The Journal of Open Source Software

Sciris: Simplifying scientific software in Python

Cliff C. Kerr [1,2¶], **Paula Sanz-Leon** [1], **Romesh G. Abeysuriya** [1,3], **George L. Chadderdon** [3], **Vlad-Ştefan Harbuz**[4], **Parham Saidi**[4], **James Jansson**[5], **Maria del Mar Quiroga** [3], **Rowan Martin-Hughes** [3], **Sherrie L. Kelly** [3], **Jamie A. Cohen** [1], **Robyn M. Stuart** [1,3], **and Anna Nachesa**[6]

1 Institute for Disease Modeling, Global Health Division, Bill & Melinda Gates Foundation, Seattle, USA **2** School of Physics, University of Sydney, Sydney, Australia **3** Burnet Institute, Melbourne, Victoria, Australia **4** Saffron Software, Bucharest, Romania **5** Kirby Institute, University of New South Wales, Sydney, Australia **6** Google Inc., Zürich, Switzerland ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

Sciris is a collection of tools that simplifies use of the foundational libraries in the scientific Python ecosystem (such as NumPy for math and Matplotlib for plotting), as well as with libraries of broader scope, such as `multiprocess` (for parallelization) and `pickle` (for saving and loading objects). The purpose of Sciris is to accelerate the development of scientific software. This is achieved by providing classes and functions that provide easy access to frequently used low-level functionality, which, while valuable for developing robust software applications, diverts focus from the scientific problem being solved. Some of Sciris' key features include: ensuring consistent dictionary, list, and array types (e.g., enabling users to provide inputs as either lists or arrays); enabling ordered dictionary elements to be referenced by index; simplifying datetime arithmetic by allowing users to provide dates in various formats, including strings; simplifying the saving and loading of files and complex objects; and simplifying the parallel execution of code. Sciris makes writing scientific code in Python faster, more pleasant, and more accessible, especially for people without extensive training in software development. With Sciris, users can achieve the same functionality with fewer lines of code, avoid reinventing the wheel, and spend less time looking up solutions on Stack Overflow. Sciris also forms the basis of ScirisWeb, an additional set of tools for building Python webapps. In summary, Sciris aims to streamline the wide spectrum of tasks commonly required when developing scientific software tools.

Statement of need

With the increasing availability of large volumes of data and computing resources, scientists across multiple fields of research have been able to tackle increasingly complex problems. But to harness these resources, the need to develop and use domain-specific software has become a ubiquitous part of scientific projects. Commensurate with the complexity of problems, these software-related activities have also become increasingly complex, creating a steep learning curve and an increasing burden of code review (Editorial Nature Methods, 2018).

The current reality of scientific code production is that any workflow (e.g., either a full cycle in the development of a new software library, or in the execution of a one-off individual analysis) very often relies on multiple codebases, including but not limited to: low-level libraries; domain-specific open-source software; and self-developed and/or inherited Swiss-army-knife toolboxes – whose original developer may or may not be around to pass on undocumented wisdom. Several scientific communities have adopted collaborative, community-driven, open-source software approaches due to the significant savings in development costs and increases in code

quality that they afford (Kerr, 2019) (e.g., `astropy` (Robitaille et al., 2013), `nilearn` (The Nilearn Team, 2010-2022) and `fmriprep` (Esteban et al., 2019), HoloViz's libraries). Despite this progress, a large fraction of scientific software efforts remain a solo adventure leading to proliferation of tools where resources are largely spent reinventing wheels. Since the core purpose of scientific software is to support scientific conclusions, it must be be: re-runnable, repeatable, reproducible, reusable, and replicable (Benureau & Rougier, 2018).

Beyond these minimum requirements, low-level programming abstractions may get in the way of clarifying the science. For instance, one of the reasons PyTorch has become popular in academic and research environments is its success in making models easier to write compared to TensorFlow (Lorica, 2017). The need for libraries that provide "simplifying interfaces" for research applications is reflected by the development of various notable libraries in scientific Python ecosystem that have enabled researchers focus their time and efforts on solving problems, prototyping solutions, deploying applications and educating their communities. Some of these include PyTorch, seaborn (Waskom, 2021), DataLad (Halchenko et al., 2021), pingouin (Vallat, 2018), hypothesis (MacIver et al., 2019), Mayavi (Ramachandran & Varoquaux, 2011) and PyVista (Sullivan & Kaszynski, 2019), among many others.

Sciris (whose name comes from a combination of "scientific" and "iris", the Greek word for "rainbow") originated in 2014, initially created to support development of the Optima suite of models (Kerr et al., 2015). We repeatedly encountering the same inconveniences while building scientific webapps, and so began collecting the tools we used to overcome them into a shared library. While Python was, and still is, considered an easy-to-use language for beginners, the motivation that shaped Sciris' evolution was to further lower the barriers to access, interact with, and orchestrate the numerous supporting libraries we were using.

For those reasons Sciris provides tools that will result in a more effective and sustainable scientific code production for solo developers and teams alike, and increased longevity (Perkel, 2020) of new scientific libraries. Some of the key functional aspects that Sciris provides are: (i) brevity through simplifying interfaces; (ii) scientific idiomaticity; (iii) locally scoped forgiving and strict exception handling; and (iv) management of versioning. We expand on each of these below, but first provide a vignette that illustrates many of Sciris' features.

Vignette

The Sciris library offers a straightforward interface to various well-established Python libraries. Writing a script that uses these libraries directly can obscure the key logic of the scientific problem, while Sciris aims to simplify common tasks as much as possible.

For example, imagine that we wish to sample random numbers from a user-defined function with varying noise levels, save the intermediate calculations, and plot the results. In vanilla Python, each of these operations is somewhat cumbersome. Figure 1 presents two functionally identical scripts and highlights that the one written with Sciris is much more succinct and readable:

The Journal of Open Source Software

```diff
 # Define random wave generator
 import numpy as np

 def randwave(std, xmin=0, xmax=10, npts=50):
     np.random.seed(int(100*std)) # Ensure differences between runs
     a = np.cos(np.linspace(xmin, xmax, npts))
     b = np.random.randn(npts)
     return a + b*std

 # Other imports
-import time
-import multiprocessing as mp
-import pickle
-import gzip
-import matplotlib.pyplot as plt
-from mpl_toolkits.mplot3d import Axes3D # Unused but must be imported
+import sciris as sc

 # Start timing
-start = time.time()
+T = sc.timer()

 # Calculate output in parallel
-multipool = mp.Pool(processes=mp.cpu_count())
-waves = multipool.map(randwave, np.linspace(0, 1, 11))
-multipool.close()
-multipool.join()
+waves = sc.parallelize(randwave, np.linspace(0, 1, 11))

 # Save to files
-filenames = []
-for i,wave in enumerate(waves):
-    filename = f'wave{i}.obj'
-    with gzip.GzipFile(filename, 'wb') as fileobj:
-        fileobj.write(pickle.dumps(wave))
-    filenames.append(filename)
+filenames = [sc.save(f'wave{i}.obj', wave) for i,wave in enumerate(waves)]

 # Create dict from files
-data_dict = {}
-for fname in filenames:
-    with gzip.GzipFile(fname) as fileobj:
-        filestring = fileobj.read()
-        data_dict[fname] = pickle.loads(filestring)
+data = sc.odict({fname:sc.load(fname) for fname in filenames})

 # Create 3D plot
-data = np.array([data_dict[fname] for fname in filenames])
-fig = plt.figure()
-ax = plt.axes(projection='3d')
-ny,nx = np.array(data).shape
-x = np.arange(nx)
-y = np.arange(ny)
-X, Y = np.meshgrid(x, y)
-surf = ax.plot_surface(X, Y, data, cmap='coolwarm')
-fig.colorbar(surf)
+sc.surf3d(data[:], plotkwargs=dict(cmap='orangeblue'))

 # Print elapsed time
-elapsed = time.time() - start
-print(f'Elapsed time: {elapsed:0.1f} s')
+T.toc()
```

Figure 1: Comparison of a functionally identical script without Sciris (left) and with Sciris (right), showing a nearly five-fold reduction in lines of code. The resulting plot is shown in Figure 2.

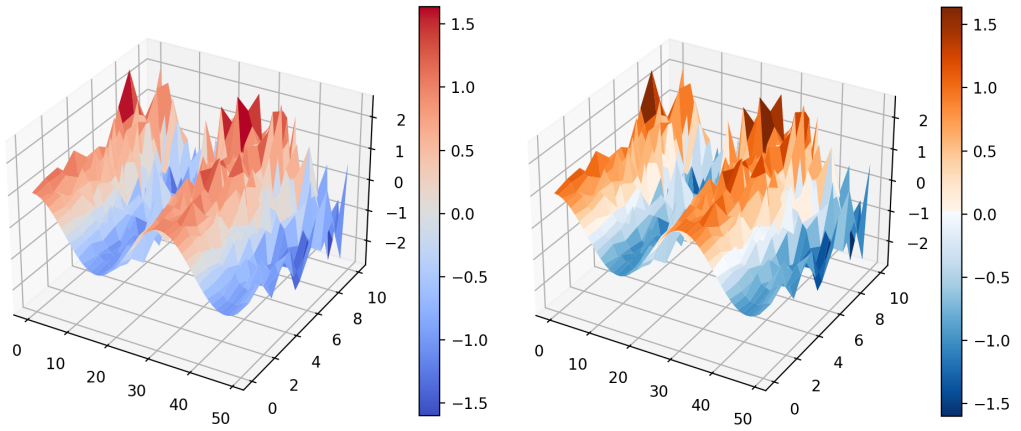

Figure 2: Output of the codes shown in Figure 1, without Sciris (left) and with Sciris (right). The two are identical except for the new high-contrast colormap available in Sciris.

This vignette illustrates many of Sciris' most-used features, including timing, parallelization,

high-performance containers, file saving and loading, and plotting. For the parts of the script that differ, Sciris reduces the number of lines required from 33 to 7, a 79% decrease.

Design philosophy

The aim of Sciris is to make common tasks simple. The current stable version of Sciris (2.0.5) includes implementations of heavily used code patterns and abstractions that facilitate the development and deployment of complex domain-specific scientific applications, and helps non-specialist audiences interact with these applications. We note that Sciris "stands on the shoulders of giants", and as such is not intended as a replacement of these libraries, but rather as an interface that facilitate a more effective and sustainable development process through the following principles:

Brevity through simplifying interfaces. Sciris packages common patterns requiring multiple lines of code into single, simple functions. With these functions one can succinctly express and execute frequent plotting tasks (e.g., `sc.commaticks`, `sc.dateformatter`, `sc.plot3d`); ensure consistent types and merging containers (e.g., `sc.toarray`, `sc.mergedicts`, `sc.mergelists`), or line-by-line performance profiling (`sc.profile`). Brevity is also achieved by extending functionality of well established objects (e.g., `OrderedDict` via `sc.odict`) or methods (e.g., `isinstance` via `sc.checktype` that enables the comparison of objects against higher-level types like `listlike`).

Dejargonification. Sciris aims to use plain function names (i.e., `sc.smooth`, `sc.findnearest`, `sc.safedivide`) so that the resulting code is as scientifically and human-readable as possible (e.g., contrast `sc.findinds()` with the functionally similar `np.nonzero()[0]`). Further, some Sciris functions map onto Matlab equivalents (e.g., `sc.tic` and `sc.toc`; `sc.boxoff`), to reduce the learning curve for scientists with this background.

Forgiving and strict exception handling. Across many complex classes and methods, Sciris uses the keyword `die`, enabling users to set a locally scoped level of strictness in the handling of exceptions. If `die=False`, Sciris is more forgiving and softly handles exceptions by using its default (opinionated) behavior. This could be simply printing a message and returning `None` so users can decide how to proceed. If `die=True`, it directly raises the corresponding exception and message. This reduces the need for try-catch blocks, which can distract from the code's scientific logic.

Management of versioning information. Keeping track of dates, authors, code version, plus additional notes or comments is an essential part of scientific projects. Sciris provides methods to easily save and load metadata to/from figure files, including Git information (`sc.savefig`, `sc.gitinfo`, `sc.loadmetadata`), as well as shortcuts for comparing module versions (`sc.compareversions`) or enforcing them (`sc.require`).

Our investments in Sciris paid off when in early 2020 its combination of brevity and simplicity proved crucial in enabling the rapid development of the Covasim model of COVID-19 transmission (Kerr et al., 2021, 2022). Covasim's relative simplicity and readability, based in large part on its heavy use of Sciris, enabled it to become one of the most widely adopted models of COVID-19, used by students, researchers and policymakers in over 30 countries.

In addition to Covasim, Sciris is currently used by a number of other scientific software tools, such as Optima HIV (Kerr et al., 2015), Optima Nutrition (Pearson et al., 2018), the Cascade Analysis Tool (Kedziora et al., 2019), Atomica (The Atomica Team, 2020), Optima TB (Goscé et al., 2021), the Health Interventions Prioritization Tool (Fraser-Hurt et al., 2021), SynthPops (Mistry et al., 2021), and FPsim (O'Brien et al., 2022).

The Journal of Open Source Software

Overview of key features

Sciris provides class- and function-based implementations of common operations ranging from parallelization to improved object representation; here we illustrate a smattering of key features in greater detail. Further information can be found at https://docs.sciris.org. Documentation includes installation instructions (`pip install sciris`), for both Sciris and ScirisWeb, usage instructions, and style guidelines.

Figure 3 illustrates the functional modules of Sciris.

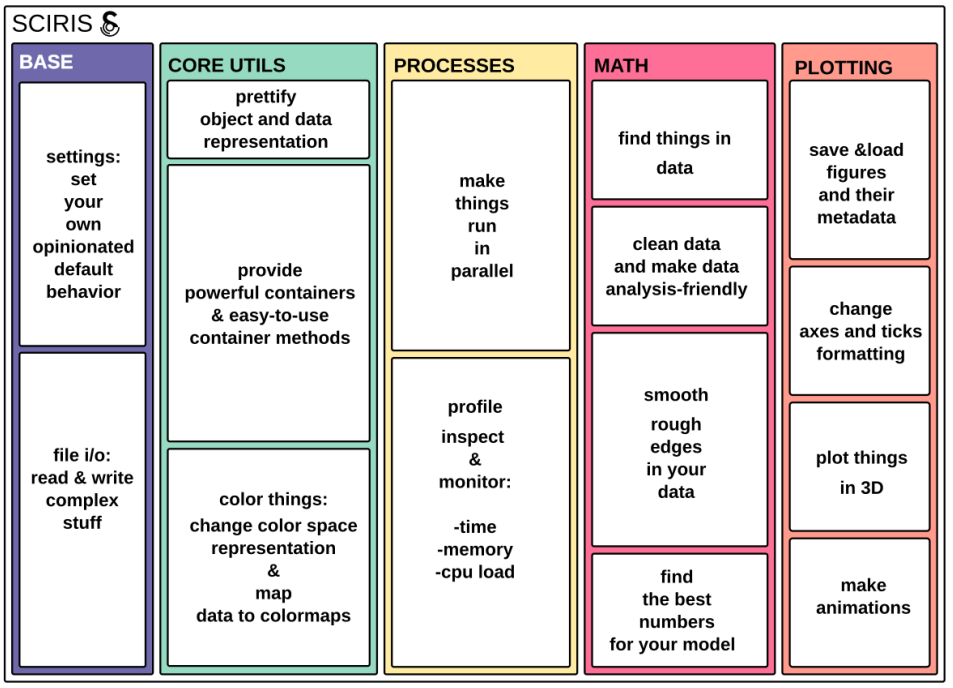

Figure 3: Block diagram of the Sciris' functionality, grouped by high-level concepts and types of tasks that are commonly performed in scientific code.

High-performance containers

One of the key features in Sciris is `sc.odict`, a flexible container representing an associative array with the best-of-all-worlds across lists, dictionaries, and numeric arrays. This is based on `OrderedDict` from `collections`, but supports list methods like integer indexing, key slicing, and item inserting:

```
> my_odict = sc.odict(foo=[1,2,3], bar=[4,5,6])
> my_odict['foo'] == my_odict[0]
> my_odict[:].sum() == 21
> for i, key, value in my_odict.enumitems():
      print(f'Item {i} is named {key} and has value {value}')
```

Numerical utilities

`sc.findinds` matches even if two things are not exactly equal due to differences in numeric type (e.g., floats vs. integers). The code shown below produces the same result as calling `np.nonzero(np.isclose(arr, val))[0]`.

```
> sc.findinds([2,3,6,3], 3)
array([1,3])
```

Parallelization

A frequent hurdle scientists face is parallelization. Sciris provides `sc.parallelize`, which acts as a shortcut for using `multiprocess.Pool()`. Importantly, this function can also iterate over more complex arguments. It can either use a fixed number of CPUs or allocate dynamically based on load (`sc.loadbalancer`). Users can also specify a fixed number of CPUs to be used. The example below shows three equivalent ways to iterate over multiple arguments:

```
> def f(x, y):
>     return x*y

> out1 = sc.parallelize(func=f, iterarg=[(1,2),(2,3),(3,4)])
> out2 = sc.parallelize(func=f, iterkwargs={'x':[1,2,3], 'y':[2,3,4]})
> out3 = sc.parallelize(func=f, iterkwargs=[{'x':1, 'y':2},
                                            {'x':2, 'y':3},
                                            {'x':3, 'y':4}])

> assert out1 == out2 == out3
```

Plotting

Numerous shortcuts for plotting customizations are available in Sciris; several commonly used features are illustrated below, with the results shown in Figure 4:

```
> sc.options(font='Garamond') # Set custom font
> x = sc.daterange('2022-06-01', '2022-12-31', as_date=True) # Create dates
> y = sc.smooth(np.random.randn(len(x))**2*1000) # Create smoothed random numbers
> c = sc.vectocolor(y, cmap='turbo') # Set colors proportional to y values
> plt.scatter(x, y, c=c) # Plot the data
> sc.dateformatter() # Automatic x-axis date formatter
> sc.commaticks() # Add commas to y-axis tick labels
> sc.setylim() # Reset the y-axis limits
> sc.boxoff() # Remove the top and right axis spines
```

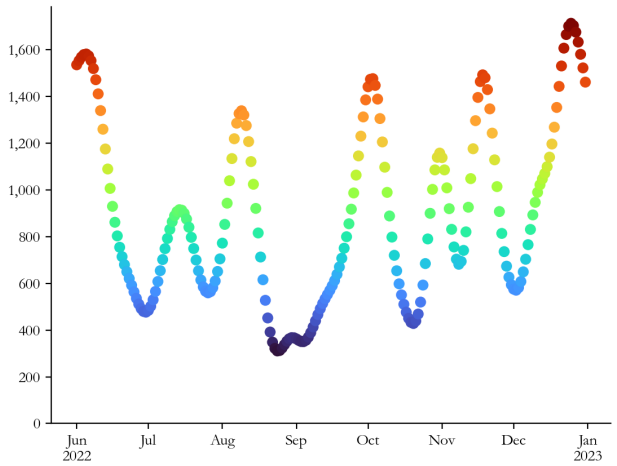

Figure 4: Example of plot customizations via Sciris, including x- and y-axis tick labels and the font.

ScirisWeb

While a full description of ScirisWeb is beyond the scope of this paper, briefly, it builds on Sciris to enable the rapid development of Python-based webapps, including those powering Optima Nutrition and Covasim. By default, ScirisWeb uses Vuejs for the frontend, Flask as the web framework, Redis for the (optional) database, and Matplotlib/mpld3 for plotting. However, ScirisWeb is completely modular, which means that it can also be used to link a React frontend to a MySQL database with Plotly figures. In contrast to Plotly Dash and Streamlit, which have limited options for customization or switching between technology stacks, ScirisWeb is completely modular, so users can freely choose which features they use for a given project.

Acknowledgements

The authors wish to thank David J. Kedziora, Dominic Delport, Kevin M. Jablonka, Meikang Wu, and Dina Mistry for providing helpful feedback on the Sciris library. David P. Wilson, William B. Lytton, and Daniel J. Klein provided in-kind support of Sciris development. Financial support has been provided by the United States Defense Advanced Research Projects Agency (DARPA) Contract N66001-10-C-2008 (2010–2014), World Bank Assignment 1045478 (2011–2015), the Australian National Health and Medical Research Council (NHMRC) Project Grant APP1086540 (2015–2017), the Australian Research Council (ARC) Discovery Early Career Research Award (DECRA) Fellowship Grant DE140101375 (2014–2019), Intellectual Ventures (2019–2020), and the Bill & Melinda Gates Foundation (2020–present).

References

Benureau, F. C., & Rougier, N. P. (2018). Re-run, repeat, reproduce, reuse, replicate: Transforming code into scientific contributions. *Frontiers in Neuroinformatics*, *11*, 69.

Editorial Nature Methods. (2018). Easing the burden of code review. *Nature Methods*, *15*(9).

Esteban, O., Markiewicz, C. J., Blair, R. W., Moodie, C. A., Isik, A. I., Erramuzpe, A., Kent, J. D., Goncalves, M., DuPre, E., Snyder, M., & others. (2019). fMRIPrep: A robust preprocessing pipeline for functional MRI. *Nature Methods*, *16*(1), 111–116.

Fraser-Hurt, N., Hou, X., Wilkinson, T., Duran, D., Abou Jaoude, G. J., Skordis, J., Chukwuma, A., Lao Pena, C., Tshivuila Matala, O. O., Gorgens, M., & others. (2021). Using allocative efficiency analysis to inform health benefits package design for progressing towards universal health coverage: Proof-of-concept studies in countries seeking decision support. *PLoS One*, *16*(11), e0260247.

Goscé, L., Abou Jaoude, G. J., Kedziora, D. J., Benedikt, C., Hussain, A., Jarvis, S., Skrahina, A., Klimuk, D., Hurevich, H., Zhao, F., & others. (2021). Optima TB: A tool to help optimally allocate tuberculosis spending. *PLoS Computational Biology*, *17*(9), e1009255.

Halchenko, Y., Meyer, K., Poldrack, B., Solanky, D., Wagner, A., Gors, J., MacFarlane, D., Pustina, D., Sochat, V., Ghosh, S., & others. (2021). DataLad: Distributed system for joint management of code, data, and their relationship. *Journal of Open Source Software*, *6*(63).

Kedziora, D. J., Abeysuriya, R., Kerr, C. C., Chadderdon, G. L., Harbuz, V.-Ș., Metzger, S., Wilson, D. P., & Stuart, R. M. (2019). The cascade analysis tool: Software to analyze and optimize care cascades. *Gates Open Research*, *3*.

Kerr, C. C. (2019). Is epidemiology ready for big software? *Pathogens and Disease*, *77*(1), ftz006.

Kerr, C. C., Stuart, R. M., Gray, R. T., Shattock, A. J., Fraser-Hurt, N., Benedikt, C., Haacker, M., Berdnikov, M., Mahmood, A. M., Jaber, S. A., & others. (2015). Optima: A model for HIV epidemic analysis, program prioritization, and resource optimization. *Journal of Acquired Immune Deficiency Syndromes*, *69*(3), 365–376.

Kerr, C. C., Stuart, R. M., Mistry, D., Abeysuriya, R. G., Cohen, J. A., George, L., Jastrzebski, M., Famulare, M., Wenger, E., & Klein, D. J. (2022). *Python vs. The pandemic: A case study in high-stakes software development*.

Kerr, C. C., Stuart, R. M., Mistry, D., Abeysuriya, R. G., Rosenfeld, K., Hart, G. R., Núñez, R. C., Cohen, J. A., Selvaraj, P., Hagedorn, B., & others. (2021). Covasim: An agent-based model of COVID-19 dynamics and interventions. *PLOS Computational Biology*, *17*(7), e1009149.

Lorica, B. (2017). *Why AI and machine learning researchers are beginning to embrace PyTorch*. https://www.oreilly.com/radar/podcast/why-ai-and-machine-learning-researchers-are-beginning-to-e

MacIver, D. R., Hatfield-Dodds, Z., & others. (2019). Hypothesis: A new approach to property-based testing. *Journal of Open Source Software*, *4*(43), 1891.

Mistry, D., Kerr, C. C., Abeysuriya, R. G., Wu, M., Fisher, M., Thompson, A., Skrip, L., Cohen, J. A., Althouse, B. M., & Klein, D. (2021). *SynthPops: A generative model of human contact networks. In preparation*.

O'Brien, M. L., Valente, A., Chabot-Couture, G., Proctor, J., Klein, D., Kerr, C., & Zimmermann, M. (2022). FPSim: An agent-based model of family planning for informed policy decision-making. *PAA 2022 Annual Meeting*.

Pearson, R., Killedar, M., Petravic, J., Kakietek, J. J., Scott, N., Grantham, K. L., Stuart, R. M., Kedziora, D. J., Kerr, C. C., Skordis-Worrall, J., & others. (2018). Optima nutrition: An allocative efficiency tool to reduce childhood stunting by better targeting of nutrition-related interventions. *BMC Public Health*, *18*(1), 1–12.

Perkel, J. M. (2020). Challenge to scientists: Does your ten-year-old code still run? *Nature*, *584*(7822), 656–659.

Ramachandran, P., & Varoquaux, G. (2011). Mayavi: 3D visualization of scientific data. *Computing in Science & Engineering*, *13*(2), 40–51.

Robitaille, T. P., Tollerud, E. J., Greenfield, P., Droettboom, M., Bray, E., Aldcroft, T., Davis, M., Ginsburg, A., Price-Whelan, A. M., Kerzendorf, W. E., & others. (2013). Astropy: A community python package for astronomy. *Astronomy & Astrophysics*, *558*, A33.

Sullivan, C., & Kaszynski, A. (2019). PyVista: 3D plotting and mesh analysis through a streamlined interface for the visualization toolkit (VTK). *Journal of Open Source Software*, *4*(37), 1450.

The Atomica Team. (2020). Atomica: A simulation engine for compartmental models. In *GitHub repository*. GitHub. https://github.com/atomicateam/atomica

The Nilearn Team. (2010-2022). Nilearn: A library for easy statistical and machine-learning analysis of brain volumes. In *GitHub repository*. GitHub. https://nilearn.github.io/stable/index.html

Vallat, R. (2018). Pingouin: Statistics in python. *Journal of Open Source Software*, *3*(31), 1026.

Waskom, M. L. (2021). Seaborn: Statistical data visualization. *Journal of Open Source Software*, *6*(60), 3021.