Proposal Title

Streamlined Special Function Development in SciPy

Two Sentence Summary of Proposal

SciPy has dozens of open bug reports and enhancement requests that are easily solved in principle, requiring only the implementation of a well-defined mathematical function, yet these may be formidable in practice because implementing such functions in SciPy requires programming in a compiled language. To streamline the process of adding special functions to SciPy, we will develop an infrastructure for three of most common techniques used to compute special functions (namely series, continued fractions, and Padé approximants), enable the definition of special functions using pure-Python code, and document the process of implementing special functions.

Description of Proposal (<750 words, < 4500 chars)

In mathematics, a “special function” is a function encountered frequently in science or mathematics that is not considered to be an “elementary function”, like the logarithm or the trigonometric functions. Examples include the normal distribution function, which is the integral of the Gaussian probability density function, and the gamma function, a generalization of the factorial function to complex numbers. In Python, implementations of the most common special functions are available in `scipy.special`. For convenience, they behave as NumPy universal functions (`ufunc`s), operating on arrays in an element-by-element fashion, and for speed, this requires that they are implemented in a compiled language like C++ or Fortran instead of Python. Consequently, development of `scipy.special` is much slower than other SciPy subpackages: fewer than 400 `scipy.special` PRs have been merged in the past decade, only about one-third the number merged into `scipy.stats`.

`scipy.special` is used not only by dependent packages, but also in other SciPy subpackages, including `scipy.optimize`, `scipy.spatial`, `scipy.interpolate`, `scipy.integrate`, and `scipy.stats`. Quite often, bug reports and enhancement requests in these subpackages could easily be resolved if only the appropriate special function were available in `scipy.special`. For example, [1] was filed nearly a decade ago, but it is held up by a need for the unregularized incomplete beta function, which is well-defined mathematically and has several methods for numerical computation, such as by power series or hypergeometric functions. Part of the hesitation to implementing the function stems from the challenges imposed by finite-precision arithmetic, but another substantial barrier is the need to work with compiled code, which only a relatively small subset of SciPy contributors and maintainers are comfortable doing.

To streamline the process of special function development, we will:

- Create an infrastructure for implementing special functions via series approximations and continued fractions. Once the contributor defines a function to evaluate individual terms of the series or continued fraction, the infrastructure will automatically generate a ufunc that uses it to compute as many terms as needed to achieve the required accuracy.
- Create an infrastructure for implementing Padé approximants. Using Taylor series coefficients provided by the contributor, the infrastructure will automatically generate a ufunc to evaluate the approximant.

- Provide a framework for implementing special functions via custom Python code, which will be translated into compiled code using Pythran.
- Provide a framework for stitching together complementary implementations of the same special function, each to be used within the domain where it is most accurate. For example, a Padé approximant may be most appropriate for small values of the argument, whereas a continued fraction implementation may be required for larger values.
- Write detailed documentation of the features described above, the process for wrapping special functions from Boost.math, best practices for implementing numerically robust functions (e.g. using logarithms to avoid underflow/overflow), and best practices for testing new special functions.

Finally, we will generate awareness of this work by applying the features to resolve open issues [1-2] and reviewing relevant PRs by other contributors.

References:

[1] Evgeni Burovski. "implement cdf/sf for logser distribution · Issue #3890 · scipy/scipy". https://github.com/scipy/scipy/issues/3890.

[2] Lander Bodyn. "Scipy.stats.poisson underflow/overflow + solution · Issue #8424 · scipy/scipy". https://github.com/scipy/scipy/issues/8424.

[3] @ivarzap. "BUG: incorrect levy_stable survival function for large values · Issue #17194 · scipy/scipy". https://github.com/scipy/scipy/issues/17194.

[4] Daniel Schmitz. "ENH: more analytic formulas to calculate the entropy of a distribution · Issue #17748 · scipy/scipy". https://github.com/scipy/scipy/issues/17748.

[5] Daniel Schmitz. "Tracking issue: custom methods for continuous distributions · Issue #17832 · scipy/scipy". https://github.com/scipy/scipy/issues/17832.

[6] 98 open scipy.special issues. https://github.com/scipy/scipy/issues?page=4&q=is%3Aissue+is%3Aopen+label%3Ascipy.special

Benefit to Project/Community (<400 words, < 2500 chars)

Please explain the benefit of this proposal including:

- *Impact to the project*
- *Impact to the scientific ecosystem*
- *Impact to the community*

This work will impact the SciPy project by addressing several open issues that require a new special function implementations, namely [1] and [2], and it will pave the way toward resolving other, similar issues [3-6].

This work will benefit the scientific Python ecosystem by seamlessly improving the performance of dependent libraries that rely on SciPy functionality fixed or enhanced by this work. Furthermore, although the frameworks themselves will be private initially, it may be possible to make them public in the future, enabling dependent libraries to define their own special functions with ease. Finally, the

documentation of best practices for implementing numerically robust functions will be beneficial for all scientific Python users, as these practical tips (e.g. the `logsumexp` trick) are rarely found in one place.

This work will strengthen the SciPy community by increasing the time that core developers can allocate to reviewing the pull requests of others, which is essential to mentoring new contributors and retaining other repeat contributors. For example, relatively new contributors @dschmitz89 and @MatteoRaso have been very actively adding special functions related to statistical distributions, and their work will benefit tremendously from these new tools and the attention of the maintainers supported by this project.

Amount Requested

$\$x + w$

Brief Budget Justification - How will the money be spent?

$\$x$ ($\$y$ per hour $\times$ z hours) will compensate work by Albert Steppi and is sufficient to fund this proposal.

Tirth Patel has agreed to review Albert's pull requests on a voluntary basis. Consequently, we request an additional $\$w$ to support Albert Steppi for other `scipy.stats` maintenance and review of `scipy.stats` pull requests to free Tirth's time for review of this work.

Timeline of Deliverables

All dates listed are the anticipated date of a PR posted to SciPy's GitHub repository. At least two weeks will be required for community review and merge.

April 2023 – Decision notification.

May 2023 – Framework for implementing special functions using Pythran.

June 2023 – Infrastructure for Padé approximants and corresponding documentation.

July 2023 – Infrastructure for series approximations and corresponding documentation.

August 2023 – Infrastructure for continued fractions and corresponding documentation.

September 2023 – Framework for stitching together complementary implementations of a given special function, and documentation of adding special functions from Boost.math.

October 2023 – PRs to resolve gh-3890 [1] and gh-8424 [2] to demonstrate the applicability of the work, and additional best practices documentation. Submit final report.

Has someone been identified to carry out the work in the proposal?

Yes: Albert Steppi and Tirth Patel. Albert and Tirth are ideally suited to the work because they are both regular maintainers of `scipy` and have complementary skill sets: Albert's doctoral work involved numerical analysis and implementation of special functions, and Tirth is one of relatively few SciPy experts in bridging compiled code and Python. As maintainers, they are able to review and merge the work of one another without relying on volunteers to review.

Please list the name and email address of a project leader(s) who has approved this proposal. *

Matt Haberland <matt.haberland@gmail.com>

I agree to submit a grant report-back if my proposal is selected for funding.

I agree.