

Toward SALib 2.0: Advancing the accessibility and interpretability of global sensitivity analyses

Takuya Iwanaga^{1*}, Will Usher², and Jon Herman³

¹ Institute for Water Futures, The Australian National University, Australia

² Division of Energy Systems, School of Industrial Engineering and Management, KTH Royal Institute of Technology, Sweden

³ Department of Civil and Environmental Engineering, University of California, Davis, CA, USA

Abstract

Sensitivity analysis is now considered a standard practice in environmental modeling. Several open-source libraries, such as the Sensitivity Analysis Library (SALib), have been published in the recent past aimed at simplifying the application of sensitivity analyses. Still, there remain issues in software usability and accessibility, as well as a lack of guidance in the interpretation of sensitivity analysis results. This paper describes the changes made and planned to SALib to advance the ease with which modelers may conduct sensitivity analysis and interpret results. We further offer our perspectives from the past 7 years of maintaining SALib for the consideration of those aspiring to launch their own software for sensitivity analysis, develop methodology, or those otherwise interested in becoming involved in a project like SALib. These include the value of a community of practice to foster best practices for sensitivity analysis, the potential for collaboration across different software (for sensitivity analysis) platforms, and the need to specifically support the software development that underpins computational science.

Keywords

sensitivity analysis; community of practice; software accessibility

Code availability

The SALib project is hosted on GitHub (at <https://github.com/SALib/SALib>) and is made available under the MIT license. Data, code, and figures for citation analysis conducted for this publication are found in Iwanaga (2021) accessible via <https://doi.org/10.5281/zenodo.5523624>.

1. Introduction

The explosive growth and availability of computational power has increased the complexity of environmental modeling analyses, both in terms of the models themselves, as well as the data produced and collected. Sensitivity analysis (SA) is now considered a standard practice in modeling workflows as it can help navigate this complexity. At its core SA illuminates the behavior of a system by exploring and mapping the relationship between the inputs and outputs. Application of SA aids in identification of the relative magnitudes at which model factors drive results (factor prioritization); screening out non-influential parameters to reduce dimensionality of a model (factor fixing); better understanding how the input space relates to an interesting area of the output space (factor mapping); and supports meta-modelling (Saltelli et al., 2008). Application of SA

Correspondence:

Contact T. Iwanaga at iwanaga.takuya@anu.edu.au

Cite this article as:

Iwanaga, T., Usher, W., & Herman, J.
Toward SALib 2.0: Advancing the accessibility and interpretability of global sensitivity analyses
Socio-Environmental Systems Modelling, vol. 4, 18155, 2022, doi:10.18174/sesmo.18155

This work is licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](https://creativecommons.org/licenses/by-nc/4.0/).



Socio-Environmental Systems Modelling

An Open-Access Scholarly Journal

<http://www.sesmo.org>

is also beneficial towards a wide range of contexts within scientific research and decision support beyond model analysis (Razavi et al., 2021; Wagener and Pianosi, 2019).

A key difficulty in applying SA is that sufficient areas of parameter space must be explored for SA to be effective particularly for models with nonlinear behavior; a global analysis (GSA) is necessary as opposed to local (LSA; Saltelli and Annoni, 2010). Although use of GSA has been slowly increasing in recent years (Douglas-Smith et al., 2020) there is some evidence that LSA remains the dominant form of SA in the literature (Ferretti et al., 2016; Saltelli et al., 2019). Past reviews have offered reasons for the hesitancy in adopting GSA as part of modeling practice. These include a lack of awareness of GSA itself (Saltelli et al., 2019); the relative complexity of the interpretation and implementation of GSA methods compared to LSA (Ferretti et al., 2016; Saltelli and Annoni, 2010); a lack of awareness, accessibility or usability of available tooling to conduct GSA (Douglas-Smith et al., 2020; Puy et al., 2021; Razavi et al., 2021); and a lack of guidance in the interpretation of SA results (Pianosi et al., 2020; Wagener and Pianosi, 2019). The computational cost of conducting such analyses may also be hampering uptake of GSA, especially as models become increasingly complex with large numbers of factors (Cuntz et al., 2015).

The Sensitivity Analysis Library (SALib; Herman and Usher, 2017) is a Python package developed and released as a collection of easy-to-use and well-tested methods for GSA. Since its initial release (circa 2013) the library has evolved to offer not only a wide range of SA methods but to further reduce the friction in their application. Usability is one area of focus to encourage adoption of SA amongst modelers. Resolution of complex issues can be brought about quicker with tools that are trusted, intuitive, and simple to use (Razavi et al., 2021). In this paper we outline our efforts to address the above barriers to adoption and provide key lessons and perspectives to be considered by SA practitioners who are aspiring to launch their own software for SA, develop new SA methodology, or are interested in becoming involved in an SA-related project such as the SALib project.

An overview of SALib is included in Section 2, followed by an outline of the improvements made since its initial release (Section 3). In Section 4, we then identify general changes planned for v2.0 with the overarching goal of improving interpretability of SA results, which also apply to other SA packages. A further discussion of the development of a community of SA practitioners, and its role in increasing the accessibility and applicability of the software, is also found therein. We conclude in Section 5 with opportunities to advance cross-language collaboration and funding support for open-source development in the SA community.

2. Overview of SALib

SALib is perhaps the oldest openly developed sensitivity analysis library for the Python ecosystem and is unique in that it is developed and maintained by a handful of volunteers from a variety of disciplinary backgrounds. It provides well-tested implementations of common and emerging SA methods and test functions, written in the Python programming language. As a software package SALib is distributed through, and installable from, the Python Package Index (PyPI) as well as the Anaconda package repository (through the conda-forge channel; Conda-Forge Community, 2015). Tutorials and guides are available through the documentation (see Section 3.3) allowing those new to SALib, as well as SA, to quickly get started.

Project code is publicly hosted on GitHub through which a community of practice has evolved (expanded on in Section 3.4). Community contributions have been essential for new methods, features, and bug reports, and have moved the library toward organized testing and versioning from its origins as an informally developed collection of scripts. Usability is a core focus, with use of the package intended to be accessible to anyone with a working understanding of Python. Currently available SA methods as of v1.4 are listed in Table 1.

Evidence of the broad applicability of SALib, and SA more generally, can be seen in the citations of the earlier 2017 paper describing v1.0 (Herman and Usher, 2017) and adoption of the software. Although dominated by the engineering fields, citations come from 17 areas of research (see Figure 1 and Figure 2). Since its initial release SALib has also been integrated into several disciplinary-specific tools and frameworks. These include (but are not limited to) an agent-based modeling framework (Agentpy; Foramitti, 2021); exploratory modeling frameworks (Rhodium and the EMA Workbench; Hadjimichael et al., 2020; Kwakkel, 2017); an urban energy simulation platform (CityEnergyAnalyst; Fonseca et al., 2021), and a probabilistic natural catastrophe impact model (CLIMADA; Aznar-Siguan and Bresch, 2019; Bresch and Aznar-Siguan, 2021).

Table 1: List of provided sensitivity analysis methods in SALib v1.4

Method	Approach	References
Sobol'	Variance	Saltelli, 2002; Saltelli et al., 2010; Sobol', 2001
Morris Elementary Effects	Variance	Campolongo et al., 2007; Morris, 1991; Ruano et al., 2012
Extended Fourier Amplitude Sensitivity Test (eFAST)	Variance	Cukier et al., 1973; Saltelli et al., 1999)
Random Balanced Designs Fourier Amplitude Sensitivity Test (an RBD-FAST implementation also known as EASI)	Density/moment-independent, given-data	Plischke, 2010; Tarantola et al., 2006; Tissot and Prieur, 2012
Delta Moment-Independent Measure	Density/moment-independent, given-data	Borgonovo, 2007; Plischke et al., 2013
Derivative-based Global Sensitivity Measure (DGSM)	Variance	Sobol' and Kucherenko, 2010
Fractional Factorial Sensitivity Analysis	Variance	Saltelli et al., 2008
PAWN	Density/moment-independent, given-data	Pianosi and Wagener, 2018, 2015
High-Dimensional Model Representation (HDMR)	Variance, given-data	Li et al., 2010; Rabitz et al., 1999

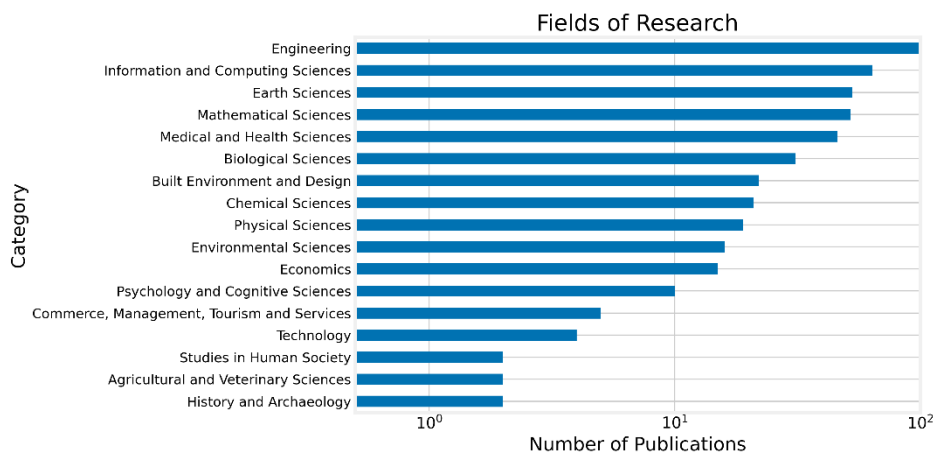


Figure 1: Publications which cite SALib come from a wide variety of research areas although concentrated in the engineering fields. Note that the figure is in log scale. Data sourced from the Dimensions.ai platform in September 2021.

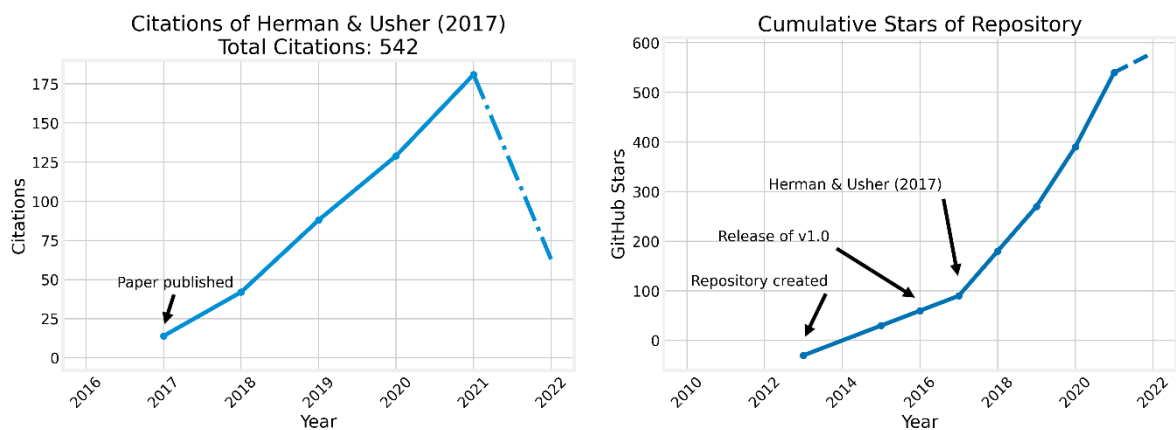


Figure 2: (a) Yearly citations of the original SALib paper Herman and Usher (2017), from Google Scholar; (b) repository “stars” used as an indicator of use, interest, and popularity of the SALib project, from GitHub. Dashed lines represent use of incomplete data (partial year, ending April 2022).

One underlying reason for the increased use of SALib is that the Python language itself has seen an increase in general popularity, from teaching contexts to professional and scientific applications. Disciplinary specialists are likely hesitant to switch to another language due to the time commitment and learning curve involved. Still, we believe that the popularity of the Python language is not the sole reason for the current level of interest in SALib. For one, the package is open-source and permissively licensed (under the MIT license), allowing for its adoption with little restriction. The package structure simplifies use and application as evidenced by the incorporation of SALib into other Python packages. Community contributions since the initial release have further enhanced the accessibility and usability of SALib to meet their evolving needs (further explored in Section 3.4). Specifically, we believe that several key features of SALib are driving the increasing adoption and use of the SALib package, including its:

- Open source and permissive licensing,
- Focus on robust, tested implementations,
- Supporting multiple workflows and user-interfaces for ease of application,
- Improved documentation, including "cookbook" style examples,
- Extensibility, and
- Community.

3. Toward improved accessibility

An area of focus for SALib has been its usability – the ease at which SA methods are applied – as it increases accessibility, allowing researchers and practitioners to minimize the time from installation to results. A user-centric approach to development has been adopted, whereby the contextual needs and concerns of the user and key barriers to their successful, and appropriate, application is identified and mitigated. In this section we outline the changes made (and user concerns addressed) since the initial release of SALib and detail the growth and benefits of a community of SA practitioners.

3.1. Supporting a broad range of users

Sensitivity analysis has wide applicability across the sciences. Users of SALib consequently come from a variety of disciplinary backgrounds with varying levels of programming experience. Contexts of its use may differ as well, from desktop analysis to applications on a supercomputer. SALib is structured along the three basic steps of (1) input sampling, (2) model evaluation (running the model, performed separately), and (3) post-processing of results (Pianosi et al., 2016). Applying these three steps is accomplished either programmatically (i.e., with code) or through a command-line (or terminal) interface. A Graphical User Interface (GUI) is a possibility but has not been explored in detail at this stage.

The command-line user interface allows the use of SALib directly from a command prompt, useful in situations where user-written Python scripts can be tedious or disallowed. In the programmatic case, users can adopt two general software development paradigms: procedural and object-oriented. In this context, "paradigm" refers to the conceptual framework through which the functionality of SALib can be applied and is briefly described here. Adopting one paradigm over the other influences the way code is written in the application of SALib, with advantages and disadvantages to each.

The procedural paradigm is the most common in the application of SALib as it is perhaps the most familiar to users with some programming experience. The procedural paradigm focuses on prescribing how the analysis is to occur in a step-by-step manner. In the context of SALib, adopting the procedural paradigm involves defining a "problem": a simple associative array (a dictionary in Python terms) that details the number of variables, their associated names, and their value range (i.e., the variability space). Additional optional details that can be included are their distributions and whether any grouping of variables is to occur. This "problem" is then passed into individual SALib functions to generate samples and to conduct analyses (see left panel in Box 1). Care must be taken to always provide the defined "problem" with their samples and associated model results for the appropriate analyses to be conducted. On a related note, the design of the SA experiment is left to the user. Guidance on assessing the appropriateness of experimental design is not currently explicitly covered in SALib documentation, a limitation explored in more detail in Section 4.

Procedural programming is perhaps the most common programming paradigm and makes SALib accessible to those who may not be entirely familiar with Python or other programming paradigms. The examples provided as part of the library are primarily written in this procedural style, which are continually expanded. Code conforming to the procedural paradigm are perhaps the simplest implementations that still allow some degree of composability – the mix-and-matching of different functions to perform a SA – while still offering a standardized interface (via the functions) for users to interact with.

A weakness of the procedural paradigm is that it arguably places too much onus on the user, requiring the preparation of input and intermediate variables, and their states to be manually tracked. Very few safeguards are in place to prevent inappropriate analyses from occurring and this leads to a cumbersome process that is often a source of errors, particularly when conducting multiple complex analyses. The Object-Oriented (OO) paradigm can potentially reduce the cognitive load placed on users. Here, cognitive load refers to the proportion of working memory that must be devoted to understanding and processing a task (Hermans, 2021; Sweller, 1988). In this context, the load is due to the required knowledge of SA theory, the target model, SALib, and the application context. Too high a cognitive load may alienate novices, introduce errors, and in the worst case, discourage use of SA altogether.

In very general terms, the OO paradigm focuses on encapsulating and abstracting data and associated functions. An Object-Oriented (OO) Interface was introduced in SALib v1.4 through which users may programmatically interact with most of the functionality offered by SALib. For users, the OO Interface simplifies use of SALib as it provides a single point of interaction that implicitly tracks the state of data for the user as it flows from one step in the workflow to the next. In practical terms this means the problem specification and the associated samples, model results, and results of analyses do not have to be stored in separate variables by the user and explicitly passed from one function to the other.

The OO-based Interface is demonstrated in Box 1 with a method chaining approach which reflects the flow of data throughout the SA workflow. Use of the interface is demonstrated in comparison to code written in the procedural style. The Interface is created by defining an SALib Problem (with `ProblemSpec()`), see point 2 in Box 1) which is assigned, by convention, to an `sp` variable. Here, `sp` refers to “SALib Problem specification” and provides a common interface to all available sampling and analysis methods. Printing the Interface displays a concise overview of the specification and available data (illustrated in Figure 3).

For the SALib development team, the design and implementation of the OO Interface simplifies future development and maintenance activities. Contributors and maintainers are not required to write code specifically to attach individual methods to the OO Interface as this is handled automatically, reducing maintenance and contribution effort. Having a single common interface additionally offers the opportunity to provide checks to safeguard users from common errors. For example, an error is raised where the number of parameters and quantities of interest do not match the problem specification. Implementing such checks in the procedural implementation would require identical (or near-identical) lines of code to be inserted into each sampling and analysis function, thereby increasing the maintenance burden.

An additional benefit is that the methods and associated documentation are made user discoverable through inspection via a REPL or modern Integrated Development Environment (see Figure 4). Users are then able to explore these as needed. The documentation itself has undergone significant changes to provide guided working examples using test functions which can help, to give one example, limit the potential for a mismatch between sampling and analysis methods. Further aspects of the documentation are explored in Section 3.3.

The import of necessary functions becomes optional when using the OO Interface and removes the necessity for the user to track intermediate results. Calling individual methods separately is possible and doing so results in a workflow that is near-identical to the procedural code. Still, users may prefer the original procedural style, which may be simpler in cases where the model cannot be interacted with from Python directly; in cases where finer-grain control over operations is needed, such as when integrating SALib into another modeling or decision-support framework; or cases where performance may be a key concern as the OO Interface introduces additional overhead.

Box 1: Demonstration of the procedural programming style compared with the Object-Oriented Interface using the Morris method on the Sobol' G-function.

Procedural	Object-Oriented
<pre> from SALib.test_functions import Sobol_G from SALib.analyze import morris from SALib.sample.morris import sample problem = { 'num_vars': 8, 'names': ['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8'], 'groups': None, 'dists': None, 'bounds': [[0.0, 1.0],] * 8 } X = sample(problem, N=1000, num_levels=4) Y = Sobol_G.evaluate(X) Si = morris.analyze(problem, X, Y, num_levels=4, num_resamples=100) ax = Si.plot() Si_df = Si.to_df() </pre>	<pre> from SALib.test_functions import Sobol_G from SALib import ProblemSpec sp = ProblemSpec({ 'names': ['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8'], 'groups': None, 'dists': None, 'bounds': [[0.0, 1.0],] * 8, }) (sp.sample_morris(1000, num_levels=4) .evaluate(Sobol_G.evaluate) .analyze_morris(num_levels=4, num_resamples=100)) # X = sp.samples # Y = sp.results # S = sp.analysis # sp.samples = X # sp.results = Y # sp.analyze_morris() print(sp) ax = sp.plot() sp.to_df() </pre>
<p>(1) Importing SALib. Specific sampling and analysis functions should be imported when using SALib procedurally.</p> <p>(2) Specifying an SALib problem as a Python dictionary, compared to the equivalent Object-Oriented (OO) Interface for the Sobol' G-function. Optional entries allow parameter groupings ('groups'), and alternate distributions ('dists', uniform being the default). An additional 'outputs' entry (not shown) can also be provided to name the expected model outputs. When using the Interface, the 'num_vars' is inferred from the number of elements in 'names'.</p> <p>(3) The trifecta of sampling, model evaluation, and analysis with the Morris method. The default values for 'num_levels' (for the Morris analysis) and 'num_resamples' (for bootstrapping) are shown. In the case of the Interface, the 'problem' specification, samples, and model results are automatically passed onto each step as necessary. A generic 'sample()' and 'analyze()' method is also provided for use with user-defined sampling and analysis methods (see Figure 4). Model evaluation and analysis can be parallelized with the OO Interface by adding an 'nprocs' argument to the 'evaluate' and 'analyze' methods, and specifying the desired number of CPU cores to use.</p> <p>(4) It is possible to extract the stored samples, model results, and analyses from the Interface, and to provide pre-existing samples and results for analysis. Methods to set pre-existing samples and results ('.set_samples()' and '.set_results()') can also be used as part of the workflow shown in (3). Previously stored 'results' are automatically cleared when providing 'samples' to avoid data mismatches.</p> <p>(5) Print results and associated information (see Figure 3).</p> <p>(6) Produces an indicative plot. For experienced Python programmers, a matplotlib axes object is returned which can be further modified to adjust the plot. Conversion to a Pandas DataFrame for further analysis is also supported.</p>	

```

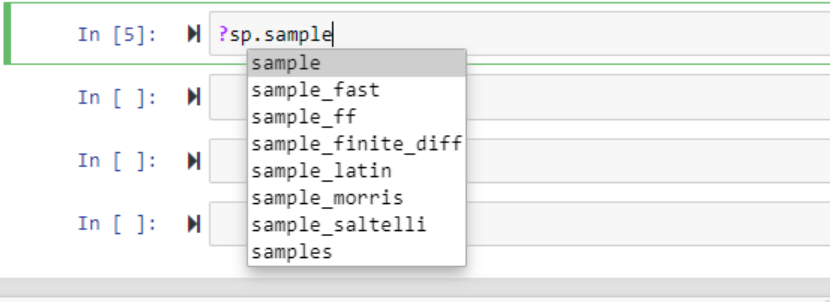
Samples:
  8 parameters: ['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8']
  9000 evaluations

Outputs:
  1 outputs: ['Y']
  9000 evaluations

Analysis:
      mu  mu_star  sigma  mu_star_conf
x1  0.094224  2.595509  2.719205  0.054289
x2  0.038762  1.486289  1.677687  0.041578
x3 -0.007628  0.595197  0.692888  0.020773
x4 -0.006204  0.326824  0.386222  0.011783
x5  0.001658  0.035332  0.041377  0.001325
x6 -0.003028  0.035522  0.041641  0.001259
x7 -0.000988  0.035416  0.041373  0.001363
x8  0.001371  0.035009  0.041213  0.001401

```

Figure 3: Using the ``print()`` function on the Object-Oriented Interface (see point 5, in Box 1) provides an overview of the available data held in the Interface.



```

In [5]: sp.sample
sample
sample_fast
sample_ff
sample_finite_diff
sample_latin
sample_morris
sample_saltelli
samples

Signature: sp.sample(func, *args, **kwargs)
Docstring:
Create sample using given function.

Parameters
-----
func : function,
      Sampling method to use. The given function must accept the SALib
      problem specification as the first parameter and return a numpy
      array.

```

Figure 4: Screenshot of a SALib Interface being inspected from a Jupyter Notebook via tab-completion, showing a list of methods beginning with “sample”. Tab-completion refers to the action where the user presses the “tab” key to bring up a list of relevant functions/methods. It is a common feature in most, if not all, modern development environments. The user can then contextually discover methods associated with the SALib Interface without having to refer to a separate manual or user guide. The documentation associated with the generic ``sample`` method is partially displayed. The ``sample`` and corresponding ``analyze`` method allow the user to provide their own methods for use with SALib, allowing use of SALib as a testbed for testing and development.

3.2. Reducing time-to-results

The Object-Oriented Interface also provides methods for quickly plotting analysis results as well as support for parallel model evaluation and analysis. Inclusion of these functionalities address two concerns. First, most diagnostic visualization tasks require (at least initially) an indicative plot of results that utilize near identical code. Second, SA is often conducted on long-running models, samples for which can be obtained independently of one another as there are typically no interactions between model runs. Similarly, analyses which focus on the relationship between specific input factors and model outputs can also be conducted independently. Such cases where the model runs are independent of each other and can be trivially parallelized (at least conceptually) is referred to as an embarrassingly parallel problem in Computer Science (Foster, 1995; Herlihy and Shavit, 2012). Yet, setting up parallel model evaluations can be daunting depending on the model context and individual skill-level of the user.

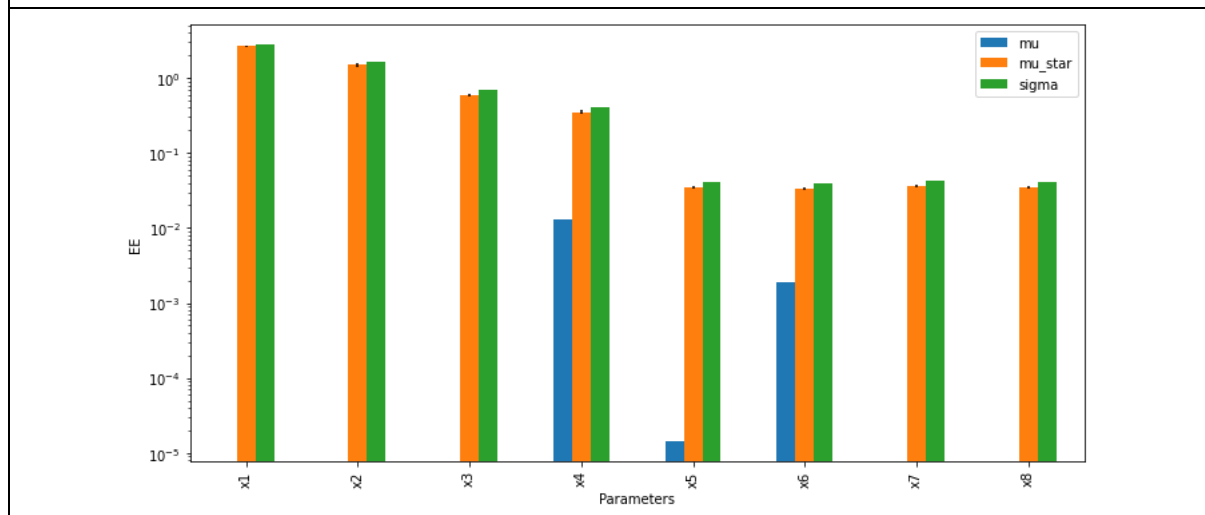
By including support for diagnostic plotting and parallel analyses as part of SALib itself mitigates to some degree the duplication of work that may otherwise occur across the sciences. Further, use of these functionality is simplified such that the user only need to invoke a single command (`.plot()`) to obtain an indicative figure after model evaluation and analysis is complete (see point 5 in Box 1). For parallelizing model evaluation itself, the user need only to specify the number of processors to use (see note for point 3 in Box 1). For suitably long-running models, the provided parallelism (implemented with the pathos library; McKerns et al., 2012) can significantly reduce evaluation and analysis time. The provided approach comes at the cost of flexibility, however, as users are not able to specify how the parallelism is to occur and assumes sufficient computer memory is available to conduct the model runs and subsequent SA.

The intention of the provided visualization functions is to allow users to obtain quick diagnostic indication of model behavior, as opposed to publish quality figures. Examples of further qualitative analyses are provided in the documentation, with a view to expand these in the future. For advanced Python users, all plotting methods return a matplotlib axes object which allows the figure to be further modified if desired (e.g., for publication purposes, see Box 2). Providing such “convenience methods” for plotting removes the need for users to write their own code for plotting which represents a duplication of work for a common workflow.

Box 2: SALib plotting methods return a standard matplotlib axes object to allow for further modification. Here the results from Box 1 are displayed using the `barplot()` method, and adjusted to be in log scale, the figure enlarged, and displayed with additional axis labels. It is also possible to modify the axes object produced by the `.plot()` method in Box 1.

```
import matplotlib.pyplot as plt
from SALib.plotting.bar import plot as barplot

fig, ax1 = plt.subplots(1,1, figsize=(12,6))
ax1 = barplot(Si.to_df(), ax=ax1)
ax1.set_yscale('log')
ax1.set_xlabel("Parameters")
ax1.set_ylabel("EE")
```



3.3. Documentation

One frequent piece of feedback received over the past years is the lack of a general and introductory guide to SA. For novices, the documentation could be the first time they are exposed to SA concepts and terminology. It is also true that experience with SA does not necessarily mean that the structure, design, and application of the software will be immediately transparent. Thus, sparsely written or poorly organized documentation can be a deterrent to effective use or adoption of the software, or in the worst-case adoption of SA in general, particularly for novices, as the relevance and importance is lost.

Since v1.0, the community has added a Read the Docs page (<https://salib.readthedocs.io/en/latest/>) which serves as a concise API reference with examples and a change log, in part, to address these concerns. This documentation continues to be updated and modified for accessibility. The decentralized development raises the challenge of keeping documentation up to date with the latest contributions. This is not to say that the documentation is perfectly accessible to users of all skill levels, and improvements are ongoing (further discussed in the Outlook section below). For one, it is desirable for the level of applicability and known limitations for each method to be described and contextualized, however briefly. More recently, a list of relevant references has been embedded in the documentation for each method, accessible through the Python REPL console, and via the built-in `help()` command (as discussed in Section 3.1 above). Cross-compatibility of sampling and analysis functions have also begun to be listed.

One driving motivation for the contributors has been to reduce the need for more documentation by better organizing the software and API itself. In the object-oriented approach, for example, auto-completion allows users to interactively explore and discover relevant functionality. A key challenge is to ensure that the sampling methods are combined with appropriate analysis methods, an issue partially ameliorated by listing the compatible methods within the documentation.

However, we also observe that many of the questions raised by the community are related to experimental design and interpretation rather than the execution of the SA methods. Efforts to better organize these questions and discussions have taken several forms, including a FAQ page on the Read the Docs website which we intend to continue expanding ahead of the v2.0 release (explored further in Section 4). The need for continually evolving documentation and support points to the value of an engaged community.

3.4. A sensitivity analysis community of practice

A relatively new advance in open-source development platforms is the adoption of functionality and features commonly associated with social networks. These include incorporating user profiles, wikis, project roadmaps, discussion forums, and other facilities to encourage and foster communities to grow around the software being developed. Users can raise questions and submit bug reports via an issue tracker, or otherwise contribute enhancements and other changes for consideration following an open code review process and automated testing. The project began as a small collaboration and the community has evolved gradually over time, bringing together developers, users, and researchers alike to contribute code and documentation and participate in discussions. A key catalyst of the growth of the community was the introduction of processes and procedures (such as documentation, coding standards, testing) which were formalized in response to needs, which then made it easier to manage contributions to the source code and documentation.

The library began as a collection of scripts authored and collated by Herman with a simple and consistent interface that allowed the interchange of different SA methods. Usher then contributed code and helped package the scripts, adding unit tests and a continuous integration service which runs the tests on each change to the code. Continuous Deployment was also added to automate the process of releasing the software to the Python Packaging Index (PyPI). With the addition of documentation and examples, the package was released and JOSS paper published (Herman and Usher, 2017). After this very intensive phase of development, the package grew in prominence in lockstep with the rapidly increasing use of Python by the scientific community and through the formal recognition of the package via academic citation. Today, Herman and Usher are less active in the development of the package, with Iwanaga driving development and coordinating the SALib community. New contributors are actively encouraged through clear documentation which provides direction on how to contribute. Suggested enhancements to the code are managed in the issue tracker. Individual contributions are collated into a pull request and reviewed by the SALib administration team. Contributions must maintain the project technical standards, some which are checked automatically (for example, code style and test coverage).

One of the most common uses of the issue tracker has turned out to be questions on the interpretation of SA results and on the problem configuration for non-standard cases. Over 260 questions, issues, bug reports, and feature requests have been made (as of April 2022). Given the questions and responses from the community are indexed by Google and other search engines and are publicly searchable, the resulting online discussions are helpful to distill and disseminate good SA practice. The benefits of an active community are highlighted by the

fact that the questions raised are often not addressed in the literature, helping to bridge the chasm between theory and implementation that is essential for the uptake of SA methods. From our experience, engaging with practitioners has led to (and is leading to) improvements in accessibility of methods and improved interpretability of SA results (cf. Niet et al., 2021; Noacco et al., 2019; Pianosi et al., 2020). Here, we posit that leveraging open development processes to actively foster a community of practice facilitates greater, and more meaningful, adoption of SA practices.

In summary, the SALib community of practice demonstrates that the software has achieved its implicit goal of removing barriers to the technical implementation of SA approaches so that users can focus on the broader conceptual issues, such as experimental design and interpretation of results. Our experience shows that this community is invaluable for improving the implementations and documentation of SA approaches, identifying novel issues and corner cases that can drive research into new implementations, and improving the interpretation of results and good SA practice.

4. Outlook

A stated aim of SALib (see Section 2) is to lower the barrier to entry so that a wide audience can make use of SA methods. The changes made since the initial release of SALib in 2017 are summarized in Section 3. However, there are opportunities for further improvements to better serve the needs of SA practitioners. These have implications not only for SALib but also for developers of similar software packages.

While the underlying theory and methodologies of SA are widely applicable across application contexts (Razavi et al., 2021), there is seemingly a wide chasm (cf. Kelly, 2007) between those focused on advancing SA methodologies and those who apply SA in practice. Past investigations into applications of SA have raised concerns over erroneous or questionable usage of SA. One aspect is the consideration of the analyses' experimental design such as the assumed parameter distributions and how these may influence results from SA which in turn influence the conclusions reached (Palaeri and Confalonieri, 2016). Still, relatively little attention or consideration appears to be given toward bridging this chasm beyond urging greater awareness of the limitations of commonly applied SA practices (e.g., Saltelli et al., 2019). As a result, SA is said to be applied unevenly across the disciplines due to a lack of uniformity on what is considered good practice or appropriate application (Razavi and Gupta, 2015; Saltelli et al., 2019).

Historically, tooling to support and encourage these best practices is often a secondary consideration, perhaps due to a lack of formal incentives within the current culture of academe (cf. Hannay et al., 2009; Iwanaga et al., 2021; Razavi et al., 2021). Where such tooling is considered, development is arguably tightly controlled through licensing or through workflows not wholly amenable to the open and collaborative practices through which interdisciplinary advances are usually made. In the recent past however, an increasing number of SA researchers have become involved in writing and releasing code to disseminate their work, or otherwise engage with existing SA frameworks (e.g., Baroni and Francke, 2020; Cuntz and Mai, 2020; Pianosi et al., 2020; Puy et al., 2021).

Perhaps more important than the availability of tooling, however, is the availability and strength of community. While accessibility can be partly improved through technical means (such as implementing user-friendly interfaces), interpretability is often subjective and context dependent, and requires engagement with users to address. If modelers are reluctant to adopt and apply Global Sensitivity Analysis (as explored in Saltelli et al., 2019; Saltelli and Annoni, 2010) then it is a social issue for which no amount of technical tooling by themselves will resolve.

The benefits of a culture of open and inclusive development are described in Section 3.4, including the ability to incorporate feedback, suggestions as well as contributions from practitioners. The collaborative spirit of such a culture benefits the SA community at large as it allows knowledge and experience to be shared and even incorporated into the software itself. Newcomers especially benefit from this development model as best-or-good practices become further "baked-in" and included as a standard process within the software itself. At the very least such practices become contextualized by the (community developed) documentation, rather than relying on practices borne from experience which often go undocumented and unpublished, or otherwise obscured by the volume of available literature. Questions raised around how best to assess appropriateness or effectiveness of experimental design (cf. Palaeri and Confalonieri, 2016; Pianosi et al., 2016), for example, could

be better served through such documentation. Community engagement is particularly important in resolving concerns over the “perfunctory” nature of current SA: in this case, fostering culture is likely more important than a directed strategy.

Documentation has evolved over the years in response to community feedback (detailed in Section 3.3), although much work remains to be done. With the v1.4 series, specific methods now include detailed “notes” sections highlighting potential pitfalls, considerations, and include references to further literature. Aspects somewhat lacking in SALib currently are the explicit inclusion of a conceptual framework to assess experimental design as well as a guide to the interpretation of results (similar to those provided in Noacco et al., 2019). Expanding this contextualization to all methods offered in SALib likely requires a broader network of SA practitioners – one that spans overlapping open-source communities – due to the specialized knowledge associated with each method. Collating such knowledge in an accessible way is likely instrumental to overcoming perceived deficiencies in SA applications.

On this note, SALib is but one node within a broader SA software community. There are, of course, other software packages available to support SA across a wide range of contexts (see for example Marelli and Sudret, 2014; Pianosi et al., 2020; Puy et al., 2021; Razavi et al., 2019). There is significant potential to collaborate across the SA software, modeling frameworks and decision support platforms regardless of the programming languages they are implemented in, leading to improved outcomes across the disciplines. In particular, the computational sciences are increasingly interdisciplinary, and the software developed and produced is one vehicle through which discussions and knowledge transfer occurs. Specific sessions in GSA-relevant conferences on software implementations and associated concerns, issues and experiences could facilitate such cross-disciplinary collaboration. Educational initiatives, similar to the Software Carpentry (Wilson, 2006) and CodeRefinery (<https://coderefinery.org>) programs, that prepare the next generation of SA practitioners to better collaborate and engage in this context are also needed. A larger cultural change, and associated funding, is necessary for this to occur however (cf. Downey, 2017; Little et al., 2019).

In the meantime, we encourage SA researchers to develop, foster and engage with these communities. Promoting the open implementations of new SA methods and sharing knowledge and experience across these networks can aid in reducing duplication of work, and crucially, open the possibility toward many research questions across the fields that leverage SA. One proffered incentive to academics is that involvement and contribution to an open platform (such as SALib) is likely to increase visibility and uptake of their own work. Citations for a specific method may also be boosted as the method and accompanying documentation is made accessible to a wider audience over time (cf. Niyazov et al., 2016; Teplitskiy et al., 2017). Adopting open development practices, such as hosting code on a public version-controlled repository (e.g., GitHub) invites collaboration and improvements to be made by a global community. The last is perhaps one incentive for those developing new SA methodologies. Aside from engagement by individual researchers, there is a need for increased support and funding for these development and engagement activities, the importance of which is often not wholly appreciated in the academic context.

Beyond accessibility and understandability, it is intended that new and emerging SA approaches be targeted for inclusion, particularly if they prove capable of more efficient sampling and analyses. These include variogram-based SA methods (e.g., Razavi et al., 2019); meta-methods which combine multiple approaches (e.g., variance and distribution-based metrics; Baroni and Francke, 2020); incorporating adaptive sampling approaches (e.g., Steiner et al., 2019); and further expansion of moment-independent, adaptive, and iterative analysis approaches (Cuntz et al., 2015; Sheikholeslami et al., 2021). Improved accessibility and interpretability unlock opportunities for new research and applications that may have previously been unviable.

5. Conclusion

Since the release of v1.0, the use of the SALib package for GSA has grown rapidly, supported by state-of-the-art practices and tools for testing, distribution, and community engagement. Key goals for the upcoming v2.0 software release focus on improving the usability and accessibility of SA, namely: supporting multiple programming paradigms; reducing time-to-results; improving documentation, especially around the experimental design and interpretation of results; and continuing to foster an active community on GitHub in support of the above. The contributors’ experience highlights the potential for decentralized scientific software

development across research groups and practitioners, which sometimes runs counter to the academic incentive structure. More broadly, it points to the role of software as a bridge between theory and practice, and the importance of usability, accessibility and design. We welcome new contributors as we work toward v2.0 and look forward to maintaining the community and continuing to promote the role of SA in scientific modeling.

Acknowledgements

The authors would additionally like to acknowledge and thank all those who have used SALib and the many who have contributed to its development over the years. A non-exhaustive list of contributors can be found in: <https://github.com/SALib/SALib/graphs/contributors>.

References

- Aznar-Siguan, G., & Bresch, D. N. (2019). CLIMADA v1: A global weather and climate risk assessment platform. *Geoscientific Model Development*, 12(7), 3085–3097. <https://doi.org/10.5194/gmd-12-3085-2019>
- Baroni, G., & Francke, T. (2020). An effective strategy for combining variance- and distribution-based global sensitivity analysis. *Environmental Modelling & Software*, 134, 104851. <https://doi.org/10.1016/j.envsoft.2020.104851>
- Borgonovo, E. (2007). A new uncertainty importance measure. *Reliability Engineering & System Safety*, 92(6), 771–784. <https://doi.org/10.1016/j.ress.2006.04.015>
- Bresch, D. N., & Aznar-Siguan, G. (2021). CLIMADA v1.4.1: Towards a globally consistent adaptation options appraisal tool. *Geoscientific Model Development*, 14(1), 351–363. <https://doi.org/10.5194/gmd-14-351-2021>
- Campolongo, F., Cariboni, J., & Saltelli, A. (2007). An effective screening design for sensitivity analysis of large models. *Environmental Modelling & Software*, 22(10), 1509–1518. <https://doi.org/10.1016/j.envsoft.2006.10.004>
- Conda-Forge Community. (2015). The conda-forge Project: Community-based Software Distribution Built on the conda Package Format and Ecosystem. <https://doi.org/10.5281/ZENODO.4774216>
- Cukier, R. I., Fortuin, C. M., Shuler, K. E., Petschek, A. G., & Schaibly, J. H. (1973). Study of the sensitivity of coupled reaction systems to uncertainties in rate coefficients. I Theory. *The Journal of Chemical Physics*, 59(8), 3873–3878. <https://doi.org/10.1063/1.1680571>
- Cuntz, M., & Mai, J. (2020). pyeee: Parameter screening using Morris' method or its extension of Efficient/Sequential Elementary Effects (2.0) [Computer software]. Zenodo. <https://doi.org/10.5281/ZENODO.3897550>
- Cuntz, M., Mai, J., Zink, M., Thober, S., Kumar, R., Schäfer, D., Schrön, M., Craven, J., Rakovec, O., Spieler, D., Prykhodko, V., Dalmasso, G., Musuuza, J., Langenberg, B., Attinger, S., & Samaniego, L. (2015). Computationally inexpensive identification of noninformative model parameters by sequential screening. *Water Resources Research*, 51(8), 6417–6441. <https://doi.org/10.1002/2015WR016907>
- Douglas-Smith, D., Iwanaga, T., Croke, B. F. W., & Jakeman, A. J. (2020). Certain trends in uncertainty and sensitivity analysis: An overview of software tools and techniques. *Environmental Modelling & Software*, 124, 104588. <https://doi.org/10.1016/j.envsoft.2019.104588>
- Downey, A. (2017). *Modeling and Simulation in Python*. Green Tea Press. <https://github.com/AllenDowney/ModSimPy> (Original work published 2016)
- Ferretti, F., Saltelli, A., & Tarantola, S. (2016). Trends in sensitivity analysis practice in the last decade. *Science of The Total Environment*, 568, 666–670. <https://doi.org/10.1016/j.scitotenv.2016.02.133>
- Fonseca, J., Thomas, D., Mok, R., Mosteiro-Romero, M., Happle, G., Rogenhofer, L., Jack-Hawthorne, Fazel Khayatian, Zhongming Shi, Riegelbauer, E., Ong, B. L., Orenkiwi, H. T., Paulneitzel, Sulzer, M., Molony, R., Elesawy, A., JOSE ANTONIO BELLO ACOSTA, Bosova, A., ... Strusoftsawen. (2021). *architecture-building-systems/CityEnergyAnalyst: CityEnergyAnalyst v3.22.0 (v3.22.0)* [Computer software]. Zenodo. <https://doi.org/10.5281/ZENODO.4646225>
- Foramitti, J. (2021). *JoelForamitti/agentpy* [Python]. <https://github.com/JoelForamitti/agentpy> (Original work published 2020)
- Foster, I. (1995). *Designing and building parallel programs: Concepts and tools for parallel software engineering*. Addison-Wesley.
- Hadjimichael, A., Gold, D., Hadka, D., & Reed, P. (2020). Rhodium: Python Library for Many-Objective Robust Decision Making and Exploratory Modeling. *Journal of Open Research Software*, 8(1), 12. <https://doi.org/10.5334/jors.293>

- Hannay, J. E., MacLeod, C., Singer, J., Langtangen, H. P., Pfahl, D., & Wilson, G. (2009). How do scientists develop and use scientific software? 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering, 1–8. <https://doi.org/10.1109/SECSE.2009.5069155>
- Herlihy, M., & Shavit, N. (2012). *The Art of Multiprocessor Programming*. Elsevier Science. http://www.123library.org/book_details/?id=53644
- Herman, J., & Usher, W. (2017, January 10). SALib: An open-source Python library for Sensitivity Analysis. *The Journal of Open Source Software*. <https://doi.org/10.21105/joss.00097>
- Hermans, F. (2021). *The Programmer's Brain: What every programmer needs to know about cognition*. Manning Publications.
- Iwanaga, T. (2021). ConnectedSystems/SALib-impact: V0.5. Zenodo. <https://doi.org/10.5281/zenodo.5523624>
- Iwanaga, T., Wang, H.-H., Hamilton, S. H., Grimm, V., Koralewski, T. E., Salado, A., Elsayah, S., Razavi, S., Yang, J., Glynn, P., Badham, J., Voinov, A., Chen, M., Grant, W. E., Peterson, T. R., Frank, K., Shenk, G., Barton, C. M., Jakeman, A. J., & Little, J. C. (2021). Socio-technical scales in socio-environmental modeling: Managing a system-of-systems modeling approach. *Environmental Modelling & Software*, 104885. <https://doi.org/10.1016/j.envsoft.2020.104885>
- Kelly, D. F. (2007). A Software Chasm: Software Engineering and Scientific Computing. *IEEE Software*, 24(6), 120–119. <https://doi.org/10.1109/MS.2007.155>
- Kwakkel, J. H. (2017). The Exploratory Modeling Workbench: An open source toolkit for exploratory modeling, scenario discovery, and (multi-objective) robust decision making. *Environmental Modelling & Software*, 96, 239–250. <https://doi.org/10.1016/j.envsoft.2017.06.054>
- Li, G., Rabitz, H., Yelvington, P. E., Oluwole, O. O., Bacon, F., Kolb, C. E., & Schoendorf, J. (2010). Global Sensitivity Analysis for Systems with Independent and/or Correlated Inputs. *The Journal of Physical Chemistry A*, 114(19), 6022–6032. <https://doi.org/10.1021/jp9096919>
- Little, J. C., Hester, E. T., Elsayah, S., Filz, G. M., Sandu, A., Carey, C. C., Iwanaga, T., & Jakeman, A. J. (2019). A tiered, system-of-systems modeling framework for resolving complex socio-environmental policy issues. *Environmental Modelling & Software*, 112, 82–94. <https://doi.org/10.1016/j.envsoft.2018.11.011>
- Marelli, S., & Sudret, B. (2014). UQLab: A Framework for Uncertainty Quantification in Matlab. *Vulnerability, Uncertainty, and Risk*, 2554–2563. <https://doi.org/10.1061/9780784413609.257>
- McKerns, M. M., Strand, L., Sullivan, T., Fang, A., & Aivazis, M. A. G. (2012). Building a Framework for Predictive Science. *ArXiv:1202.1056 [Cs]*. <http://arxiv.org/abs/1202.1056>
- Morris, M. D. (1991). Factorial Sampling Plans for Preliminary Computational Experiments. *Technometrics*, 33(2), 161–174. <https://doi.org/10.1080/00401706.1991.10484804>
- Niet, T., Shivakumar, A., Gardumi, F., Usher, W., Williams, E., & Howells, M. (2021). Developing a community of practice around an open source energy modelling tool. *Energy Strategy Reviews*, 35, 100650. <https://doi.org/10.1016/j.esr.2021.100650>
- Niyazov, Y., Vogel, C., Price, R., Lund, B., Judd, D., Akil, A., Mortonson, M., Schwartzman, J., & Shron, M. (2016). Open Access Meets Discoverability: Citations to Articles Posted to Academia.edu. *PLOS ONE*, 11(2), e0148257. <https://doi.org/10.1371/journal.pone.0148257>
- Noacco, V., Sarrazin, F., Pianosi, F., & Wagener, T. (2019). Matlab/R workflows to assess critical choices in Global Sensitivity Analysis using the SAFE toolbox. *MethodsX*, 6, 2258–2280. <https://doi.org/10.1016/j.mex.2019.09.033>
- Paleari, L., & Confalonieri, R. (2016). Sensitivity analysis of a sensitivity analysis: We are likely overlooking the impact of distributional assumptions. *Ecological Modelling*, 340, 57–63. <https://doi.org/10.1016/j.ecolmodel.2016.09.008>
- Pianosi, F., Beven, K., Freer, J., Hall, J. W., Rougier, J., Stephenson, D. B., & Wagener, T. (2016). Sensitivity analysis of environmental models: A systematic review with practical workflow. *Environmental Modelling & Software*, 79, 214–232. <https://doi.org/10.1016/j.envsoft.2016.02.008>
- Pianosi, F., Sarrazin, F., & Wagener, T. (2020). How successfully is open-source research software adopted? Results and implications of surveying the users of a sensitivity analysis toolbox. *Environmental Modelling & Software*, 124, 104579. <https://doi.org/10.1016/j.envsoft.2019.104579>
- Pianosi, F., & Wagener, T. (2015). A simple and efficient method for global sensitivity analysis based on cumulative distribution functions. *Environmental Modelling & Software*, 67, 1–11. <https://doi.org/10.1016/j.envsoft.2015.01.004>
- Pianosi, F., & Wagener, T. (2018). Distribution-based sensitivity analysis from a generic input-output sample. *Environmental Modelling & Software*, 108, 197–207. <https://doi.org/10.1016/j.envsoft.2018.07.019>

- Plischke, E. (2010). An effective algorithm for computing global sensitivity indices (EASI). *Reliability Engineering & System Safety*, 95(4), 354–360. <https://doi.org/10.1016/j.res.2009.11.005>
- Plischke, E., Borgonovo, E., & Smith, C. L. (2013). Global sensitivity measures from given data. *European Journal of Operational Research*, 226(3), 536–550. <https://doi.org/10.1016/j.ejor.2012.11.047>
- Puy, A., Piano, S. L., Saltelli, A., & Levin, S. A. (2021). sensobol: An R package to compute variance-based sensitivity indices. *ArXiv Preprint ArXiv:2101.10103*. <https://arxiv.org/abs/2101.10103>
- Rabitz, H., Aliş, Ö. F., Shorter, J., & Shim, K. (1999). Efficient input–Output model representations. *Computer Physics Communications*, 117(1), 11–20. [https://doi.org/10.1016/S0010-4655\(98\)00152-0](https://doi.org/10.1016/S0010-4655(98)00152-0)
- Razavi, S., & Gupta, H. V. (2015). What do we mean by sensitivity analysis? The need for comprehensive characterization of “global” sensitivity in Earth and Environmental systems models. *Water Resources Research*, 51(5), 3070–3092. <https://doi.org/10.1002/2014WR016527>
- Razavi, S., Jakeman, A. J., Saltelli, A., Prieur, C., Iooss, B., Borgonovo, E., Plischke, E., Lo Piano, S., Iwanaga, T., Becker, W., Tarantola, S., Guillaume, J. H. A., Jakeman, J., Gupta, H., Melillo, N., Rabitti, G., Chabridon, V., Duan, Q., Sun, X., ... Maier, H. R. (2021). The Future of Sensitivity Analysis: An Essential Discipline for Systems Modeling and Policy Support. *Environmental Modelling & Software*, 137, 104954. <https://doi.org/10.1016/j.envsoft.2020.104954>
- Razavi, S., Sheikholeslami, R., Gupta, H. V., & Haghnegahdar, A. (2019). VARS-TOOL: A toolbox for comprehensive, efficient, and robust sensitivity and uncertainty analysis. *Environmental Modelling & Software*, 112, 95–107. <https://doi.org/10.1016/j.envsoft.2018.10.005>
- Ruano, M. V., Ribes, J., Seco, A., & Ferrer, J. (2012). An improved sampling strategy based on trajectory design for application of the Morris method to systems with many input factors. *Environmental Modelling & Software*, 37, 103–109. <https://doi.org/10.1016/j.envsoft.2012.03.008>
- Saltelli, A. (2002). Making best use of model evaluations to compute sensitivity indices. *Computer Physics Communications*, 145, 280–297. [https://doi.org/10.1016/S0010-4655\(02\)00280-1](https://doi.org/10.1016/S0010-4655(02)00280-1)
- Saltelli, A., Aleksankina, K., Becker, W., Fennell, P., Ferretti, F., Holst, N., Li, S., & Wu, Q. (2019). Why So Many Published Sensitivity Analyses Are False: A Systematic Review of Sensitivity Analysis Practices. *Environmental Modelling & Software*, 114, 29–39.
- Saltelli, A., & Annoni, P. (2010). How to avoid a perfunctory sensitivity analysis. *Environmental Modelling and Software*, 25(12), 1508–1517. <https://doi.org/10.1016/j.envsoft.2010.04.012>
- Saltelli, A., Annoni, P., Azzini, I., Campolongo, F., Ratto, M., & Tarantola, S. (2010). Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index. *Computer Physics Communications*, 181(2), 259–270. <https://doi.org/10.1016/j.cpc.2009.09.018>
- Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., & Tarantola, S. (2008). *Global Sensitivity Analysis: The Primer*. Wiley. <https://dx.doi.org/10.1002/9780470725184>
- Saltelli, A., Tarantola, S., & Chan, K. P.-S. (1999). A Quantitative Model-Independent Method for Global Sensitivity Analysis of Model Output. *Technometrics*, 41(1), 39–56. <https://doi.org/10.1080/00401706.1999.10485594>
- Sheikholeslami, R., Gharari, S., Papalexioiu, S. M., & Clark, M. P. (2021). VISCOUS: A Variance-Based Sensitivity Analysis Using Copulas for Efficient Identification of Dominant Hydrological Processes. *Water Resources Research*, 57(7), e2020WR028435. <https://doi.org/10.1029/2020WR028435>
- Sobol', I. M. (2001). Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates. *Mathematics and Computers in Simulation*, 55(1–3), 271–280. [https://doi.org/10.1016/S0378-4754\(00\)00270-6](https://doi.org/10.1016/S0378-4754(00)00270-6)
- Sobol', I. M., & Kucherenko, S. (2010). Derivative based global sensitivity measures. *Procedia - Social and Behavioral Sciences*, 2(6), 7745–7746. <https://doi.org/10.1016/j.sbspro.2010.05.208>
- Steiner, M., Bourinet, J.-M., & Lahmer, T. (2019). An adaptive sampling method for global sensitivity analysis based on least-squares support vector regression. *Reliability Engineering & System Safety*, 183, 323–340. <https://doi.org/10.1016/j.res.2018.11.015>
- Sweller, J. (1988). Cognitive Load During Problem Solving: Effects on Learning. *Cognitive Science*, 12(2), 257–285. https://doi.org/10.1207/s15516709cog1202_4
- Tarantola, S., Gatelli, D., & Mara, T. A. (2006). Random balance designs for the estimation of first order global sensitivity indices. *Reliability Engineering & System Safety*, 91(6), 717–727. <https://doi.org/10.1016/j.res.2005.06.003>
- Teplitskiy, M., Lu, G., & Duede, E. (2017). Amplifying the impact of open access: Wikipedia and the diffusion of science. *Journal of the Association for Information Science and Technology*, 68(9), 2116–2127. <https://doi.org/10.1002/asi.23687>

- Tissot, J.-Y., & Prieur, C. (2012). Bias correction for the estimation of sensitivity indices based on random balance designs. *Reliability Engineering & System Safety*, 107, 205–213. <https://doi.org/10.1016/j.res.2012.06.010>
- Wagener, T., & Pianosi, F. (2019). What has Global Sensitivity Analysis ever done for us? A systematic review to support scientific advancement and to inform policy-making in earth system modelling. *Earth-Science Reviews*, 194, 1–18. <https://doi.org/10.1016/j.earscirev.2019.04.006>
- Wilson, G. (2006). Software Carpentry: Getting Scientists to Write Better Code by Making Them More Productive. *Computing in Science Engineering*, 8(6), 66–69. <https://doi.org/10.1109/MCSE.2006.122>