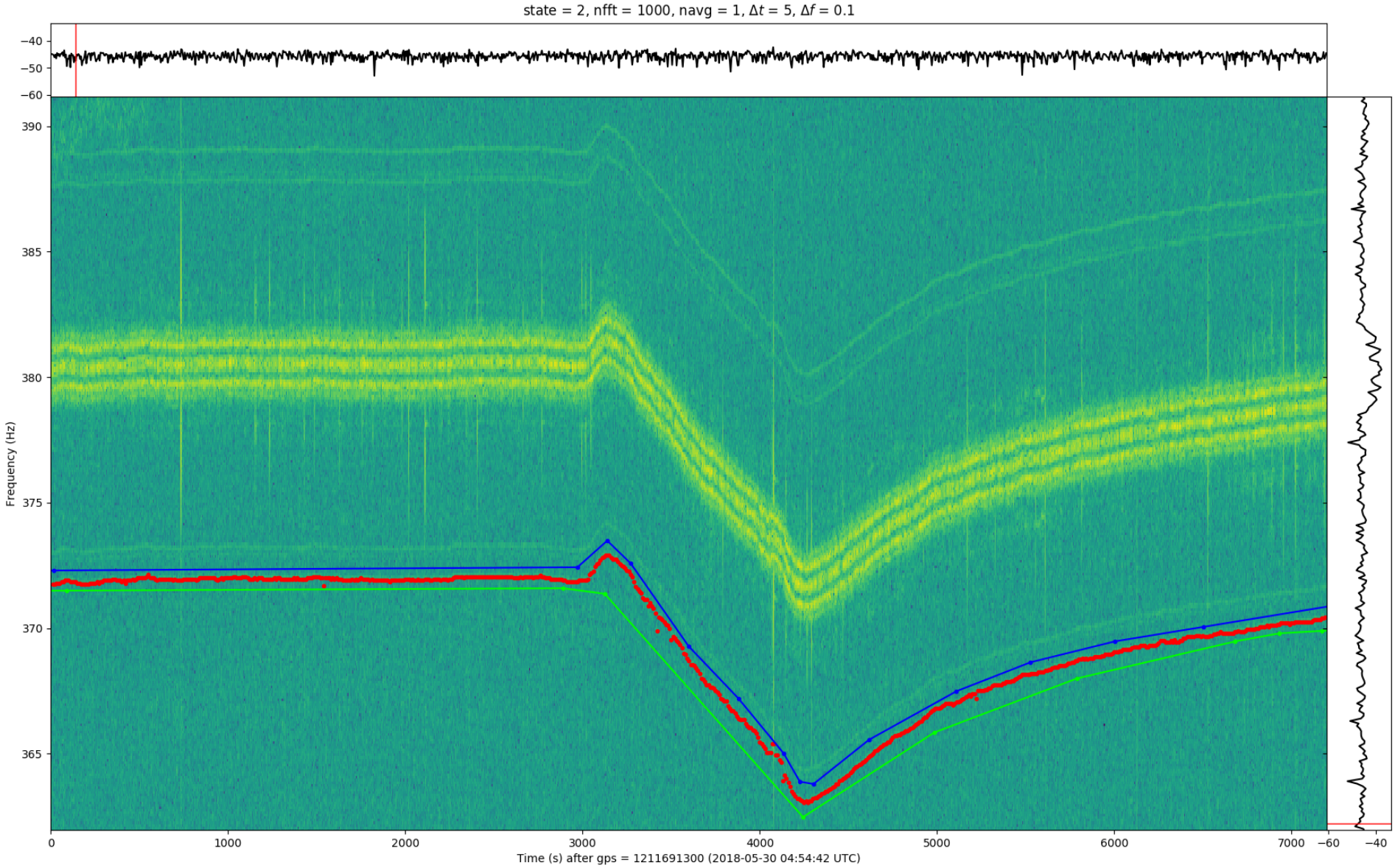
Step 1: speeding up spectrograms

```
>>> ./demod.py -h
usage: demod.py [-h] [--ffl FFL] [--chan CHAN] --f_demod F_DEMOD --f_out F_OUT
                [--gstart GSTART] [--gstop GSTOP]

Tool to demodulate and decimate channels

optional arguments:
  -h, --help          show this help message and exit
  --ffl FFL           ffl file to read input from
  --chan CHAN         channel name, needed if source contains multiple channels
  --f_demod F_DEMOD   demodulation frequency, must be integer
  --f_out F_OUT       sample frequency of output signal, must be a sub-multiple
                      of the one of the input signal
  --start START       gps time of start of period to process, accepts any valid
                      input to gwpy.to_gps
  --stop STOP         gps time of end of period to process, accepts any valid
                      input to gwpy.to_gps
```

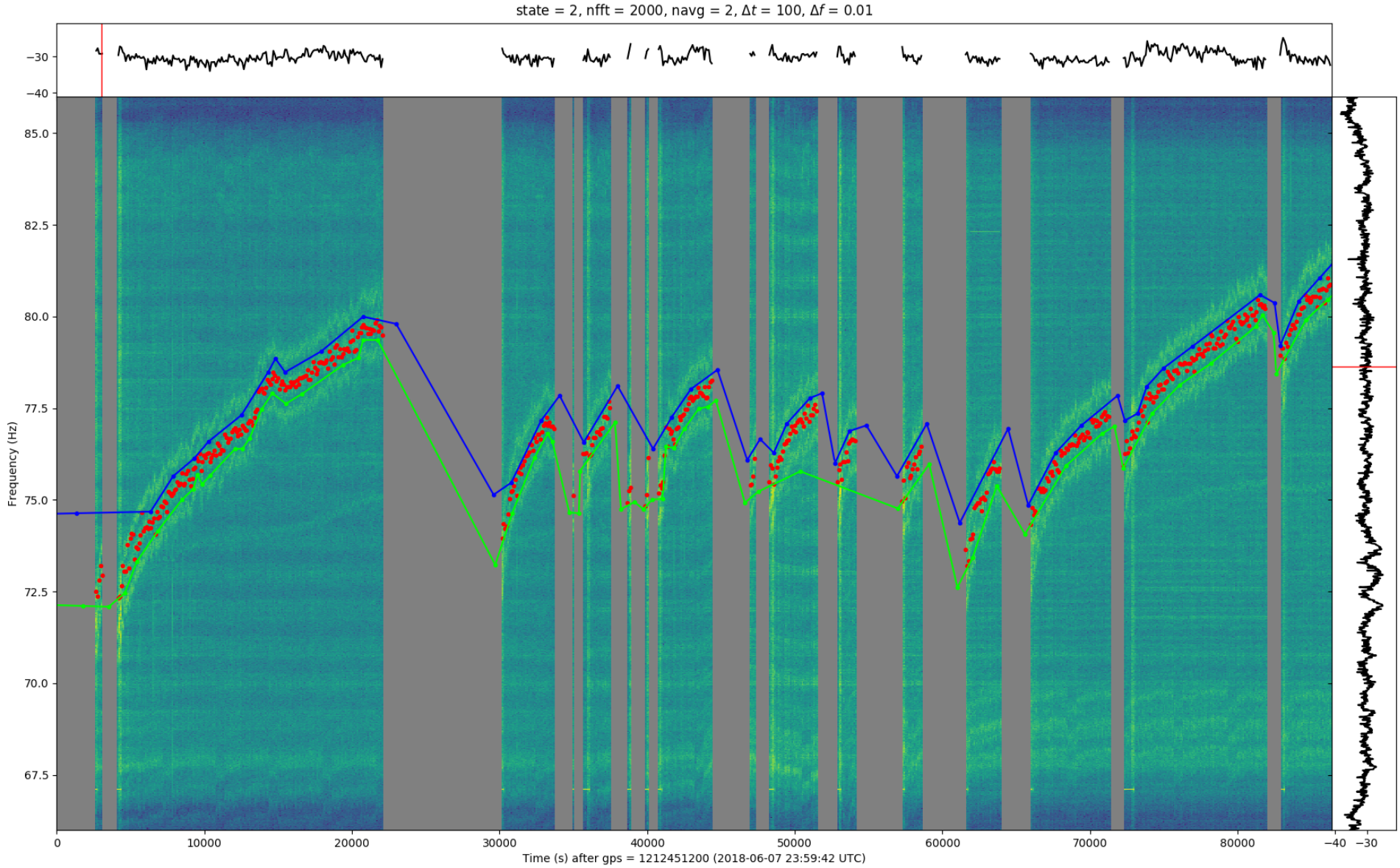
Step 2: tracking lines

- No peak is the same, need to tune NFFT, number of averages, to resolve the line in the best way
- Drifting lines can cross, can be very close, ...
- Idea: no artificial intelligence is will do better than your brain, so do this step by hand using a fast GUI

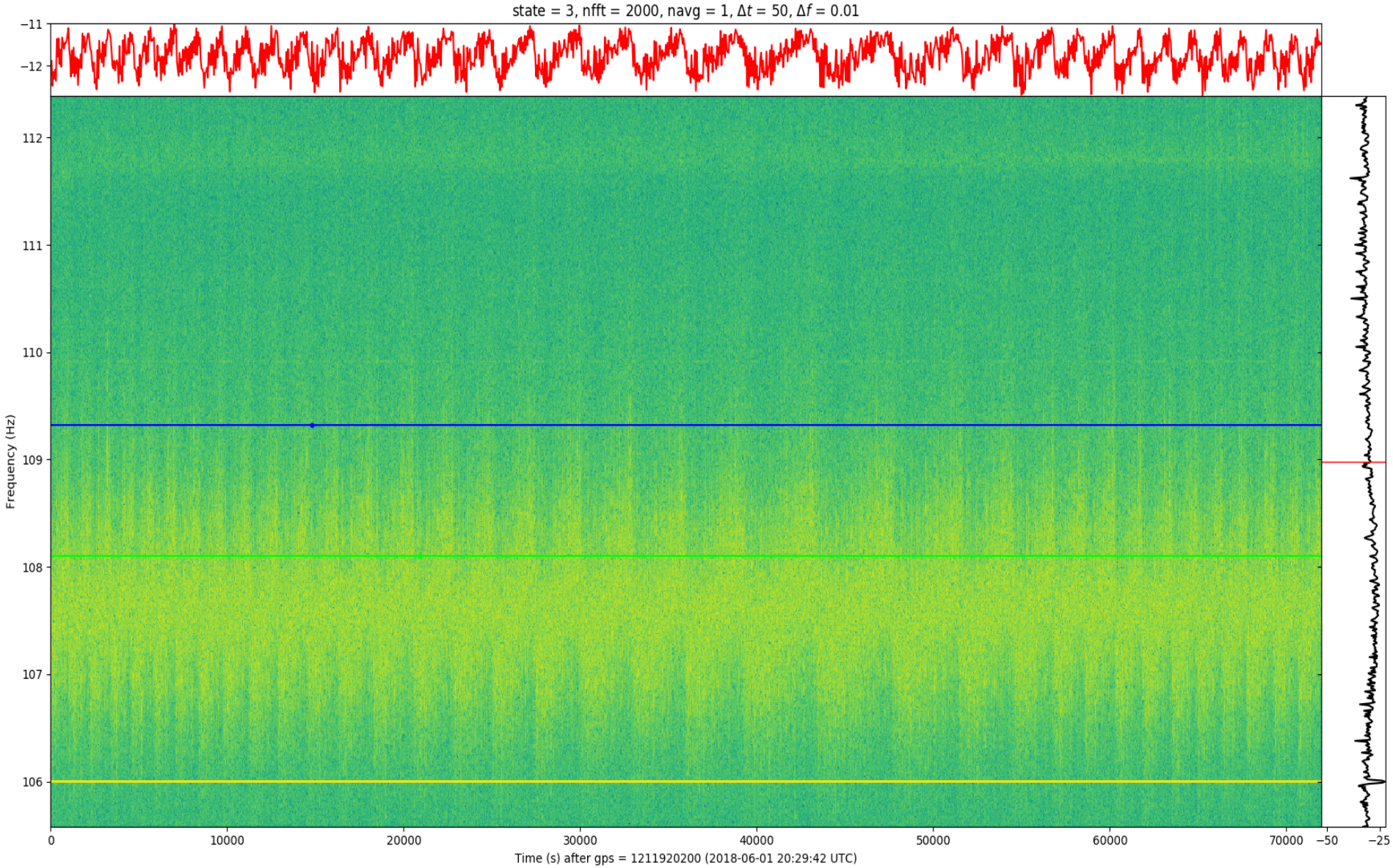
Step 2: tracking lines



Step 2: tracking lines



Step 2: tracking lines



Step 2: tracking lines

- Based on Matplotlib: things like zooming, panning, saving images come for free
- Interactive: can use keyboard to change NFFT, number of averages
- **Very fast**: redrawing spectrogram costs about 1 second!
- User can define bounding-box using mouse clicks, exclude bad regions (unlocks)
- Maximum frequency is searched within this box, saved to file in hdf5 format
- Can also compute BRMS of the same box

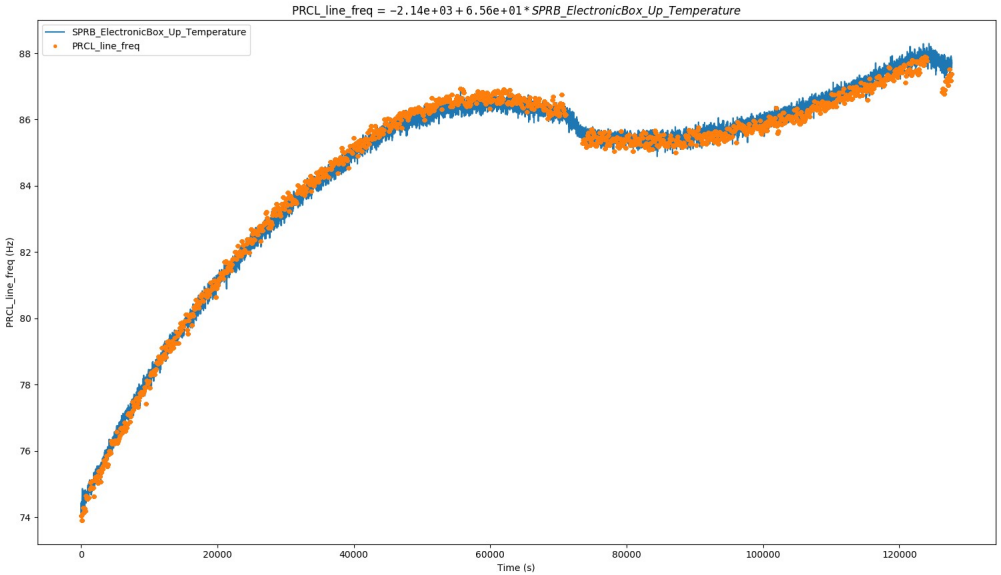
Step 3: correlate

- Pretty standard brute-force correlator (see e.g. lb #24349), this has been done by many others
- Loop over all channels in ffl, blacklist a few, calculate a FOM based on correlation, rank channels
- Use residuals of a polyfit as figure of merit, so could potentially find quadratic relations
- Very basic algorithm, can in future be swapped out for more sophisticated algorithms like Francesco's code (VIR-0406A-18) or LIGO's LASSO code (LIGO-P1800173)
- Speed: about 1-2 days of 10000 channels from trend takes ~10 min, already uses multiprocessing

Step 3: correlate

```
>>> ./correlate.py -h
usage: correlate.py [-h] --source SOURCE [--chans CHANS [CHANS ...]]
                    --f_target F_TARGET --aux_source AUX_SOURCE
                    [--fit_order FIT_ORDER] [--ntop NTOP] [--start START]
                    [--stop STOP]

Tool to correlate channels

optional arguments:
  -h, --help            show this help message and exit
  --source SOURCE       source for channel for which to find correlation
  --chans CHANS [CHANS ...]
                        channel names (can be omitted if source constains
                        single channel)
  --f_target F_TARGET   sample frequency at which correlation is performed,
                        ideally same as aux data
  --aux_source AUX_SOURCE
                        source for auxiliary channels (name of ffl for now)
  --fit_order FIT_ORDER
                        order of polynomial fit
  --ntop NTOP           number of winning channels to report
  --start START         gps time of start of period to process, accepts any
                        valid input to gwpy.to_gps
  --stop STOP           gps time of end of period to process, accepts any
                        valid input to gwpy.to_gps
```

Step 4: report results




```
Brute-force search for correlation from
gps = 1211922075.0 (2018-06-01 21:00:57 UTC) to
gps = 1212049775.0 (2018-06-03 08:29:17 UTC)
Order of fit: 1


*** Best correlation for channel peak_freq ***

rank  residual  channel
   1  7.08e+03  V1:SPRB_ElectronicBox_Up_Temperature
   2  8.43e+03  V1:SPRB_ElectronicBox_Down_Temperature
   3  2.40e+04  V1:SPRB_B4_Cam2_temp
   4  3.94e+04  V1:SPRB_B4_Cam2_press
   5  4.59e+04  V1:ENV_TCS_NE_RH_TE
   6  4.87e+04  V1:SPRB_LC_LVDT_BR_H_FL_V_in_mag_mean
   7  5.70e+04  V1:SPRB_LC_LVDT_FR_H_BL_V_in_mag_mean
   8  6.94e+04  V1:SPRB_LC_LVDT_FL_H_BR_V_in_mag_mean
   9  7.71e+04  V1:SPRB_LC_NULL_H_mean
  10  7.91e+04  V1:Sc_IB_MAR_TX_CORR_mean
```

- Print ranking, make plots of winning channel, ...
- Follow-up investigations ...

Data types

- Thinking about these tools, you see they all use the same basic data types:
  - time series data
  - data sources (e.g. raw.ffl, LIGO's datafinder, something saved by the user)
  - channels, with option to blacklist
  - segments (e.g. you only want to analyze data when ITF is locked)
  - lists of events (e.g. omicron triggers for UPV/Excavator)
- Most of these are well supported in Duncan Macleod's gwpy library (still working on ffl support) https://gwpy.github.io/
- gwpy can read and write these to many different file formats (gwf, txt, hdf5, ...). Try to allow all possible inputs, write by default to hdf5 only
- Should be possible to make code work at LIGO with minimal modifications
- Planning to deprecate parts of virgotools in favor of gwpy: gps conversion, reading data from gwf, ...

Some comments

- All of this is work in progress, will do first release of code in the next days
- The basic algorithms are straightforward, the complications come from boring things like missing samples, comparing channels with different sample rates
- Gaps in target data (e.g. unlocks) can be handled by using NaNs's (for short gaps) or by considering data as a list of segments (for long gaps, not yet implemented). Will add threshold channel (e.g. META_ITF_LOCK_index > 120)
- Missing data in auxiliary channels is bad: you only know there is one at the moment you get the data. Skipping channel can hide interesting channels. Could interpolate gaps?
- Need to consider channels at different sample rates (at first missed SPRB temperature at 0.2 Hz). Sample rates of 1-2-5-10 are a PITA when doing up/down sampling, **power of 2 sample rates would simplify things a lot**
- Blacklisting: for now skipping *_min, *_max, *_rms, VAC_* and Daq_* to speed things up. Need to periodically check full list
- Slow monitoring channels on all noisy machines are essential, commissioning time/missed Mpc is much more expensive than some 'cheap' temperature sensors