Preliminary asymmetry analysis

Rob Mahurin, Corwin Lester, Sydney Smith
MTSU

2015-11-23 Mon

Outline

An independent analysis tool, written in pure Python

Git repository: `ssh://basestar.phys.utk.edu/~mahurin/n3he_py.git`

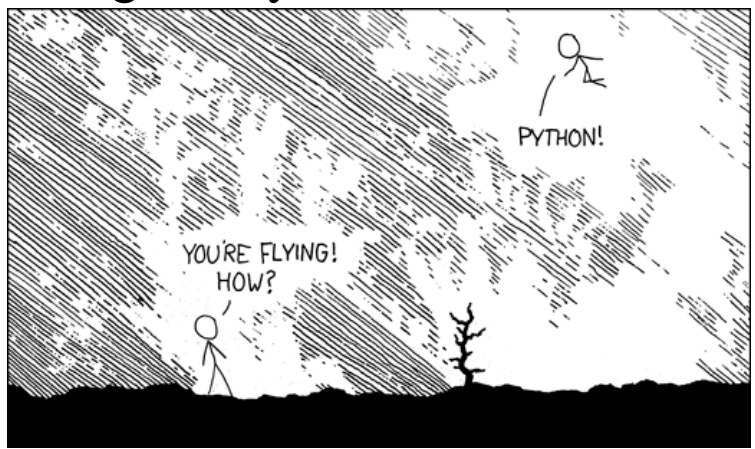
- This presentation documents v0-02.
- Earlier PyROOT-based version (v0-01) included in n3he repository.
- ROOT dependence excised (save one example).

Python is a robust interpreted language

- Explicit loops are slow, but implicit loops may access compiled code.
- Rapid prototyping and development
- Easy to attach new metadata to objects
- Exception handling rather than segfaulting
- No distinction between 'typed-in' code and 'scripts'

Python has a rich library ecosystem

- `ipython`: extra-featured interpreter
- `numpy`: fast library for numerical analysis, especially arrays
- `matplotlib`: MATLAB-esque plotting library
- `antigravity`:



ADCdata

- Raw data files are accessed as arrays of integers using `numpy.memmap`. The memory-map allocates space to store the entire data file in memory[1] but delays the read until data is actually accessed. Short random reads are fast, and there is no penalty associated with loading an entire file.
- `ADCdata` objects present datafiles to the interpreter as structured arrays of 'header' and 'data'. The data array is a 3D array with indices [pulse, tbin, channel]. There are some utility functions for making common tasks easier.

```
In [0]: import n3he_mm; from n3he_mm import *

In [0]: adc = ADCdata(run=39390, adc=21)

In [0]: adc['header'].dtype, adc['data'].dtype # data types
Out[0]: (dtype('uint32'), dtype('int32'))

# print, from ADC headers, pulse 10, first five elements
In [0]: print adc['header'][10, :5]
[2857759060 2857759060 2857759060 2857759060         11]

# from pulse 10, first 5 time bins of channel 4 (upstream center wire)
In [0]: print adc['data'][10, :5, 4] # raw data
In [0]: print adc['data'][10, :5, 4].adc_value()
In [0]: print adc['data'][10, :5, 4].volts()
[265427236 258715940 262409508 276724004 294627876]
[1036825 1010609 1025037 1080953 1150890]
[ 1.23599172  1.20473981  1.22193933  1.28859639  1.37196779]
```

[1] The big memory allocation for `ADCdata` objects is annoying, but I get around it in `DAQdata` objects by using the same sort of trick: only creating the actual `ADCdata` objects briefly, when there is a request for their data.

DAQdata

A `DAQdata` object is a container of the five `ADCdata` objects for a run, with some utility functions. For instance, data from the target may be accessed by [adc](channel) or by (plane, wire):

```
In [0]: print adc.run, adc.adc
39390 21

In [0]: d = DAQdata(run=adc.run)

In [0]: print adc['data'][10, :5, 4].volts()
In [0]: print d[21](channel=4)[10, :5].volts()
In [0]: print d(plane=0, wire=4)[10, :5].volts()
[ 1.23599172  1.20473981  1.22193933  1.28859639  1.37196779]
[ 1.23599172  1.20473981  1.22193933  1.28859639  1.37196779]
[ 1.23599172  1.20473981  1.22193933  1.28859639  1.37196779]

In [0]: compare = (d(plane=0, wire=4) == adc(channel=4)) # churn through file
In [0]: print repr(compare)
ADCdata(filename=None, dtype=dtype('bool'), shape=(25000, 49))

In [0]: print compare.all() # check all 1.2M comparisons
True
```

DAQsummary

A `DAQsummary` is a special `DAQdata` which tries to read a summary file rather than raw data files. If the summary file isn't found, `DAQsummary` will compute one and try to save it. The summary contains (as of v0-02):

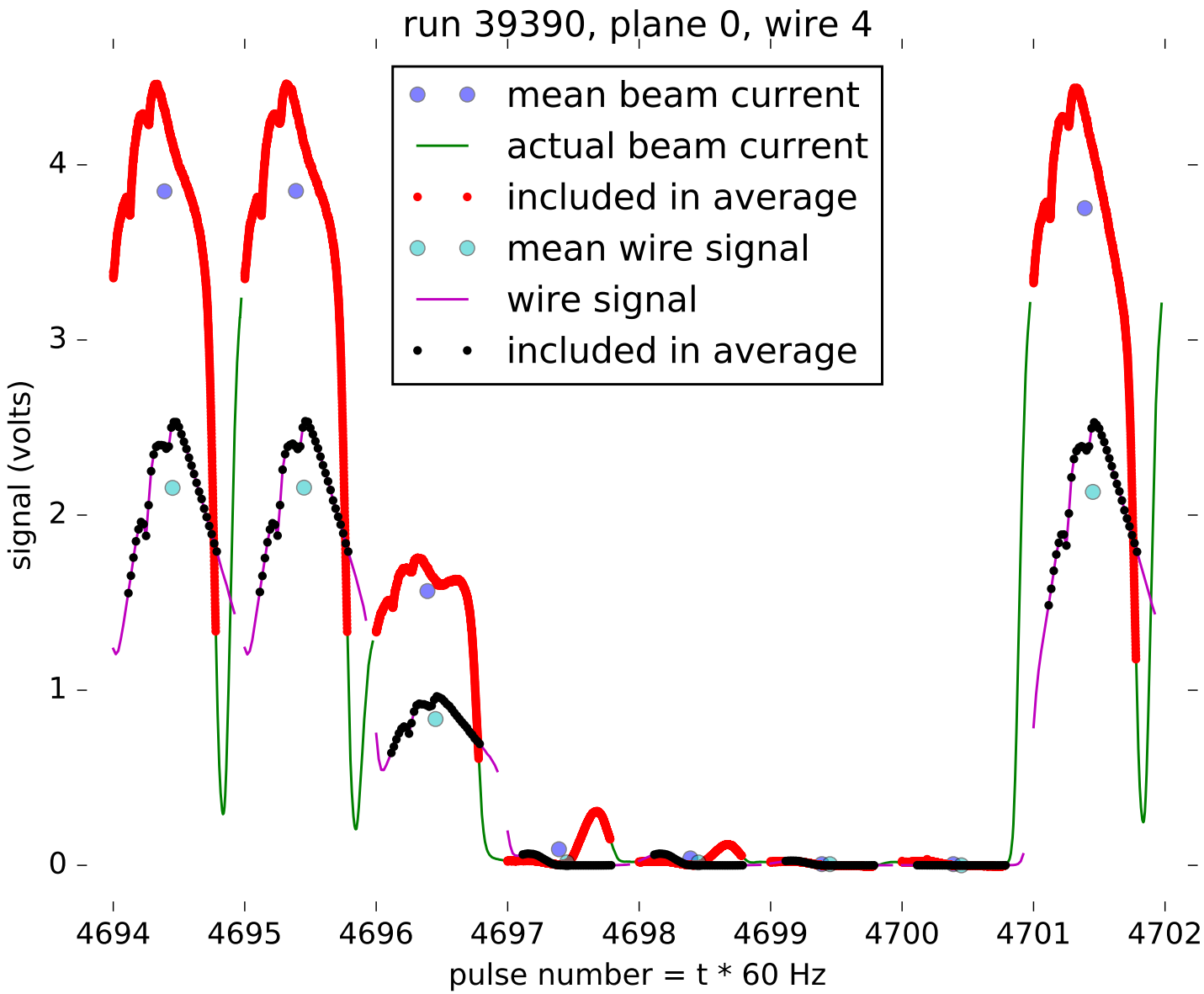
- 'rfsf': boolean array, spin flipper on/off for pulse
- 'beam': float64 array, mean beam monitor voltage per pulse
- 21-24: int32 arrays, sum of adc.values() per channel per pulse

The time bins used in the summing/averaging are stored with the arrays. The detector data is guaranteed to fit in an int32: even summing 98 time bins from adjacent pulses, log2(98 * 2**24) < 31.
A `DAQsummary` condenses a 1.3 GB run down to about 20 MB.

Code sample: input

```
In [0]: summary = DAQsummary(run=d.run)
In [0]: print find(summary['beam'] < 1.5) # use 1.5V as the beam cutoff
[   88   688  1288  1888  2488  3088  3688  4288  4697  4698  4699 ... ]

# Let's look at the consective dropped pulses around 4700
In [0]: interval = r_[4694:4702] # a helpful numpy range operator

# First some plots of the beam monitor
In [0]: plot(summary['beam'].tbins.flatten() + interval, # xdata, minor ugh
   ...:      summary['beam'][interval],                  # ydata
   ...:      'o', label='mean beam current', alpha=0.5)
In [0]: d[30](0).plot(pulselist=interval, fmt='-', label='actual beam current')
In [0]: d[30](0).plot(pulselist=interval, tbins=summary['beam'].processed,
   ...:               fmt='.', label='included in average')

# Next some similar plots of the front-and-center wire
In [0]: plane,wire = 0,4
In [0]: plot(summary(plane,wire).tbins.flatten() + interval, # xdata, minor ugh
   ...:      summary(plane,wire)[interval].volts(),          # ydata
   ...:      'o', label='mean wire signal', alpha=0.5)
In [0]: d(plane,wire).plot(pulselist=interval, fmt='-', label='wire signal')
In [0]: d(plane,wire).plot(pulselist=interval,
   ...:                    tbins=summary(plane,wire).processed,
   ...:                    fmt='.k', label='included in average')
```

Code sample: output



Asymmetry computation: formalism

Definitions and assumptions

Beam current	$I^{\pm}$
Beam polarization	$P^{\pm}$
i-th detector "efficiency"	e_i
Geometric sensitivity	g_i
Physics asymmetry	A

Yield from i-th detector:

$$Y_i^{\pm} = e_i I^{\pm} \left(1 + g_i P^{\pm} A\right)$$

Spin flipper efficiency, detector symmetry are good:

$$P^+ \approx -P^-$$

$$g_i \approx -g_j$$

Possible asymmetries

$$A_{\text{beam}} = \frac{I^+ - I^-}{I^+ + I^-}$$

Wire asymmetry:

$$A_i = \frac{Y_i^+ - Y_i^-}{Y_i^+ + Y_i^-} = \frac{A_{\text{beam}} + g_i PA}{1 + A_{\text{beam}} g_i PA}$$

$$A_i + A_j \approx 2A_{\text{beam}} + (g_i + g_j)PA$$

$$A_i - A_j \approx 2g_i PA(1 - A_{\text{beam}}^2)$$

Ratio of pairs:

$$R_{ij}^+ = Y_i^+ / Y_j^+$$

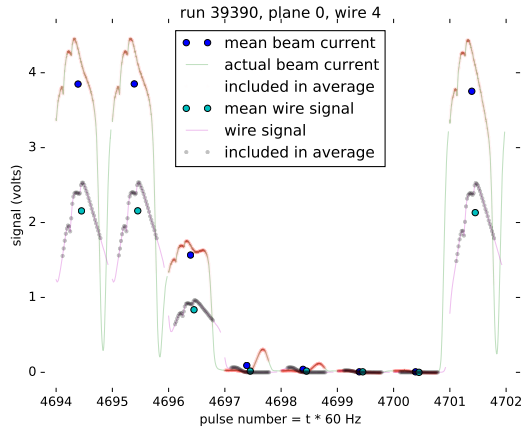
$$A_{R_{ij}} = \frac{R_{ij}^+ - R_{ij}^-}{R_{ij}^+ + R_{ij}^-} \approx \frac{2g_i PA}{1 + (g_i PA)^2}$$

Non-leading terms are a little iffy.

Caching evaluation of asymmetries on summary data

```
In [34]: summary = DAQsummary(run=14858)
In [35]: summary.keys()
['rfsf', 'beam', 21, 22, 23, 24]

In [36]: summary._dispatch_table
{'miss':        <function n3he_mm.low_beam>,
 'nearmiss':    <function n3he_mm.near_low_beam>,
 'diff21':      <functools.partial at 0x2b45d60>,
 'asym21':      <functools.partial at 0x2b54100>,
 ...}

# Most runs from this era appear to have
# the spin flipper phased different from
# the fall runs. A configuration change?
In [45]: cp = [True, False]
In [46]: summary['badrfsf'] = \
    ...:   summary.rfsf_bad(correct_pattern=cp)

# Calling this function ...
In [47]: summary('asym', plane=0, wire=4)
ADCdata([[  2.29702134e-04],
       [  6.66346858e-04],
       ...,
       [  3.91685902e-04],
       [ -1.88695423e-04]])

# ... computes and stores intermediate results
In [48]: summary.keys()
['rfsf', 'beam', 21, 22, 23, 24,
 'badrfsf', 'miss', 'nearmiss', 'good21', 'pol',
 'diff21', 'sum21', 'asym21']
```

Asymmetry computations are done starting from the "summed" data (the big fat dots, one per pulse).

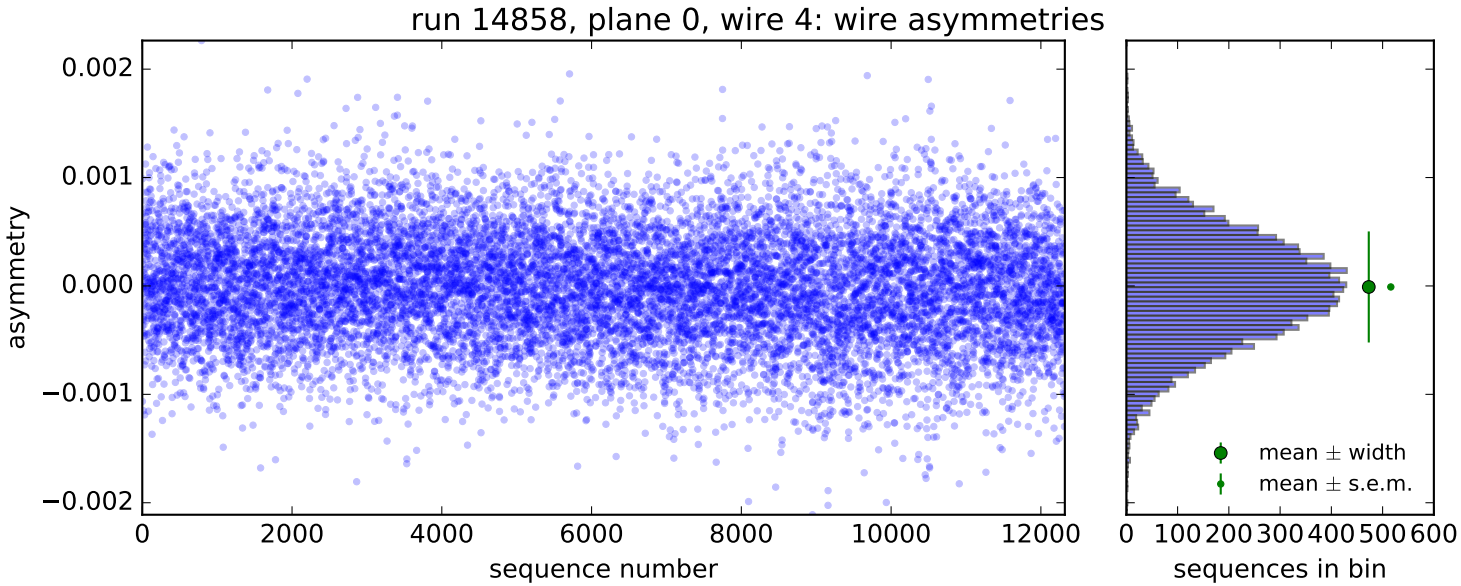


Cuts at this stage are

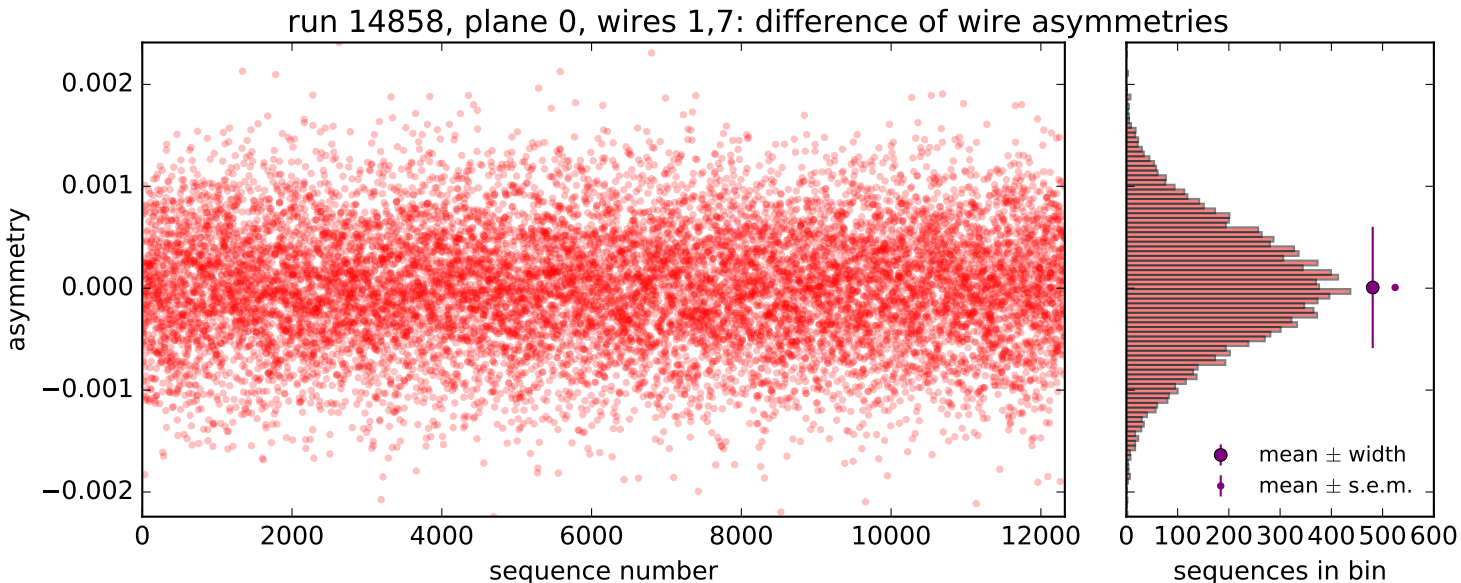
- Nearby missed pulses. One before miss is bad, five after miss is good again.
- Spin flipper phase. Apparently during the PC data the spin flipper phase was shifted compared to now?

Run-level asymmetries in each channel

For each run, find individual asymmetries. This one is $-9.4(4.6) \times 10^{-6}$.

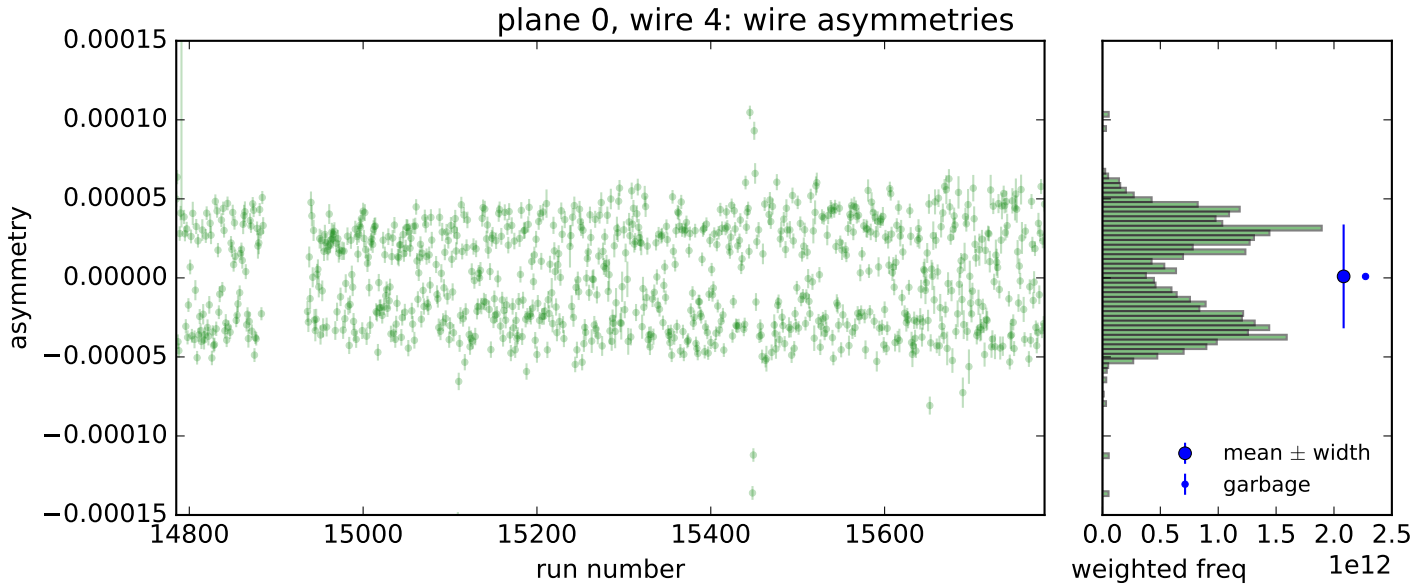


Also find the combination asymmetries. Here's $A_{01} - A_{07} = 7.0(5.4) \times 10^{-6}$

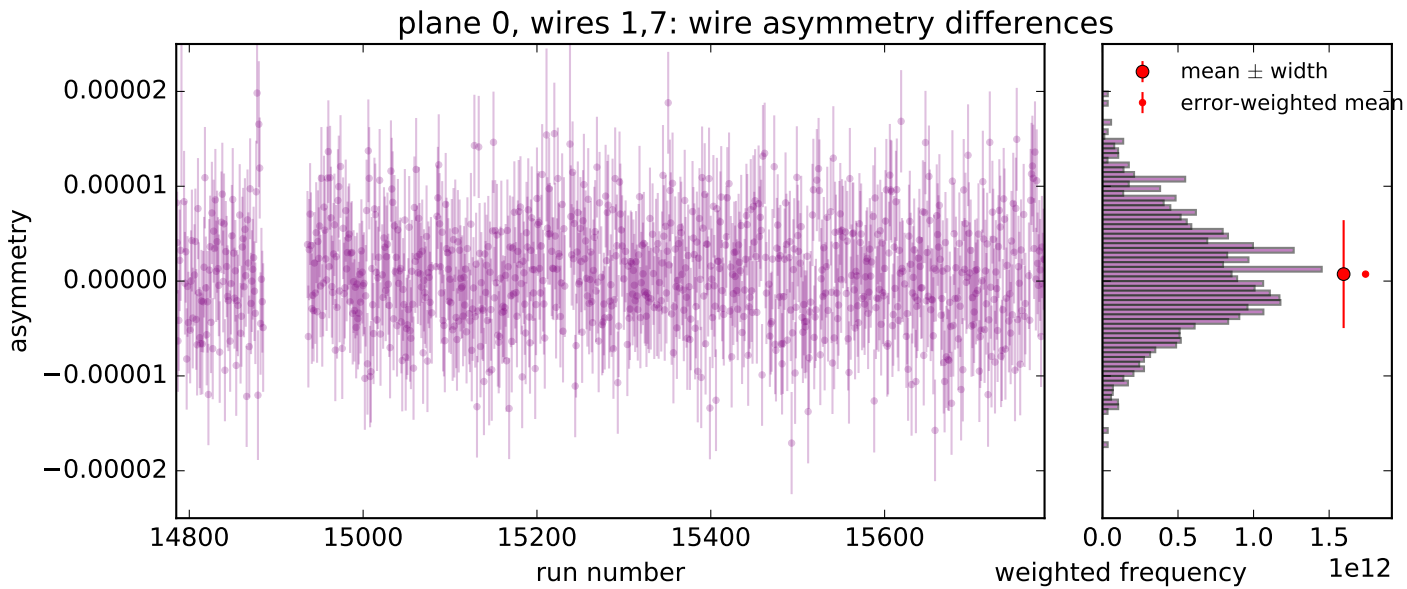


Combine run-level asymmetries across dataset

Combine run-level asymmetries over many runs. Single-wire asymmetries: ugh.



The wire difference asymmetries are drawn from a simpler distribution.

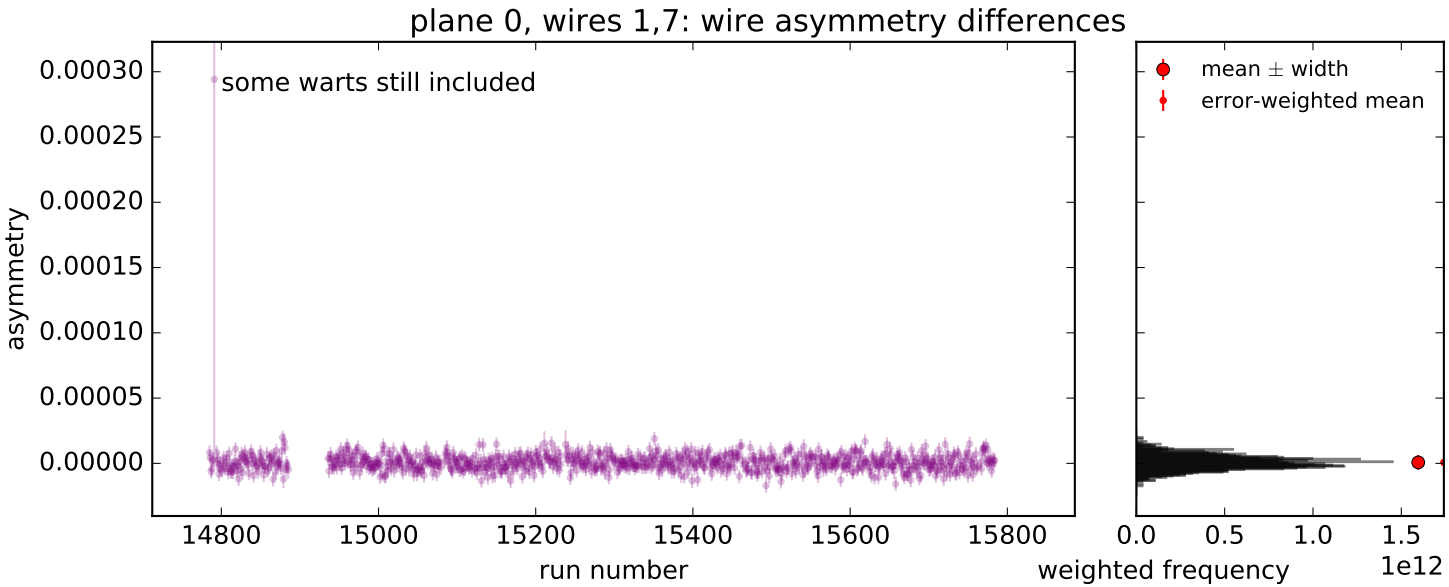


Some warts still included

The error-weighted average for the (purple) asymmetry difference is

$$A_{01} - A_{07} = (7.3 \pm 1.8) \times 10^{-7}$$

with χ^2/d.o.f. $= 994/895$. Not bad!



No further cuts so far.

So close, but not there yet

Current status

- Have computed results for 896 runs in the parity-conserving configuration 14785–15784
  - wire asymmetries (144)
  - conjugate-pair sums of wire asymmetries (64)
  - conjugate-pair differences of wire asymmetries (64)
  - asymmetries of conjugate-pair signal ratios (64)
- Asymmetry computation from completed run summaries at rate of about 10/minute

In progress

- Currently computing run summaries at 100–150 per hour, with 20k/40k completed.
- Nice plotting/tabulating of existing results (bummer)
- More results: after holiday

Not done

- Geometry factors
- Correlations