Numpy

Adarsh Chagantipati, Sai Balasubrahmanya Dheeraj Gandikota, Khalid Hourani

May 1, 2022

Contents

1 Introduction

The Numpy package is a "fundamental package for scientific computing with Python"[5]. Created in 2005, the package contains a mature set of libraries for numerical applications and a robust set of tests for testing both Python and C code. Numpy is used for linear algebra, Fourier Transform, and random number capabilities. NumPy can also be used as an efficient multi-dimensional container of generic data. The project is sophisticated with functions and tools for integrating C/C++ and Fortran code.

Having a large project takes time to build, e.g. we must acquire all libraries with the exact versions supported for the build, check out all requirements, test requirements, etc. In order to build the project, we cloned our given repository from the Git link provided. It took us around 2 hours to have the setup with all the requirements, installing libraries, test suite requirements, and by using the command

```
pip install -r doc_requirements.txt\
            -r linter_requirements.txt\
            -r release_requirements.txt\
            -r test_requirements.txt
python runtests.py -coverage -m
```

After building, we see how many tests have passed, how many failed, how many have been skipped, and the total coverage of files. A folder coverage is added to the repository, containing a coverage report for the overall project, as well as each individual file in the project. This report covers the number of lines, number of missing statements, number of excluded statements, number of branches, partial code coverage, and overall code coverage of each file in the project, as well as for the project as a whole.

PyDriller is used to parse the data in repo. With it, we parse commits and observe the properties of files in each commit, such as when the file has been added to git, by whom, what type of commit (modify vs. new file vs. delete vs. rename), how many commits, date of modification., etc. Using this tool, we generate a report based on when the test files have been added to the project, how often they are modified, and how many people were involved.

Looking at the project from the perspective of a developer, the NumPy project covers all functions and is well written. A developer must be careful to test his code (i.e., ensure that the program is working as expected) by having tests for each of the scenarios, such as

- positive test cases
- negative test cases

- neutral test cases (bug free, exception handling, etc.)

while moving fast. On the other hand, a *tester* needs to design a system that performs different types of testing, such as

- black box testing
- white box testing
- regression testing

to test all functionality for the project.

2 Testing

2.1 Code Structure

The filenames of all tests are prefixed with `test_` and are found inside of a testing folder.

2.2 How to Test

The NumPy library provides a testing module, `runtests.py`, which can be run using with various configuration flags[8].

Included is a bash script[6] which

1. clones the data from GitHub[1]
2. runs the included test suite
3. generates the coverage reports[2]
4. parses the repo for relevant information (using `grep`[4], `find`[3], and PyDriller[7])
5. and generates this report[3]

Our coverage report is run using the `--mode fast` flag. A more comprehensive set of tests can be run by generating the report with the `--mode full` flag, i.e., by calling

```
$ bash generate_data.sh -m full
```

2.3 Number of Tests

Altogether, there are 273 test files and 221 production (Python) files. Using `find`, we are able to locate all files inside of `test` directories with the following command:

```
$ find "$SRCDIR"
    -type f\
    ! -wholename "*/build/*"\
    -wholename "*/tests/*.py"
```

and similarly, we can locate all production (Python) files via

```
$ find "$SRCDIR"
    -type f\
    ! -wholename "*/build/*"\
    ! -wholename "*/tests/*"\
    -wholename "*.py"
```

[1]We use the release tagged `v1.21.6`.
[2]Note, this script assumes no tests fail. If a test fails, the script will terminate early.
[3]We use LaTeXto generate this report.

Tables of assert statement counts are given in tables 1 and 2.

File Name	Number of Assert Statements
utils.py	142
timer_comparison.py	75
testArray.py	66
testutils.py	42
linalg.py	33
testVector.py	32
testTensor.py	29
testSuperTensor.py	28
testMatrix.py	28
ccompiler_opt.py	20
misc_util.py	12
testFarray.py	10
utils.py	9
__init__.py	8
cythonize.py	7
versioneer.py	6
parameterized.py	5
__init__.py	5
openblas_support.py	5
function_base.py	4
nanfunctions.py	4
Total	622

Table 1: File Name and number of assert statements in production files. Only counts for the top 20 files are given, but the total is across all production files.

File Name	Number of Assert Statements
test_multiarray.py	1806
test_core.py	1602
test_numeric.py	938
test_function_base.py	926
test_umath.py	905
test_datetime.py	834
test_nditer.py	629
test_generator_mt19937.py	527
test_ufunc.py	526
test_extras.py	494
test_regression.py	471
test_randomstate.py	431
test_linalg.py	396
test_random.py	366
test_old_ma.py	351
test_io.py	351
test_utils.py	330
test_dtype.py	268
test_indexing.py	256
test_einsum.py	242
Total	19657

Table 2: File Name and number of assert statements in test files. Only counts for the top 20 files are given, but the total is across all test files.

2.4 Coverage

Coverage is given in table 3. Notice that both the Python and C code achieve greater than 80% coverage, a sign of extensive regression testing. The full coverage report, including a breakdown by file (and an ability to inspect coverage on a line-by-line, function-by-function, etc., basis), is given in coverage/index.html. The C code coverage is given in lcov/index.html.

Language	Statements	Missing	Excluded	Branches	Partial	Coverage
Python	99472	12806	0	27149	2254	84%
C	54333	13540				80.1%

Table 3: Table of Coverage information. Python coverage is generated using the `Coverage` library[1]. C coverage is generated using `gcov`[2].

2.5 Activity

The *number* of committers (both major and minor — a file's minor committers are those who contribute to less 5% of the file's changes) is fairly stable (see fig. 1). However, the number of commits overall appears to be trending upwards, with local peaks every two to three years (see fig. 2). Additionally, the graph of the number of commits and the graph of the number of major/minor contributors are all qualitatively similar.

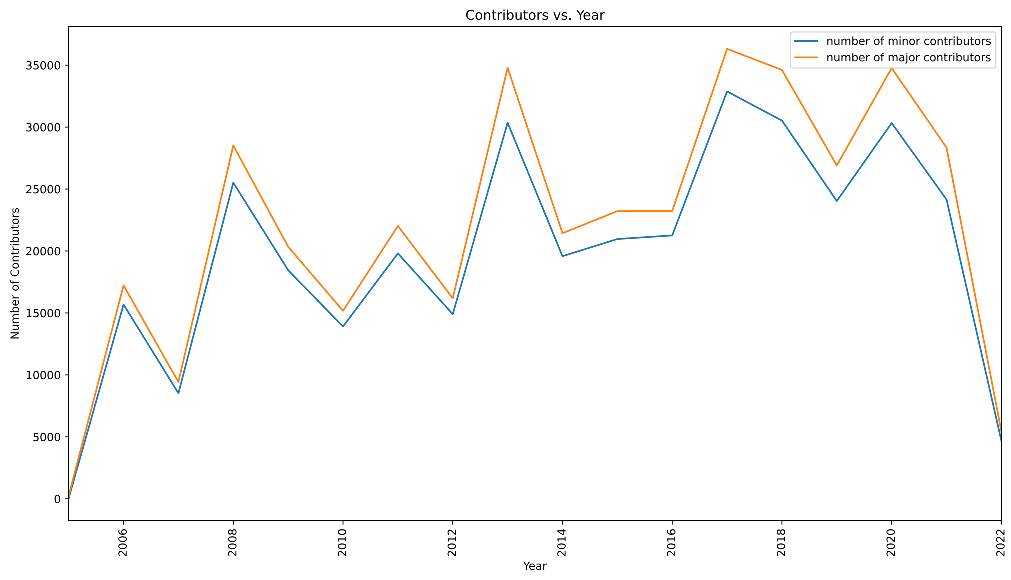

Figure 1: Number of major/minor contributors vs. year.

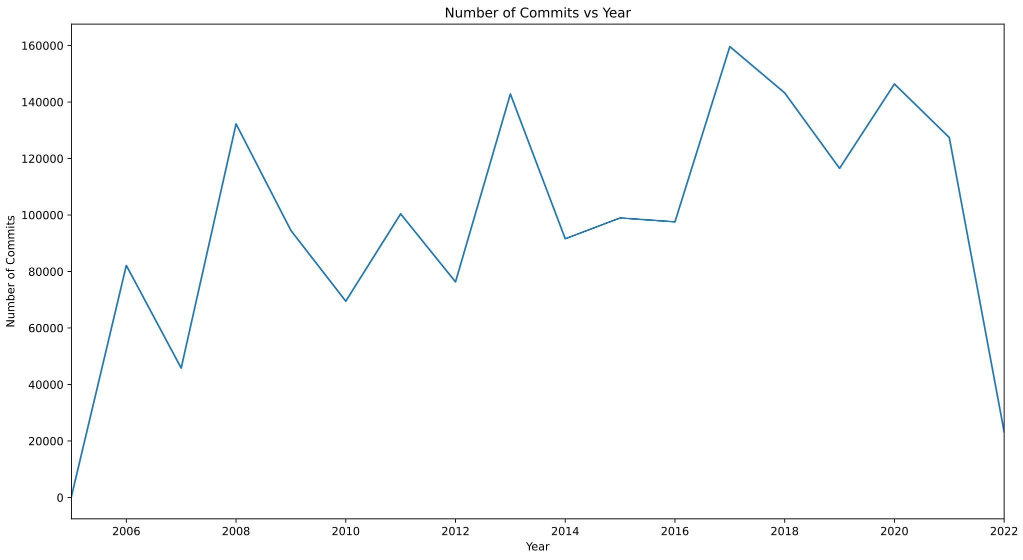

Figure 2: Number of commits vs. year. Notice the graph is qualitatively very similar to fig. 1.

As expected for a project like this, the commits overwhelmingly consist of modifications (see fig. 3).

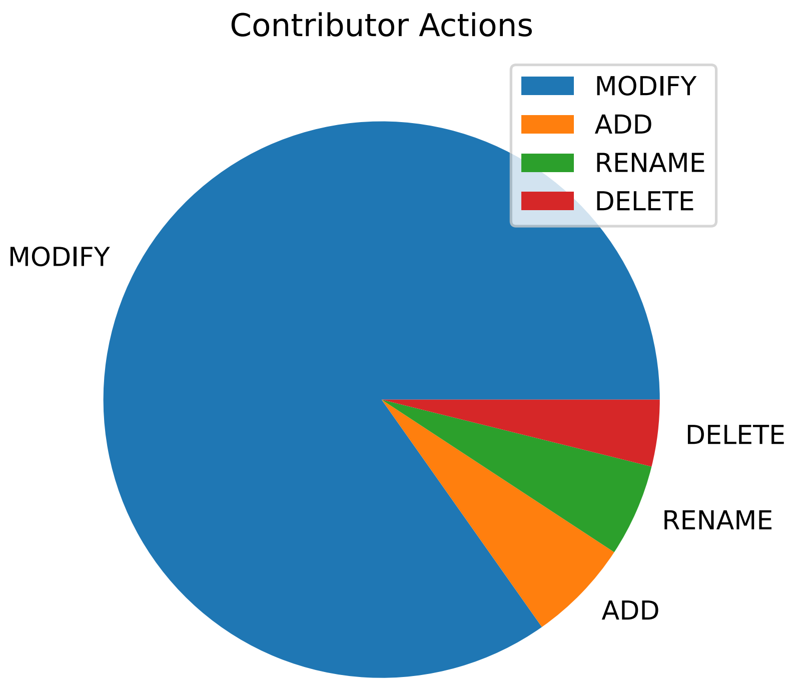

Figure 3: A pie chart of commit types. `Modify` commits are overwhelmingly the most common.

Finally, we notice that the top 10 major contributors are also the top 10 minor contributors (figs. 4 to 6) and also make the most overall contributions (both major and minor).

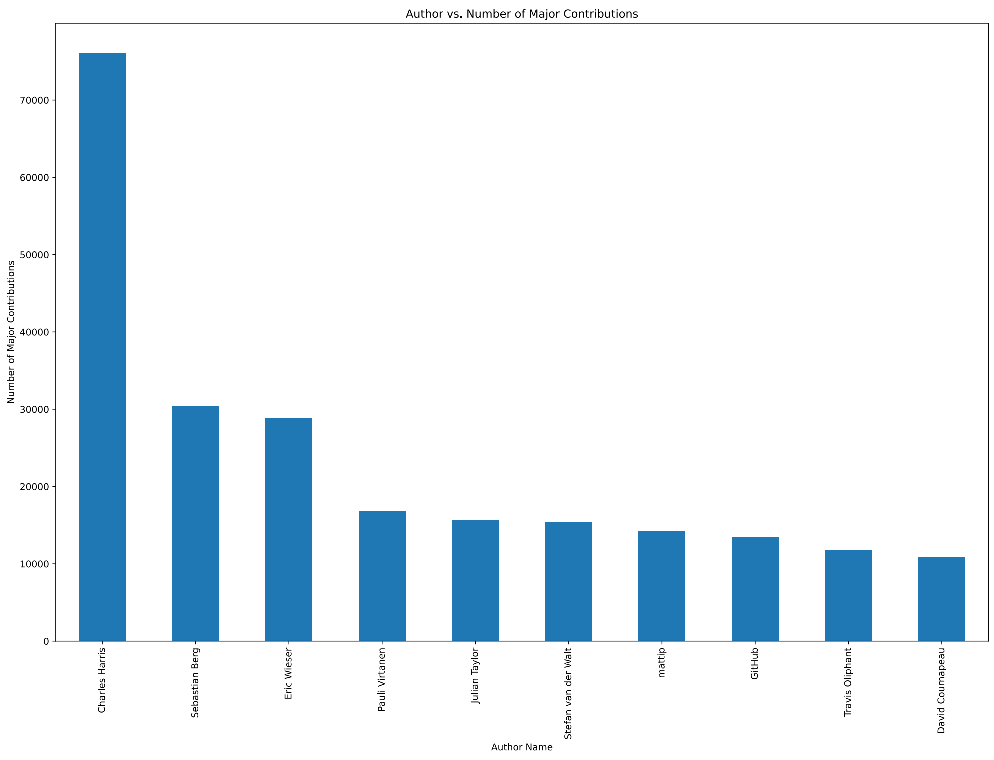

Figure 4: A bar graph of the top 10 largest major contributors. A major contributor for a file is any contributor who contributed more than 5% to the file. Notice that Charles Harris is the largest contributor by a substantial margin.

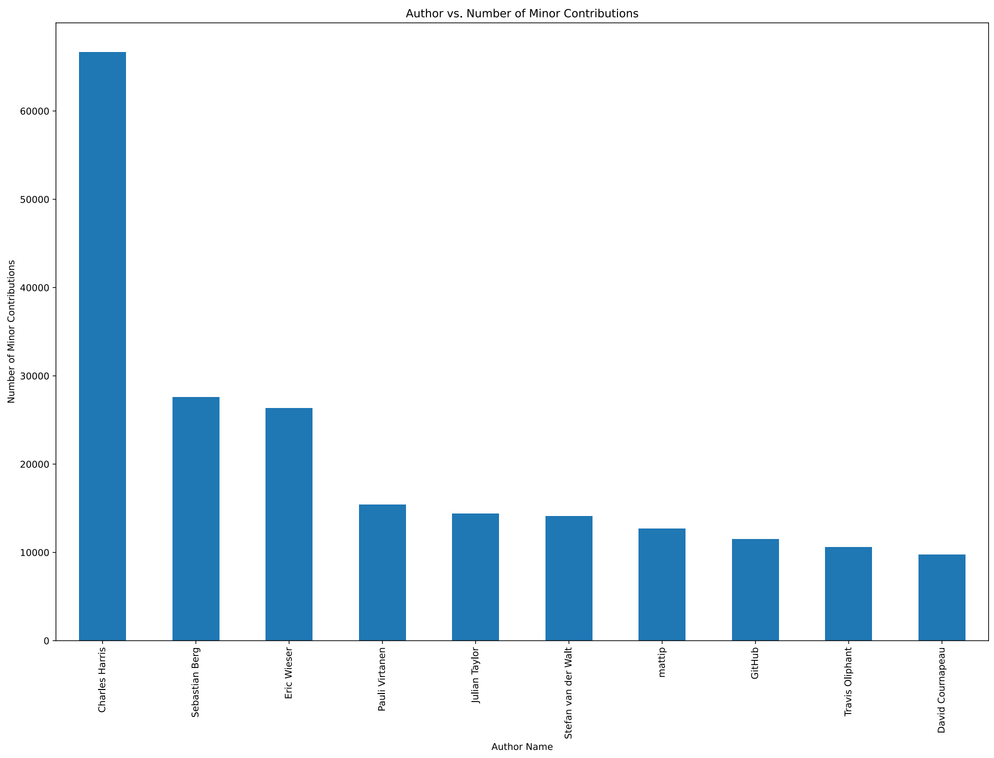

Figure 5: A bar graph of the top 10 largest *minor* contributors. A *minor* contributor for a file is any contributor who contributed less than 5% to the file. Notice that Charles Harris is the largest contributor by a substantial margin.

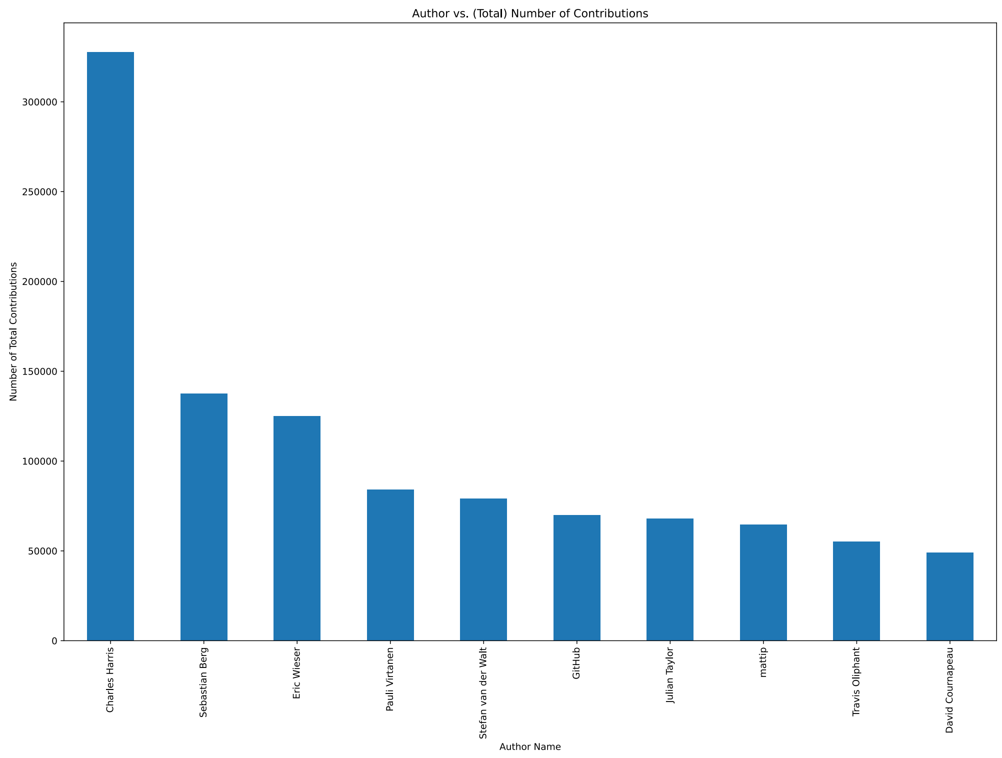

Figure 6: A bar graph of the top 10 largest *total* contributors. Once again, notice that Charles Harris is the largest contributor by a substantial margin.

3 Conclusion

Numpy is a mature library with very high quality tests. As we can see from fig. 1, tests are added and modified regularly, with test modifications being overwhelmingly more common (see fig. 3). With test coverage greater than 80% for both Python and C code (table 3) as well as a large number of assert statements (especially in test files), this code base is set up to be productive for developers — able to focus on development — while having tests that allow for quickly finding regressions. Additionally, the discrete separation between testing and production files allows for testers to quickly design and update unit and regression tests.

References

[1] *Coverage.py*. URL: https://coverage.readthedocs.io/en/6.3.2/.

[2] *gcov*. URL: https://gcc.gnu.org/onlinedocs/gcc/Invoking-Gcov.html#Invoking-Gcov.

[3] *Gnu Findutils*. URL: https://www.gnu.org/software/findutils/manual/html_mono/find.html.

[4] *Grep*. URL: https://www.gnu.org/software/grep/.

[5] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585 (2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[6] *PAT Project*. URL: https://github.com/patdak/pat-project.

[7] Davide Spadini, Maurício Aniche, and Alberto Bacchelli. "PyDriller: Python Framework for Mining Software Repositories". In: *The 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 2018. DOI: 10.1145/3236024.3264598.

[8] *Testing guidelines¶*. URL: https://numpy.org/devdocs/reference/testing.html.