SciPy 1.0—Fundamental Algorithms for Scientific Computing in Python

Pauli Virtanen[1], Ralf Gommers[2,*], Tyler Reddy[3,4], Anne Archibald[5], Andrew R. J. Nelson[6], Charles Harris[7], CJ Carey[8], Denis Laxalde[9], Eric Larson[10], Eric Moore[11], Eric Quintero[12], Evgeni Burovski[13], Jaime Fernández del Río[14], Josef Perktold[15], Josh Wilson[16], Matthew Brett[17], Nikolay Mayorov[18], Warren Weckesser[19], Matt Haberland[20], Scott Sievert[21], Yu Feng[22], Antonio Horta Ribeiro[23], Ian Henriksen[24], K. Jarrod Millman[3,25], Stéfan J. van der Walt[3], and İlhan Polat[26]

[1]Affiliation, department, city, postcode, country
[2]Affiliation, department, city, postcode, country
[2]Affiliation, department, city, postcode, country
[3]Berkeley Institute for Data Science, University of California, Berkeley, CA, 94720, USA
[4]Los Alamos National Laboratory, Theoretical Division 6, Los Alamos, NM, 87545, USA
[5]Affiliation, department, city, postcode, country
[6]Australian Nuclear Science and Technology Organisation, Locked Bag 2001, Kirrawee DC, NSW 2232, Australia
[7]Affiliation, department, city, postcode, country
[8]Affiliation, department, city, postcode, country
[9]Affiliation, department, city, postcode, country
[10]University of Washington, Institute for Learning and Brain Sciences, Seattle, WA, 98195, USA
[11]Affiliation, department, city, postcode, country
[12]Affiliation, department, city, postcode, country
[13]Affiliation, department, city, postcode, country
[14]Affiliation, department, city, postcode, country
[15]Affiliation, department, city, postcode, country
[16]Affiliation, department, city, postcode, country
[17]Affiliation, department, city, postcode, country
[18]Affiliation, department, city, postcode, country
[19]Affiliation, department, city, postcode, country
[20]BioResource and Agricultural Engineering, California Polytechnic State University, San Luis Obispo, CA, 93407, USA
[21]Affiliation, department, city, postcode, country
[22]Affiliation, department, city, postcode, country
[23]Affiliation, department, city, postcode, country
[24]University of Texas at Austin, Oden Institute for Computational Engineering and Sciences, Austin, TX, 78712, USA
[25]Division of Biostatistics, University of California, Berkeley, CA, 94720, USA
[26]Affiliation, department, city, postcode, country
[*]ralf.gommers@gmail.com

ABSTRACT

SciPy is an open source scientific computing library for the Python programming language. SciPy 1.0 was released in late 2017, about 16 years after the original version 0.1 release. SciPy has become a *de facto* standard for leveraging scientific algorithms in the Python programming language, with more than 600 unique code contributors, millions of downloads per year, 161 dependent packages, and 28700 dependent repositories. This includes usage of SciPy in almost half of all machine learning projects on GitHub, and usage by high profile projects including LIGO gravitational wave analysis and creation of the first-ever image of a black hole (M87). The library includes functionality spanning clustering, Fourier transforms, integration, interpolation, file I/O, linear algebra, image processing, orthogonal distance regression, minimization algorithms, signal processing, sparse matrix handling, computational geometry, and statistics. In this work, we provide an overview of the capabilities and development practices of the SciPy library and highlight some recent technical developments.

Introduction

SciPy is a library of numerical routines for the Python programming language that provides fundamental building blocks for modeling and solving scientific problems. SciPy includes algorithms for optimization, integration, interpolation, eigenvalue problems, algebraic equations, differential equations, and many other classes of problems; it also provides specialized data structures, such as sparse arrays and *k*-dimensional trees. SciPy is built on top of NumPy[1,2], which provides array data structures and related fast numerical routines, and SciPy is itself the foundation upon which higher level scientific libraries, including scikit-learn[3] and scikit-image[4], are built. SciPy is relied upon by scientists, engineers, and others around the world. For example, published scripts[5,6] used in the analysis of gravitational waves[7,8] import several subpackages of SciPy, and the M87 black hole imaging project directly cites SciPy[9].

Recently, SciPy released version 1.0, a milestone that traditionally signals a library's API (Applications Programming Interface) being mature enough to be be trusted in production pipelines. This version numbering convention, however, belies the history of a project that has become the standard which others follow and has seen extensive adoption in research and industry.

SciPy's arrival at this point is surprising and somewhat anomalous. When started, the library had little funding, and was written mainly by graduate students—many of them without a computer science education, and often without the blessing of their advisors. To even imagine that a small group of "rogue" student programmers could upend the already well-established ecosystem of research software—backed by millions in funding and many hundreds of highly qualified engineers[10–12]—was preposterous.

Yet the philosophical motivations behind a fully open toolstack, combined with an excited, friendly community with a singular focus, have proven auspicious in the long run. They led not only to the library described in this paper, but also to an entire ecosystem of related packages[13], and a variety of social activities centered around them[14]. The packages in the SciPy ecosystem share high standards of implementation, documentation, and testing, and a culture eager to learn and adopt better practices—both for community management and software development.

In the background section that follows, we capture a selective history of some milestones and important events in the growth of SciPy. Despite what we might highlight here, it is important to understand that a project like SciPy is only possible because of the contributions of very many contributors—too many to mention individually, but each bringing an important piece to the puzzle.

Background

Python is an interpreted, high-level, general-purpose computer programming language, designed by Guido van Rossum in the late 1980s, with a dynamic type system and an emphasis on readability and rapid prototyping. The reference and most popular implementation of Python is CPython[15,16], which is written in the C and Python languages and assumed throughout this paper. As a general purpose programming language, it had no special support for scientific data structures or algorithms, unlike many of the other established computation platforms of the time. Yet, scientists soon discovered the language's virtues, such as its ability to wrap C and Fortran libraries and to then drive those libraries interactively. Scientists could thereby gain access to a wide variety of existing computational libraries without concerning themselves with low-level programming concepts such as memory management.

In 1995, Jim Hugunin, a graduate student from MIT, wrote the first message in a new Python Matrix Special Interest Group (Matrix-SIG) mailing list[17]:

> There seems to be a fair amount of interest in the Python community concerning the addition of numeric operations to Python. My own desire is to have as large a library of matrix based functions available as possible (linear algebra, eigenfunctions, signal processing, statistics, etc.). In order to ensure that all of these libraries interoperate, there needs to be agreement on a basic matrix object that can be used to represent arrays of numbers.

Over the next several months, conversations on that mailing list by, among others, Jim Fulton, Jim Hugunin, Paul Dubois, Konrad Hinsen, and Guido van Rossum led to the creation of a package called Numeric with an array object that supported a high number of dimensions. Jim Hugunin explained the utility of Python for numerical computation[18]:

> I've used almost all of the available numerical languages at one time or another over the past 8 years. One thing I've noticed is that over time, the designers of these languages are steadily adding more of the features that one would expect to find in a general-purpose programming language.

This remains a distinguishing feature of Python for science, and one of the reasons why it has been so successful in the realm of data science: instead of adding general features to a language designed for numerical and scientific computing, here scientific features are added to a general purpose language. This broadens the scope of problems that can be addressed easily, expands the sources of data that are readily accessible, and increases the size of the community that develops code for the platform.

```
SciPy is an open source package that builds on the strengths of Python and
Numeric providing a wide range of fast scientific and numeric functionality.
SciPy's current module set includes the following:

    Special Functions (Bessel, hanker, Airy, etc.)
    Signal/Image Processing
    2D Plotting capabilities
    Integration
    ODE solvers
    Optimization (simplex, BFGS, Netwon-CG, etc.)
    Genetic Algorithms
    Numeric -> C++ expression compiler
    Parallel programming tools
    Splines and Interpolation
    And other stuff.
```

Figure 1. Excerpt from SciPy 0.1 release announcement (typos included).

The availability of a standard numerical array data structure in Python led to a sudden, rapid growth in the number of scientific packages available for solving common numeric problems. A number of these packages were written by graduate students and postdoctoral researchers to solve the very practical research problems that they faced on a daily basis. While they had access to and were familiar with specialized (often commercial) systems, many found it easier to implement the domain-specific functionality they needed in a general purpose programming language. And given Python's innate ability to function as a systems language, controlling specialized or custom-built hardware was also possible.

SciPy Begins

By the late 1990s, discussions appeared on Matrix-SIG expressing a desire for a complete scientific data analysis environment[19]. Travis Oliphant, a PhD student at the Mayo Clinic, released a number of packages[20,21] that built on top of the Numeric array package, and provided algorithms for signal processing, special functions, sparse matrices, quadrature, optimization, Fast Fourier Transforms, and more. One of these packages, Multipack[22], was a set of extension modules that wrapped Fortran and C libraries such as ODEPACK[23], QUADPACK[24], and MINPACK[25], to solve and minimize nonlinear equations, integrate differential equations, and fit splines (the latter implemented by Pearu Peterson). Robert Kern, then an undergraduate student (and currently a SciPy core developer), provided compilation instructions under Windows. Around the same time, Pearu Peterson, a PhD student from Estonia, released F2PY[26], a command line tool for binding Python and Fortran codes, and wrote modules for linear algebra and interpolation. Eric Jones, while a graduate student at Duke, wrote a number of packages to support his dissertation, including a parallel job scheduler and genetic optimizer. Gary Strangman, a postdoctoral fellow at Harvard Medical School, published several descriptive and inferential statistical routines[27].

With a rich programming environment and a numerical array object in place, the time was ripe for the development of a full scientific software stack. In 2001, Eric Jones and Travis Vaught founded Enthought Scientific Computing Solutions (now Enthought, Inc.) in Austin, Texas. In an effort to simplify the tool stack, they created the SciPy project, centered around the SciPy library which would subsume all the above-mentioned packages. The new project quickly gained momentum, with a website and code repository appearing in February[28], and a mailing list announced in June[29]. By August, a first release was announced[30], an excerpt of which is shown in Fig. 1. In September, the first documentation was published[31]. The first SciPy workshop[32] was held in September 2002 at Caltech—a single track, two day event with 50 participants, many of them developers of SciPy and surrounding libraries.

At this point, scientific Python started attracting more serious attention; code that started out as side projects by graduate students had grown into essential infrastructure at national laboratories and research institutes. For example, Paul Dubois at Lawrence Livermore National Laboratory (LLNL) took over the maintenance of Numeric and funded the writing of its manual[33], and the Space Telescope Science Institute (STScI), which was in charge of Hubble Space Telescope science operations, decided to replace their custom scripting language and analysis pipeline with Python[34].

As SciPy, the algorithms library of the ecosystem, began attracting the attention of large research organizations, the next generation of graduate students and postdocs were already exploring other aspects of the computational environment. In 2000, Prabhu Ramachandran, a PhD student at the Indian Institute of Technology, started work on a 3D visualization application, based on Kitware's C++ Visualization Toolkit[35], called Mayavi[36]. In 2001, Fernando Pérez, a graduate student at the University

of Colorado, Boulder, created the IPython interactive shell, which eventually blossomed into Project Jupyter[37]. John Hunter, a postdoc at the University of Chicago, liked the plotting functionality available in MATLAB[38], but had problems accessing their laboratory's license, which was governed by a hardware dongle. In response, he wrote a plotting library from scratch, and Matplotlib 0.1 was released April 2003[39].

As STScI continued to use Python for an increasingly large portion of the Hubble Space Telescope data analysis pipeline, they encountered problems with the Python numerical array container. Numeric, the original array package, was suitable for small arrays, but not for the large images processed by STScI. With the Numeric maintainer's blessing, the decision was made to write NumArray[40], a library that could handle data on a larger scale. Unfortunately, NumArray proved inefficient for small arrays, presenting the community with a rather unfortunate choice. In 2005, Travis Oliphant combined the best elements of Numeric and NumArray, thereby solving the dilemma and paving the way for Python to become a very significant player in the Data Science movement. NumPy 1.0 was released in October 2006[41], with about 30 authors recognized for major contributions in its release notes.

In a May/June 2007 special issue of IEEE Computing in Science and Engineering, Paul Dubois wrote[42]:

> LLNL now has many Python-based efforts built from scratch or wrapped around legacy codes [. . .] hundreds of thousands of lines of C++, Python, and Fortran 95, all working together just as we hoped, doing compute-intensive calculations on massively parallel computers.

While it is now more commonly accepted, Dubois took the time to explain to the 2007 reader that "interpreted doesn't mean slow or only interactive." He also shared his own interest in Python for "computational steering" where "Python serves as the input language to a scientific application, and the actual computations are performed both in Python itself and in compiled extensions." Using Python for computational steering was one of the earliest uses of Python in large scientific processing pipelines. In addition to Dubois' overview there were also articles introducing Python and SciPy[43], IPython[44], and Matplotlib[45]. There was also a diverse set of research applications including systems biology[46], astronomy[47], robotics[48], nanophotonics[49], partial differential equations[50], neuroimaging[51], geographic information systems[52], and education[53,54].

SciPy Matures

By the middle to late 2000s, SciPy was starting to mature after a long phase of significant growth and adoption. The informal workshops grew into international conferences with many hundreds of attendees. Special issues were organized and published in a leading scientific journal[42], and the scope of the SciPy library narrowed, while the breadth of the ecosystem grew through a new type of auxiliary package: the scikit[55]. The tooling, development, and release processes became more professional. SciPy was expanded carefully, with the patience affordable in open source projects and via best practices common in industry[56].

In the early workshops, recurrent topics reflected the state of development, with emphasis being placed on the underlying array package, plotting, parallel processing, acceleration / wrapping, and user interfaces. By 2004, a significant shift occurred towards application of SciPy to scientific problems. The event also started to draw in more keynote speakers from outside the community, such as Guido van Rossum (creator of Python, 2006), Ivan Krstić (One Laptop per Child, 2007), Alex Martelli (Google, 2008), and Peter Norvig (Google Research, 2009). The SciPy conference went from being a small gathering of core developers to a multi-location event with increased funding, a published proceedings, and scholarships for attending students. By 2010, the US SciPy conference had multiple tracks, and satellite conferences were being organized by volunteers elsewhere, such as EuroSciPy (2008–) and SciPy India (2009–). Special sessions and minisymposia dedicated to scientific Python began appearing at many other events. For example, a three-part minisymposium organized for CSE 2009 was featured in SIAM News[57].

In 2007, Python had a strong enough presence in science and engineering that the editors of IEEE Computing in Science and Engineering (CiSE) solicited a special issue[42], edited by Paul Dubois. However, Python was still sufficiently niche that the average reader would need additional information in order to decide whether it would "be useful in their own work." The follow-up CiSE March/April 2011 Python for Scientists and Engineers special issue[58], focused more on the core parts of the scientific Python ecosystem[59] including NumPy[1], Cython[60], and Mayavi[61]. Python became so pervasive that journals began publishing domain-specific special issues. For example, in 2015 Frontiers in Neuroinformatics published a collection of 25 articles—covering topics including modeling and simulation, data collection, electrophysiology, visualization, as well as stimulus generation and presentation—called Python in Neuroscience[62].

In 2012, Perry Greenfield, John Hunter, Jarrod Millman, Travis Oliphant, and Fernando Pérez founded NumFOCUS[63], a 501(c)3 public charity with a mission "to promote sustainable high-level programming languages, open code development, and reproducible scientific research." While NumFOCUS is language agnostic, many of the early sponsored projects came from the scientific Python stack. Today it has 50 affiliated and sponsored projects including NumPy, SciPy, IPython, and Matplotlib. Among its other projects, NumFOCUS organizes a global network of community driven educational programs, meetups, and conferences called PyData.

SciPy Today

At the time of writing, the SciPy library consists of nearly 600,000 lines of code organized in 16 subpackages. Over 85,000 GitHub repositories and almost 5000 packages depend on SciPy. Some of the major feature highlights from the three years preceding SciPy 1.0 are discussed in the Key Technical Improvements section below, and milestones in its history are highlighted in Figure 2.

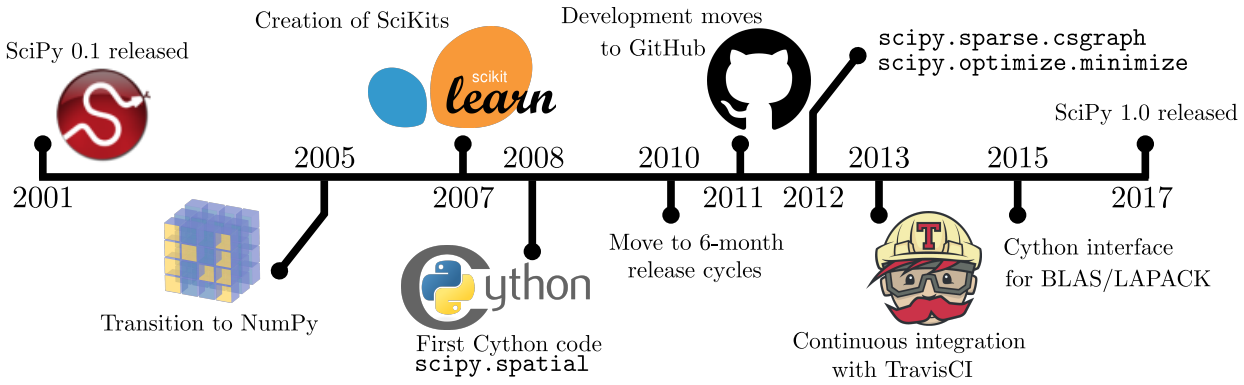

Figure 2. Major milestones from SciPy's initial release in 2001 to the release of SciPy 1.0 in 2017.

Architecture and implementation choices

Project scope

SciPy provides fundamental algorithms for scientific computing. The breadth of its scope was derived from the Guide to Available Mathematical Software classification system (GAMS[64]). In areas that move relatively slowly, e.g. linear algebra, SciPy aims to provide complete coverage. In other areas it aims to provide fundamental building blocks while interacting well with other packages specialized in that area. For example, SciPy provides what one expects to find in a statistics textbook (probability distributions, hypothesis tests, frequency statistics, correlation functions, and more), while Statsmodels[65] provides more advanced statistical estimators and inference methods; scikit-learn[3] covers machine learning; and PyMC3[66], emcee[67] and PyStan[68] cover Bayesian statistics and probabilistic modeling. scikit-image[4] provides image processing capabilities beyond SciPy's `ndimage`, sympy[69] provides a Python interface for symbolic computation, and while `sparse.csgraph` or `spatial` offers basic tools for working with graphs and networks compared to a more specialized Python library like NetworkX[70].

We use the following criteria to determine whether or not to include new functionality in SciPy:

- The algorithm is of relevance to multiple fields of science.
- The algorithm is demonstrably important. For example, it is classic enough to be included in textbooks, or it is based on a peer-reviewed article which has a significant number of citations.

In terms of software systems and architecture, SciPy's scope matches NumPy's: algorithms for in-memory computing on single machines, with support for a wide range of data types and process architectures. Distributed computing and support for graphics processing units (GPUs) are explicitly out of scope.

Package organization

The SciPy library is organized as a collection of subpackages. The 16 subpackages include mathematical building blocks (e.g. linear algebra, Fourier transforms, special functions), data structures (e.g. sparse matrices, k-D trees), algorithms (e.g. numerical optimization and integration, clustering, interpolation, graph algorithms, computational geometry), and higher-level data analysis functionality (e.g. signal and image processing, statistical methods).

Here we summarize the scope and capabilities of each subpackage.

`cluster` The `cluster` subpackage contains `cluster.vq`, which provides vector quantization and k-means algorithms, and `cluster.hierarchy`, which provides functions for hierarchical and agglomerative clustering.

`constants` Physical and mathematical constants, including the CODATA recommended values of the fundamental physical constants[71].

fftpack Fast Fourier Transform routines. In addition to the FFT itself, the subpackage includes functions for the discrete sine and cosine transforms and for pseudo-differential operators.

integrate The `integrate` subpackage provides tools for the numerical computation of single and multiple definite integrals and for the solution of ordinary differential equations, including initial value problems and two-point boundary value problems.

interpolate The `interpolate` subpackage contains spline functions and classes, one-dimensional and multi-dimensional (univariate and multivariate) interpolation classes, Lagrange and Taylor polynomial interpolators, and wrappers for FITPACK[72] and DFITPACK functions.

io A collection of functions and classes for reading and writing MATLAB[38], IDL, Matrix Market[73], Fortran, NetCDF[74], Harwell-Boeing[75], WAV and ARFF data files.

linalg Linear algebra functions, including elementary functions of a matrix, such as the trace, determinant, norm and condition number; basic solver for $Ax = b$; specialized solvers for Toeplitz matrices, circulant matrices, triangular matrices and other structured matrices; least squares solver and pseudo-inverse calculations; eigenvalue and eigenvector calculations (basic and generalized); matrix decompositions, including Cholesky, Schur, Hessenberg, LU, $LDL^{\intercal}$, QR, QZ, singular value, and polar; and functions to create specialized matrices, such as diagonal, Toeplitz, Hankel, companion, Hilbert, and more.

ndimage This subpackage contains various functions for multi-dimensional image processing, including convolution and assorted linear and nonlinear filters (Gaussian filter, median filter, Sobel filter, etc.); interpolation; region labeling and processing; and mathematical morphology functions.

misc A collection of functions that did not fit into the other subpackages. While this subpackage still exists in SciPy 1.0, effort is underway to deprecate or relocate the contents of this subpackage and remove it.

odr Orthogonal distance regression, including Python wrappers for the Fortran library ODRPACK[76].

optimize This subpackage includes simplex and interior-point linear programming solvers, implementations of many nonlinear minimization algorithms, a routine for least-squares curve fitting, and a collection of general nonlinear solvers for root-finding.

signal The `signal` subpackage focuses on signal processing and basic linear systems theory. Functionality includes convolution and correlation, splines, filtering and filter design, continuous and discrete time linear systems, waveform generation, window functions, wavelet computations, peak finding, and spectral analysis.

sparse This subpackage includes implementations of several representations of sparse matrices. `scipy.sparse.linalg` provides a collection of linear algebra routines that work with sparse matrices, including linear equation solvers, eigenvalue decomposition, singular value decomposition and LU factorization. `scipy.sparse.csgraph` provides a collections of graph algorithms for which the graph is represented using a sparse matrix. Algorithms include connected components, shortest path, minimum spanning tree and more.

spatial This subpackage provides spatial data structures and algorithms, including the k-d tree, Delaunay triangulation, convex hulls and Voronoi diagrams. `scipy.spatial.distance` provides a large collection of distance functions, along with functions for computing the distance between all pairs of vectors in a given collection of points or between all pairs from two collections of points.

special The name comes from the class of functions traditionally known as "special functions", but over time, the subpackage has grown to include functions beyond the classical special functions. A more appropriate characterization of this subpackage is simply "useful functions". It includes a large collection of the classical special functions such as Airy, Bessel, etc.; orthogonal families of polynomials; the Gamma function, and functions related to it; functions for computing the PDF, CDF and quantile function for several probability distributions; information theory functions; combinatorial functions `comb` and `factorial`; and more.

stats The `stats` subpackage provides a large collection of continuous and discrete probability distributions, each with methods to compute the PDF or PMF, CDF, moments and other statistics, generation of random variates, and more; statistical tests, including tests on equality of means/medians/variance (such as the t-test) and tests whether a sample is drawn from a certain distribution (such as the Kolmogorov-Smirnov test); measures of correlation, including Pearson's r, Kendall's τ, and Spearman's ρ coefficients; descriptive statistics including trimmed values; kernel density estimation; and transformations of data such as the Box-Cox power transformation.

Language	Percent
Python	49.5
Fortran	25.6
C	19.5
Cython	3.0
C++	2.3
TEX, Matlab, Shell, and Makefile	<0.5

Table 1. Language composition of SciPy codebase: lines of code in each programming language as determined by the `linguist` package. The last row denotes tools used in supporting roles in tests, building, and documentation.

Language choices

Python, Cython, Fortran, C and C++ are the programming languages used to implement scientific algorithms in the SciPy library. An analysis of our codebase using the `linguist` library[77] provides a detailed breakdown of SciPy's composition (Table 1).

Fortran, despite its age, is still a high-performance scientific programming language with continued contemporary usage[78]. Thus, we wrap the following excellent, field-tested Fortran libraries in order to provide Python convenience while benefiting from their performance: FFTPACK[79,80], ODEPACK[23], QUADPACK[24], FITPACK[72], ODRPACK[76], MINPACK[25], ARPACK[81], ALGORITHM 644[82], and CDFLIB[83].

Rounding out the top three languages in SciPy is C, which is also extremely well-established over several decades[84] of scientific computing. The C libraries that we wrap in SciPy include trlib[85], SuperLU[86,87], Qhull[88], and Cephes[89].

Cython has been described as a creole language that mixes the best parts of Python and lower-level C / C++ paradigms[60]. We often use Cython as a glue between well-established, low-level scientific computing libraries written in C / C++ and the Python interface offered by SciPy. We also use Cython to enable performance enhancements in Python code, especially for cases where heavily used inner loops benefit from a compiled code with static typing.

For implementing new functionality, Python is the still the language of choice. If Python performance is an issue, then we prefer the use of Cython followed by C, C++, or Fortran (in that order). The main motivation for this is maintainability: Cython has the highest abstraction level and most Python developers will understand it. C is also widely known, and easier for the current core development team to manage than C++ and especially Fortran.

The distribution of secondary programming languages in SciPy is a compromise between a powerful, performance-enhancing language that interacts well with Python (i.e. Cython) and the usage of languages (and their libraries) that have proven reliable and performant over many decades. The position that SciPy occupies near the foundation of the scientific Python ecosystem is such that adoption of new languages or major dependencies is generally unlikely—our choices are strongly driven by long-term stability. GPU acceleration, new transpiling libraries, and the latest JIT compilation approaches (i.e., Numba[90]) are very powerful, but currently fall outside the remit of the main SciPy library. That said, we have recently increased our efforts to support compatibility with some of these options, and having our full test suite pass with the PyPy JIT compiler[91] is now a requirement in our development workflow.

API and ABI evolution

The application programming interface (API) for SciPy consists of approximately 1500 functions and classes. Our policy for evolving the API over time is that new functionality can be added, while removing or changing existing functionality can only be done if the benefits of that exceeds the (often significant) costs to users, *and* only after giving clear deprecation warnings to those users for at least one year. In general, we encourage changes that improve clarity in the API of the library but strongly discourage breaking backwards compatibility given our position near the base of the scientific Python computing stack.

In addition to the Python API, SciPy has C and Cython interfaces. Therefore, we have to also consider the application binary interface (ABI). This ABI has been stable for a long time, and we aim to evolve it only in a backwards compatible way.

Key technical improvements

Here we describe key technical improvements made in the last three years.

Data structures

Sparse matrices

`scipy.sparse` offers seven sparse matrix data structures, also known as sparse formats. The most important ones are the row- and column-compressed formats (CSR and CSC, respectively). These offer fast major-axis indexing and fast matrix-vector multiplication, and are used heavily throughout SciPy and dependent packages.

Over the last three years, our sparse matrix handling internals have been rewritten and performance has been improved. Iterating over and slicing of CSC and CSR matrices is now faster by up to 35%, and the coordinate (COO) / diagonal (DIA) to CSR / CSC matrix format conversions are now faster. Importantly, SuperLU[87] was updated to version 5.2.1, enhancing the low-level implementations leveraged by a subset of our `sparse` offerings.

From a new features standpoint, `scipy.sparse` matrices and linear operators now support the Python matrix multiplication (@) operator. We've added `scipy.sparse.norm` and `scipy.sparse.random` for computing sparse matrix norms and drawing random variates from arbitrary distributions, respectively. Also, we've made a concerted effort to bring the `scipy.sparse` API into line with the equivalent NumPy API where possible.

`cKDTree`

The `scipy.spatial.ckdtree` module, which implements a space-partitioning data structure that organizes points in k-dimensional space, was rewritten in C++ with templated classes. Support was added for periodic boundary conditions, which are often used in simulations of physical processes.

In 2013, the time complexity of the k-nearest neighbor search from `cKDTree.query` was approximately loglinear[92], consistent with its formal description[93]. Since then, we've enhanced `cKDTree.query` by reimplementing it in C++, removing memory leaks, and allowing release of the global interpreter lock (GIL) so that multiple threads may be used[94]. This generally improved performance on any given problem (Figure 4) while preserving the asymptotic complexity.

In 2015, SciPy added the `sparse_distance_matrix` routine for generating approximate sparse distance matrices between `KDTree` objects by ignoring all distances that exceed a user-provided value. This routine is not limited to the conventional L2 (Euclidean) norm but supports any Minkowski p-norm between 1 and infinity. By default, the returned data structure is a Dictionary Of Keys (DOK) based sparse matrix, which is very efficient for matrix construction. This hashing approach to sparse matrix assembly can be 7 times faster than constructing with CSR format[95], and the C++ level sparse matrix construction releases the Python GIL for increased performance. Once the matrix is constructed, distance value retrieval has an amortized constant time complexity[96], and the DOK structure can be efficiently converted to a CSR, CSC, or COO matrix to allow for speedy arithmetic operations.

In 2015 the `cKDTree` dual tree counting algorithm[97] was enhanced to support weights[98], which are essential in many scientific applications, e.g. computing correlation functions of galaxies[99].

Unified bindings to compiled code

LowLevelCallable

As of SciPy version 0.19, it is possible for users to wrap low-level functions in a `scipy.LowLevelCallable` object that reduces the overhead of calling compiled C functions, such as those generated using `numba` or Cython, directly from Python. Supported low-level functions include `PyCapsule` objects, `ctypes` function pointers, and `cffi` function pointers. Furthermore, it is even possible to generate a low-level callback function automatically from a Cython module using `scipy.LowLevelCallable.from_cython`.

Cython bindings for BLAS, LAPACK, and special

SciPy has provided special functions and leveraged BLAS and LAPACK[100] routines for many years. SciPy now additionally includes Cython[60] wrappers for many BLAS and LAPACK routines (added in 2015) and the special functions provided in the `scipy.special` subpackage (added in 2016). These Cython wrappers are available in `scipy.linalg.cython_blas`, `scipy.linalg.cython_lapack`, and `scipy.special.cython_special` respectively. When writing algorithms in Cython, it is typically more efficient to call directly into the libraries SciPy wraps rather than indirectly, using SciPy's Python APIs. These low-level interfaces for Cython can also be used outside of the SciPy codebase to gain access to the functions in the wrapped libraries while avoiding the overhead of Python function calls. This can give performance gains of one or two orders of magnitude for many use cases.

Developers can also use the low-level Cython interfaces without linking against the wrapped libraries[101]. This lets other extensions avoid the complexity of finding and using the correct libraries. Avoiding this complexity is especially important when wrapping libraries written in Fortran. Not only can these low-level wrappers be used without a Fortran compiler, they can also be used without having to handle all the different Fortran compiler ABIs and name mangling schemes.

Most of these low-level Cython wrappers are generated automatically to help with both correctness and ease of maintenance. The wrappers for BLAS and LAPACK are primarily generated using type information that is parsed from the BLAS and

LAPACK source files using F2PY[26], though a small number of routines use hand-written type signatures instead. The input and output types of each routine are saved in a data file that is read at build time and used to generate the corresponding Cython wrapper files. The wrappers in `scipy.special.cython_special` are also generated from a data file containing type information for the wrapped routines.

Since SciPy can be built with LAPACK 3.4.0 or later, Cython wrappers are only provided for the routines that maintain a consistent interface across all supported LAPACK versions. The standard BLAS interface provided by the various existing BLAS libraries is not currently changing, so changes are not generally needed in the wrappers provided by SciPy. Changes to the Cython wrappers for the functions in `scipy.special` follow corresponding changes to the interface of that subpackage.

Numerical optimization

The `scipy.optimize` subpackage provides functions for the numerical solution of several classes of root finding and optimization problems. Here we highlight recent additions through SciPy 1.0.

Linear Optimization

A new interior-point optimizer for continuous linear programming problems, `linprog` with `method='interior-point'`, was released with SciPy 1.0. Implementing the core algorithm of the commercial solver MOSEK[102], it solves all of the 90+ NETLIB LP benchmark problems[103] tested. Unlike some interior point methods, this homogeneous self-dual formulation provides certificates of infeasibility or unboundedness as appropriate.

A presolve routine[104] solves trivial problems and otherwise performs problem simplifications, such as bound tightening and removal of fixed variables, and one of several routines for eliminating redundant equality constraints is automatically chosen to reduce the chance of numerical difficulties caused by singular matrices. Although the main solver implementation is pure Python, end-to-end sparse matrix support and heavy use of SciPy's compiled linear system solvers—often for the same system with multiple right hand sides due to the predictor-corrector approach—provide speed sufficient for problems with tens of thousands of variables and constraints.

Compared to the previously implemented simplex method, the new interior-point method is faster for all but the smallest problems, and is suitable for solving medium- and large-sized problems on which the existing simplex implementation fails. However, the interior point method typically returns a solution near the center of an optimal face, yet basic solutions are often preferred for sensitivity analysis and for use in mixed integer programming algorithms. This motivates the need for a crossover routine or a new implementation of the simplex method for sparse problems in a future release, either of which would require an improved sparse linear system solver with efficient support for rank-one updates.

Nonlinear Optimization

Local Minimization The `minimize` function provides a unified interface for finding local minima of nonlinear optimization problems. Four new methods for unconstrained optimization were added to `minimize` in recent versions of SciPy: `dogleg`, `trust-ncg`, `trust-exact`, and `trust-krylov`. All are trust-region methods that build a local model of the objective function based on first and second derivative information, approximate the best point within a local "trust region", and iterate until a local minimum of the original objective function is reached, but each has unique characteristics that make it appropriate for certain types of problems. For instance, `trust-exact` achieves fast convergence by solving the trust-region subproblem almost exactly, but it requires the second derivative Hessian matrix to be stored and factored every iteration, which may preclude the solution of large problems ($\geq$ 1000 variables). On the other hand, `trust-ncg` and `trust-krylov` are well-suited to large-scale optimization problems because they do not need to store and factor the Hessian explicitly, instead using second derivative information in a faster, approximate way. A detailed comparison of the characteristics of all `minimize` methods is presented in Table 2; it illustrates the level of completeness that SciPy aims for when covering a numerical method or topic.

Global Minimization As `minimize` may return any local minimum, some problems require the use of a global optimization routine. The new `scipy.optimize.differential_evolution` function[105,106] is a stochastic global optimizer that works by evolving a population of candidate solutions. In each iteration, trial candidates are generated by combination of candidates from the existing population. If the trial candidates represent an improvement, then the population is updated. Most recently, the SciPy benchmark suite gained a comprehensive set of 196 global optimization problems for tracking the performance of existing solvers over time and for evaluating whether the performance of new solvers merits their inclusion in the package.

Table 2. Optimization methods from `minimize`, which solves problems of the form $\min_x f(x)$, where $x \in \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}$. The field *version added* specifies the algorithm's first appearance in SciPy. Algorithms with *version added* "0.6*" were added in version 0.6 or before. The field *wrapper* indicates whether the implementation available in SciPy wraps a function written in a compiled language (e.g. C or FORTRAN). The fields *1st* and *2nd derivatives* indicates whether first or second order derivatives are required. When *2nd derivatives* is flagged with $\sim$ the algorithm does not requires second-order derivatives from the user; it computes an approximation internally and uses it to accelerate method convergence. *Iterative Hessian factorization* denotes algorithms that factorize the Hessian in an iterative way, which does not require explicit matrix factorization or storage of the Hessian. *Local convergence* gives a lower bound on the rate of convergence of the iterations sequence once the iterate is sufficiently close to the solution: linear (L), superlinear (S) and quadratic (Q). Convergence rates denoted S* indicate that the algorithm has a superlinear rate for the parameters used in SciPy, but can achieve a quadratic convergence rate with other parameter choices. *Global convergence* is marked for the algorithms with guarantees of convergence to a stationary point (i.e. a point x^* for which $\nabla f(x^*) = 0$); this is *not* a guarantee of convergence to a global minimum. The table also indicates which algorithms can deal with constraints on the variables. We distinguish among *bound constraints* (i.e. $x^l \leq x \leq x^u$), *equality constraints* (i.e. $c_{\text{eq}}(x) = 0$) and *inequality constraints* (i.e. $c_{\text{ineq}}(x) \geq 0$).

	Nelder-Mead	*Powell*	*COBYLA*	*CG*	*BFGS*	*L-BFGS-B*	*SLSQP*	*TNC*	*Newton-CG*	*dogleg*	*trust-ncg*	*trust-exact*	*trust-krylov*
Version added	0.6*	0.6*	0.6*	0.6*	0.6*	0.6*	0.9	0.6*	0.6*	0.13	0.13	0.19	1.0
Wrapper			✓			✓	✓	✓					✓
1st derivatives				✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
2nd derivatives					∼	∼	∼	✓	✓	✓	✓	✓	✓
Iterative Hessian factorization								✓	✓		✓		✓
Local convergence				L	S	L	S	S*	S*	Q	S*	Q	S*
Global convergence					✓	✓	✓	✓	✓	✓	✓	✓	✓
Line-search (LS) or trust-region (TR)	Neither	LS	TR	LS	LS	LS	LS	LS	LS	TR	TR	TR	TR
Bound constraints			✓				✓	✓	✓				
Equality constraints							✓						
Inequality constraint			✓				✓						
References	[107,108]	[109]	[110–112]	[113,114]	[114]	[115,116]	[117–120]	[121]	[114]	[114,122]	[114,123]	[124,125]	[126,127]

Statistical distributions

The `scipy.stats` subpackage contains more than 100 probability distributions: 96 continuous and 13 discrete univariate distributions and 10 multivariate distributions. The implementation relies on a consistent framework that provides methods to sample random variates, to evaluate the cumulative distribution function (cdf) and the probability density function (pdf) and to fit parameters for every distribution. Generally, the methods rely on specific implementations for each distribution such as a closed form expression of the cdf or a sampling algorithm, if available. Otherwise, default methods are used based on generic code, e.g., numerical integration of the pdf to obtain the cdf. Key recent distributions added to `scipy.stats` include the histogram-based distribution in `scipy.stats.rv_histogram` and the multinomial distribution in `scipy.stats.multinomial` (used, for example, in natural language processing, see[128]).

Polynomial interpolators

Historically, SciPy relied heavily on the venerable FITPACK Fortran library by P. Dierckx[72,129] for univariate interpolation and approximation of data. The original monolithic design and API for interaction between SciPy and FITPACK was limiting for both users and developers, as detailed in the Supporting Information.

Implementing a new, modular design of polynomial interpolators was spread over several releases. The goals of this effort were to have a set of basic objects representing piecewise polynomials, to implement a collection of algorithms for constructing various interpolators, and to provide users with building blocks for constructing additional interpolators.

At the lowest level of the new design are classes which represent univariate piecewise polynomials: `PPoly` (SciPy 0.13)[130], `BPoly` (SciPy 0.13) and `BSpline` (SciPy 0.19)[131], which allow efficient vectorized evaluations, differentiation, integration

and root-finding. `PPoly` represents piecewise polynomials in the power basis in terms of breakpoints and coefficients at each interval. `BPoly` is similar, and represents piecewise polynomials in the Bernstein basis (which is suitable for e.g., constructing Bezier curves). `BSpline` represents spline curves, i.e., linear combinations of B-spline basis elements.[132]

In the next layer, these polynomial classes are used for constructing several common ways of interpolating data: `CubicSpline` (SciPy 0.18)[133] constructs a twice differentiable piecewise cubic function, `Akima1DInterpolator` and `PCHIPInterpolator` implement two classic prescriptions for constructing a C^1 continuous monotone shape-preserving interpolator.[134,135]

Test and benchmark suite

Test suite

The SciPy test suite is orchestrated by a continuous integration matrix that includes POSIX and Windows (32/64-bit) platforms managed by Travis CI and AppVeyor, respectively. Our tests cover Python versions 2.7, 3.4, 3.5, 3.6, and include code linting with `pyflakes` and `pycodestyle`. There are more than 13,000 unit tests in the test suite, which is written for usage with the `pytest` framework, and with 87% and 45% line coverages for Python and compiled code, respectively at the SciPy 1.0 release point (Figure 3). Documentation for the code is automatically built and published by the CircleCI service to facilitate evaluation of documentation changes / integrity. Our full test suite also passes with PyPy3[91], a just-in-time compiled version of the Python language.

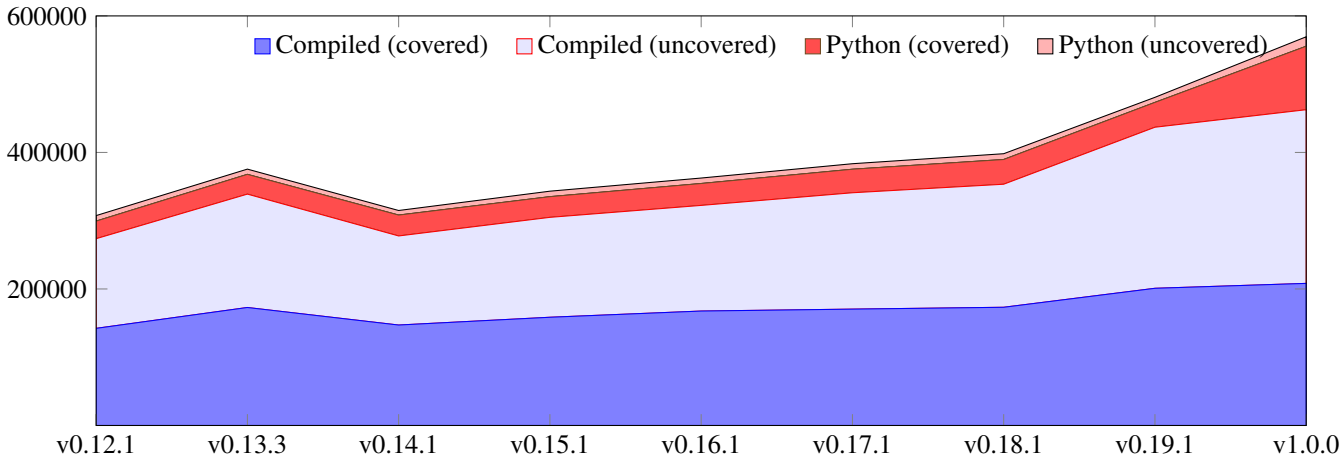

Figure 3. Python (red) and compiled (blue) code volume in SciPy over time. Deep-shaded area represents lines of code covered by units tests; light-shaded area represents lines not covered. With the exception of the removal of ≈ 61,000 lines of compiled code for SciPy v0.14, the volume of both compiled and Python code has increased between releases, as has the number of lines covered by unit tests. SciPy v1.0.0 was composed of 462,574 lines of compiled code with test coverage near 45% and 106,878 lines of Python code with test coverage near 87%.

The reconstitution of historical build and test environments was performed using a docker-based approach[136]. Analysis of lines of source code covered is performed automatically in our continuous integration suite using the `pytest-cov` and `gcov` libraries. Some decreases in compiled code coverage may reflect inclusion of additional vendored code.

Benchmark suite

In addition to ensuring that unit tests are passing, it is important to confirm that the the performance of the SciPy codebase improves over time. Since February 2015, the performance of SciPy has been monitored with Airspeed Velocity (`asv`[137]). SciPy's `run.py` script conveniently wraps `asv` features such that benchmark results over time can be generated with a single console command. For example, Figure 4 illustrates the improvement of`scipy.spatial.cKDTree.query` over roughly nine years of project history.

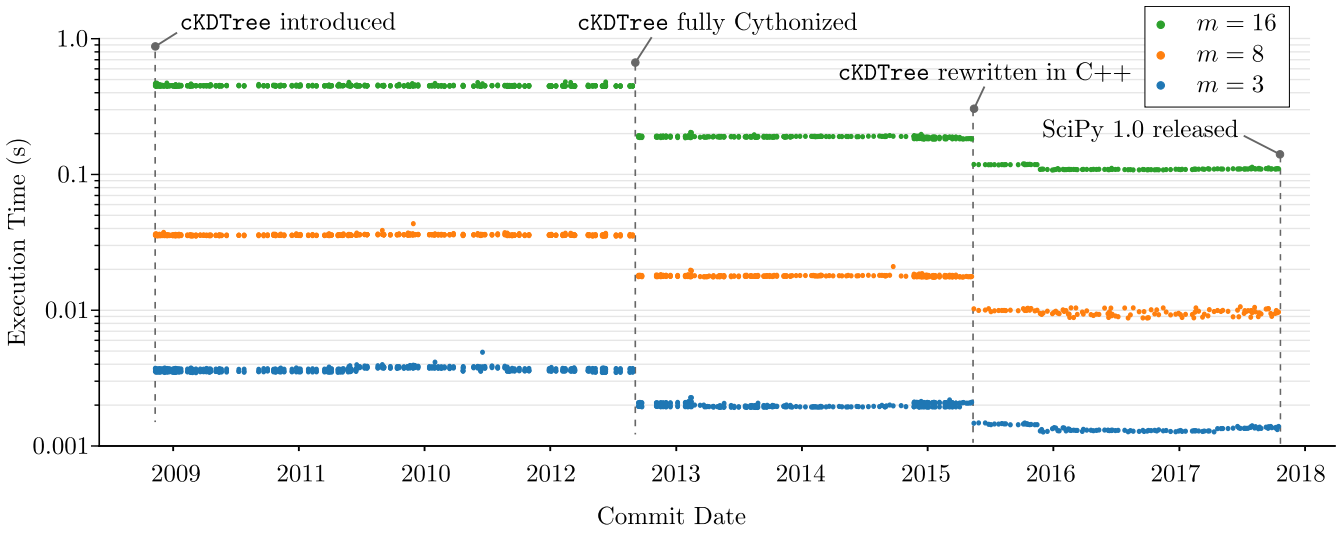

Figure 4. Results of the `scipy.spatial.cKDTree.query` benchmark from the introduction of `cKDTree` to the release of SciPy 1.0. The benchmark generates a k-d tree from 10,000 uniformly distributed points in an m-dimensional unit hypercube, then finds the nearest (Euclidean) neighbor in the tree for each of 1,000 query points. Each marker in the figure indicates the execution time of the benchmark for a commit in the master branch of SciPy. Substantial performance improvements were realized when `cKDTree` was fully Cythonized and again when it was rewritten in C++. The tree was generated without application of toroidal topology (`boxsize=None`), and tests were performed by Airspeed Velocity 0.4 using Python 2.7, NumPy 1.8.2, and Cython versions 0.27.3, 0.21.1, and 0.18 (for improved backward compatibility).

Project organization and community

Governance

SciPy adopted an official Governance Document on August 3, 2017[138]. A Steering Council, currently composed of 18 members, oversees daily development of the project by contributing code and reviewing contributions from the community. Council Members have commit rights to the project repository, but they are expected to merge changes only when there are no substantive community objections. The Chair of the Steering Council, Ralf Gommers, is responsible for initiating biannual technical reviews of project direction and summarizing any private Council activities to the broader community. The project's Benevolent Dictator for Life (BDFL), Pauli Virtanen, has overruling authority on any matter, but the BDFL is expected to act in good faith and only exercise this authority when the Steering Council cannot reach agreement.

SciPy's official Code of Conduct was approved on October 24, 2017. In summary, there are five Specific Guidelines: *be open* to everyone participating in our community; *be empathetic and patient* in resolving conflicts; *be collaborative*, as we depend on each other to build the library; *be inquisitive*, as early identification of issues can prevent serious consequences; and *be careful with wording*. The Code of Conduct specifies how breaches can be reported to a Code of Conduct Committee, and outlines procedures for the Committee's response. Our Diversity Statement "welcomes and encourages participation by everyone".

Maintainers and contributors

The SciPy project has approximately 100 unique contributors for every 6-monthly release cycle. Anyone with the interest and skills can become a contributor; the SciPy Developer Guide[139] provides guidance on how to do that. In addition, the project currently has 15 active maintainers: people who review the contributions of others and do everything else needed to ensure that the software and the project move forward. Maintainers are critical to the health of the project[140]; their skills and efforts largely determine how fast the project moves forward, and they enable input from the much larger group of contributors. Anyone can become a maintainer, too, as they are selected on a rolling basis from contributors with a significant history of high-quality contributions.

Downstream projects

The scientific Python ecosystem includes many examples of domain-specific software libraries building on top of SciPy features and then returning to the base SciPy library to suggest and even implement improvements. For example, there are common contributors to the SciPy and Astropy core libraries[141], and what works well for one of the codebases, infrastructures, or

communities is often transferred in some form to the other. At the codebase level, the `binned_statistic` functionality is one such cross-project contribution: it was initially developed in an Astropy-affiliated package and then placed in SciPy afterwards. In this perspective, SciPy serves as a catalyst for cross-fertilisation throughout the Python scientific computing community.

Discussion

SciPy has a strong developer community and a massive user base. GitHub traffic metrics report roughly 20,000 unique visitors to the source website between May 14 and May 27, 2018, with 721 unique copies ("clones") of the codebase over that roughly two-week period. The developer community at that time consisted of 610 unique contributors of source code, with $> 19,000$ commits accepted into the codebase (GitHub page data).

From the user side, there were 13,096,468 downloads of SciPy from the Python Packaging Index (PyPI)[142] and 5,776,017 via the default channel of the `conda`[143] package manager during the year 2017. These numbers establish a lower bound on the total number of downloads by users given that PyPI and `conda` are only two of several popular methods for installing SciPy. The SciPy website[144], which has been the default citation in the absence of a peer-reviewed paper, has been cited > 3000 times[145]. Some of the most prominent usages of or demonstrations of credibility for SciPy include the LIGO / Virgo scientific collaboration that lead to the observation of gravitational waves[146], the fact that SciPy is shipped directly with MacOS and in the Intel Distribution for Python[147], and that SciPy is used by 47% of all machine learning projects on GitHub[148].

Nevertheless, SciPy continually strives to improve. The SciPy Roadmap[149,150], summarized in Table 3, is a continuously-updated document maintained by the community that describes some of the major directions for improvement for the project, as well as specific limitations and matters that require assistance moving forward. In addition to the items on the roadmap, we are still working to increase the number of SciPy usage tutorials beyond our current 15 section offering[151]. Also, the low-level Cython code in our library (which interacts with C-level code and exposes it for Python usage) could use some measure of modernization, including migration to typed memoryviews to handle NumPy arrays.

Table 3. Summary of SciPy Roadmap items following 1.0 release

SciPy subpackage	Summary of Change
`optimize`	a few more high quality global optimizers
`fftpack`	reduce overlap with NumPy equivalent
`linalg`	reduce overlap with NumPy equivalent
`interpolate`	new spline fitting and arithmetic routines
`interpolate`	new transparent tensor-product splines
`interpolate`	new Non-Uniform Rational B-Splines
`interpolate`	mesh refinement & coarsening of B-splines and tensor products
`signal`	migrate spline functionality to `interpolate`
`signal`	Second Order Sections update to match capabilities in other routines
`linalg`	support a more recent version of LAPACK
`ndimage`	clarify usage of the "data point" coordinate model, and add additional wrapping modes
`sparse`	incorporate sparse arrays from Sparse package[152]
`sparse.linalg`	add PROPACK wrappers for faster SVD
`spatial`	add support for (quaternion) rotation matrices
`special`	precision improvements for hypergeometric, parabolic cylinder, and spheroidal wave functions

A problem faced by many open source projects is attracting and retaining developers. While it is normal for some individuals to contribute to a project for a while and then move on, too much turnover can result in the loss of institutional memory, leading to mistakes of the past being repeated, APIs of new code becoming inconsistent with the old code, and a drifting project scope. We are fortunate that the SciPy project continues to attract enthusiastic and competent new developers while maintaining the involvement of a small but dedicated "old guard". There are contributors who were present in the early years of the project who still contribute to discussions of bug reports and reviews of new code contributions. Our BDFL has been with the project for more than 10 years and is still actively contributing code, and the head of our Steering Council, who also acts as a general manager, is approaching his eleventh anniversary. An additional half dozen or so active developers have been contributing steadily for five or more years. The combination of a committed old guard and a host of new contributors ensures that SciPy will continue to grow while maintaining a high level of quality.

Data Availability

All SciPy library source code and all data generated for the current study are available in the SciPy GitHub repository, https://github.com/scipy.

References

1. van der Walt, S., Colbert, S. C. & Varoquaux, G. The NumPy array: a structure for efficient numerical computation. *Comput. Sci. & Eng.* **13**, 22–30 (2011).
2. Oliphant, T. E. *Guide to NumPy* (CreateSpace Independent Publishing Platform, USA, 2015), 2nd edn.
3. Pedregosa, F. *et al.* Scikit-learn: Machine learning in Python. *J. machine learning research* **12**, 2825–2830 (2011).
4. van der Walt, S. *et al.* scikit-image: image processing in Python. *PeerJ* **2**, e453 (2014).
5. Nitz, A. *et al.* gwastro/pycbc: Pycbc v1.13.2 release (2018). URL https://doi.org/10.5281/zenodo.1596771. DOI 10.5281/zenodo.1596771.
6. Vallisneri, M., Kanner, J., Williams, R., Weinstein, A. & Stephens, B. The LIGO Open Science Center. *J. Physics: Conf. Ser.* **610**, 012021 (2015). URL http://stacks.iop.org/1742-6596/610/i=1/a=012021.
7. Abbott, B. P. *et al.* Gw150914: First results from the search for binary black hole coalescence with advanced ligo. *Phys. Rev. D* **93**, 122003 (2016). URL https://link.aps.org/doi/10.1103/PhysRevD.93.122003. DOI 10.1103/PhysRevD.93.122003.
8. Abbott, B. P. *et al.* Gw170817: observation of gravitational waves from a binary neutron star inspiral. *Phys. Rev. Lett.* **119**, 161101 (2017).
9. Event Horizon Telescope Collaboration *et al.* First M87 Event Horizon Telescope Results. III. Data Processing and Calibration. *The Astrophys. J. Lett.* **875**, L3 (2019). DOI 10.3847/2041-8213/ab0c57.
10. Blanton, K. At Mathworks, support + fun = success CEO Jack Little believes in power of his workers – and their ideas. *The Boston Globe* J5 (1997). URL https://www.newspapers.com/clip/26952040/the_boston_globe/.
11. Howell, D. Jack Dangermond's Digital Mapping Lays It All Out. *Investor's Bus. Dly.* (2009). URL https://web.archive.org/web/20100510064955/http://www.investors.com/NewsAndAnalysis/Article.aspx?id=503454.
12. Port, O. Simple Solutions. *BusinessWeek* 24–24 (2005). Historic Article about Stephen Wolfram and Mathematica.
13. Open Source Community. Numeric and Scientific (2017). URL https://wiki.python.org/moin/NumericAndScientific.
14. Open Source Community. Python Conferences (2018). URL https://wiki.python.org/moin/PythonConferences.
15. Guido van Rossum. Python/C API Reference Manual (2001). URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.211.6702&rep=rep1&type=pdf.
16. CPython Developers. The Python programming language (2019). URL https://github.com/python/cpython.
17. Jim Hugunin. The Matrix Object Proposal (very long) (1995). URL https://mail.python.org/pipermail/matrix-sig/1995-August/000002.html.
18. Jim Hugunin. Extending Python for Numerical Computation (1995). URL http://hugunin.net/papers/hugunin95numpy.html.
19. Oliphant, T. Moving Forward from the Last Decade of SciPy (2010). URL https://conference.scipy.org/scipy2010/slides/travis_oliphant_keynote.pdf.
20. Oliphant, T. Some Python Modules (1999). URL https://web.archive.org/web/19990125091242/http://oliphant.netpedia.net:80/.
21. Oliphant, T. Modules to enhance Numerical Python (2000). URL https://web.archive.org/web/20001206213500/http://oliphant.netpedia.net:80/.
22. Travis Oliphant. Multipack (a collection of FORTRAN routines interfaced with NumPy). URL http://pylab.sourceforge.net/multipack.html.

23. Hindmarsh, A. C. *ODEPACK, A Systematized Collection of ODE Solvers , R. S. Stepleman et al. (eds.), North-Holland, Amsterdam, (vol. 1 of), pp. 55-64.*, vol. 1 of *IMACS Transactions on Scientific Computation* (North-Holland Amsterdam, 1983).

24. Piessens, R., de Doncker-Kapenga, E. & Ueberhuber, C. W. *Quadpack. A subroutine package for automatic integration* (1983).

25. More, J. J., Garbow, B. S. & Hillstrom, K. E. *User guide for MINPACK-1. [In FORTRAN]* (Argonne National Laboratory Report ANL-80-74, 1980).

26. Peterson, P. F2PY: a tool for connecting Fortran and Python programs. *Int. J. Comput. Sci. Eng.* **4**, 296–305 (2009).

27. Strangman, G. PYTHON MODULES (2000). URL https://web.archive.org/web/20001022231108/http://www.nmr.mgh.harvard.edu/Neural_Systems_Group/gary/python.html.

28. Developers, S. SciPy.org (2001). URL https://web.archive.org/web/20010309040805/http://scipy.org:80/.

29. Vaught, T. N. SciPy Developer mailing list now online (2001). URL https://mail.python.org/pipermail/scipy-dev/2001-June/000000.html.

30. Jones, E. ANN: SciPy 0.10 – Scientific Computing with Python (2001). URL https://mail.python.org/pipermail/python-list/2001-August/106419.html.

31. Vaught, T. N. Reference documentation and Tutorial documentation are now available for download as tarballs (2002). URL https://web.archive.org/web/20021013204556/http://www.scipy.org:80/scipy/site_content/site_news/docs_released1.

32. Vaught, T. N. [ANN] SciPy '02 - Python for Scientific Computing Workshop (2002). URL https://mail.python.org/pipermail/numpy-discussion/2002-June/001511.html.

33. Ascher, D., Dubois, P. F., Hinsen, K., Hugunin, J. & Oliphant, T. An Open Source Project: Numerical Python (2001). URL http://www.numpy.org/_downloads/numeric-manual.pdf.

34. Greenfield, P. How Python Slithered Into Astronomy (2011). URL https://conference.scipy.org/scipy2011/slides/greenfield_keynote_astronomy.pdf.

35. Schroeder, W., Martin, K. & Lorensen, B. *The Visualization Toolkit* (Kitware, 2006), 4 edn.

36. Ramachandran, P. MayaVi Uses Python for Scientific Data Visualization (2003). URL https://web.archive.org/web/20030731122452/http://pythonology.org/success&story=mayavi.

37. Kluyver, T. *et al.* Jupyter notebooks – a publishing format for reproducible computational workflows. In Loizides, F. & Schmidt, B. (eds.) *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, 87 – 90 (IOS Press, 2016).

38. *MATLAB* (The MathWorks, Inc., Natick, MA, United States).

39. Hunter, J. ANN: matplotlib 0.1 (2003). URL https://mail.python.org/pipermail/python-list/2003-April/193167.html.

40. Greenfield, P., Miller, J. T., Hsu, J. & White, R. L. numarray: A new scientific array package for Python. *PyCon DC* (2003).

41. NumPy Developers. v1.0 (2006). URL https://github.com/numpy/numpy/releases/tag/v1.0.

42. Dubois, P. F. Python—batteries included. *Comput. Sci. & Eng.* **9**, 7–9 (2007).

43. Oliphant, T. E. Python for scientific computing. *Comput. Sci. & Eng.* **9**, 10–20 (2007).

44. Pérez, F. & Granger, B. E. IPython: a system for interactive scientific computing. *Comput. Sci. & Eng.* **9** (2007).

45. Hunter, J. D. Matplotlib: A 2d graphics environment. *Comput. science & engineering* **9**, 90–95 (2007).

46. Myers, C. R., Gutenkunst, R. N. & Sethna, J. P. Python unleashed on systems biology. *Comput. Sci. & Eng.* **9** (2007).

47. Greenfield, P. Reaching for the stars with python. *Comput. Sci. & Eng.* **9**, 38–40 (2007).

48. Krauss, R. W. & Book, W. J. A python module for modeling and control design of flexible robots. *Comput. Sci. & Eng.* **9**, 41–45 (2007).

49. Bienstman, P., Vanholme, L., Bogaerts, W., Dumon, P. & Vandersteegen, P. Python in nanophotonics research. *Comput. Sci. & Eng.* **9**, 46–47 (2007).

50. Mardal, K.-A., Skavhaug, O., Lines, G. T., Staff, G. A. & Ødegård, Å. Using python to solve partial differential equations. *Comput. Sci. & Eng.* **9**, 48–51 (2007).

51. Millman, K. J. & Brett, M. Analysis of functional magnetic resonance imaging in python. *Comput. Sci. & Eng.* **9**, 52–55 (2007).

52. Shi, X. Python for internet gis applications. *Comput. Sci. & Eng.* **9**, 56–59 (2007).

53. Bäcker, A. Computational physics education with python. *Comput. Sci. & Eng.* **9**, 30–33 (2007).

54. Myers, C. R. & Sethna, J. P. Python for education: Computational methods for nonlinear systems. *Comput. Sci. & Eng.* **9**, 75–79 (2007).

55. SciPy Developers. Scikits (2019). URL https://www.scipy.org/scikits.html.

56. Millman, K. J. & Pérez, F. Developing open-source scientific practice. *Implement. Reproducible Res. CRC Press. Boca Raton, FL* 149–183 (2014).

57. Pérez, F., Langtangen, H. P. & LeVeque, R. CSE 2009: Python for Scientific Computing at CSE 2009. SIAM News (2009). URL https://archive.siam.org/news/news.php?id=1595.

58. Millman, K. J. & Aivazis, M. Python for scientists and engineers. *Comput. Sci. & Eng.* **13**, 9–12 (2011).

59. Pérez, F., Granger, B. E. & Hunter, J. D. Python: an ecosystem for scientific computing. *Comput. Sci. & Eng.* **13**, 13–21 (2011).

60. Behnel, S. *et al.* Cython: The best of both worlds. *Comput. Sci. & Eng.* **13**, 31–39 (2011).

61. Ramachandran, P. & Varoquaux, G. Mayavi: 3d visualization of scientific data. *Comput. Sci. & Eng.* **13**, 40–51 (2011).

62. Gwaltig, M.-O. *et al.* Research Topic: Python in Neuroscience (2008). URL https://www.frontiersin.org/research-topics/8/python-in-neuroscience.

63. NumFocus Team. NumFocus: Open Code = Better Science (2019). URL https://numfocus.org.

64. Boisvert, R. F., Howe, S. E. & Kahaner, D. K. The guide to available mathematical software problem classification system1. *Commun. Stat. Comput.* **20**, 811–842 (1991).

65. Seabold, J. & Perktold, J. Statsmodels: Econometric and statistical modeling with python. In *Proceedings of the 9th Python in Science Conference* (2010).

66. Salvatier, J., Wiecki, T. V. & Fonnesbeck, C. Probabilistic programming in python using pymc3. *PeerJ Comput. Sci.* **2**, e55 (2016). URL https://doi.org/10.7717/peerj-cs.55. DOI 10.7717/peerj-cs.55.

67. Foreman-Mackey, D., Hogg, D. W., Lang, D. & Goodman, J. emcee: The MCMC Hammer. *Publ. Astron. Soc. Pac.* **125**, 306 (2013). DOI 10.1086/670067. 1202.3665.

68. Stan Development Team. PyStan: the Python interface to Stan (2018). URL http://mc-stan.org.

69. Meurer, A. *et al.* SymPy: symbolic computing in Python. *PeerJ Comput. Sci.* **3**, e103 (2017).

70. Hagberg, A., Schult, D. & Swart, P. Exploring network structure, dynamics, and function using networkx.(2008). In Varoquaux, G., Vaught, T. & Millman, K. J. (eds.) *Proceedings of the 7th Python in Science Conference*, 11–15 (2008).

71. NIST. CODATA 2014 recommended values of the fundamental physical constants. http://physics.nist.gov/cuu/Constants/index.html (2014).

72. Dierckx, P. *Curve and Surface Fitting with Splines* (Oxford University Press, Inc., New York, NY, USA, 1993).

73. Boisvert, R. F., Pozo, R., Remington, K., Barrett, R. F. & Dongarra, J. J. Matrix market: a web resource for test matrix collections. In *Quality of Numerical Software*, 125–137 (Springer, 1997).

74. Rew, R. & Davis, G. Netcdf: an interface for scientific data access. *IEEE computer graphics applications* **10**, 76–82 (1990).

75. Duff, I. S., Grimes, R. G. & Lewis, J. G. Users' guide for the harwell-boeing sparse matrix collection (release i) (1992).

76. Boggs, P. T., Byrd, R. H., Rogers, J. E. & Schnabel, R. B. *User's Reference Guide for ODRPACK Version 2.01 Software for Weight Orthogonal Distance Regression* (U.S. Department of Commerce, National Institute of Standards and Technology, 1992).

77. Linguist developers; open source community. linguist (2019). URL https://github.com/github/linguist.

78. Koelbel, C. H. & Zosel, M. E. *The High Performance FORTRAN Handbook* (MIT Press, Cambridge, MA, USA, 1993).

79. Swarztrauber, P. N. Fft algorithms for vector computers. *Parallel Comput.* **1**, 45 – 63 (1984). URL http://www.sciencedirect.com/science/article/pii/S0167819184904137. DOI https://doi.org/10.1016/S0167-8191(84)90413-7.

80. Swarztrauber, P. N. Vectorizing the ffts**this chapter was written while the author was visiting the scientific computing division at the national bureau of standards. In RODRIGUE, G. (ed.) *Parallel Computations*, 51 – 83 (Academic Press, 1982). URL http://www.sciencedirect.com/science/article/pii/B9780125921015500075. DOI https://doi.org/10.1016/B978-0-12-592101-5.50007-5.

81. Lehoucq, Sorensen & Yang, C. ARPACK Users' Guide: Solution of Large Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods. Tech. Rep., Rice University (1996).

82. Amos, D. E. Algorithm 644: A portable package for bessel functions of a complex argument and nonnegative order. *ACM Trans. Math. Softw.* **12**, 265–273 (1986). URL http://doi.acm.org/10.1145/7921.214331. DOI 10.1145/7921.214331.

83. Brown, B., Lovato, J. & Russell, K. CDFLIB. URL https://people.sc.fsu.edu/~jburkardt/f_src/cdflib/cdflib.html.

84. Kernighan, B. W. *The C Programming Language* (Prentice Hall Professional Technical Reference, 1988), 2nd edn.

85. Lenders, F., Kirches, C. & Potschka, A. trlib: a vector-free implementation of the gltr method for iterative solution of the trust region problem. *Optim. Methods Softw.* **33**, 420–449 (2018). URL https://doi.org/10.1080/10556788.2018.1449842. DOI 10.1080/10556788.2018.1449842. https://doi.org/10.1080/10556788.2018.1449842.

86. Li, X. S. An overview of SuperLU: Algorithms, implementation, and user interface. *ACM Trans. Math. Softw.* **31**, 302–325 (2005).

87. Li, X. *et al.* SuperLU Users' Guide. Tech. Rep. LBNL-44289, Lawrence Berkeley National Laboratory (1999). http://crd.lbl.gov/~xiaoye/SuperLU/. Last update: August 2011.

88. Barber, C. B., Dobkin, D. P. & Huhdanpaa, H. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.* **22**, 469–483 (1996). URL http://doi.acm.org/10.1145/235815.235821. DOI 10.1145/235815.235821.

89. Moshier, S. L. Cephes. URL http://www.netlib.org/cephes/readme.

90. Lam, S. K., Pitrou, A. & Seibert, S. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, 7:1–7:6 (ACM, New York, NY, USA, 2015). URL http://doi.acm.org/10.1145/2833157.2833162. DOI 10.1145/2833157.2833162.

91. Bolz, C. F., Cuni, A., Fijalkowski, M. & Rigo, A. Tracing the meta-level: Pypy's tracing jit compiler. In *Proceedings of the 4th Workshop on the Implementation, Compilation, Optimization of Object-Oriented Languages and Programming Systems*, ICOOOLPS '09, 18–25 (ACM, New York, NY, USA, 2009). URL http://doi.acm.org/10.1145/1565824.1565827. DOI 10.1145/1565824.1565827.

92. Jake VanderPlas. Benchmarking Nearest Neighbor Searches in Python (2013). URL https://jakevdp.github.io/blog/2013/04/29/benchmarking-nearest-neighbor-searches-in-python/.

93. Maneewongvatana, S. & Mount, D. M. Analysis of approximate nearest neighbor searching with clustered point sets (1999). URL https://arxiv.org/pdf/cs/9901013.pdf.

94. Sturla Molden. ENH: Enhancements to spatial.cKDTree (2015). URL https://github.com/scipy/scipy/pull/4374/.

95. Aspnäs, M., Signell, A. & Westerholm, J. Efficient assembly of sparse matrices using hashing. In Kågström, B., Elmroth, E., Dongarra, J. & Waśniewski, J. (eds.) *Applied Parallel Computing. State of the Art in Scientific Computing*, 900–907 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2007).

96. Cormen, T. H., Stein, C., Rivest, R. L. & Leiserson, C. E. *Introduction to Algorithms* (McGraw-Hill Higher Education, 2001), 2nd edn.

97. Moore, A. W. *et al.* Fast algorithms and efficient statistics: N-point correlation functions. In *Proceedings, MPA / ESO / MPE Joint Astronomy Conference: Mining the Sky: Garching, Germany, July 31-August 4, 2000*, 71–82 (2001). DOI 10.1007/10849171_5. astro-ph/0012333.

98. Yu Feng. ENH: Faster count_neighour in cKDTree weighted input data (2015). URL https://github.com/scipy/scipy/pull/5647.

99. Martin, A. M., Giovanelli, R., Haynes, M. P. & Guzzo, L. The clustering characteristics of HI-selected galaxies from the 40% ALFALFA survey. *The Astrophys. J.* **750**, 38 (2012). URL http://stacks.iop.org/0004-637X/750/i=1/a=38.

100. Anderson, E. *et al. LAPACK Users' Guide* (Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999), third edn.

101. Ian Henriksen. Circumventing The Linker: Using SciPy's BLAS and LAPACK Within Cython. In Kathryn Huff & James Bergstra (eds.) *Proceedings of the 14th Python in Science Conference*, 49 – 52 (2015).

102. Andersen, E. D. & Andersen, K. D. The mosek interior point optimizer for linear programming: an implementation of the homogeneous algorithm. In *High performance optimization*, 197–232 (Springer, 2000).

103. The NETLIB LP test problem set. http://www.numerical.rl.ac.uk/cute/netlib.html. Accessed: 2018-02-20.

104. Andersen, E. D. & Andersen, K. D. Presolving in linear programming. *Math. Program.* **71**, 221–245 (1995).

105. Wormington, M., Panaccione, C., M., M. K. & Bowen, D. K. Characterization of structures from x-ray scattering data using genetic algorithms. *Phil. Trans. R. Soc. Lond. A* **357**, 2827–2848 (1999).

106. Storn, R. & Price, K. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *J. Glob. Optim.* **11**, 341–359 (1997).

107. Nelder, J. A. & Mead, R. A simplex method for function minimization. *The computer journal* **7**, 308–313 (1965).

108. Wright, M. H. Direct search methods: Once scorned, now respectable. *Pitman Res. Notes Math. Ser.* 191–208 (1996).

109. Powell, M. J. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The computer journal* **7**, 155–162 (1964).

110. Powell, M. J. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in Optimization and Numerical Analysis*, 51–67 (Springer, 1994).

111. Powell, M. Direct search algorithms for optimization calculations. *Acta numerica* **7**, 287–336 (1998).

112. Powell, M. J. A view of algorithms for optimization without derivatives. *Math. Today-Bulletin Inst. Math. its Appl.* **43**, 170–174 (2007).

113. Polak, E. & Ribiere, G. Note sur la convergence de méthodes de directions conjuguées. *Revue française d'informatique et de recherche opérationnelle. Série rouge* **3**, 35–43 (1969).

114. Nocedal, J. & Wright, S. *Numerical Optimization* (Springer Science & Business Media, 2006), second edn.

115. Byrd, R. H., Lu, P., Nocedal, J. & Zhu, C. A limited memory algorithm for bound constrained optimization. *SIAM J. on Sci. Comput.* **16**, 1190–1208 (1995).

116. Zhu, C., Byrd, R. H., Lu, P. & Nocedal, J. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Math. Softw. (TOMS)* **23**, 550–560 (1997).

117. Schittkowski, K. The nonlinear programming method of Wilson, Han, and Powell with an augmented Lagrangian type line search function. Part 1: Convergence analysis. *Numer. Math.* **38**, 83–114 (1982). DOI 10.1007/BF01395810.

118. Schittkowski, K. The nonlinear programming method of Wilson, Han, and Powell with an augmented Lagrangian type line search function. Part 2: An efficient implementation with linear least squares subproblems. *Numer. Math.* **38**, 115–127 (1982). DOI 10.1007/BF01395811.

119. Schittkowski, K. On the convergence of a sequential quadratic programming method with an augmented lagrangian line search function. *Math. Oper. und Stat. Ser. Optim.* **14**, 197–216 (1983). DOI 10.1080/02331938308842847.

120. Kraft, D. A software package for sequential quadratic programming. *Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt* (1988).

121. Nash, S. G. Newton-type minimization via the Lanczos method. *SIAM J. on Numer. Analysis* **21**, 770–788 (1984).

122. Powell, M. J. A new algorithm for unconstrained optimization. *Nonlinear programming* 31–65 (1970).

123. Steihaug, T. The conjugate gradient method and trust regions in large scale optimization. *SIAM J. on Numer. Analysis* **20**, 626–637 (1983).

124. Conn, A. R., Gould, N. I. & Toint, P. L. *Trust Region Methods* (SIAM, 2000).

125. Moré, J. J. & Sorensen, D. C. Computing a trust region step. *SIAM J. on Sci. Stat. Comput.* **4**, 553–572 (1983).

126. Gould, N. I. M., Lucidi, S., Roma, M. & Toint, P. L. Solving the trust-region subproblem using the Lanczos method. *SIAM J. on Optim.* **9**, 504–525 (1999).

127. Lenders, F., Kirches, C. & Potschka, A. trlib: A vector-free implementation of the GLTR method for iterative solution of the trust region problem. *Optim. Methods Softw.* (2018). URL https://doi.org/10.1080/10556788.2018.1449842. DOI 10.1080/10556788.2018.1449842.

128. Griffiths, T. L. & Steyvers, M. Finding scientific topics. *Proc. Natl. Acad. Sci.* **101**, 5228–5235 (2004). URL http://www.pnas.org/content/101/suppl_1/5228. DOI 10.1073/pnas.0307752101. http://www.pnas.org/content/101/suppl_1/5228.full.pdf.

129. Dierckx, P. FITPACK. URL http://netlib.org/dierckx/.

130. Pauli Virtanen. ENH: interpolate: rewrite ppform evaluation in Cython (2013). URL https://github.com/scipy/scipy/pull/2885.

131. Evgeni Burovski. add b-splines (2013). URL https://github.com/scipy/scipy/pull/3174.

132. de Boor, C. *A practical guide to splines* (Springer-Verlag, 1978).

133. Nikolay Mayorov. ENH: CubicSpline interpolator (2016). URL https://github.com/scipy/scipy/pull/5653.

134. Fritsch, F. N. & Carlson, R. E. Monotone piecewise cubic interpolation. *SIAM J. on Numer. Analysis* **17**, 238–246 (1980).

135. Akima, H. A new method of interpolation and smooth curve fitting based on local procedures. *J. ACM* **17**, 589–602 (1970).

136. Tyler Reddy. Track SciPy code base % line coverage over time (2019). URL https://github.com/tylerjereddy/scipy-cov-track.

137. Airspeed Velocity developers; open source community. Airspeed Velocity: A simple Python benchmarking tool with web-based reporting (2019). URL https://github.com/airspeed-velocity/asv.

138. The SciPy Community. SciPy project governance (2017). URL https://docs.scipy.org/doc/scipy/reference/dev/governance/governance.html.

139. SciPy Developers. SciPy Developer Guide (2019). URL https://scipy.github.io/devdocs/dev/index.html.

140. Eghbal, N. Roads and Bridges: The Unseen Labor Behind Our Digital Infrastructure. Tech. Rep., Ford Foundation (2016).

141. Collaboration, T. A. *et al.* The astropy project: Building an open-science project and status of the v2.0 core package. *The Astron. J.* **156**, 123 (2018). URL http://stacks.iop.org/1538-3881/156/i=3/a=123.

142. Lev, O., Dufresne, J., Kasim, R., Skinn, B. & Wilk, J. pypinfo: View PyPI download statistics with ease (2018). URL https://github.com/ofek/pypinfo.

143. Anaconda, Inc. Conda Package Download Data (2019). URL https://github.com/ContinuumIO/anaconda-package-data.

144. Jones, E., Oliphant, T., Peterson, P. *et al.* SciPy: Open source scientific tools for Python (2001–). URL http://www.scipy.org/. [Online; accessed 2018-06-28].

145. Google Scholar. Scipy - Google Scholar (2019). URL https://scholar.google.com/scholar?q=SciPy.

146. Abbott, B. P. *et al.* Observation of gravitational waves from a binary black hole merger. *Phys. Rev. Lett.* **116**, 061102 (2016). URL https://link.aps.org/doi/10.1103/PhysRevLett.116.061102. DOI 10.1103/PhysRevLett.116.061102.

147. Intel Software. The Intel Distribution for Python (2018). URL https://software.intel.com/en-us/articles/intel-optimized-packages-for-the-intel-distribution-for-python.

148. Thomas Elliott. The State of the Octoverse: Machine Learning (2019). URL https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/.

149. The SciPy Project Community. Scipy Roadmap (2017). URL https://docs.scipy.org/doc/scipy-1.0.0/reference/roadmap.html.

150. The SciPy Project Community. Scipy Roadmap (2019). URL https://scipy.github.io/devdocs/roadmap.html.

151. The SciPy Project Community. Scipy Tutorial (2018). URL https://docs.scipy.org/doc/scipy/reference/tutorial/.

152. Hameer Abbasi. Sparse: A more modern sparse array library. In Fatih Akici, David Lippa, Dillon Niederhut & Pacer, M. (eds.) *Proceedings of the 17th Python in Science Conference*, 27–30 (2018). DOI 10.25080/Majora-4af1f417-00a.

Author Contributions Statement

All authors have contributed code to SciPy, and all authors reviewed the manuscript. In the author list, the names of authors who contributed text to the manuscript are indicated with †, and the primary manuscript editors are denoted with ‡.

Competing Interests

The authors declare no competing interests.