

blimpy Test Coverage

Lukas Finkbeiner

May 1, 2020

1 Introduction and Background

Test coverage is good. We want more. Also documentation. Keep issues alive and open. Also, test centralization.

At various points the tone of this report will shift toward advice, in case some future intern wishes to contribute.

2 Methods

I isolate the script that I want to work on.

The blimpy repository includes two scripts necessary for running the complete suite of tests: `download_data.sh` and `run_tests.sh`. In this repository I have included two pared-down bash scripts, `fast.sh`, `fail.sh`. `fail.sh` is the fastest, but can only be used to check whether new test cases pass. `fast.sh` is the next fastest, and will check the coverage of all scripts in the blimpy directory. Simpler bash scripts have the advantage of lower overhead, which allows the programmer to easily switch between writing tests and examining their impact. The principle downside of this pared-down approach is that `fast.sh` ignores repository warnings regarding which scripts are to be tested. Consequently, the programmer will receive coverage results about scripts in the directories ‘`calib_utils`’ and ‘`deprecated`’ (observe the absence of such scripts from the tables below).

3 Coverage Results

The following two tables record the initial (as of February 2020) state of test coverage as well as the state as of the writing of this report. Each table represents a similar view to what one will receive from running `run_tests.sh` in the shell. I have shortened the script names unless the directory is necessary for disambiguation.

Script	Statements	Misses	Coverage (%)
__init__	25	8	68
bl_scrunch	45	25	44
dice	103	48	53
ephemeris/__init__	3	0	100
compute_lsrk	28	0	100
compute_lst	15	4	73
ephemeris/config	9	2	78
observatory	41	12	71
fil2h5	42	21	50
guppi	271	137	49
h52fil	42	21	50
io/__init__	2	0	100
fil_writer	41	8	80
file_wrapper	397	101	75
hdf_writer	87	20	77
sigproc	157	28	82
match_fils	74	61	18
plotting/__init__	7	0	100
plotting/config	11	2	82
plot_all	69	5	93
plot_kurtosis	16	2	88
plot_spectrum	39	6	85
plot_spectrum_min_max	44	5	89
plot_time_series	26	3	88
plot_utils	28	4	86
plot_waterfall	28	4	86
utils	65	0	100
waterfall	228	57	75
TOTAL	1925	581	70

Script	Misses	Decrease	Latest Coverage (%)
bl_scrunch	25	12	72
dice	38	10	63
fil2h5	8	13	82
guppi	77	60	72
h52fil	8	13	82
file_wrapper	397	84	79
match_fils	0*	10	100*
waterfall	51	6	78
TOTAL [†]	1925	581	70

* asterisk indicates deprecation-assisted numbers * dagger indicates we are including numbers from unchanged tests

4 Current Efforts and Obstacles

The guppi file-handling script `guppi.py` represented a thorn in the side of this project. Interacting with this script was especially slow. On many occasions, even running tests on this script in isolation, the code-coverage routine would experience a time-out, killing the test session. In visually analyzing the script, I have not come across major areas which could be sped up to resolve this issue. However, virtually all of the user-facing methods (e.g. `histogram`) deplete the file. As a consequence, each test case must create a new guppi-handler object from scratch. This places a major strain on the hard disk, the slowest piece of a computer.

`bl_scrunch` and `h52fil.py`

* The `cmd_tool` does not accept arguments passed into the function (since it is supposed to be run from the command line).

* However, there is one script which allows this: `guppi.py`

Do we want more data? The test cases will continue to slow down. Diminishing returns, but we will be able to weed out if statements every here and there

5 Concluding Recommendations

In conclusion, we need some deprecation notices (`match_fils.py`). We also need to exclude some methods from our tests, because they raise `NotImplementedError` (`file_wrapper.py`).

Guppi sucks!! I HATE it! Especially because we keep getting killed. File depletion is a major nuisance. And, hard-coding represents a total off-shoot of this project.

We need more documentation all-around. Some methods are fairly self-explanatory, they just run one function and make sure that there are no runtime errors. Other methods, especially those involving command line options, are a bit more involved.

Do we want more rigorous checks for accuracy? The development process will continue to slow down. Error checking? Not likely, but it is something to think about...