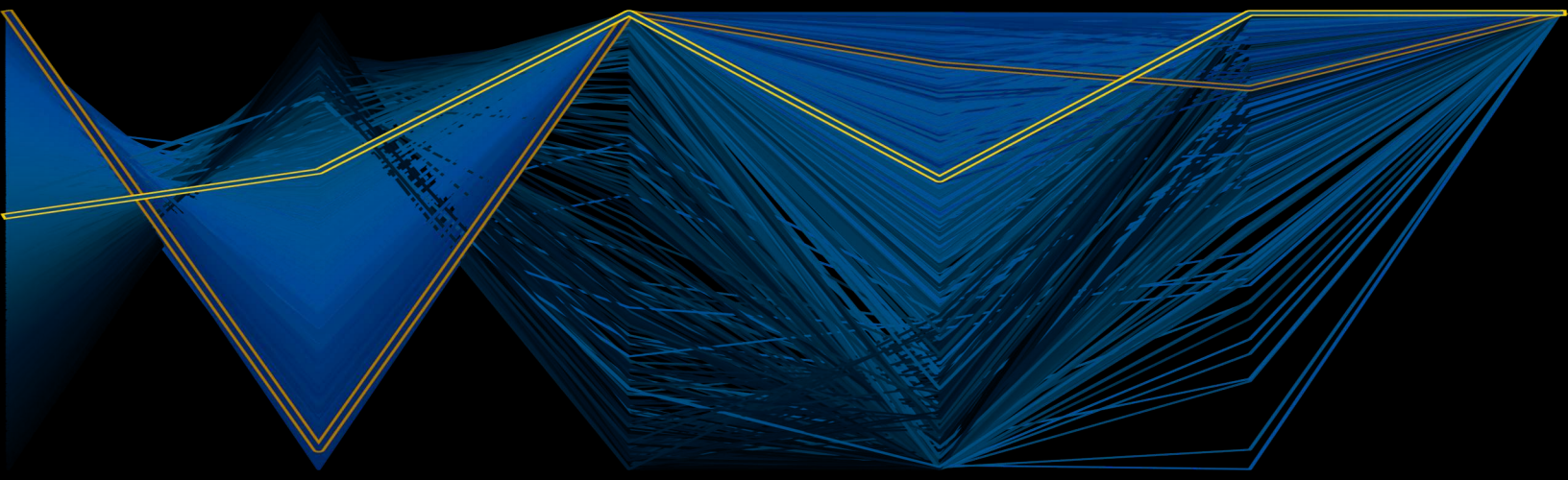
Addressing Uncertainty *in* MultiSector Dynamics Research

Patrick M. Reed, Antonia Hadjimichael, Keyvan Malek
Tina Karimi, Chris R. Vernon, Vivek Srikrishnan, Rohini S. Gupta,
David F. Gold, Ben Lee, Klaus Keller, Travis B. Thurber, Jennie S. Rice

This e-book was developed by the Integrated Multisector, Multiscale Modeling (IM3) project, supported by the U.S. Department of Energy, Office of Science, as part of research in the MultiSector Dynamics, Earth and Environmental System Modeling Program.



Addressing Uncertainty in MultiSector Dynamics Research

Patrick M. Reed, Antonia Hadjimichael, Keyvan Malek
Tina Karimi, Chris R. Vernon, Vivek Srikrishnan, Rohini S. Gupta
David F. Gold, Ben Lee, Klaus Keller, Travis B. Thurber, Jennie S. Rice

Last updated: Sep 24, 2025

PREFACE

This online book is meant to provide an open science "living" resource on uncertainty characterization methods for the MultiSector Dynamics (MSD) community and other technical communities confronting sustainability, climate, and energy transition challenges. The last decade has seen rapid growth in science efforts seeking to address the interconnected nature of these challenges across scales, sectors, and systems. Accompanying these advances is the growing realization that the deep integration of research from many disciplinary fields is non-trivial and raises important questions. How and why models are developed seems to have an obvious answer ("to gain understanding"). But what does it actually mean to gain understanding? What if a small change in a model or its data fundamentally changes our perceptions of what we thought we understood? What controls the outcomes of our model(s)? How do we understand the implications of model coupling, such as when one model is on the receiving end of several other models that are considered "input data"?

The often quoted "All models are wrong, but some are useful." (George Box) is a bit of a conflation trap, often used to excuse known weaknesses in complex models as just an unavoidable outcome of being a modeler. In fact, the quote actually refers to a specific class of small-scale statistical models within an application context that assures a much higher degree of understanding and data quality control than is typical for the coupled human-natural systems applications in the MSD area. Moreover, Box was actually warning readers to avoid overparameterization and emphasizing the need to better understand what underlying factors cause your model to be wrong [1].

So, in short, there is a tension when attaining better performance by means of increasing the complexity of a model or model-based workflow. Box highlights that a modeler requires a clear diagnostic understanding of this performance-complexity tradeoff. If we move from small-scale models simulating readily-observed phenomena to the MSD context, things get quite a bit more complicated. How can we provide robust insights for unseen futures that emerge across a myriad of human and natural systems? Sometimes even asking, "what is a model?" or "what is data?" is complicated (e.g., data assimilated weather products, satellite-based signals translated through retrieval algorithms, demographic changes, resource demands, etc.). This MSD guidance text seeks to help readers navigate these challenges. It is meant to serve as an evolving resource that helps the MSD community learn how to better address uncertainty while working with complex chains of models bridging sectors, scales, and systems. It is not intended to be an exhaustive resource, but instead should be seen as a guided tour through state-of-the-science methods in uncertainty characterization, including global sensitivity analysis and exploratory modeling, to provide insights into complex human-natural systems interactions.

To aid readers in navigating the text, the key goals for each chapter are summarized below.

Chapter 1 uses the Integrated Multisector Multiscale Modeling project as a living lab to encapsulate the challenges that emerge in bridging disciplines to make consequential model-based insights while acknowledging the tremendous array of uncertainties that shape them.

Chapter 2 helps the reader to better understand the importance of using diagnostic modeling to interrogate why uncertain model behaviors may emerge. The chapter also aids readers to better understand the diverse disciplinary perspectives that exist on how best to pursue consequential model-based discoveries.

Chapter 3 is a technical tools-focused primer for readers on the key elements of uncertainty characterization that includes ensemble-based design of experiments, quantitative methods for computing global sensitivities, and a summary of existing software packages.

Chapter 4 narrates for readers how and why the tools from the previous chapter can be applied in a range of tasks from diagnosing model performance to formal exploratory modeling methods for making consequential model-based discoveries.

The supplemental appendices provided in the text are also important resources for readers. They provide a glossary to help bridge terminology challenges, a brief summary of uncertainty quantification tools for more advanced readers, and a suite of Jupyter notebook tutorials that provide hands-on training tied to the contents of Chapter 3 and Chapter 4.

This text was written with a number of different audiences in mind.

Technical experts in uncertainty may find this to be a helpful and unique resource bridging a number of perspectives that have not been combined in prior books (e.g., formal model diagnostics, global sensitivity analysis, and exploratory modeling under deep uncertainty).

Readers from different sector-specific and disciplinary-specific backgrounds can use this text to better understand potential differences and opportunities in how to make model-based insights.

Academic or junior researchers can utilize this freely available text for training and teaching resources that include hands-on coding experiences.

This text itself represents our strong commitment to open science and will evolve as a living resource as the communities of researchers provide feedback, innovations, and future tools.

SUGGESTED CITATION

Reed, P.M., Hadjimichael, A., Malek, K., Karimi, T., Vernon, C.R., Srikrishnan, V., Gupta, R.S., Gold, D.F., Lee, B., Keller, K., Thurber, T.B, & Rice, J.S. (2022). Addressing Uncertainty in Multisector Dynamics Research [Book]. Zenodo. https://doi.org/10.5281/zenodo.6110623

0.1 Supplemental Appendices Citations

Srikrishnan, V. (2025). Model Calibration with Markov Chain Monte Carlo Tutorial (v1.0.0). MSD-LIVE Data Repository. https://doi.org/10.57931/2565322

ACKNOWLEDGEMENTS

This e-book was developed by the Integrated Multisector, Multiscale Modeling (IM3) project, supported by the U.S. Department of Energy, Office of Science, as part of research in the MultiSector Dynamics, Earth and Environmental System Modeling Program.

The authors would like to thank Casey Burleyson (PNNL) and Isaac Thompson (PNNL) for their insightful feedback on drafts of the text and tutorials.

0.2 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

0.3 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

0.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

0.5 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

Our eBook is a living product that we hope continues to grow over time to stay relevant with methodological and technological advancements in the field of uncertainty quantification and MultiSector Dynamics at large. We extend an invitation to you, the reader, to contribute to our interactive Jupyter notebook tutorials. If you feel you have a contribution that would be relevant and generalizable to the MSD community, please submit a proposal idea here. Your proposal will be reviewed by our team, and feedback and/or a decision will be provided shortly. Any accepted contribution will receive it's own DOI and citation so that you may provide an independent reference to your work as you see fit.

Please consider the following requirements for contribution:

- All elements of your contribution MUST be fully open-source and can be distributed with an Open Source Initiative approved license with an understanding that they may be used in community demonstrations and other activities. Author citations will be present in the notebook for any contributed work to ensure the author(s) receive full credit for their contribution.
- Any data or code reused in your submission must be correctly cited giving credit to the original authors.
- The notebook provided must be written in English and able to be a stand-alone product that needs no further explanation past what is written in the notebook to make use of it.
- The provided work is not merely a regurgitation of an existing tutorial or demonstration but represents a novel contribution.
- All contributions and communication thereof must abide by our code of conduct.

If you feel your work meets the criteria above, please submit a proposal issue here. If your proposal is approved, please create a pull request with the submission template copied into the pull request description and filled out. We will then review your pull request and provide feedback. Once your contribution has been deemed ready to deploy, we will generate a DOI for your work, launch it to our MSD-LIVE set of interactive notebooks, and feature the contribution in the index of our eBook.

Please feel free to reach out with any further questions.

0.6 Development workflow

The following is the recommended workflow for contributing:

1. Fork the msd_uncertainty_ebook repository and then clone it locally:

   ```
   git clone https://github.com/<your-user-name>/msd_uncertainty_ebook
   ```

2. Set up the repository for development:

   Make sure that you are using an **msd_uncertainty_ebook** compatible python version. It is important to install the package in development mode. This will give you the flexibility to make changes in the code without having to rebuild the package:

   ```
   cd msd_uncertainty_ebook

   pip install -e ".[dev]"
   ```

   Install `pre-commit`, a code format checker, in the repo:

```
pre-commit install
```

3. Add your changes and commit them:

```
git add <my-file-name>

git commit -m "<my short message>"
```

4. Ensure all tests pass:

   Ensure your tests pass locally before pushing to your remote branch where GitHub actions will launch CI services to build the package, run the test suite, and evaluate code coverage. To do this, ensure that `pytest` is installed then navigate to the root of your cloned directory (e.g., <my-path>/msd_uncertainty_ebook) and run `pytest` in the terminal.

```
pip install -e ".[dev]"

pytest
```

5. Update the Documentation:

   Changes to the documentation can be made in the `msd_uncertainty_ebook/docs/source` directory containing the RST files. To view your changes, ensure you have the documentation dependencies of **msd_uncertainty_ebook** installed and run the following from the `msd_uncertainty_ebook/docs/source` directory:

```
pip install -e ".[docs]"

make html
```

   This will generate your new documentation in a directory named `msd_uncertainty_ebook/docs/build/html`. You can open the `index.html` in your browser to view the documentation site locally. If your changes are merged into the main branch of **msd_uncertainty_ebook**, changes in your documentation will go live on the uc-ebook.org documentation. If your changes are merged into the dev branch of **msd_uncertainty_ebook**, changes in your documentation will go live on the dev site.

6. Push your changes to the remote

```
git push origin <my-branch-name>
```

7. Submit a pull request with the submission template copied into the pull request description and filled out.
8. If approved, your pull request will be merged first into the dev, and then into the main branch by a **msd_uncertainty_ebook** admin and a release will be conducted subsequently. **msd_uncertainty_ebook** uses semantic naming for versioned releases. Each release receives a DOI via a linked Zenodo service automatically.

0.7 Miscellaneous Developer Notes

- Dockerfile for the uc-ebook site.
- Dockerfile for the uc-ebook dev site.

CONTENTS

CHAPTER

ONE

INTRODUCTION

This guidance text has been developed in support of the Integrated Multisector Multiscale Modeling (IM3) Science Focus Area's objective to formally integrate uncertainty into its research tasks. IM3 is focused on innovative modeling to explore how human and natural system landscapes in the United States co-evolve in response to short-term shocks and long-term influences. The project's challenging scope is to advance our ability to study the interactions between energy, water, land, and urban systems, at scales ranging from local (~1km) to the contiguous United States, while consistently addressing influences such as population change, technology change, heat waves, and drought. Uncertainty and careful model-driven scientific insights are central to the project's science objectives shown below.

IM3 Key MSD Science Objectives:

Develop flexible, open-source, and integrated modeling capabilities that capture the structure, dynamic behavior, and emergent properties of the multiscale interactions within and between human and natural systems.

Use these capabilities to study the evolution, vulnerability, and resilience of interacting human and natural systems and landscapes from local to continental scales, including their responses to the compounding effects of long-term influences and short-term shocks.

Understand the implications of uncertainty in data, observations, models, and model coupling approaches for projections of human-natural system dynamics.

Addressing the objectives above poses a strong transdisciplinary challenge that depends on a diversity of models and, more specifically, a consistent framing for making model-based science inferences. The term transdisciplinary science as used here formally implies a deep integration of disciplines to aid our hypothesis-driven understanding of coupled human-natural systems–bridging differences in theory, hypothesis generation, modeling, and modes of inference [2]. The IM3 MSD research foci and questions require a deep integration across disciplines, where new modes of analysis can emerge that rapidly synthesize and exploit advances for making decision-relevant insights that at minimum acknowledge uncertainty and more ideally promote a rigorous quantitative mapping of its effects on the generality of claimed scientific insights. More broadly, diverse scientific disciplines engaged in the science of coupled human-natural systems, ranging from natural sciences to engineering and economics, employ a diversity of numerical computer models to study and understand their underlying systems of focus. The utility of these computer models hinges on their ability to represent the underlying real systems with sufficient fidelity and enable the inference of novel insights. This is particularly challenging in the case of coupled human-natural systems where there exists a multitude of interdependent human and natural processes taking place that could potentially be represented. These processes usually translate into modeled representations that are highly complex, non-linear, and exhibit strong interactions and threshold behaviors [3, 4, 5]. Model complexity and detail have also been increasing as a result of our improving understanding of these processes, the availability of data, and the rapid growth in computing power [6]. As model complexity grows, modelers need to specify a lot more information than before: additional model inputs and relationships as more processes are represented, higher resolution data as more observations are collected, new coupling relationships and interactions as diverse models are being used in combination to answer multisector questions (e.g., the land-water-energy nexus). Typically, not all of this information is well known, nor is the impact of these many uncertainties on model outputs well understood. It is further especially difficult to distinguish the effects of individual as well as interacting sources of uncertainty when modeling coupled systems with multisector and multiscale dynamics [7].

Given the challenge and opportunity posed by the disciplinary diversity of IM3, we utilized an informal team-wide

survey to understand how the various disciplines typically address uncertainty, emphasizing key literature examples and domain-specific reviews. The feedback received provided perspectives across diverse areas within the Earth sciences, different engineering fields, as well as economics. Although our synthesis of this survey information highlighted some commonality across areas (e.g., the frequent use of scenario-based modeling), we identified key differences in vocabulary, the frequency with which formal uncertainty analysis appears in the disciplinary literature, and technical approaches. The IM3 team's responses captured a very broad conceptual continuum of methodological traditions, ranging from deterministic (no uncertainty) modeling to the theoretical case of fully engaging in modeling sources of uncertainty. Overall, error-driven analyses that focus on replicating prior observed conditions were reported to be the most prevalent types of studies for all disciplines. It was generally less common for studies to strongly engage with analyzing uncertainty via more formal ensemble analyses and design of experiments, though some areas did show significantly higher levels of activity. Another notable finding from our survey was the apparent lack of focus on understanding how model coupling relationships shape uncertainty. Although these observations are limited to the scope of feedback attained in the team-wide IM3 survey responses and the bodies of literature reported by respondents, we believe they reflect challenges that are common across the MSD community.

In the IM3 uncertainty-related research that has occurred since this survey, we have observed that differences in terminology and interpretation of terminology across modeling teams can be confounding. One of the goals of this eBook is to provide a common language for uncertainty analysis within IM3 and, hopefully, for the broader MSD community. While individual scientific disciplines would be expected to retain their own terminology, by providing explicit definitions of terms we can facilitate the translation of concepts across transdisciplinary science teams. To begin, we use the term Uncertainty Analysis (UA) as an umbrella phrase covering all methods in this eBook. Next, we distinguish the key terms of uncertainty quantification (UQ) and uncertainty characterization (UC). UQ refers to the formal focus on the full specification of likelihoods as well as the distributional forms necessary to infer the joint probabilistic response across all modeled factors of interest [8]. UC refers to exploratory modeling of alternative hypotheses to understand the co-evolutionary dynamics of influences and stressors, as well as path dependent changes in the form and function of modelled systems [9, 10]. As discussed in later sections, the choice of UC or UQ depends on the specific goals of studies, the availability of data, the types of uncertainties (e.g., well-characterized or deep), and the complexity of underlying models as well as computational limits. Definitions of key uncertainty analysis terms used in this eBook appear below, and our Glossary (Chapter C.2.2) contains a complete list of terms.

- **Exploratory modeling**: Use of large ensembles of uncertain conditions to discover decision-relevant combinations of uncertain factors
- **Factor**: Any model component that can affect model outputs: inputs, resolution levels, coupling relationships, model relationships and parameters. In models with acceptable model fidelity these factors may represent elements of the real-world system under study.
- **Sensitivity analysis: Model evaluation to understand the factors and processes that most (or least) control a model's outputs**
  - **Local sensitivity analysis**: Varying uncertain factors around specific reference values
  - **Global sensitivity analysis**: Varying uncertain factors throughout their entire feasible value space
- **Uncertainty characterization**: Model evaluation under alternative factor hypotheses to explore their implications for model output uncertainty
- **Uncertainty quantification**: Representation of model output uncertainty using probability distributions

At present, there is no singular guide for confronting the computational and conceptual challenges of the multi-model, transdisciplinary workflows that characterize ambitious projects such as IM3 [11]. The primary aim of this text is to begin to address this gap and provide guidance for facing these challenges. Chapter 2 provides an overview of diagnostic modeling and the different perspectives for how we should evaluate our models, Chapter 3 summarizes basic methods and concepts for sensitivity analysis, and Chapter 4 delves into more technical applications of sensitivity analysis to support diagnostic model evaluation and exploratory modeling. Finally, Chapter 5 provides some concluding remarks across the UC and UQ topics covered in this text. The appendices of this text include a glossary of the key concepts, an overview of UQ methods, and coding-based illustrative examples of key UC concepts discussed in earlier chapters.

CHAPTER

TWO

DIAGNOSTIC MODELING OVERVIEW AND PERSPECTIVES

This text prescribes a formal model diagnostic approach that is a deliberative and iterative combination of state-of-the-art UC and global sensitivity analysis techniques that progresses from observed history-based fidelity evaluations to forward looking resilience and vulnerability inferences [12, 13].

2.1 Overview of model diagnostics

Model diagnostics provide a rich basis for hypothesis testing, model innovation, and improved inferences when classifying what is controlling highly consequential results (e.g., vulnerability or resilience in coupled human-natural systems). Fig. 2.1, adapted from [6], presents idealized illustrations of the relationship between UC and global sensitivity analysis for two coupled simulation models. The figure illustrates how UC can be used to address how uncertainties in various modeling decisions (e.g., data inputs, parameters, model structures, coupling relationships) can be sampled and simulated to yield the empirical model output distribution(s) of interest. Monte Carlo frameworks allow us to sample and propagate (or integrate) the ensemble response of the model(s) of focus. The first step of any UC analysis is the specification of the initial input distributions as illustrated in Fig. 2.1. The second step is to perform the Monte Carlo simulations. The question can then be raised, which of the modeling assumptions in our Monte Carlo experiment are the most responsible for the resulting output uncertainty. We can answer this question using global sensitivity analysis as illustrated in Fig. 2.1. Global sensitivity analysis can be defined as a formal Monte Carlo sampling and analysis of modeling choices (structures, parameters, inputs) to quantify their influence on direct model outputs (or output-informed metrics). UC experiments by themselves do not explain why a particular uncertain outcome is produced, but produce distributions of model outcomes, as portrayed by the yellow curve. The pie chart shown in Fig. 2.1 is a conceptual representation of the results of using a global sensitivity analysis to identify those factors that are most dominantly influencing results, either individually or interactively [14].

UC and global sensitivity analysis are not independent modeling analyses. As illustrated here, any global sensitivity analysis requires an initial UC hypothesis in the form of statistical assumptions and representations for the modeling choices of focus (structural, parametric, and data inputs). Information from these two model diagnostic tools can then be used to inform data needs for future model runs, experiments to reduce the uncertainty present, or the simplification or enhancement of the model where necessary. Together UC and global sensitivity analysis provide a foundation for diagnostic exploratory modeling that has a consistent focus on the assumptions, structural model forms, alternative parameterizations, and input data sets that are used to characterize the behavioral space of one or more models.

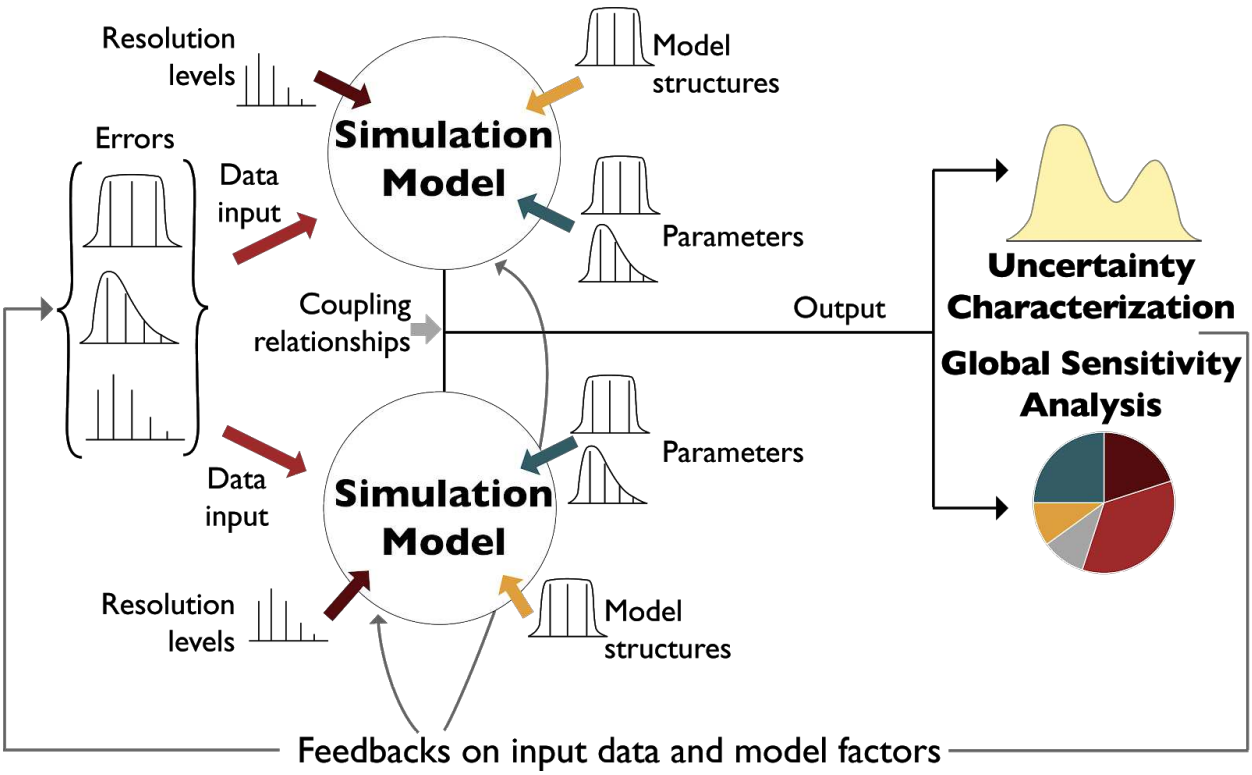


Fig. 2.1: Idealized uncertainty characterization and global sensitivity analysis for two coupled simulation models. Uncertainty coming from various sources (e.g., inputs, model structures, coupling relationships) is propagated through the coupled model(s) to generate empirical distributions of outputs of interest (uncertainty characterization). This model output uncertainty can be decomposed to its origins, by means of sensitivity analysis. Figure adapted from Saltelli *et al.* [6].

2.2 Perspectives on diagnostic model evaluation

When we judge or diagnose models, the terms "verification" and "validation" are commonly used. However, their appropriateness in the context of numerical models representing complex coupled human-natural systems is questionable [15, 16]. The core issue relates to the fact that these systems are often not fully known or perfectly implemented when modeled. Rather, they are defined within specific system framings and boundary conditions in an evolving learning process with the goal of making continual progress towards attaining higher levels of fidelity in capturing behaviors or properties of interest. Evaluating the fidelity of a model's performance can be highly challenging. For example, the observations used to evaluate the fidelity of parameterized processes are often measured at a finer resolution than what is represented in the model, creating the challenge of how to manage their relative scales when performing evaluation. In other cases, numerical models may neglect or simplify system processes because sufficient data is not available or the physical mechanisms are not fully known. If sufficient agreement between prediction and observation is not achieved, it is challenging to know whether these types of modeling choices are the cause, or if other issues, such as deficiencies in the input parameters and/or other modeling assumptions are the true cause of errors. Even if there is high agreement between prediction and observation, the model cannot necessarily be considered validated, as it is always possible that the right values were produced for the wrong reasons. For example, low error can stem from a situation where different errors in underlying assumptions or parameters cancel each other out ("compensatory errors"). Furthermore, coupled human-natural system models are often subject to "equifinality", a situation where multiple parameterized formulations can produce similar outputs or equally acceptable representations of the observed data. There is therefore no uniquely "true" or validated model, and the common practice of selecting "the best" deterministic calibration set is more of an assumption than a finding [17, 18]. The situation becomes even more tenuous when observational data is limited in its scope and/or quality to be insufficient to distinguish model representations or their performance differences.

These limitations on model verification undermine any purely positivist treatment of model validity: that a model should correctly and precisely represent reality to be valid. Under this perspective, closely related to empiricism, statistical tests should be used to compare the model's output with observations and only through empirical verification can a model or theory be deemed credible. A criticism to this viewpoint (besides the aforementioned challenges for model verification) is that it reduces the justification of a model to the single criterion of predictive ability and accuracy [19]. Authors have argued that this ignores the explanatory power held in models and other procedures, which can also advance scientific knowledge [20]. These views gave rise to relativist perspectives of science, which instead place more value on model utility in terms of fitness for a specific purpose or inquiry, rather than representational accuracy and predictive ability [21]. This viewpoint appears to be most prevalent among practitioners seeking decision-relevant insights (i.e., inspire new views vs. predict future conditions). The relativist perspective argues for the use of models as heuristics that can enhance our understanding and conceptions of system behaviors or possibilities [22]. In contrast, natural sciences favor a positivist perspective, emphasizing similarity between simulation and observation even in application contexts where it is clear that projections are being made for conditions that have never been observed and the system of focus will have evolved structurally beyond the model representation being employed (e.g., decadal to centennial evolution of human-natural systems).

These differences in prevalent perspectives are mirrored in how model validation is defined by the two camps: From the relativist perspective, validation is seen as a process of incremental "confidence building" in a model as a mechanism for insight [23], whereas in natural sciences validation is framed as a way to classify a model as having an acceptable representation of physical reality [16]. Even though the relativist viewpoint does not dismiss the importance of representational accuracy, it does place it within a larger process of establishing confidence through a variety of tools. These tools, not necessarily quantitative, include communicating information between practitioners and modelers, interpreting a multitude of model outputs, and contrasting preferences and viewpoints.

On the technical side of the argument, differing views on the methodology of model validation appear as early as in the 1960's. Naylor and Finger [24] argue that model validation should not be limited to a single metric or test of performance (e.g., a single error metric), but should rather be extended to multiple tests that reflect different aspects of a model's structure and behavior. This and similar arguments are made in literature to this day [12, 25, 26, 27, 28] and are primarily founded on two premises. First, that even though modelers widely recognize that their models are abstractions of the truth, they still make truth claims based on traditional performance metrics that measure the divergence of their model from observation [28]. Second, that the natural systems mimicked by the models contain many processes that exhibit significant heterogeneity at various temporal and spatial scales. This heterogeneity is

lost when a single performance measure is used, as a result of the inherent loss of process information occurring when transitioning from a highly dimensional and interactive system to the dimension of a single metric [15]. These arguments are further elaborated in Chapter 4.

Multiple authors have proposed that the traditional reliance on single measures of model performance should be replaced by the evaluation of several model signatures (characteristics) to identify model structural errors and achieve a sufficient assessment of model performance [12, 29, 30, 31]. There is however a point of departure here, especially when models are used to produce inferences that can inform decisions. When agencies and practitioners use models of their systems for public decisions, those models have already met sufficient conditions for credibility (e.g., acceptable representational fidelity), but may face broader tests on their salience and legitimacy in informing negotiated decisions [22, 32, 33]. This presents a new challenge to model validation, that of selecting decision-relevant performance metrics, reflective of the system's stakeholders' viewpoints, so that the most consequential uncertainties are identified and addressed [34]. For complex multisector models at the intersection of climatic, hydrologic, agricultural, energy, or other processes, the output space is made up of a multitude of states and variables, with very different levels of salience to the system's stakeholders and to their goals being achieved [35]. This is further complicated when such systems are also institutionally and dynamically complex. As a result, a broader set of qualitative and quantitative performance metrics is necessary to evaluate models of such complex systems, one that embraces the plurality of value systems, agencies and perspectives present. For IM3, even though the goal is to develop better projections of future vulnerability and resilience in co-evolving human-natural systems and not to provide decision support per se, it is critical for our multisector, multiscale model evaluation processes to represent stakeholders' adaptive decision processes credibly.

As a final point, when a model is used in a projection mode, its results are also subject to additional uncertainty, as there is no guarantee that the model's functionality and predictive ability will stay the same as the baseline, where the verification and validation tests were conducted. This challenge requires an additional expansion of the scope of model evaluation: a broader set of uncertain conditions needs to be explored, spanning beyond historical observation and exploring a wide range of unprecedented conditions. This perspective on modeling, termed exploratory [36], views models as computational experiments that can be used to explore vast ensembles of potential scenarios to identify those with consequential effects. Exploratory modeling literature explicitly orients experiments toward stakeholder consequences and decision-relevant inferences and shifts the focus from predicting future conditions to *discovering* which conditions lead to undesirable or desirable consequences.

This evolution in modeling perspectives can be mirrored by the IM3 family of models in a progression from evaluating models relative to observed history to advanced formalized analyses to make inferences on multisector, multiscale vulnerabilities and resilience. Exploratory modeling approaches can help fashion experiments with large numbers of alternative hypotheses on the co-evolutionary dynamics of influences, stressors, as well as path-dependent changes in the form and function of human-natural systems [37]. The aim of this text is to therefore guide the reader through the use of sensitivity analysis (SA) methods across these perspectives on diagnostic and exploratory modeling.

Note: The following articles are suggested as fundamental reading for the information presented in this section:

- Naomi Oreskes, Kristin Shrader–Frechette, and Kenneth Belitz. Verification, Validation, and Confirmation of Numerical Models in the Earth Sciences. *Science*, 263 (5147): 641-646, February 1994. URL: https://science.sciencemag.org/content/263/5147/641. DOI: https://doi.org/10.1126/science.263.5147.641.
- Keith Beven. Towards a coherent philosophy for modelling the environment. *Proceedings of the Royal Society of London*. Series A: mathematical, physical and engineering sciences, 458 (2026): 2465-2484, 2002.
- Eker, S., Rovenskaya, E., Obersteiner, M., Langan, S., 2018. Practice and perspectives in the validation of resource management models. *Nature Communications* 9, 1–10. https://doi.org/10.1038/s41467-018-07811-9

The following articles can be used as supplemental reading:

- Canham, C.D., Cole, J.J., Lauenroth, W.K. (Eds.), 2003. Models in Ecosystem Science. Princeton University Press. https://doi.org/10.2307/j.ctv1dwq0tq

---

CHAPTER

THREE

SENSITIVITY ANALYSIS: THE BASICS

3.1 Global Versus Local Sensitivity

Out of the several definitions for sensitivity analysis presented in the literature, the most widely used has been proposed by Saltelli *et al.* [38] as "the study of how uncertainty in the output of a model (numerical or otherwise) can be apportioned to different sources of uncertainty in the model input". In other words, sensitivity analysis explores the relationship between the model's N input variables, $x = [x_1, x_2, ..., x_N]$, and M output variables, $y = [y_1, y_2, ..., y_M]$ with $y = g(x)$, where g is the model that maps the model inputs to the outputs [39].

Historically, there have been two broad categories of sensitivity analysis techniques: local and global. Local sensitivity analysis is performed by varying model parameters around specific reference values, with the goal of exploring how small input perturbations influence model performance. Due to its ease-of-use and limited computational demands, this approach has been widely used in literature, but has important limitations [40, 41]. If the model is not linear, the results of local sensitivity analysis can be heavily biased, as they are strongly influenced by independence assumptions and a limited exploration of model inputs (e.g., Tang *et al.* [42]). If the model's factors interact, local sensitivity analysis will underestimate their importance, as it does not account for those effects (e.g., [43]). In general, as local sensitivity analysis only partially and locally explores a model's parametric space, it is not considered a valid approach for nonlinear models [44]. This is illustrated in Fig. 3.1 (a-b), presenting contour plots of a model response (y) with an additive linear model (a) and with a nonlinear model (b). In a linear model without interactions between the input terms x_1 and x_2, local sensitivity analysis (assuming deviations from some reference values) can produce appropriate sensitivity indices (Fig. 3.1 (a)). If however, factors x_1 and x_2 interact, the local and partial consideration of the space can not properly account for each factor's effects on the model response (Fig. 3.1 (b)), as it is only informative at the reference value where it is applied. In contrast, a global sensitivity analysis varies uncertain factors within the entire feasible space of variable model responses (Fig. 3.1 (c)). This approach reveals the global effects of each parameter on the model output, including any interactive effects. For models that cannot be proven linear, global sensitivity analysis is preferred and this text is primarily discussing global sensitivity analysis methods. In the text that follows, whenever we use the term sensitivity analysis we are referring to its global application.

3.2 Why Perform Sensitivity Analysis

It is important to understand the many ways in which a SA might be of use to your modeling effort. Most commonly, one might be motivated to perform sensitivity analysis for the following reasons:

Model evaluation: Sensitivity analysis can be used to gauge model inferences when assumptions about the structure of the model or its parameterization are dubious or have changed. For instance, consider a numerical model that uses a set of calibrated parameter values to produce outputs, which we then use to inform decisions about the real-world system represented. One might like to know if small changes in these parameter values significantly change this model's output and the decisions it informs or if, instead, our parameter inferences yield stable model behavior regardless of the uncertainty present in the specific parameterized processes or properties. This can either discredit or lend credence

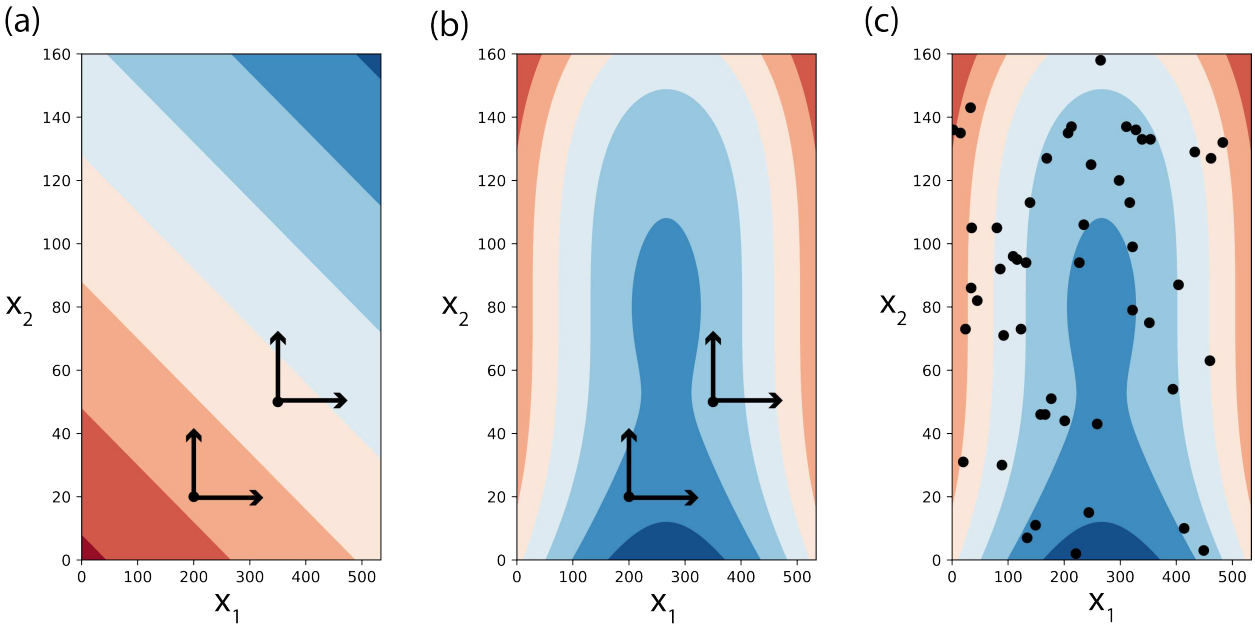


Fig. 3.1: Treatment of a two-dimensional space of variability by local (panels a-b) and global (panel c) sensitivity analyses. Panels depict contour plots with the value of a model response (y) changing with changes in the values of input terms x_1 and x_2. Local sensitivity analysis is only an appropriate approach to sensitivity in the case of linear models without interactions between terms, for example in panel (a), where $y = 3x_1 + 5x_2$. In the case of more complex models, for example in panels (b-c), where $y = \frac{1}{e^{x_1^2+x_2^2}} + \frac{50}{e^{(0.1x_1)^2+(0.1x_2)^3}}$, local sensitivity will miscalculate sensitivity indices as the assessed changes in the value y depend on the assumed base values chose for x_1 and x_2 (panel (b)). In these cases, global sensitivity methods should be used instead (panel (c)). The points in panel (c) are generated using a uniform random sample of $n = 50$, but many other methods are available.

to the model at hand, as well as any inferences drawn that are founded on its accurate representation of the system. Sensitivity analysis can identify which uncertain model factors cause this undesirable model behavior.

Model simplification: Sensitivity analysis can also be used to identify factors or components of the model that appear to have limited effects on direct outputs or metrics of interest. Consider a model that has been developed in an organization for the purposes of a specific research question and is later used in the context of a different application. Some processes represented in significant detail might no longer be of the same importance while consuming significant data or computational resources, as different outputs might be pertinent to the new application. Sensitivity analysis can be used to identify unimportant model components and simplify them to nominal values and reduced model forms. Model complexity and computational costs can therefore be reduced.

Model refinement: Alternatively, sensitivity analysis can reveal the factors or processes that are highly influential to the outputs or metrics of interest, by assessing their relative importance. In the context of model evaluation, this can inform which model components warrant additional investigation or measurement so the uncertainty surrounding them and the resulting model outputs or metrics of interest can be reduced.

Exploratory modeling: When sufficient credence has been established in the model, sensitivity analysis can be applied to a host of other inquiries. Inferences about the factors and processes that most (or least) control a model's outputs of interest can be extrapolated to the real system they represent and be used in a heuristic manner to inform model-based inferences. On this foundation, a model paired with the advanced techniques presented in this text can be used to "discover" decision relevant and highly consequential outcomes (i.e., scenario discovery, discussed in more detail in Chapter 4.3 [36, 45]).

The nature and context of the model shapes the specific objectives of applying a sensitivity analysis, as well as methods and tools most appropriate and defensible for each application setting [35, 38, 46]. The three most common sensitivity analysis modes (*Factor Prioritization*, *Factor Fixing*, and *Factor Mapping*) are presented below, but the reader should be aware that other uses have been proposed in the literature (e.g., [47, 48]).

Factor prioritization: This sensitivity analysis application mode (also referred to as *factor ranking*) refers to when one would like to identify the uncertain factors that have the greatest impact on the variability of the output, and which, when fixed to their true value (i.e., if there were no uncertainty regarding their value), would lead to the greatest reduction in output variability [49]. Information from this type of analysis can be crucial to model improvement as these factors can become the focus of future measurement campaigns or numerical experiments so that uncertainty in the model output can be reduced. The impact of each uncertain input on the variance of the model output is often used as the criterion for factor prioritization. Fig. 3.2 (a) shows the effects of three uncertain variables (X_1, X_2, and X_3) on the variance of output Y. $V(E(Y|X_i))$ indicates the variance in Y if factor X_i is left to vary freely while all other factors remain fixed to nominal values. In this case, factor X_2 makes the largest contribution to the variability of output Y and it should therefore be prioritized. In the context of risk analysis, factor prioritization can be used to reduce output variance to below a given tolerable threshold (also known as variance cutting).

Factor fixing: This mode of sensitivity analysis (also referred to as *factor screening*) aims to identify the model components that have a negligible effect or make no significant contributions to the variability of the outputs or metrics of interest (usually referred to as non-influential [49]). In the stylized example of Fig. 3.2 (a), X_1 makes the smallest contribution to the variability of output Y suggesting that the uncertainty in its value could be negligible and the factor itself fixed in subsequent model executions. Eliminating these factors or processes in the model or fixing them to a nominal value can help reduce model complexity as well as the unnecessary computational burden of subsequent model runs, results processing, or other sensitivity analyses (the fewer uncertain factors considered, the fewer runs are necessary to illuminate their effects on the output). Significance of the outcome can be gauged in a variety of manners, depending on the application. For instance, if applying a variance-based method, a minimum threshold value of contribution to the variance could be considered as a significance 'cutoff', and factors with indices below that value can be considered non-influential. Conclusions about factor fixing should be made carefully, considering all of the effects a factor has, individually and in interaction with other factors (explained in more detail in the Chapter 3.4.5).

Factor mapping: Finally, factor mapping can be used to pinpoint which values of uncertain factors lead to model outputs within a given range of the output space [49]. In the context of model diagnostics, it is possible that the model's output changes in ways considered impossible based on the represented processes, or other observed evidence. In this situation, factor mapping can be used to identify which uncertain model factors cause this undesirable model

behavior by 'filtering' model runs that are considered 'non-behavioral' [50, 51, 52]. In Fig. 3.2 (b), region B of the output space Y denotes the set of behavioral model outcomes and region $\bar{B}$ denotes the set of non-behavioral outcomes, resulting from the entirety of input space X. Factor mapping refers to the process of tracing which factor values of input space X produce the behavioral model outcomes in the output space.

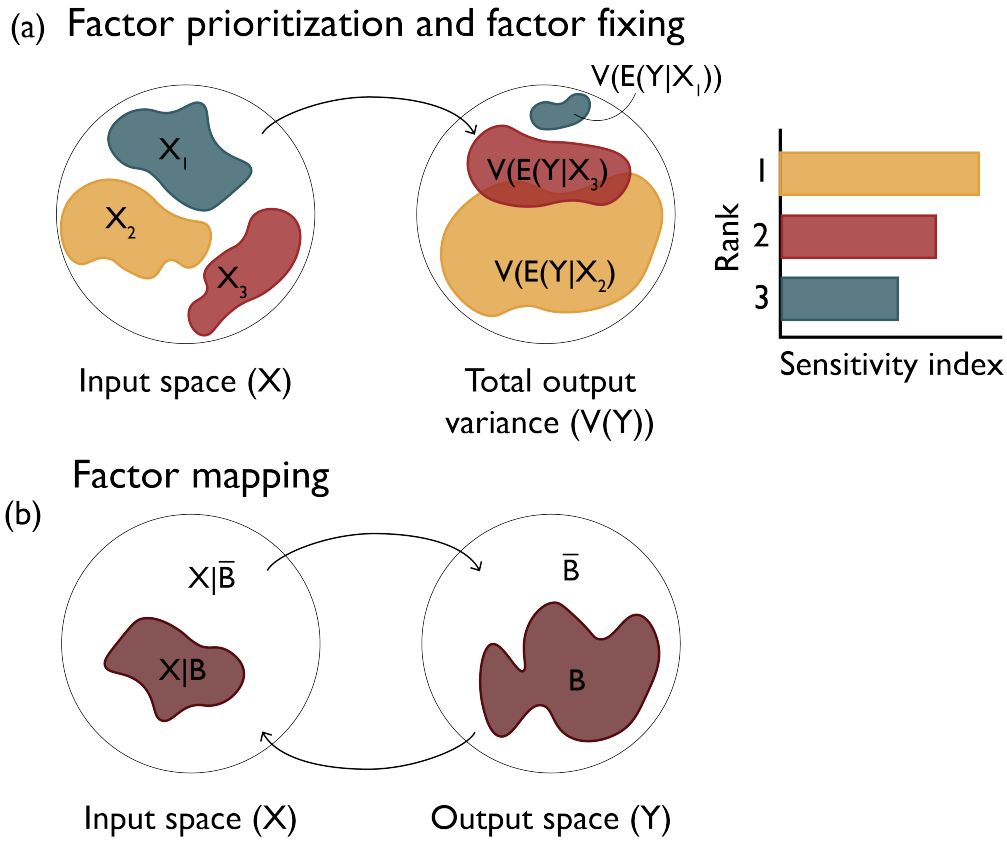


Fig. 3.2: Factor prioritization, factor fixing and factor mapping settings of sensitivity analysis.

The language used above reflects a use of sensitivity analysis for model fidelity evaluation and refinement. However, as previously mentioned, when a model has been established as a sufficiently accurate representation of the system, sensitivity analysis can produce additional inferences (i.e., exploratory modeling and scenario discovery). For instance, under the factor mapping use, the analyst can now focus on undesirable system states and discover which factors are most responsible for them: for instance, "population growth of above 25% would be responsible for unacceptably high energy demands". Factor prioritization and factor fixing can be used to make equivalent inferences, such as "growing populations and increasing temperatures are the leading factors for changing energy demands" (prioritizing of factors) or "changing dietary needs are inconsequential to increasing energy demands for this region" (a factor that can be fixed in subsequent model runs). All these inferences hinge on the assumption that the real system's stakeholders consider the model states faithful enough representations of system states. As elaborated in Chapter 2.2, this view on sensitivity analysis is founded on a relativist perspective on modeling, which tends to place more value on model usefulness rather than strict accuracy of representation in terms of error. As such, sensitivity analysis performed with decision-making relevance in mind will focus on model outputs or metrics that are consequential and decision relevant (e.g., energy demand in the examples above).

3.3 Design of Experiments

Before conducting a sensitivity analysis, the first element that needs to be clarified is the uncertainty space of the model [51, 53]. In other words, how many and which factors making up the mathematical model are considered uncertain and can potentially affect the model output and the inferences drawn from it. Uncertain factors can be model parameters, model structures, inputs, or alternative model resolution levels (scales), all of which can be assessed through the tools presented in this text. Depending on the kind of factor, its variability can be elicited through various means: expert opinion, values reported in the literature, historical observations, its physical meaning (e.g., population values in a city can never be negative), or through the use of more formal UQ methods (Chapter A). The model uncertainty space represents the entire space of variability present in each of the uncertain factors of a model. The complexity of most real-world models means that the response function, $y = g(x)$, mapping inputs to outputs, is hardly ever available in an analytical form and therefore analytically computing the sensitivity of the output to each uncertain factor becomes impossible. In these cases, sensitivity analysis is only feasible through numerical procedures that employ different strategies to sample the uncertainty space and calculate sensitivity indices.

A sampling strategy is often referred to as a *design of experiments* and represents a methodological choice made before conducting any sensitivity analysis. Experimental design was first introduced by Fisher [54] in the context of laboratory or field-based experiments. Its application in sensitivity analysis is similar to setting up a physical experiment in that it is used to discover the behavior of a system under specific conditions. An ideal design of experiments should provide a framework for the extraction of all plausible information about the impact of each factor on the output of the model. The design of experiments is used to set up a simulation platform with the minimum computational cost to answer specific questions that cannot be readily drawn from the data through analytical or common data mining techniques. Models representing coupled human-natural systems usually have a large number of inputs, state variables and parameters, but not all of them exert fundamental control over the numerical process, despite their uncertainty, nor have substantial impacts on the model output, either independently or through their interactions. Each factor influences the model output in different ways that need to be discovered. For example, the influence of a parameter on model output can be linear or non-linear and can be continuous or only be active during specific times or at particular states of the system [55, 56]. An effective and efficient design of experiments allows the analyst to explore these complex relationships and evaluate different behaviors of the model for various scientific questions [57]. The rest of this section overviews some of the most commonly used designs of experiments. Table 1 summarizes the designs discussed.

Table 3.1: Summary of designs of experiments overviewed in this section.
* Depends on the sample size.

Design of experiments	*Factor interactions considered*	*Treatment of factor domains*
One-At-a-Time (OAT)	No - main effects only	Continuous (distributions)
Full Factorial Sampling	Yes - including total effects	Discrete (levels)
Fractional Factorial Sampling	Yes - only lower-order effects*	Discrete (levels)
Latin Hypercube (LH) Sampling	Yes - including total effects*	Continuous (distributions)
Quasi-Random Sampling with Low-Discrepancy Sequences	Yes - including total effects*	Continuous (distributions)

There are a few different approaches to the design of experiments, closely related to the chosen sensitivity analysis approach, which is in turn shaped by the research motivations, scientific questions, and computational constraints at hand (additional discussion of this can be found at the end of Chapter 3). For example, in a sensitivity analysis using perturbation and derivatives methods, the model input parameters vary from their nominal values one at a time, something that the design of experiments needs to reflect. If, instead, one were to perform sensitivity analysis using a multiple-starts perturbation method, the design of experiments needs to consider that multiple points across the factor space are used. The design of experiments specifically defines two key characteristics of samples that are fed to the numerical model: the number of samples and the range of each factor.

Generally, sampling can be performed randomly or by applying a stratifying approach. In random sampling, such as

Monte Carlo [58], samples are randomly generated by a pseudo-random number generator with an a-priori assumption about the distribution of parameters and their possible ranges. Random seeds can also be used to ensure consistency and higher control over the random process. However, this method could leave some gaps in the parameter space and cause clustering in some spaces, especially for a large number of parameters [59]. Most sampling strategies use stratified sampling to mitigate these disadvantages. Stratified sampling techniques divide the domain of each factor into subintervals, often of equal lengths. From each subinterval, an equal number of samples is drawn randomly, or based on the specific locations within the subintervals [49].

3.3.1 One-At-a-Time (OAT)

In this approach, only one model factor is changed at a time while all others are kept fixed across each iteration in a sampling sequence. The OAT method assumes that model factors of focus are linearly independent (i.e., there are no interactions) and can analyze how factors individually influence model outputs or metrics of interest. While popular given its ease of implementation, OAT is ultimately limited in its exploration of a model's sensitivities [49]. It is primarily used with local sensitivity techniques with similar criticisms: applying this sampling scheme on a system with nonlinear and interactive processes will miss important information on the effect uncertain factors have on the model. OAT samplings can be repeated multiple times in a more sophisticated manner and across different locations of the parameter space to overcome some of these challenges, which would increase computational costs and negate the main reasons for its selection. Given these limitations OAT methods could be used as preliminary, low-cost analyses of the factors' individual effects, but should ultimately be complemented with more sophisticated methods.

3.3.2 Full and Fractional Factorial Sampling

In full factorial sampling each factor is treated as being discrete by considering two or more levels (or intervals) of its values. The sampling process then generates samples within each possible combination of levels, corresponding to each parameter. This scheme produces a more comprehensive sampling of the factors' variability space, as it accounts for all candidate combinations of factor levels (Fig. 3.3 (a)). If the number of levels is the same across all factors, the number of generated samples is estimated using n^k, where n is the number of levels and k is the number of factors. For example, Fig. 3.3 (a) presents a full factorial sampling of three uncertain factors (x_1, x_2, and x_3), each considered as having four discrete levels. The total number of samples necessary for such an experiment is $4^3 = 64$. As the number of factors increases, the number of simulations necessary will also grow exponentially, making full factorial sampling computationally burdensome (Fig. 3.3 (b)). As a result, it is common in the literature to apply full factorial sampling at only two levels per factor, typically the two extremes [60]. This significantly reduces computational burden but is only considered appropriate in cases where factors can indeed only assume two discrete values (e.g., when testing the effects of epistemic uncertainty and comparing between model structure A and model structure B). In the case of physical parameters on continuous distributions (e.g., when considering the effects of measurement uncertainty in a temperature sensor), discretizing the range of a factor to only extreme levels can bias its estimated importance.

Fractional factorial sampling is a widely used alternative to full factorial sampling that allows the analyst to significantly reduce the number of simulations by focusing on the main effects of a factor and seeking to avoid model runs that yield redundant response information [49]. In other words, if one can reasonably assume that higher-order interactions are negligible, information about the most significant effects and lower-order interactions (e.g., effects from pairs of factors) can be obtained using a fraction of the full factorial design. Traditionally, fractional factorial design has also been limited to two levels [60], referred to as Fractional Factorial designs 2k-p [61]. Recently, Generalized Fractional Factorial designs have also been proposed that allow for the structured generation of samples at more than two levels per factor [62]. Consider a case where the modeling team dealing with the problem in Fig. 3.3 (a) cannot afford to perform 64 simulations of their model. They can afford 32 runs for their experiment and instead decide to fractionally sample the variability space of their factors. A potential design of such a sampling strategy is presented in Fig. 3.3 (c).

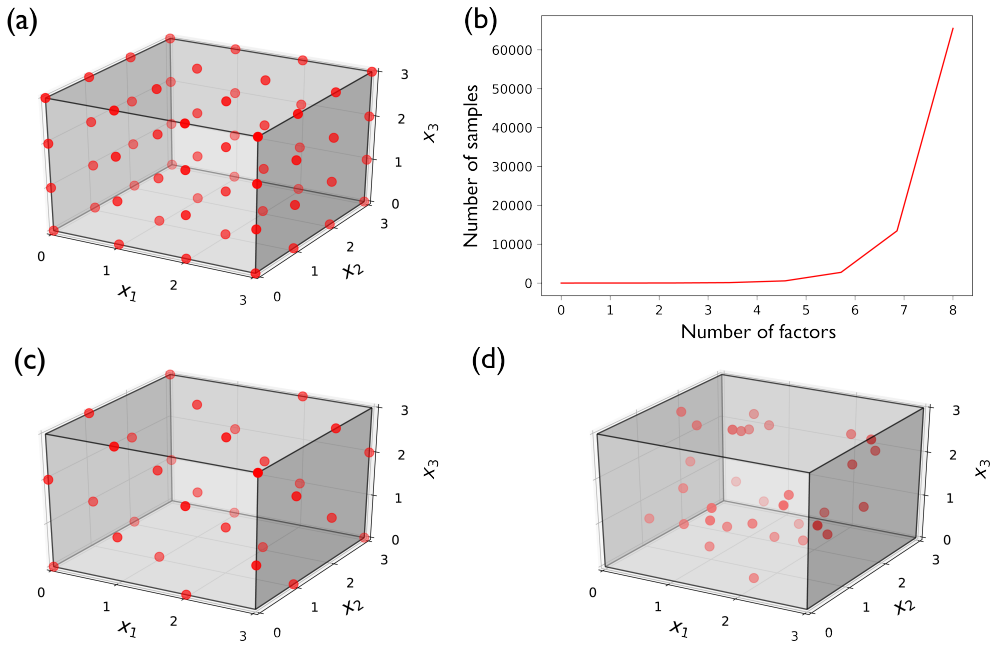


Fig. 3.3: Alternative designs of experiments and their computational costs for three uncertain factors (x_1, x_2, and x_3). (a) Full factorial design sampling of three factors at four levels, at a total of 64 samples; (b) exponential growth of necessary number of samples when applying full factorial design at four levels; (c) fractional factorial design of three factors at four levels, at a total of 32 samples; and (d) Latin Hypercube sample of three factors with uniform distributions, at a total of 32 samples.

3.3.3 Latin Hypercube Sampling (LHS)

Latin hypercube sampling (LHS) [63] is one of the most common methods in space-filling experimental designs. With this sampling technique, for N uncertain factors, an N-dimensional hypercube is generated, with each factor divided into an equal number of levels depending on the total number of samples to be generated. Equal numbers of samples are then randomly generated at each level, across all factors. In this manner, latin hypercube design guarantees sampling from every level of the variability space and without any overlaps. When the number of samples generated is much larger than the number of uncertain factors, LHS can be very effective in examining the effects of each factor [49]. LHS is an attractive technique, because it guarantees a diverse coverage of the space, through the use of subintervals, without being constrained to discrete levels for each factor - compare Fig. 3.3 (c) with Fig. 3.3 (d) for the same number of samples.

LHS is less effective when the number of samples is not much larger than the number of uncertain factors, and the effects of each factor cannot be appropriately distinguished. The samples between factors can also be highly correlated, biasing any subsequent sensitivity analysis results. To address this, the sampling scheme can be modified to control for the correlation in parameters while maximizing the information derived. An example of such modification is through the use of orthogonal arrays [64].

3.3.4 Low-Discrepancy Sequences

Low-discrepancy sequences is another sampling technique that employs a pseudo-random generator for Monte Carlo sampling [65, 66]. These quasi-Monte Carlo methods eliminate 'lumpiness' across samples (i.e, the presence of gaps and clusters) by minimizing discrepancy across the hypercube samples. Discrepancy can be quantitatively measured using the deviations of sampled points from a uniform distribution [65, 67]. Low-discrepancy sequences ensure that the number of samples in any subspace of the variability hypercube is approximately the same. This is not something guaranteed by Latin Hypercube sampling, and even though its design can be improved through optimization with various criteria, such adjustments are limited to small sample sizes and low dimensions [67, 68, 69, 70, 71]. In contrast, the Sobol sequence [72, 73], one of the most widely used sampling techniques, utilizes the low-discrepancy approach to uniformly fill the sampled factor space. A core advantage of this style of sampling is that it takes far fewer samples (i.e., simulations) to attain a much lower level of error in estimating model output statistics (e.g., the mean and variance of outputs).

Note: Put this into practice! Click the following link to try out an interactive tutorial which uses Sobol sequence sampling for the purposes of a Sobol sensitivity analysis: Sobol SA using SALib Jupyter Notebook

---

3.3.5 Other types of sampling

The sampling techniques mentioned so far are general sampling methods useful for a variety of applications beyond sensitivity analysis. There are however techniques that have been developed for specific sensitivity analysis methods. Examples of these methods include the Morris One-At-a-Time [74], Fourier Amplitude Sensitivity Test (FAST; [75]), Extended FAST [76], and Extended Sobol methods [77]. For example, the Morris sampling strategy builds a number of trajectories (usually referred to as repetitions and denoted by r) in the input space each composed of $N+1$ factor points, where N is the number of uncertain factors. The first point of the trajectory is selected randomly and the subsequent N points are generated by moving one factor at a time by a fixed amount. Each factor is perturbed once along the trajectory, while the starting points of all of the trajectories are randomly and uniformly distributed. Several variations of this strategy also exist in the literature; for more details on each approach and their differences the reader is directed to Pianosi *et al.* [51].

3.3.6 Synthetic generation of input time series

Models often have input time series or processes with strong temporal and/or spatial correlations (e.g., streamflow, energy demand, pricing of commodities, etc.) that, while they might not immediately come to mind as factors to be examined in sensitivity analysis, can be treated as such. Synthetic input time series are used for a variety of reasons, for example, when observations are not available or are limited, or when past observations are not considered sufficiently representative to capture rare or extreme events of interest [78, 79]. Synthetic generation of input time series provides a valuable tool to consider non-stationarity and incorporate potential stressors, such as climate change impacts into input time series [80]. For example, a century of record will be insufficient to capture very high impact rare extreme events (e.g., persistent multi-year droughts). A large body of statistical literature exists focusing on the topics of synthetic weather [81, 82] and streamflow [83, 84] generation that provides a rich suite of approaches for developing history-informed, well-characterized stochastic process models to better estimate rare individual or compound (hot, severe drought) extremes. It is beyond the scope of this text to review these methods, but readers are encouraged to explore the studies cited above as well as the following publications for discussions and comparisons of these methods: [78, 80, 85, 86, 87, 88, 89]. The use of these methods for the purposes of exploratory modeling, especially in the context of well-characterized versus deep uncertainty, is further discussed in Chapter 4.3.

3.4 Sensitivity Analysis Methods

In this section, we describe some of the most widely applied sensitivity analysis methods along with their mathematical definitions. We also provide a detailed discussion on applying each method, as well as a comparison of and their features and limitations.

3.4.1 Derivative-based Methods

Derivative-based methods explore how model outputs are affected by perturbations in a single model input around a particular input value. These methods are local and are performed using OAT sampling. For simplicity of mathematical notations, let us assume that the model $g(X)$ only returns one output. Following [90] and [51], the sensitivity index, S_i , of the model's *i*-th input factor, x_i , can be measured using the partial derivative evaluated at a nominal value, $\bar{x}$, of the vector of inputs:

$$S_i(\bar{x}) = \frac{\partial g}{\partial x}|_{\bar{x}^{c}{}_i}$$

where c_i is the scaling factor. In most applications however, the relationship $g(X)$ is not fully known in its analytical form, and therefore the above partial derivative is usually approximated:

$$S_i(\bar{x}) = \frac{g(\bar{x}_1, ...\bar{x}_i + \Delta_i, ...\bar{x}_N) - g(\bar{x}_1, ...\bar{x}_i, ...\bar{x}_N)}{\Delta_i} c_i$$

Using this approximation, the *i*-th input factor is perturbed by a magnitude of Δ_i, and its relative importance is calculated. Derivative-based methods are some of the oldest sensitivity analysis methods as they only require $N + 1$ model evaluations to estimate indices for N uncertain factors. As described above, being computationally very cheap comes at the cost of not being able to explore the entire input space, but only (local) perturbations to the nominal value. Additionally, as these methods examine the effects of each input factor one at a time, they cannot assess parametric interactions or capture the interacting nature of many real systems and the models that abstract them.

3.4.2 Elementary Effect Methods

Elementary effect (EE) SA methods provide a solution to the local nature of the derivative-based methods by exploring the entire parametric range of each input parameter [91]. However, EE methods still use OAT sampling and do not vary all input parameters simultaneously while exploring the parametric space. The OAT nature of EEs methods therefore prevents them from properly capturing the interactions between uncertain factors. EEs methods are computationally efficient compared to their All-At-a-Time (AAT) counterparts, making them more suitable when computational capacity is a limiting factor, while still allowing for some inferences regarding factor interactions. The most popular EE method is the Method of Morris [74]. Following the notation by [51], this method calculates global sensitivity using the mean of the EEs (finite differences) of each parameter at different locations:

$$S_i = \mu_i^* = \frac{1}{r}\sum_{j=1}^{r} EE_i^j = \frac{1}{r}\sum_{j=1}^{r} \frac{g(\bar{x}_1, ...\bar{x}_i + \Delta_i, ...\bar{x}_N) - g(\bar{x}_1, ...\bar{x}_i, ...\bar{x}_N)}{\Delta_i} c_i$$

with r representing the number of sample repetitions (also refered to as trajectories) in the input space, usually set between 4 and 10 [38]. Each x_j represents the points of each trajectory, with $j = 1, \ldots, r$, selected as described in the sampling strategy for this method, found above. This method also produces the standard deviation of the EEs:

$$\sigma_i = \sqrt{\frac{1}{r}\sum_{j=1}^{r}(EE_i^j - \frac{1}{r}\sum_{j=1}^{r} EE_i^j)^2}$$

which is a measure of parametric interactions. Higher values of σ_i suggest model responses at different levels of factor x_i are significantly different, which indicates considerable interactions between that and other uncertain factors. The

values of μ_i^* and σ_i for each factor allow us to draw several different conclusions, illustrated in Fig. 3.4, following the example by [91]. In this example, factors x_1, x_2, x_4, and x_5 can be said to have an influence on the model outputs, with x_1, x_4, and x_5 having some interactive or non-linear effects. Depending on the orders of magnitude of μ_i^* and σ_i one can indirectly deduce whether the factors have strong interactive effects, for example if a factor $\sigma_i << \mu_i^*$ then the relationship between that factor and the output can be assumed to be largely linear (note that this is still an OAT method and assumptions on factor interactions should be strongly caveated). Extensions of the Method of Morris have also been developed specifically for the purposes of factor fixing and explorations of parametric interactions (e.g., [48, 92, 93]).

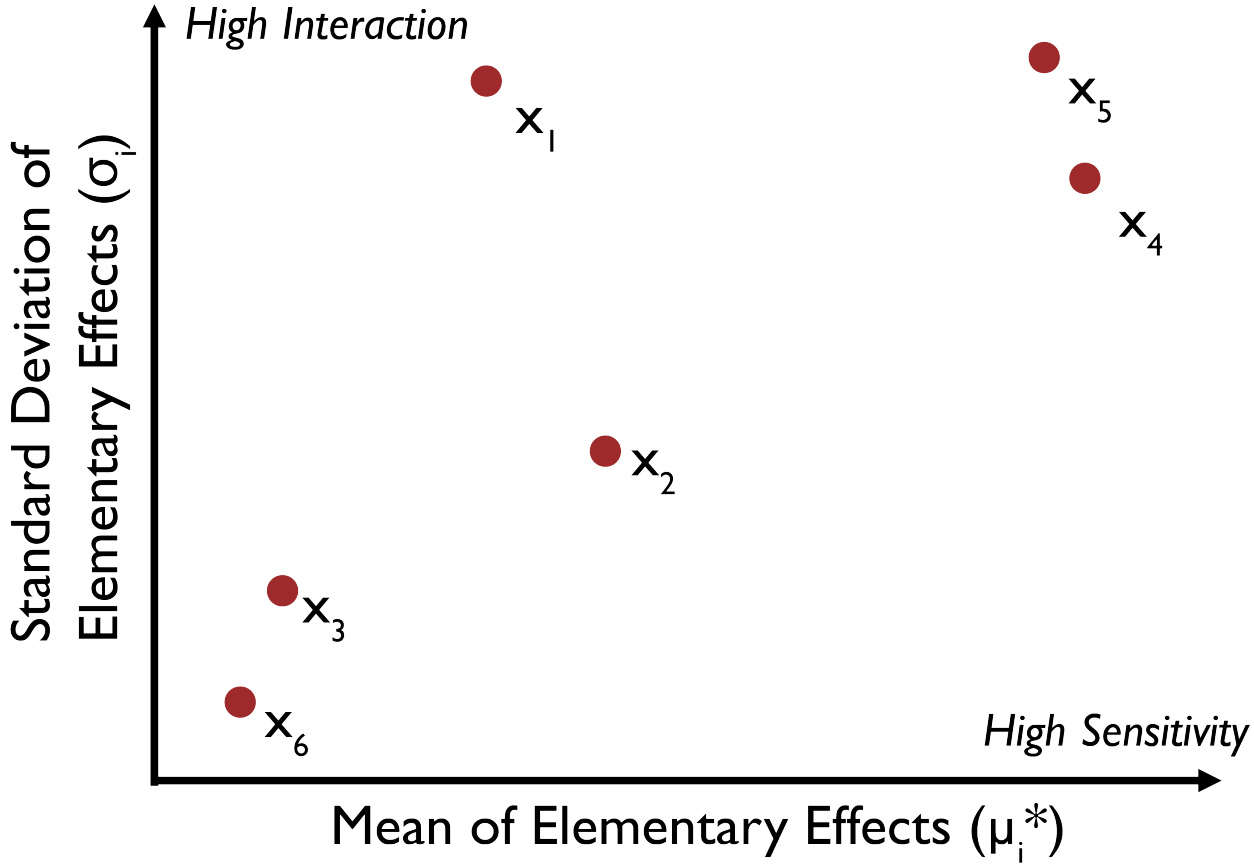


Fig. 3.4: Illustrative results of the Morris Method. Factors x_1, x_2, x_4, and x_5 have an influence on the model outputs, with x_1, x_4, and x_5 having interactive or non-linear effects. Whether or not a factor should be considered influential to the output depends on the output selected and is specific to the research context and purpose of the analysis, as discussed in Chapter 3.2.

3.4.3 Regression-based Methods

Regression analysis is one of the oldest ways of investigating parametric importance and sensitivity [38]. Here, we describe some of the most popular regression-based sensitivity indices. One of the main sensitivity indices of this category is the standardized regression coefficient (SRC). To calculate SRC, a linear regression relationship needs to be fitted between the input vector, x, and the model output of interest by using a least-square minimizing method:

$$y = b_0 + \sum_{i=1}^{N} b_i x_i$$

where b_0 and b_i (corresponding to the i-th model input) are regression coefficients. The following relationship can then be used to calculate the SRCs for different input values:

$$S_i = SRC_i = b_i \frac{\sigma_i}{\sigma_y}$$

where σ_i and σ_y are standard deviations of i-th model input and output, respectively.

Several other regression-based indices explore the correlation between input and output parameters as a proxy to model parametric sensitivity [91, 94, 95]. The Pearson correlation coefficient (PCC) can be used when a linear relationship exists between an uncertain factor, x_i, and the output y:

$$S_i = PCC = \frac{cov(x_i, y)}{\sigma_i \sigma_y}$$

In cases when there are outliers in the data or the relationship between the uncertain factors and the output is not linear, rank-based correlation coefficients are preferred, for example, Spearman's rank correlation coefficient (SRCC):

$$S_i = SRCC = \frac{cov(rx_i, ryi)}{\sigma_{ri} \sigma_{ry}}$$

where the raw values of x_i and y and converted to ranks rx_i and ry respectively, which instead represent a measurement of the strength of the monotonic relationship, rather than linear relationship, between the input and output. Other regression-based metrics include the partial correlations coefficient, the partial rank correlations coefficient, and the Nash-Sutcliffe coefficient, more discussion on which can be found in [39, 91].

Tree-based regression techniques have also been used for sensitivity analysis in an effort to address the challenges faced with nonlinear models [96]. Examples of these methods include the Patient Rule Induction Method (PRIM; [97]) and Classification And Regression Trees (CART; [98]). CART-based approaches also include boosting and bagging extensions [99, 100]. These methods are particularly useful when sensitivity analysis is used for factor mapping (i.e., when trying to identify which uncertain model factors produce a certain model behavior). Chapter 4.3 elaborates on the use of these methods. Regression-based sensitivity analysis methods are global by nature and can explore the entire space of variables. However, the true level of comprehensiveness depends on the design of experiments and the number of simulations providing data to establish the regression relationships. Although they are usually computationally efficient, they do not produce significant information about parametric interactions [38, 39].

3.4.4 Regional Sensitivity Analysis

Another method primarily applied for basic factor mapping applications is Regional Sensitivity Analysis (RSA; [101]). RSA is a global sensitivity analysis method that is typically implemented using standard sampling methods such as latin hypercube sampling. It is performed by specifying a condition on the output space (e.g., an upper threshold) and classifying outputs that meet the condition as behavioral and the ones that fail it as non-behavioral (illustrated in Fig. 3.2 (b)). Note that the specified threshold depends on the nature of the problem, model, and the research question. It can reflect model-performance metrics (such as errors) or consequential decision-relevant metrics (such as unacceptable system outcomes). The behavioral and non-behavioral outputs are then traced back to their originating sampled factors, where differences between the distributions of samples can be used to determine their significance in producing each part of the output. The Kolmogorov-Smirnov divergence is commonly used to quantify the difference between the distribution of behavioral and non-behavioral parameters [51]:

$$S_i = |F_{x_i|y_b}(y \in Y_b) - F_{x_i|y_{nb}}(y \in Y_{nb})|$$

where Y_b represents the set of behavioral outputs, and $F_{x_i|y_b}$ is the empirical cumulative distribution function of the values of x_i associated with values of y that belong in the behavioral set. The nb notation indicates the equivalent elements related to the non-behavioral set. Large differences between the two distributions indicate stronger effects by the parameters on the respective part of the output space.

Used in a factor mapping setting, RSA can be applied for scenario discovery [102, 103], the Generalized Likelihood Uncertainty Estimation method (GLUE; [18, 104, 105]) and other hybrid sensitivity analysis methods (e.g., [106,

107]). The fundamental shortcomings of RSA are that, in some cases, it could be hard to interpret the difference between behavioral and non-behavioral sample sets, and that insights about parametric correlations and interactions cannot always be uncovered [38]. For more elaborate discussions and illustrations of the RSA method, readers are directed to Tang *et al.* [42], Saltelli *et al.* [49], Young [108] and references therein.

3.4.5 Variance-based Methods

Variance-based sensitivity analysis methods hypothesize that various specified model factors contribute differently to the variation of model outputs; therefore, decomposition and analysis of output variance can determine a model's sensitivity to input parameters [38, 77]. The most popular variance-based method is the Sobol method, which is a global sensitivity analysis method that takes into account complex and nonlinear factor interaction when calculating sensitivity indices, and employs more sophisticated sampling methods (e.g., the Sobol sampling method). The Sobol method is able to calculate three types of sensitivity indices that provide different types of information about model sensitivities. These indices include first-order, higher-order (e.g., second-, third-, etc. orders), and total-order sensitivities.

The first-order sensitivity index indicates the percent of model output variance contributed by a factor individually (i.e., the effect of varying x_i alone) and is obtained using the following [77, 109]:

$$S_i^1 = \frac{V_{x_i}[E_{x_{\sim i}}(x_i)]}{V(y)}$$

with E and V denoting the expected value and the variance, respectively. $x_{\sim i}$ denotes all factors expect for x_i. The first-order sensitivity index (S_i^1) can therefore also be thought of as the portion of total output variance (V_y) that can be reduced if the uncertainty in factor x_i is eliminated [110]. First-order sensitivity indices are usually used to understand the independent effect of a factor and to distinguish its individual versus interactive influence. It would be expected for linearly independent factors that they would only have first order indices (no interactions) that should correspond well with sensitivities obtained from simpler methods using OAT sampling.

Higher-order sensitivity indices explore the interaction between two or more parameters that contribute to model output variations. For example, a second-order index indicates how interactions between a pair of factors can lead to change in model output variance and is calculated using the following relationship:

$$S_{ij}^2 = \frac{V_{x_{i,j}}[E_{x_{\sim i,j}}(x_i, x_j)]}{V(y)}$$

with $i \neq j$. Higher order indices can be calculated by similar extensions (i.e., fixing additional operators together), but it is usually computationally expensive in practice.

The total sensitivity analysis index represents the entire influence of an input factor on model outputs including all of its interactions with other factors [111]. In other words, total-order indices include first-order and all higher-order interactions associated with each factor and can be estimated calculated using the following:

$$S_i^T = \frac{E_{x_{\sim i}}[V_{x_i}(x_{\sim i})]}{V(y)} = 1 - \frac{V_{x_{\sim i}}[E_{x_i}(x_{\sim i})]}{V(y)}$$

This index reveals the expected portion of variance that remains if uncertainty is eliminated in all factors but x_i [110]. The total sensitivity index is the overall best measure of sensitivity as it captures the full individual and interactive effects of model factors.

Besides the Sobol method, there are some other variance-based sensitivity analysis methods, such as the Fourier amplitude sensitivity test (FAST; [75, 112]) and extended-FAST [113, 114], that have been used by the scientific community. However, Sobol remains by far the most common method of this class. Variance-based techniques have been widely used and have proved to be powerful in a variety of applications. Despite their popularity, some authors have expressed concerns about the methods' appropriateness in some settings. Specifically, the presence of heavy-tailed distributions or outliers, or when model outputs are multimodal can bias the sensitivity indices produced by these methods [115, 116, 117]. Moment-independent measures, discussed below, attempt to overcome these challenges.

Note: Put this into practice! Click the following link to try out an interactive tutorial which demonstrates the application of a Sobol sensitivity analysis: Sobol SA using SALib Jupyter Notebook

3.4.6 Analysis of Variance (ANOVA)

Analysis of Variance (ANOVA) was first introduced by Fisher and others [118] and has since become a popular factor analysis method in physical experiments. ANOVA can be used as a sensitivity analysis method in computational experiments with a factorial design of experiment (referred to as factorial ANOVA). Note that Sobol can also be categorized as an ANOVA sensitivity analysis method, and that is why Sobol is sometimes referred to as a functional ANOVA [119]. Factorial ANOVA methods are particularly suited for models and problems that have discrete input spaces, significantly reducing the computational time. More information about these methods can be found in [119, 120, 121].

3.4.7 Moment-Independent (Density-Based) Methods

These methods typically compare the entire distribution (i.e., not just the variance) of input and output parameters in order to determine the sensitivity of the output to a particular input variable. Several moment-independent sensitivity analysis methods have been proposed in recent years. The delta (δ) moment-independent method calculates the difference between unconditional and conditional cumulative distribution functions of the output. The method was first introduced by [122, 123] and has become widely used in various disciplines. The δ sensitivity index is defined as follows:

$$S_i = \delta_i = \frac{1}{2} E_{x_i} |f_y(y) - f_{y|x_i}(y)| dy$$

where $f_y(y)$ is the probability density function of the entire model output y, and $f_{y|x_i}(y)$ is the conditional density of y, given that factor x_i assumes a fixed value. The δ_i sensitivity indicator therefore represents the normalized expected shift in the distribution of y provoked by x_i. Moment-independent methods are advantageous in cases where we are concerned about the entire distribution of events, such as when uncertain factors lead to more extreme events in a system [13]. Further, they can be used with a pre-existing sample of data, without requiring a specific sampling scheme, unlike the previously reviewed methods [124]. The δ sensitivity index does not include interactions between factors and it is therefore akin to the first order index produced by the Sobol method. Interactions between factors can still be estimated using this method, by conditioning the calculation on more than one uncertain factor being fixed [123].

3.5 How To Choose A Sensitivity Analysis Method: Model Traits And Dimensionality

Fig. 3.5, synthesized from variants found in [51, 91], presents a graphical synthesis of the methods overviewed in this section, with regards to their appropriateness of application based on the complexity of the model at hand and the computational limits on the number of model evaluations afforded. The bars below each method also indicate the sensitivity analysis purposes they are most appropriate to address, which are in turn a reflection of the motivations and research questions the sensitivity analysis is called to address. Computational intensity is measured as a multiple of the number of model factors that are considered uncertain (d). Increasing model complexity mandates that more advanced sensitivity analysis methods are applied to address potential nonlinearities, factor interactions, and discontinuities. Such methods can only be performed at increasing computational expense. For example, computationally cheap linear regression should not be used to assess factors' importance if the model cannot be proven linear and the factors independent, because important relationships will invariably be missed (recall the example in Fig. 3.5). When computational limits do constrain applications to make simplified assumptions and sensitivity techniques, any conclusions in such cases should be delivered with clear statements of the appropriate caveats.

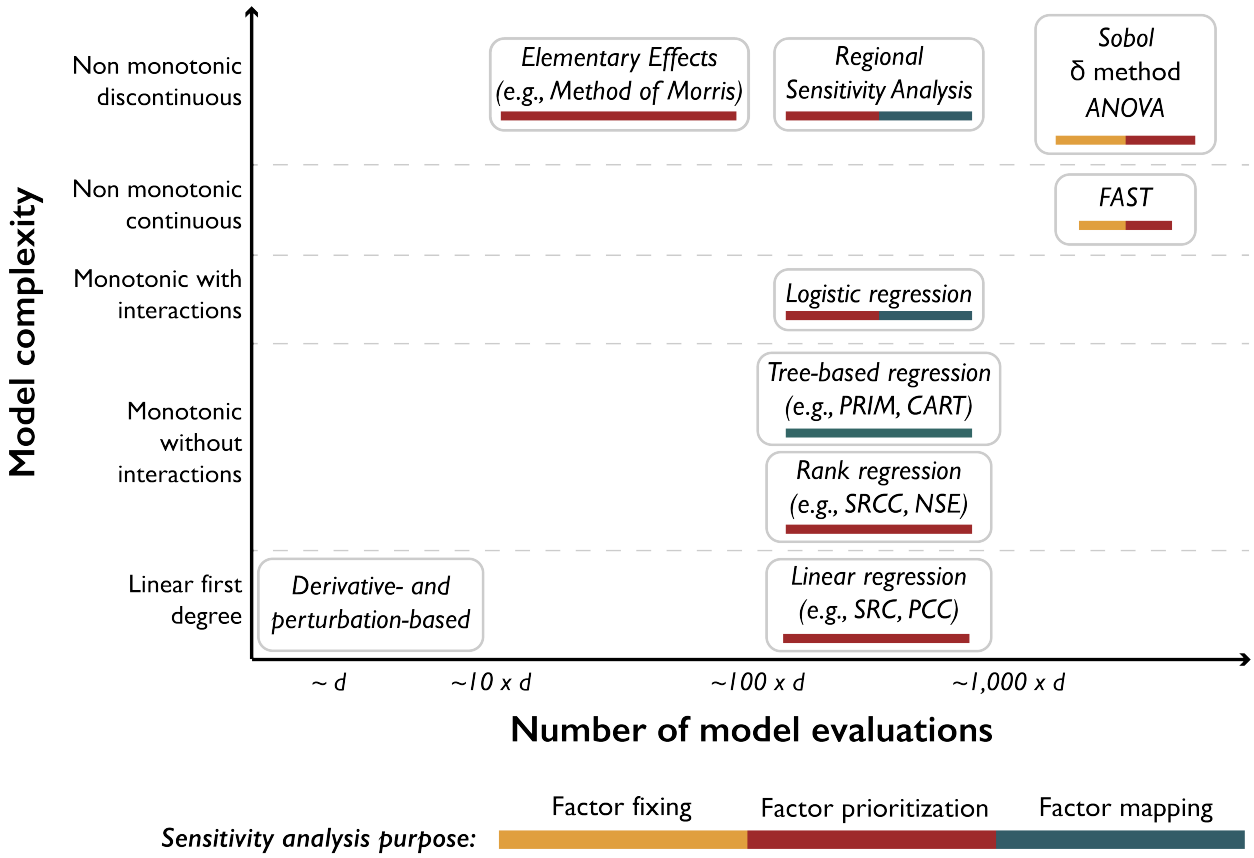


Fig. 3.5: Classification of the sensitivity analysis methods overviewed in this section, with regards to their computational cost (horizontal axis), their appropriateness to model complexity (vertical axis), and the purpose they can be used for (colored bars). d: number of uncertain factors considered; ANOVA: Analysis of Variance; FAST: Fourier Amplitude Sensitivity Test; PRIM: Patient Rule Induction Method; CART: Classification and Regression Trees; SRCC: Spearman's rank correlation coefficient: NSE: Nash–Sutcliffe efficiency; SRC: standardized regression coefficient; PCC: Pearson correlation coefficient. This figure is synthesized from variants found in [51, 91].

The reader should also be aware that the estimates of computational intensity that are given here are indicative of magnitude and would vary depending on the sampling technique, model complexity and the level of information being asked. For example, a Sobol sensitivity analysis typically requires a sample of size $n * d + 2$ to produce first- and total-order indices, where d is the number of uncertain factors and n is a scaling factor, selected ad hoc, depending on model complexity [46]. The scaling factor n is typically set to at least 1000, but it should most appropriately be set on the basis of index convergence. In other words, a prudent analyst would perform the analysis several times with increasing n and observe at what level the indices converge to stable values [125]. The level should be the minimum sample size used in subsequent sensitivity analyses of the same system. Furthermore, if the analyst would like to better understand the degrees of interaction between factors, requiring second-order indices, the sample size would have to increase to $n * 2d + 2$ [46].

Another important consideration is that methods that do not require specific sampling schemes can be performed in conjunction with others without requiring additional model evaluations. None of the regression-based methods, for example, require samples of specific structures or sizes, and can be combined with other methods for complementary purposes. For instance, one could complement a Sobol analysis with an application of CART, using the same data, but to address questions relating to factor mapping (e.g., we know factor x_i is important for a model output, but we would like to also know which of its values specifically push the output to undesirable states). Lastly, comparing results from different methods performed together can be especially useful in model diagnostic settings. For example, [13] used δ indices, first-order Sobol indices, and R^2 values from linear regression, all performed on the same factors, to derive insights about the effects on factors on different moments of the output distribution and about the linearity of their relationship.

3.6 Software Toolkits

This section presents available open source sensitivity analysis software tools, based on the programming language they use and the methods they support Fig. 3.6. Our review covers five widely used programming languages: R, MATLAB, Julia, Python, and C++, as well as one tool that provides a graphical user interface (GUI). Each available SA tool was assessed on the number of SA methods and design of experiments methods it supports. For example, the *sensobol* package in R only supports the variance-based Sobol method. However, it is the only package we came across that calculates third-order interactions among parameters. On the other side of the spectrum, there are SA software packages that contain several popular SA methods. For example, *SALib* in Python [126] supports seven different SA methods. The *DifferentialEquations* package is a comprehensive package developed for Julia, and *GlobalSensitivityAnalysis* is another Julia package that has mostly adapted SALib methods. Fig. 3.6 also identifies the SA packages that have been updated since 2018, indicating active support and development.

Note: The following articles are suggested as fundamental reading for the information presented in this section:

- 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
- 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

The following articles can be used as supplemental reading:

- Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., Tarantola, S., 2008. Global Sensitivity Analysis: The Primer, 1st edition. ed. Wiley-Interscience, Chichester, England; Hoboken, NJ.
- Montgomery, D.C., 2017. Design and analysis of experiments. John Wiley & Sons.
- Iooss, B., Lemaître, P., 2015. A Review on Global Sensitivity Analysis Methods, in: Dellino, G., Meloni, C. (Eds.), Uncertainty Management in Simulation-Optimization of Complex Systems: Algorithms and Appli-

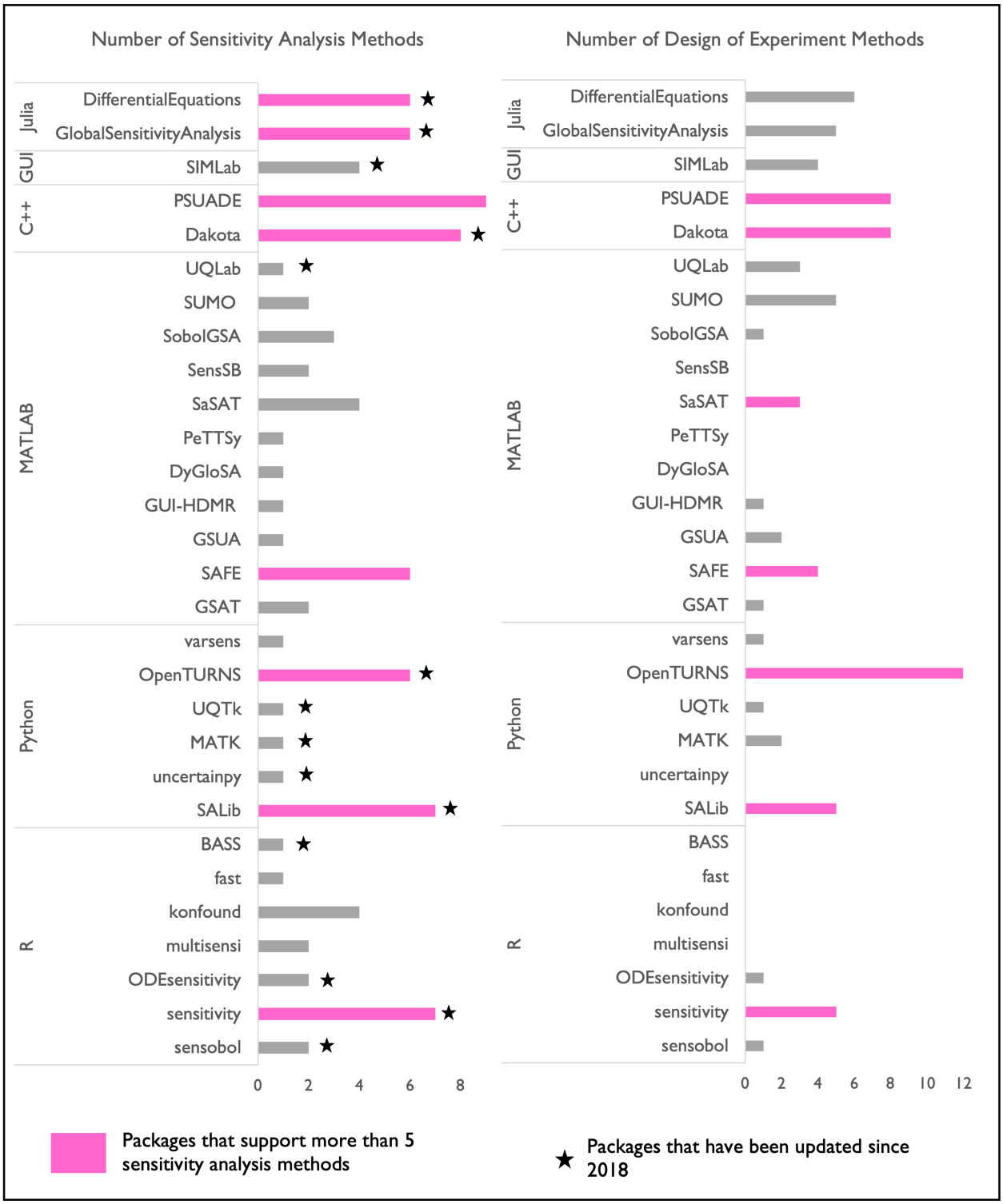


Fig. 3.6: Sensitivity analysis packages available in different programming language platforms (R, Python, Julia, MATLAB, and C++), with the number of methods they support. Packages supporting more than five methods are indicated in pink. Packages updated since 2018 are indicated with asterisks.

cations, Operations Research/Computer Science Interfaces Series. Springer US, Boston, MA, pp. 101–122. https://doi.org/10.1007/978-1-4899-7547-8_5

CHAPTER

FOUR

SENSITIVITY ANALYSIS: DIAGNOSTIC & EXPLORATORY MODELING

4.1 Understanding Errors: What Is Controlling Model Performance?

Sensitivity analysis is a diagnostic tool when reconciling model outputs with observed data. It is helpful for clarifying how and under what conditions modeling choices (structure, parameterization, data inputs, etc.) propagate through model components and manifest in their effects on model outputs. This exploration is performed through carefully designed sampling of multiple combinations of input parameters and subsequent evaluation of the model structures that are emerging as controlling factors. Model structure and parameterization are two of the most commonly explored aspects of models that have been a central focus when evaluating their performance relative to available observations [17]. Addressing these issues plays an important role in establishing credibility in model predictions, particularly in the positivist natural sciences literature. Traditional model evaluations compare the model with observed data, and then rely on expert judgements of its acceptability based on the closeness between simulation and observation with one or a small number of selected metrics. This approach can be myopic, as it is often impossible to use one metric to attribute a certain error and to link that with different parts of the model and its parameters [127]. This means that, even when the error or fitting measure between the model estimates and observations is very small, it is not guaranteed that all the components in the model accurately represent the conceptual reality that the model is abstracting: propagated errors in different parts of the model might cancel each other out, or multiple parameterized implementations of the model can yield similar performance (i.e., equifinality [17]).

The inherent complexity of a system hinders accepting or rejecting a model based on one performance measure and different types of measures can aid in evaluating the various components of a model as essentially a multiobjective problem [25, 26]. In addition, natural systems mimicked by the models contain various interacting components that might act differently across spatial and temporal domains [51, 55]. This heterogeneity is lost when a single performance measure is used, as a highly dimensional and interactive system becomes aggregated through the averaging of spatial or temporal output errors [15]. Therefore, diagnostic error analyses should consider multiple error signatures across different scales and states of concern when seeking to understand how model performance relates to observed data (Fig. 4.1). Diverse error signatures can be used to measure the consistency of underlying processes and behaviors of the model and to evaluate the dynamics of model controls under changing temporal and spatial conditions [128]. Within this framework, even minimal information extracted from the data can be beneficial as it helps us unearth structural inadequacies in the model. In this context, proper selection of measures of model performance and the number of measures could play consequential roles in our understanding of the model and its predictions [129].

As discussed earlier, instead of the traditional focus on using deterministic prediction that results in a single error measure, many plausible states and spaces could be searched for making different inferences and quantifying uncertainties. This process also requires estimates of prior probability distributions of all the important parameters and quantification of model behavior across input space. One strategy to reduce the search space is filtering of some model alternatives that are not consistent with observations and known system behaviors. Those implausible parts of the search space can be referred to as non-physical or non-behavioral alternatives [50, 104]. This step is conducted before the Bayesian calibration exercise (see Chapter A).

A comprehensive model diagnostic workflow typically entails the components demonstrated in Fig. 4.1. The workflow begins with the selection of model input parameters and their plausible ranges. After the parameter selection, we need to specify the design of experiment (Chapter 3.3) and the sensitivity analysis method (Chapter 3.4) to be used.

As previously discussed, these methods require different numbers of model simulations, and each method provides a different insights into the direct effects and interactions of the uncertain factors. In addition, the simulation time of the model and the available computational resources are two of the primary considerations that influence these decisions. After identifying the appropriate methods, we generate a matrix of input parameters, where each set of input parameters will be used to conduct a model simulation. The model can include one or more output variables that fluctuate in time and space. The next step is to analyze model performance by comparing model outputs with observations. As discussed earlier, the positivist model evaluation paradigm focuses on a single model performance metric (error), leading to a loss of information about model parameters and the suitability of the model's structure. However, a thorough investigation of the temporal and spatial signatures of model outputs using various performance metrics or time- and space-varying sensitivity analyses can shed more light on the fitness of each parameter set and the model's internal structure. This analysis provides diagnostic feedback on the importance and range of model parameters and can guide further improvement of the model algorithm.

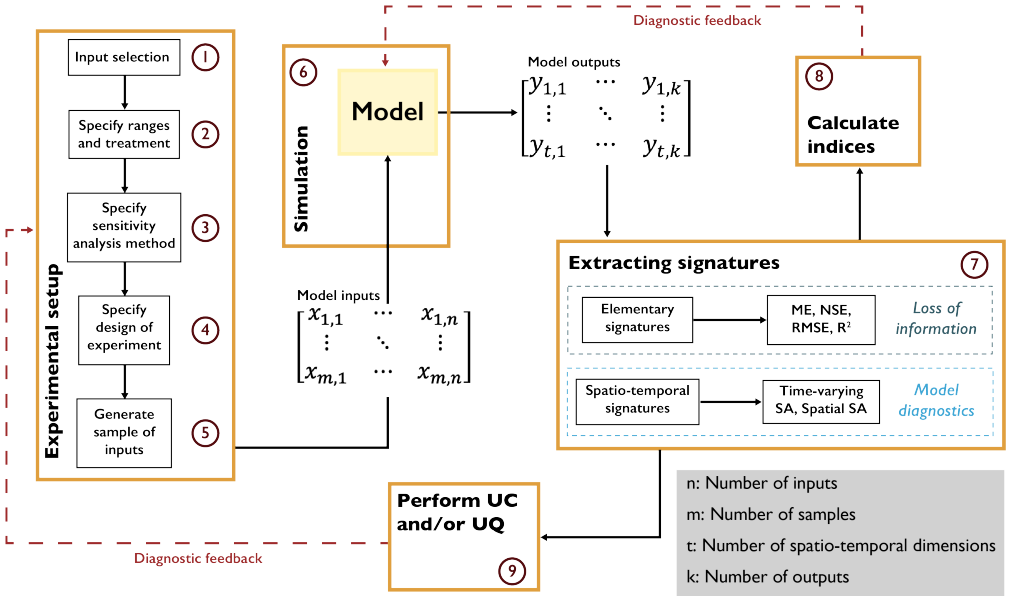


Fig. 4.1: Diagnostic evaluation of model fidelity using sensitivity analysis methods.

Note: Put this into practice! Click the following badge to try out an interactive tutorial on implementing a time-varying sensitivity analysis of HYMOD model parameters: HYMOD Jupyter Notebook

4.2 Consequential Dynamics: What is Controlling Model Behaviors of Interest?

Consequential changes in dynamic systems can take many forms, but most dynamic behavior can be categorized in a few basic patterns. Feedback structures inherent in a system, be they positive or negative, generate these patterns which can be grouped into three groups for the simplest of systems: exponential growth, goal seeking, and oscillation (Fig. 4.2). A positive (or self-reinforcing) feedback gives rise to exponential growth, a negative (or self-correcting) feedback gives rise to a goal seeking mode, and negative feedbacks with time delays give rise to oscillatory behavior. Nonlinear interactions between the system's feedback structures can give rise to more complex dynamic behavior modes, examples of which are also shown in Fig. 4.2, adapted from Sterman [130].

The nature of feedback processes in a dynamic system shapes its fundamental behavior: positive feedbacks generate their own growth, negative feedbacks self-limit, seeking balance and equilibrium. In this manner, feedback processes

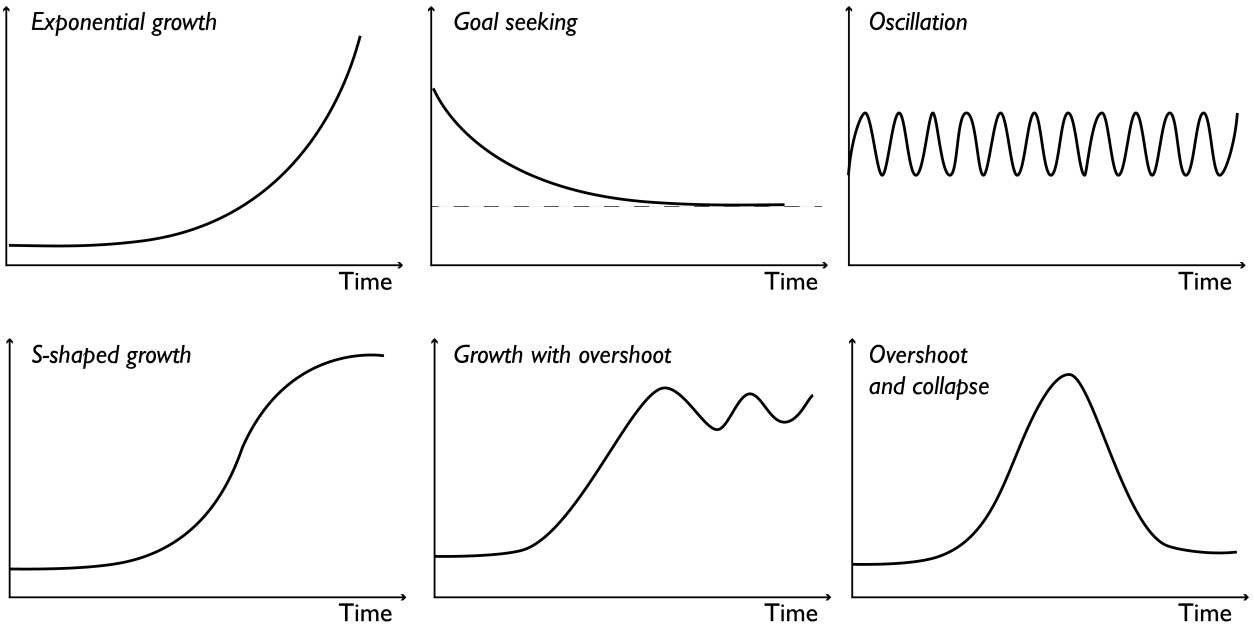


Fig. 4.2: Common modes of behavior in dynamic systems, occurring based on the presence of positive and negative feedback relationships, and linear and non-linear interactions. Adapted from Sterman [130].

give rise to different regimes, multiples of which could be present in each mode of behavior. Consider a population of mammals growing exponentially until it reaches the carrying capacity of its environment (referred to as S-shaped growth). When the population is exponentially growing, the regime is dominated by positive feedback relationships that reinforce its growth. As the population approaches its carrying capacity limit, negative feedback structures begin to dominate, counteracting the growth and establishing a stable equilibrium. Shifts between regimes can be thought of as tipping points, mathematically defined as unstable equilibria, where the presence of positive feedbacks amplifies disturbances and moves the system to a new equilibrium point. In the case of stable equilibria, the presence of negative feedbacks dampens any small disturbance and maintains the system at a stable state. As different feedback relationships govern each regime, different factors (those making up each feedback mechanism) are activated and shape the states the system is found in, as well as define the points of equilibria.

For simple stylized models with a small number of states, system dynamics analysis can analytically derive these equilibria, the conditions for their stability and the factors determining them. The ability for this to be performed is, however, significantly challenged when it comes to systems that attempt to more closely resemble real complex systems. We argue this is the case for several reasons. First, besides generally exhibiting complex nonlinear dynamics, real world systems are also made up from larger numbers of interacting elements, which often makes the analytic derivation of system characteristics intractable [131, 132]. Second, human-natural systems temporally evolve and transform when human state-aware action is present. Consider, for instance, humans recreationally hunting the aforementioned population of mammals. Humans act based on the mammal population levels by enforcing hunting quotas or establishing protected territories or eliminating other predators. The mammal population reacts in a response, giving birth to ever changing state-action-consequence feedbacks, the path dependencies of which become difficult to diagnose and understand (e.g., [133]). Trying to simulate the combination of these two challenges (large numbers of state-aware agents interacting with a natural resource and with each other) produces intractable models that require advanced heuristics to analyze their properties and establish useful inferences.

Sensitivity analysis paired with exploratory modeling methods offers a promising set of tools to address these challenges. We present a simple demonstrative application based on Quinn *et al.* [134]. This stylistic example was first developed by Carpenter *et al.* [135] and represents a town that must balance its agricultural and industrial productivity with the pollution it creates in a downstream lake. Increased productivity allows for increased profits, which the town aims to maximize, but it also produces more pollution for the lake. Too much phosphorus pollution can cause irreversible eutrophication, a process known as "tipping" the lake. The model of phosphorus in the lake X_t at time t is

governed by:

$$X_{t+1} = X_t + a_t + \frac{X_t^q}{1 + X_t^q} - bX_t + \varepsilon$$

where $a_t \in [0, 0.1]$ is the town's pollution release at each timestep, b is the natural decay rate of phosphorus in the lake, q defines the lake's recycling rate (primarily through sediments), and ε represents uncontrollable natural inflows of pollution modeled as a log-normal distribution with a given mean, μ, and standard deviation σ.

Panels (a-c) in Fig. 4.3 plot the fluxes of phosphorus into the lake versus the mass accumulation of phosphorus in the lake. The red line corresponds to the phosphorus sinks in the lake (natural decay), given by bX_t. The grey shaded area represents the lake's phosphorus recycling flux, given by $\frac{X_t^q}{1+X_t^q}$. The points of intersection indicate the system's equilibria, two of which are stable, and one is unstable (also known as the tipping point). The stable equilibrium in the bottom left of the figure reflects an oligotrophic lake, whereas the stable equilibrium in the top right represents a eutrophic lake. With increasing phosphorus values, the tipping point can be crossed, and the lake will experience irreversible eutrophication, as the recycling rate would exceed the removal rate even if the town's pollution became zero. In the absence of anthropogenic and natural inflows of pollution in the lake (a_t and ε respectively), the area between the bottom-left black point and the white point in the middle can be considered as the safe operating space, before emission levels cross the tipping point.

The town has identified two potential policies that can be used to manage this lake, one that maximizes its economic profits ("best economic policy") and one that maximizes the time below the tipping point ("most reliable policy"). Panels (b-c) in Fig. 4.3 add the emissions from these policies to the recycling flux and show how the equilibria points shift as a result. In both cases the stable oligotrophic equilibrium increases and the tipping point decreases, narrowing the safe operating space [131, 138]. The best economic policy results in a much narrower space of action, with the tipping point very close to the oligotrophic equilibrium. The performance of both policies depends significantly on the system parameters. For example, a higher value of b, the natural decay rate, would shift the red line upward, moving the equilibria points and widening the safe operating space. Inversely, a higher value of q, the lake's recycling rate, would shift the recycling line upward, moving the tipping point lower and decreasing the safe operating space. The assumptions under which these policies were identified are therefore critical to their performance and any potential uncertainty in the parameter values could be detrimental to the system's objectives being met.

Sensitivity analysis can be used to clarify the role these parameters play on policy performance. Fig. 4.3 (d) shows the results of a Sobol sensitivity analysis on the reliability of the "most reliable" policy in a radial convergence diagram. The significance of each parameter is indicated by the size of circles corresponding to it. The size of the interior dark circle indicates the parameter's first-order effects and the size of the exterior circle indicates the parameter's total-order effects. The thickness of the lines between two parameters indicated the extent of their interaction (second-order effects). In this case, parameters b and q appear to have the most significant importance on the system, followed by the mean, μ, of the natural inflows. All these parameters function in a manner that shifts the location of the three equilibria and therefore policies that are identified ignoring this parametric uncertainty might fail to meet their intended goals.

It is worth mentioning that current sensitivity analysis methods are somewhat challenged in addressing several system dynamics analysis questions. The fundamental reason is that sensitivity analysis methods and tools have been developed to gauge numerical sensitivity of model output to changes in factor values. This is natural, as most simulation studies (e.g., all aforementioned examples) have been traditionally concerned with this type of sensitivity. In system dynamics modeling, however, a more important and pertinent concern is changes between regimes or between behavior modes (also known as bifurcations) as a result of changes in model factors [130, 139]. This poses two new challenges. First, identifying a change in regime depends on several characteristics besides a change in output value, like the rate and direction of change. Second, behavior mode changes are qualitative and discontinuous, as equilibria change in stability but also move in and out of existence.

Despite these challenges, recent advanced sensitivity analysis methods can help illuminate which factors in a system are most important in shaping boundary conditions (tipping points) between different regimes and determining changes in behavior modes. Reviewing such methods is outside the scope of this text, but the reader is directed to the examples of Eker *et al.* [22] and Hadjimichael *et al.* [133], who apply parameterised perturbation on the functional relationships of a system to study the effects of model structural uncertainty on model outputs and bifurcations, and Hekimoğlu and

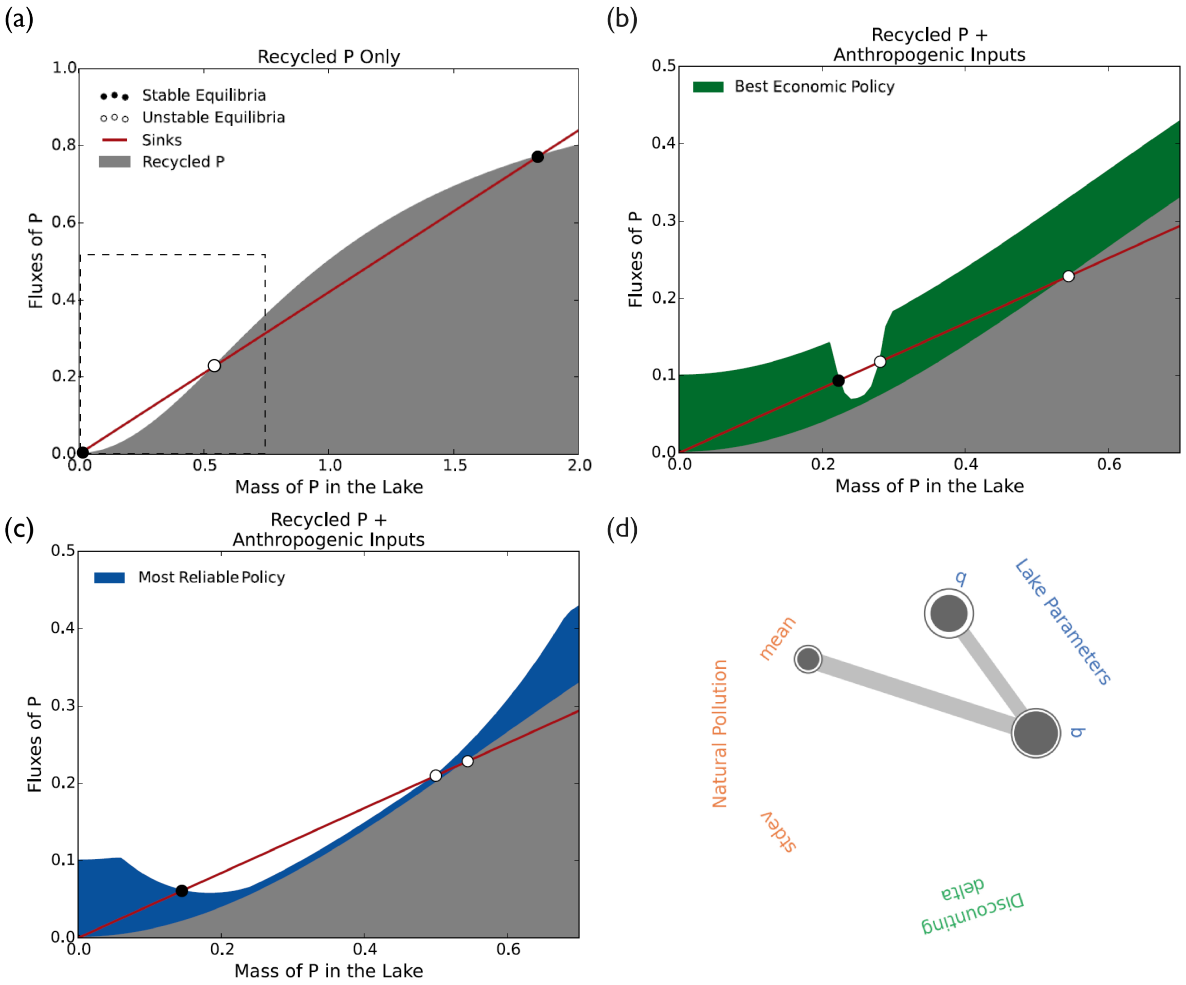


Fig. 4.3: Fluxes of phosphorus with regards to mass of phosphorus in the lake and sensitivity analysis results, assuming $b = 0.42$ and $q = 2$. (a) Fluxes of phosphorus assuming no emmisions policy and no natural inflows. (b-c) Fluxes phosphorus when applying two different emissions policies. The "Best economic policy" and the "Most reliable policy" have been identified by Quinn *et al.* [134] and can be found at Quinn [136]. (d) Results of a sensitivity analysis on the parameters of the model most consequential to the reliability of the "Most reliable policy". The code to replicate the sensitivity analysis can be found at Hadka [137]. Panels (a-c) are used courtesy of Julianne Quinn, University of Virginia.

Barlas [139] and Steinmann *et al.* [140] who, following wide sampling of uncertain inputs, cluster the resulting time series in modes of behavior and identify most important factors for each.

Note: Put this into practice! Click the following badge to try out an interactive tutorial on performing a sensitivity analysis to discover consequential dynamics: Factor Discovery Jupyter Notebook

4.3 Consequential Scenarios: What is Controlling Consequential Outcomes?

As overviewed in Chapter 2.2, most models are abstractions of systems in the real world. When sufficient confidence has been established in a model, it can then act as a surrogate for the actual system, in that the consequences of potential stressors, proposed actions or other changes can be evaluated by computer model simulations [36]. A model simulation then represents a computational experiment, which can be used to assess how the modeled system would behave should the various changes come to be. Steven Bankes coined the term exploratory modeling to describe the use of large sets of such computational experiments to investigate their implications on the system. Fig. 4.4 presents a typical workflow of an exploratory modeling application. Exploratory modeling approaches typically use sampling designs to generate large ensembles of states that represent combinations of changes happening together, spanning the entire range of potential values a factor might take (indicated in Fig. 4.4 by numbers 2-5). This perspective on modeling is particularly relevant to studies making long term projections into the future.

In the long-term policy analysis literature, exploratory modeling has prominently placed itself as an alternative to traditional narrative scenario or assumptions-based planning approaches, in what can be summarized in the following two-pronged critique [36, 141, 142]. The most prevalent criticism sees that the future and how it might evolve is both highly complex and deeply uncertain. Despite its benefits for interpretation and intuitive appeal, a small number of scenarios invariably misses many other potential futures that did not get selected as sufficiently representative of the future. This is especially the case for aggregate, narrative scenarios that describe simultaneous changes in multiple sectors together (e.g., "increased energy demand, combined with high agricultural land use and large economic growth"), such as the emission scenarios produced by the Intergovernmental Panel on Climate Change [143]. The bias introduced by this reduced set of potential changes can skew inferences drawn from the model, particularly when the original narrative scenarios are focused on a single or narrow set of measures of system behavior.

The second main criticism of traditional narrative scenario-based planning methods is that they provide no systematic way to distinguish which of the constituent factors lead to the undesirable consequences produced by a scenario. Narrative scenarios (e.g., the scenario matrix framework of RCPs-SSPs-SPAs; [144]) encompass multiple changes happening together selected to span the range of potential changes but are not typically generated in a systematic factorial manner that considers the multiple ways the factors can be combined. This has two critical limitations. It obfuscates the role each component factor plays in the system, both in isolation and in combination with others (e.g., "is it the increased energy demand or the high agricultural land use that cause unbearable water stress?"). It also renders the delineation of how much change in a factor is critical near impossible. Consider, for example, narrative scenario A with a 5% increase in energy demand, and scenario B with a 30% increase in energy demand, which would have dire consequences. At which point between 5% and 30% do the dire consequences actually begin to occur? Such questions cannot be answered without a wide exploration of the space of potential changes. It should be noted that for some levels of model complexity and computational demands (e.g., global-scale models) there is little feasible recourse beyond the use of narrative scenarios.

Exploratory modeling is typically paired with scenario discovery methods (indicated by number 9 in Fig. 4.4) that identify which of the scenarios (also known as states of the world) generated indeed have consequences of interest for stakeholders and policy makers, in an approach referred to as ensemble-based scenario-discovery [45, 102, 103]. This approach therefore flips the planning analysis from one that attempts to predict future system conditions to one that attempts to discover the (un)desirable future conditions. Ensemble-based scenario discovery can thus inform what modeling choices yield the most consequential behavioral changes or outcomes, especially when considering deeply uncertain, scenario-informed projections [9, 145]. The relative likelihoods and relevance of the discovered scenarios

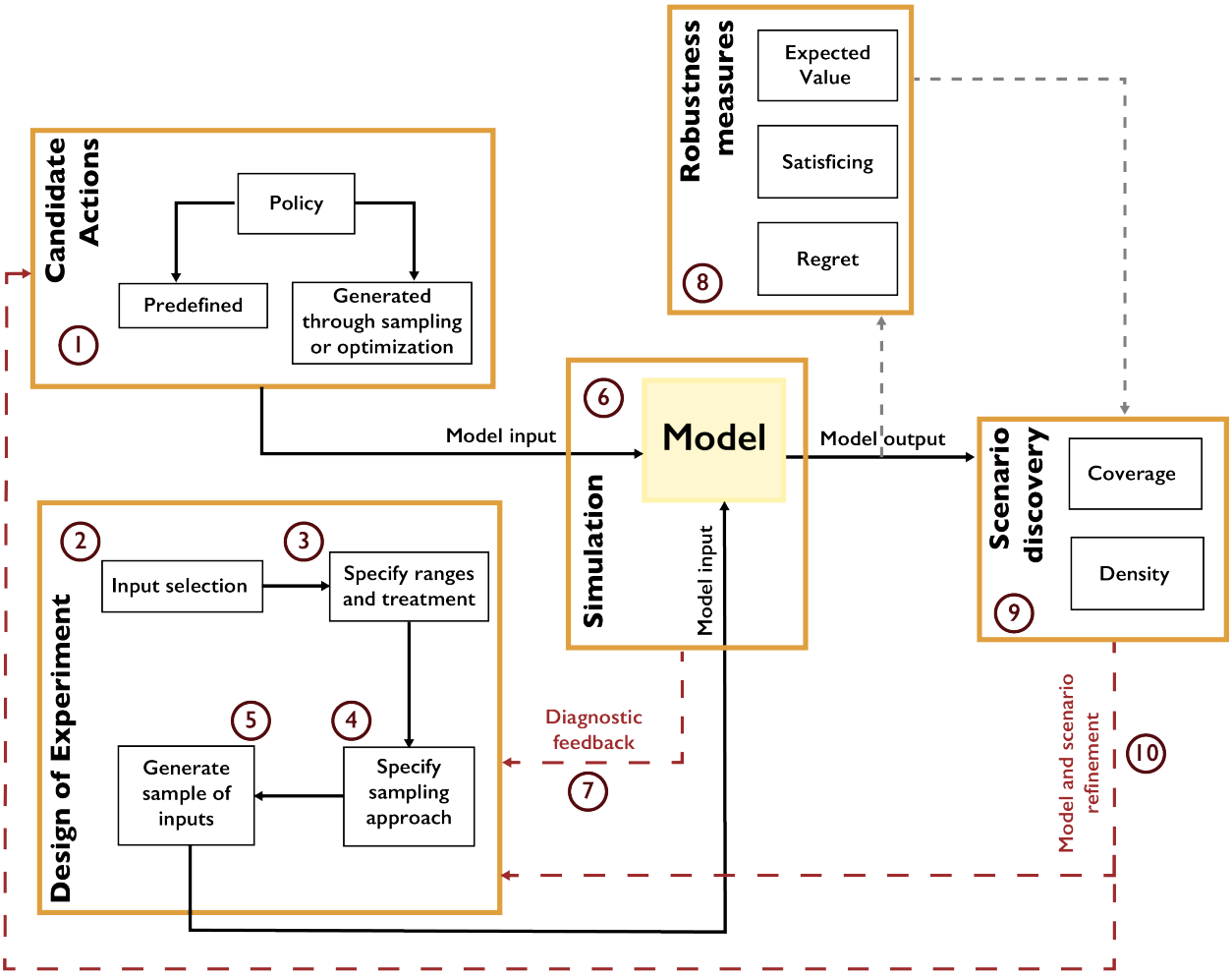


Fig. 4.4: A typical exploratory modeling workflow

can be subsequently evaluated by the practitioners a posteriori, within a richer context of knowing the wider set of potential consequences [146]. This can include changing how an analysis is framed (number 10 in Fig. 4.4). For instance, one could initially focus on ensemble modeling of vulnerability using a single uncertain factor that is assumed to be well characterized by historical observations (e.g., streamflow; this step is represented by numbers 2-3 in Fig. 4.4). The analysis can then shift to include projections of more factors treated as deeply uncertain (e.g., urbanization, population demands, temperature, and snow-melt) to yield a far wider space of challenging projected futures. UC experiments contrasting these two framings can be highly valuable for tracing how vulnerability inferences change as the modeled space of futures expands from the historical baseline [134].

An important nuance to be clarified here is that the focus or purpose of a modeling exercise plays a major role in whether a given factor of interest is considered well-characterized or deeply uncertain. Take the example context of characterizing temperature or streamflow extremes, where for each state variable of interest for a given location of focus there is a century of historical observations. Clearly, the observation technologies will have evolved over time uniquely for temperature and streamflow measurements and they likely lack replicate experiments (data uncertainty). A century of record will be insufficient to capture very high impact and rare extreme events (i.e., increasingly poor structural/parametric inference for the distributions of specific extreme single or compound events). The mechanistic processes as well as their evolving variability will be interdependent but uniquely different for each of these state variables. A large body of statistical literature exists focusing on the topics of synthetic weather [81, 82] or streamflow [83, 84] generation that provides a rich suite of approaches for developing history-informed, well-characterized stochastic process models to better estimate rare individual or compound extremes. These history-focused approaches can be viewed as providing well-characterized quantifications of streamflow or temperature distributions; however, they do not capture how coupled natural-human processes can fundamentally change their dynamics when transitioning to projections of longer-term futures (e.g., streamflow and temperature in 2055). Consequently, changing the focus of the modeling to making long term projections of future streamflow or temperature now makes these processes deeply uncertain.

Scenario discovery methods (number 9 in Fig. 4.4) can be qualitative or quantitative and they generally attempt to distinguish futures in which a system or proposed policies to manage the system meet or miss their goals [103]. The emphasis placed by exploratory modeling on model outputs that have decision relevant consequences represents a shift toward a broader class of metrics that are reflective of the stakeholders' concerns, agency and preferences (also discussed in Chapter 2.2). As a result, sensitivity analysis and scenario discovery methods in this context are therefore applied to performance metrics that go beyond model error but are rather focused on broader metrics such as the resilience of a sector, the reliability of a process, or the vulnerability of a population in the face of uncertainty. In exploratory modeling literature, this metric is most typically—but not always—a measure of robustness (number 8 in Fig. 13). Robustness is a property of a system or a design choice capturing its insensitivity to uncertainty and can be measured via a variety of means, most recently reviewed by Herman *et al.* [147] and McPhail *et al.* [129].

Scenario discovery is typically performed through the use of algorithms applied on large databases of model runs, generated through exploratory modeling, with each model run representing the performance of the system in one potential state of the world. The algorithms seek to identify the combinations of factor values (e.g., future conditions) that best distinguish the cases in which the system does or does not meet its objectives. The most widely known classification algorithms are the Patient Rule Induction Method (PRIM; [97]) and Classification and Regression Trees (CART; [98]). These factor mapping algorithms create orthogonal boundaries (multi-dimensional hypercubes) between states of the world that are successful or unsuccessful in meeting the system's goals [65]. The algorithms attempt to strike a balance between simplicity of classification (and as a result, interpretability) and accuracy [45, 103, 148].

Even though these approaches have been shown to yield interpretable and relevant scenarios [149], several authors have pointed out the limitations of these methods with regards to their division of space in orthogonal behavioral and non-behavioral regions [150]. Due to their reliance on boundaries orthogonal to the uncertainty axes, PRIM and CART cannot capture interactions between the various uncertain factors considered, which can often be significant [151]. More advanced methods have been proposed to address this drawback, with logistic regression being perhaps the most prominent [151, 152, 153]. Logistic regression can produce boundaries that are not necessarily orthogonal to each uncertainty axis, nor necessarily linear, if interactive terms between two parameters are used to build the regression model. It also describes the probability that a state of the world belongs to the scenarios that lead to failure. This feature allows users to define regions of success based on a gradient of estimated probability of success in those worlds, unlike PRIM which only classifies states of the world in two regions [151, 154].

Another more advanced factor mapping method is boosted trees [99, 155]. Boosted trees can avoid two limitations inherent to the application of logistic regression: i) to build a nonlinear classification model the interactive term between two uncertainties needs to be pre-specified and cannot be discovered (e.g., we need to know a-priori whether factor x_1 interacts with x_2 in a relationship that looks like $x_1 \cdot x_2$ or $x_1^{x_2}$); and ii) the subspaces defined are always convex. The application of such a factor mapping algorithm is limited in the presence of threshold-based rules with discrete actions in a modeled system (e.g., "if network capacity is low, build new infrastructure"), which results in failure regions that are nonlinear and non-convex [150]. Boosting works by creating an ensemble of classifiers and forcing some of them to focus on the hard-to-learn parts of the problem, and others to focus on the easy-to-learn parts. Boosting applied to CART trees can avoid the aforementioned challenges faced by other scenario discovery methods, while resisting overfitting [156], assuring the identified success and failure regions are still easy to interpret.

Below we provide an example application of two scenario discovery methods, PRIM and logistic regression, using the lake problem introduced in the previous section. From the sensitivity analysis results presented in Fig. 4.3 (d), we can already infer that parameters b and q have important effects on model outputs (i.e., we have performed factor prioritization). Scenario discovery (i.e., factor mapping) complements this analysis by further identifying the specific values of b and q that can lead to consequential and undesirable outcomes. For the purposes of demonstration, we can assume the undesirable outcome in this case is defined as the management policy failing to achieve 90% reliability in a state of the world.

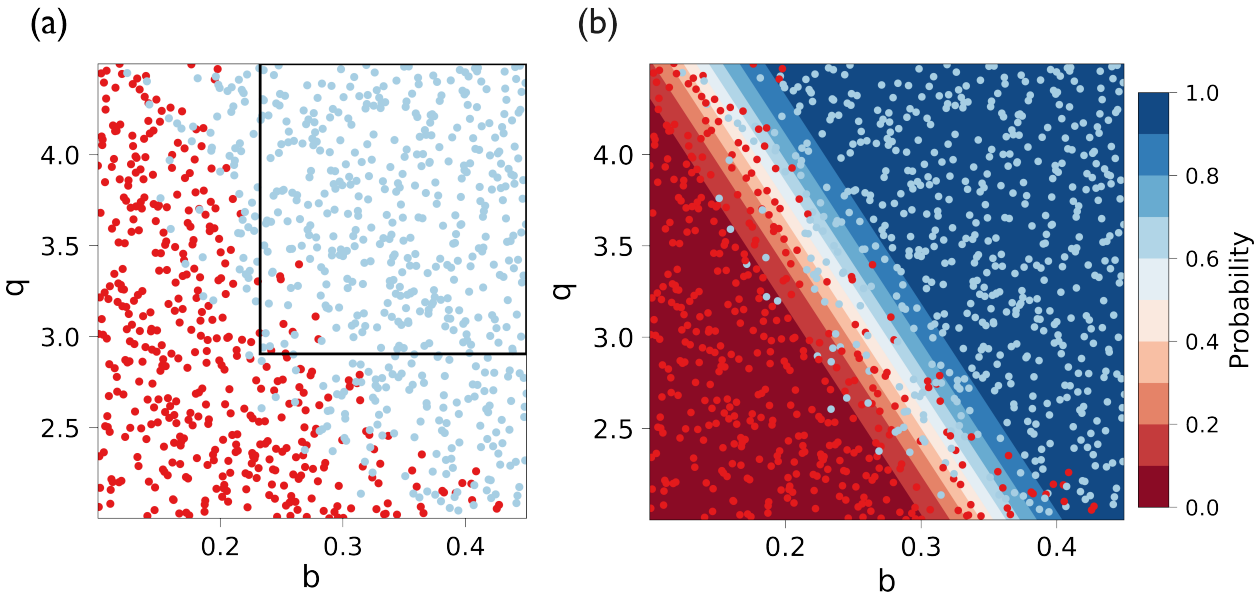


Fig. 4.5: Scenario discovery for the lake problem, using (a) PRIM and (b) logistic regression.

Fig. 4.5 shows the results of scenario discovery, performed through (a) PRIM and (b) logistic regression. Each point in the two panels indicates a potential state of the world, generated through Latin Hypercube Sampling. Each point is colored by whether the policy meets the above performance criterion, with blue indicating success and red indicating failure. PRIM identifies several orthogonal areas of interest, one of which is shown in panel (a). As discussed above, this necessary orthogonality limits how PRIM identifies areas of success (the area within the box). As factors b and q interact in this system, the transition boundary between the regions of success and failure is not orthogonal to any of the axes. As a result, a large number of points in the bottom right and the top left of the figure are left outside of the identified region. Logistic regression can overcome this limitation by identifying a diagonal boundary between the two regions, seen in panel (b). This method also produces a gradient of estimated probability of success across these regions.

Note: Put this into practice! Click the following link to try out an interactive tutorial on performing factor mapping using logistic regression: Logistic Regression Jupyter Notebook

Note: The following articles are suggested as fundamental reading for the information presented in this section:

- Gupta, H.V., Wagener, T., Liu, Y., 2008. Reconciling theory with observations: elements of a diagnostic approach to model evaluation. *Hydrological Processes: An International Journal* 22, 3802–3813.
- Bankes, S., 1993. Exploratory Modeling for Policy Analysis. *Operations Research* 41, 435–449. https://doi.org/10.1287/opre.41.3.435
- Groves, D.G., Lempert, R.J., 2007. A new analytic method for finding policy-relevant scenarios. *Global Environmental Change* 17, 73–85. https://doi.org/10.1016/j.gloenvcha.2006.11.006

The following articles can be used as supplemental reading:

- Marchau, V.A.W.J., Walker, W.E., Bloemen, P.J.T.M., Popper, S.W. (Eds.), 2019. Decision Making under Deep Uncertainty: From Theory to Practice. Springer International Publishing. https://doi.org/10.1007/978-3-030-05252-2
- Sterman, J.D., 1994. Learning in and about complex systems. System Dynamics Review 10, 291–330. https://doi.org/10.1002/sdr.4260100214

CHAPTER

FIVE

CONCLUSION

As noted in the Introduction (Chapter 1), the computational and conceptual challenges of the multi-model, transdisciplinary workflows that characterize ambitious projects such as IM3 have limited UC and UQ analyses. Moreover, the very nature and purpose of modeling and diagnostic model evaluation can have very diverse philosophical framings depending on the disciplines involved (see `Figure_1_1` and Chapter 2.2). The guidance provided in this text can be used to frame consistent and rigorous experimental designs for better understanding the consequences and insights from our modeling choices when seeking to capture complex human-natural systems. The progression of sections of this text provide a thorough introduction of the concepts and definitions of diagnostic model evaluation, sensitivity analysis and UC. In addition, we comprehensively discuss how specific modeling objectives and applications should guide the selection of appropriate techniques; broadly, these can include model diagnostics, in-depth analysis of the behavior of the abstracted system, and projections under conditions of deep uncertainty. This text also contains a detailed presentation of the main sensitivity analysis methods and a discussion of their features and main limitations. Readers are also provided with an overview of computer tools and platforms that have been developed and could be considered in addressing IM3 scientific questions. The appendices of this text include an overview of UQ methods, a terminology glossary of the key concepts as well as example test cases and scripts to showcase various UC related capabilities.

Although we distinguish the UC and UQ model diagnostics, the reader should note that we suggest an overall consistent approach to both in this text by emphasizing "exploratory modeling" (see review by Moallemi *et al.* [9]). Although data support, model complexity, and computational limits strongly distinguish the feasibility and appropriateness of various UC diagnostic tools (e.g., see Fig. 3.5), we overall recommend that modelers view their work through the lens of cycles of learning. Iterative and deliberative exploration of model-based hypotheses and inferences for transdisciplinary teams is non-trivial and ultimately critical for mapping where innovations or insights are most consequential. Overall, we recommend approaching modeling with an openness to the diverse disciplinary perspectives such as those mirrored by the IM3 family of models in a progression from evaluating models relative to observed history to advanced formalized analyses to make inferences on multi-sector, multi-scale vulnerabilities and resilience. Exploratory modeling approaches can help fashion experiments with large numbers of alternative hypotheses on the co-evolutionary dynamics of influences, stressors, as well as path-dependent changes in the form and function of coupled human-natural systems [37]. This text guides the reader through the use of sensitivity analysis and uncertainty methods across the diverse perspectives that have shaped modern diagnostic and exploratory modeling.

APPENDIX

A

UNCERTAINTY QUANTIFICATION

A.1 Introduction

As defined in Chapter 1, uncertainty quantification (UQ) refers to the formal focus on the full specification of likelihoods as well as distributional forms necessary to infer the joint probabilistic response across all modeled factors of interest [8]. This is in contrast to UC (the primary focus of the main document of this book), which is instead aimed at identifying which modeling choices yield the most consequential changes or outcomes and exploring alternative hypotheses related to the form and function of modeled systems [9, 10].

UQ is important for quantifying the relative merits of hypotheses for at least three main reasons. First, identifying model parameters that are consistent with observations is an important part of model development. Due to several effects, including correlations between parameters, simplified or incomplete model structures (relative to the full real-world dynamics), and uncertainty in the observations, many different combinations of parameter values can be consistent with the model structure and the observations to varying extents. Accounting for this uncertainty is conceptually preferable to selecting a single "best fit" parameter vector, particularly as consistency with historical or present observations does not necessarily guarantee skillful future projections.

The act of quantification requires specific assumptions about distributional forms and likelihoods, which may be more or less justified depending on prior information about the system or model behavior. As a result, UQ is well-suited for studies accounting for or addressing hypotheses related to systems with a relatively large amount of available data and models which are computationally inexpensive, particularly when the emphasis is on prediction. As shown in Fig. 1.1, there is a fundamental tradeoff between the available number of model evaluations (for a fixed computational budget) and the number of parameters treated as uncertain. Sensitivity analyses are therefore part of a typical UQ workflow to identify which factors can be fixed and which ought to be prioritized in the UQ.

The choice of a particular UQ method depends on both the desired level of quantification and the ability to navigate the tradeoff between computational expense and the number of uncertain parameters (Fig. 1.1). For example, Markov chain Monte Carlo with a full system model can provide an improved representation of uncertainty compared to the coarser pre-calibration approach [157], but requires many more model evaluations. The use of a surrogate model to approximate the full system model can reduce the number of needed model evaluations by several orders of magnitude, but the uncertainty quantification can only accommodate a limited number of parameters.

The remainder of this appendix will focus on introducing workflows for particular UQ methods, including a brief discussion of advantages and limitations.

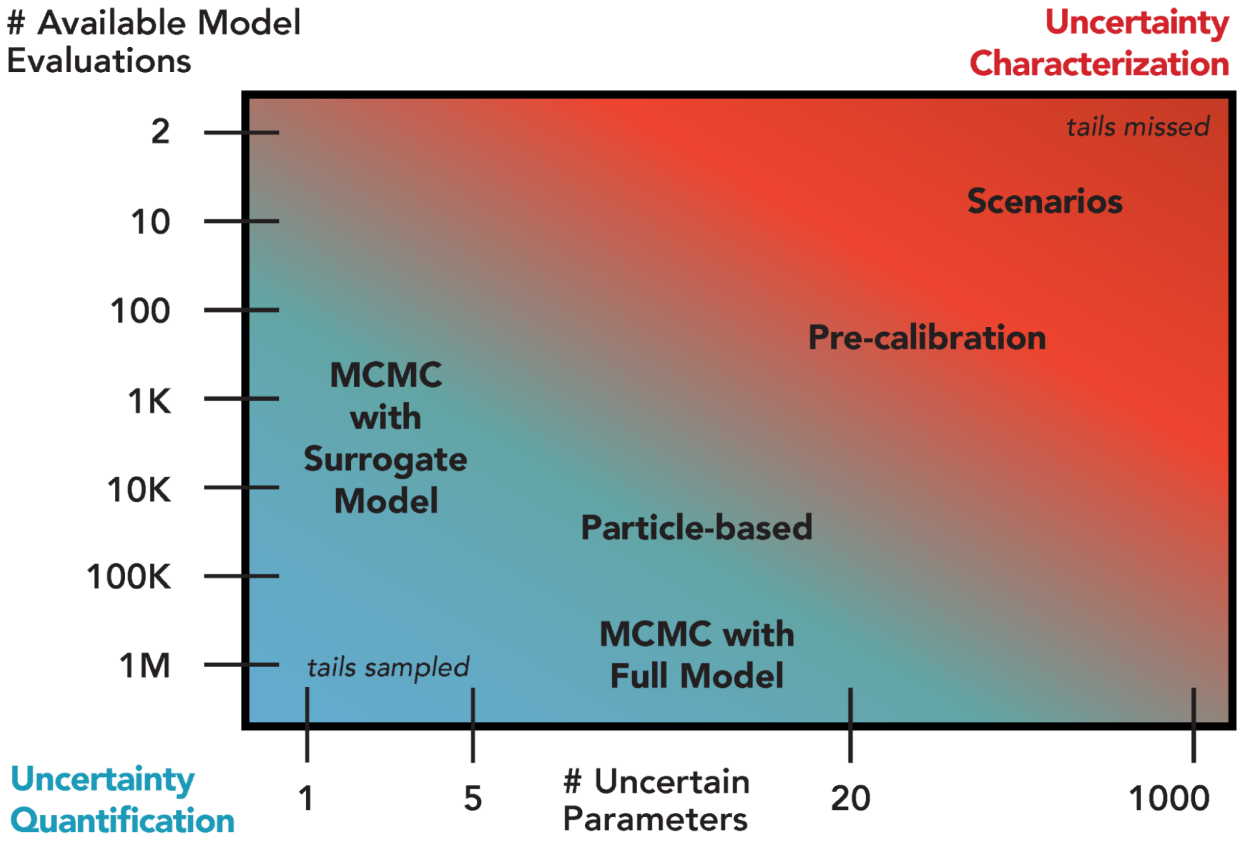


Fig. 1.1: Overview of selected existing approaches for uncertainty quantification and their appropriateness given the number of uncertain model parameters and the number of available model simulations. Green shading denotes regions suitable for uncertainty quantification and red shading indicates regions more appropriate for uncertainty characterization.

A.2 Parametric Bootstrap

The parametric bootstrap [158] refers to a process of model recalibration to alternate realizations of the data. The bootstrap was originally developed to estimate standard errors and confidence intervals without ascertaining key assumptions that might not hold given the available data. In a setting where observations can be viewed as independent realizations of an underlying stochastic process, a sufficiently rich dataset can be treated as a population representing the data distribution. New datasets are then generated by resampling from the data with replacement, and the model can be refit to each new dataset using maximum-likelihood estimation. The resulting distribution of estimates can then be viewed as a representation of parametric uncertainty.

A typical workflow for the parametric bootstrap is shown in Fig. 1.2. After identifying outputs of interest and preparing the data, the parametric model is fit by some procedure such as minimizing root-mean-square-error or maximizing the likelihood. Alternate datasets are constructed by resampling from the population or by generating new samples from the fitted data-generating process. It is important at this step that the resampled quantities are independent of one another. For example, in the context of temporally- or spatially-correlated data, such as time series, the raw observations cannot be treated as independent realizations. However, the residuals resulting from fitting the model to the data could be (depending on their structure). For example, if the residuals are treated as independent, they can then be resampled with replacement, and these residuals added to the original model fit to create new realizations. If the residuals are assumed to be the result of an autoregressive process, this process could be fit to the original residual series and new residuals be created using this model [159]. The model is then refit to each new realization.

The bootstrap is computationally convenient, particularly as the process of fitting the model to each realization can be easily parallelized. This approach also requires minimal prior assumptions. However, due to the assumption that the available data are representative of the underlying data distribution, the bootstrap can neglect key uncertainties which might influence the results. For example, when using an autoregressive process to generate new residuals, uncertainty in the autocorrelation parameter and innovation variance is neglected, which may bias estimates of, for example, low-probability but high-impact events [160].

A.3 Pre-Calibration

Pre-calibration [18, 161, 162] involves the identification of a plausible set of parameters using some prespecified screening criterion, such as the distance from the model results to the observations (based on an appropriate metric for the desired matching features, such as root-mean-squared error). A typical workflow is shown in Fig. 1.3. Parameter values are obtained by systematically sampling the input space (see Chapter 3.3). After the model is evaluated at the samples, only those passing the distance criterion are retained. This selects a subset of the parameter space as "plausible" based on the screening criterion, though there is no assignment of probabilities within this plausible region.

Pre-calibration can be useful for models which are inexpensive enough that a reasonable number of samples can be used to represent the parameter space, but which are too expensive to facilitate full uncertainty quantification. High-dimensional parameter spaces, which can be problematic for the uncertainty quantification methods below, may also be explored using pre-calibration. One key prerequisite to using this method is the ability to place a meaningful distance metric on the output space.

However, pre-calibration results in a very coarse characterization of uncertainty, especially when considering a large number of parameters, as more samples are needed to fully characterize the parameter space. Due to the inability to evaluate the relative probability of regions of the parameter space beyond the binary plausible-and-implausible characterization, pre-calibration can also result in degraded hindcast and projection skills and parameter estimates [157, 163, 164].

A related method, widely used in hydrological studies, is generalized likelihood uncertainty estimation, or GLUE [18]. Unlike pre-calibration, the underlying argument for GLUE relies on the concept of equifinality [165], which posits that it is impossible to find a uniquely well-performing parameter vector for models of abstract environmental systems [165, 166]. In other words, there exist multiple parameter vectors which perform equally or similarly well. As

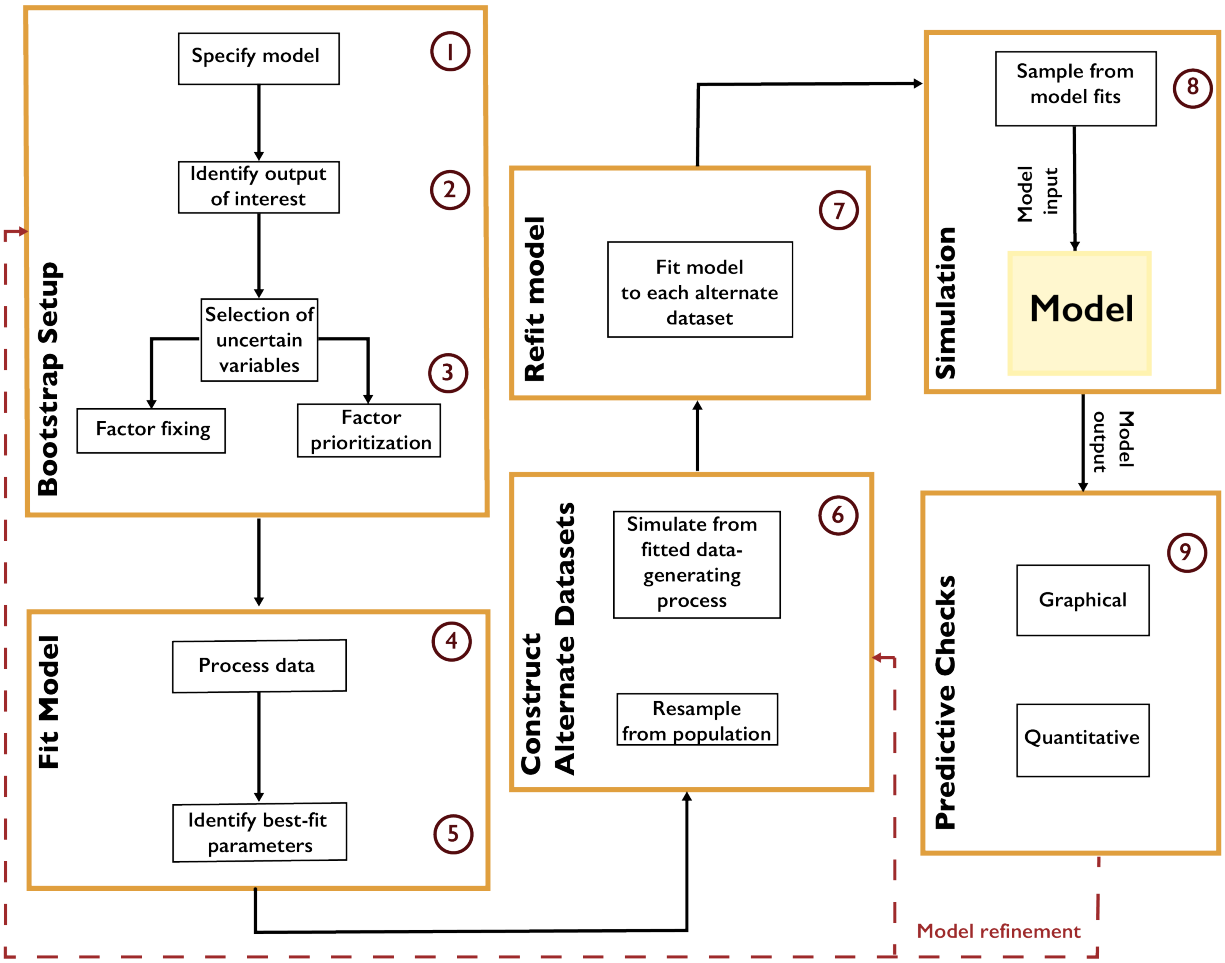


Fig. 1.2: Workflow for the parametric bootstrap.

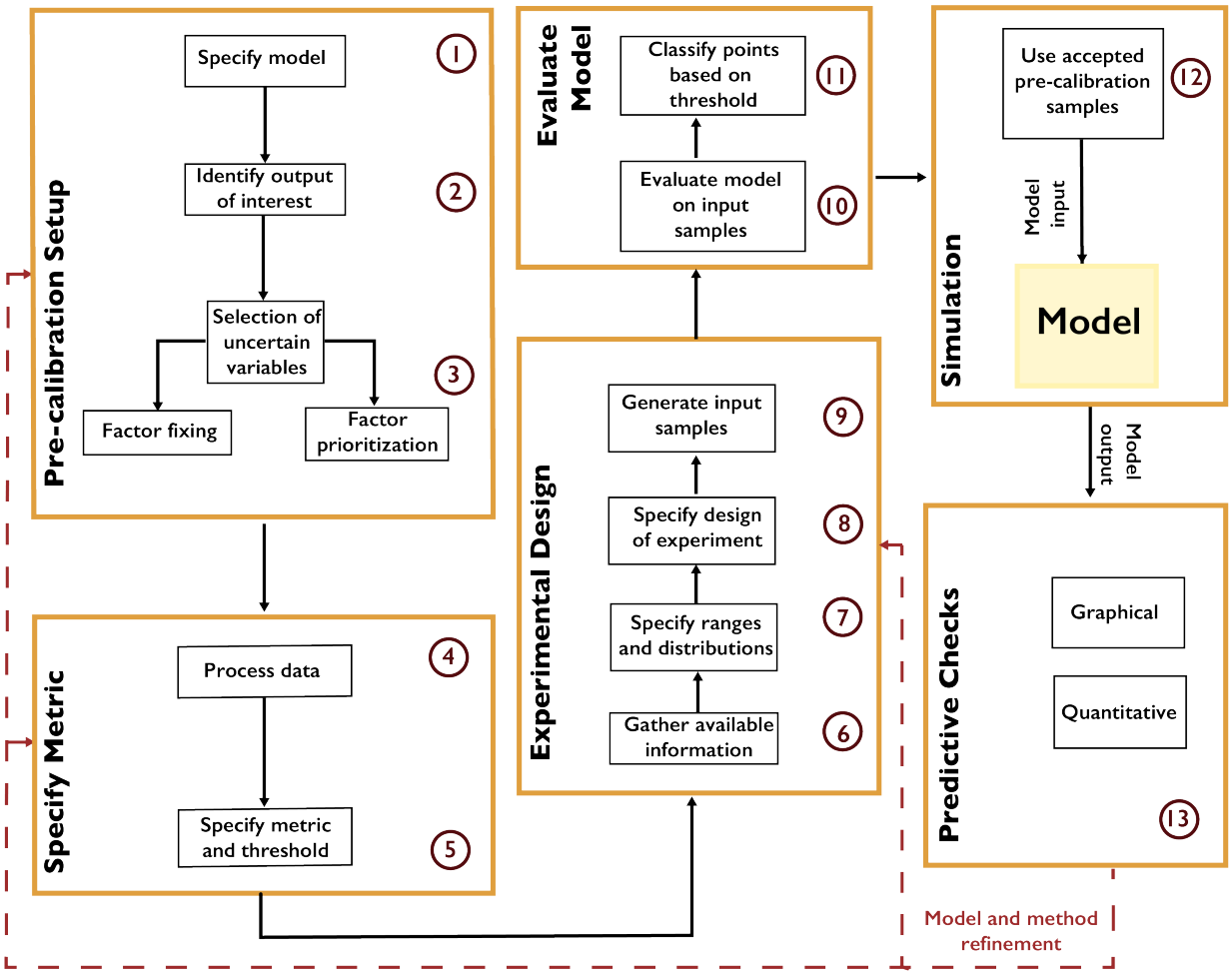


Fig. 1.3: Workflow for pre-calibration.

with pre-calibration, GLUE uses a goodness-of-fit measure (though this is called a "likelihood" in the GLUE literature, as opposed to a statistical likelihood function [167]) to evaluate samples. After setting a threshold of acceptable performance with respect to that measure, samples are evaluated and classified into "behavioral" or "non-behavioral" according to the threshold.

A.4 Markov Chain Monte Carlo

Markov chain Monte Carlo (MCMC) is a "gold standard" approach to full uncertainty quantification. MCMC refers to a category of algorithms which systematically sample from a target distribution (in this case, the posterior distribution) by constructing a Markov chain. A Markov chain is a probabilistic structure consisting of a state space, an initial probability distribution over the states, and a transition distribution between states. If a Markov chain satisfies certain properties [168, 169], the probability of being in each state will eventually converge to a stable, or stationary, distribution, regardless of the initial probabilities.

MCMC algorithms construct a Markov chain of samples from a parameter space (the combination of model and statistical parameters). This Markov chain is constructed so that the stationary distribution is a target distribution, in this case the (Bayesian) posterior distribution. As a result, after the transient period, the resulting samples can be viewed as a set of dependent samples from the posterior (the dependence is due to the autocorrelation between samples resulting from the Markov chain transitions). Expected values can be computed from these samples (for example, using batch-means estimators [170]), or the chain can be sub-sampled or thinned and the resulting samples used as independent Monte Carlo samples due to the reduced or eliminated autocorrelation.

A general workflow for MCMC is shown in Fig. 1.4. The first decision is whether to use the full model or a surrogate model (or emulator). Typical surrogates include Gaussian process emulation [171, 172], polynomial chaos expansions [173, 174], support vector machines [175, 176], and neural networks [177, 178]. Surrogate modeling can be faster, but requires a sufficient number of model evaluations for the surrogate to accurately represent the model's response surface, and this typically limits the number of parameters which can be included in the analysis.

After selecting the variables which will be treated as uncertain, the next step is to specify the likelihood based on the selected surrogate model or the structure of the data-model residuals. For example, it may not always be appropriate to treat the residuals as independent and identically distributed (as is commonly done in linear regression). A misspecification of the residual structure can result in biases and over- or under-confident inferences and projections [179].

After specifying the prior distributions (see Chapter A.6), the selected MCMC algorithm should be used to draw samples from the posterior distribution. There are many MCMC algorithms, all of which have advantages and disadvantages for a particular problem. These include the Metropolis-Hastings algorithm [169] and Hamiltonian Monte Carlo [180, 181]. Software packages typically implement one MCMC method, sometimes designed for a particular problem setting or likelihood specification. For example, R's *adaptMCMC* implements an adaptive Metropolis-Hastings algorithm [182], while *NIMBLE* [183, 184] uses a user-customizable Metropolis-Hastings implementation, as well as functionality for Gibbs sampling (which is a special case of Metropolis-Hastings where the prior distribution has a convenient mathematical form). Some recent implementations, such as *Stan* [185], *pyMC3* [186], and *Turing* [187] allow different algorithms to be used.

A main consideration when using MCMC algorithms is testing for convergence to the target distribution. As convergence is guaranteed only for a sufficiently large number of transitions, it is impossible to conclude for certain that a chain has converged for a fixed number of iterations. However, several heuristics have been developed [170, 188] to increase evidence that convergence has occurred.

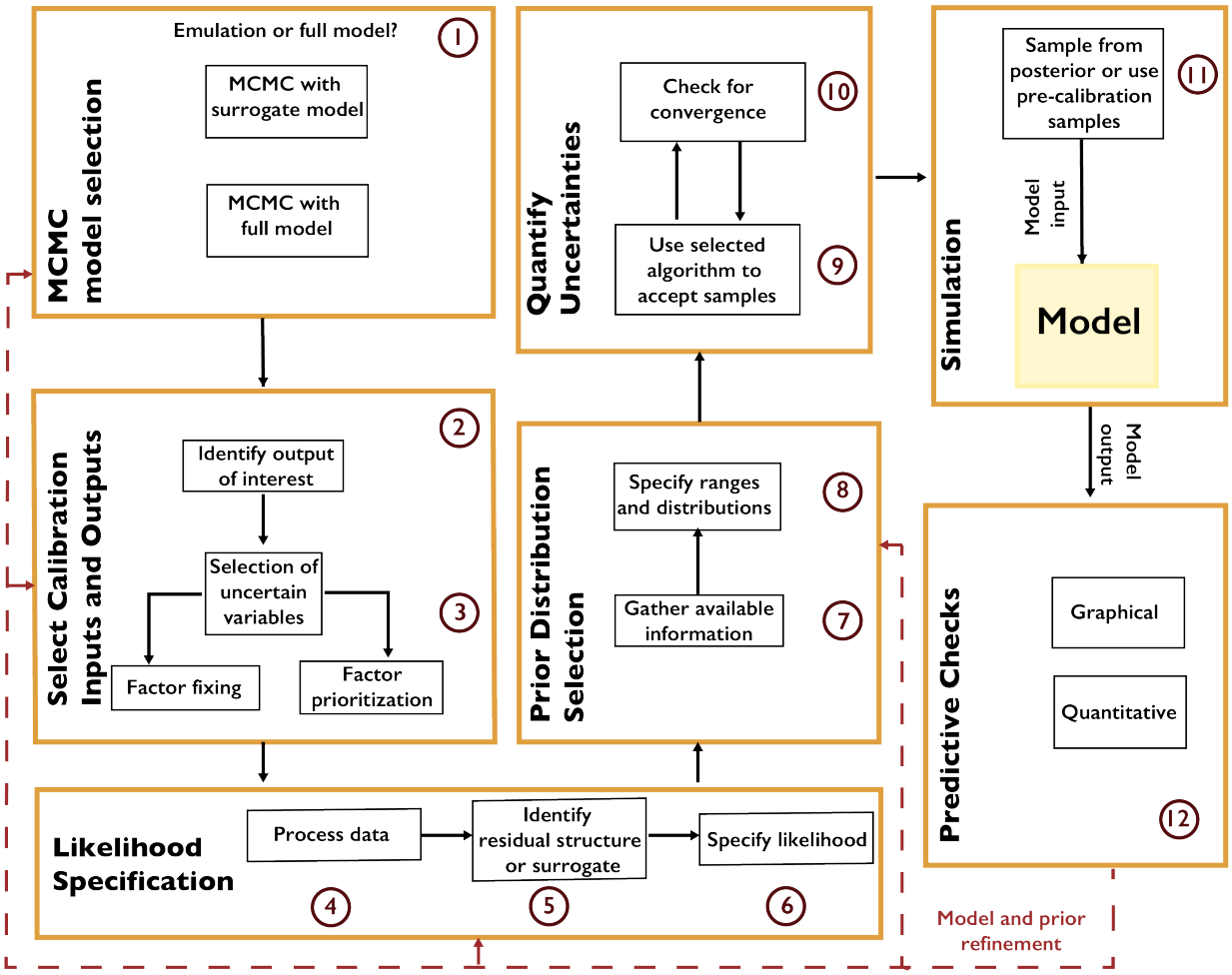


Fig. 1.4: Workflow for Markov chain Monte Carlo.

A.5 Other Methods

Other common methods for UQ exist. These include sequential Monte Carlo, otherwise known as particle filtering [189, 190, 191], where a number of particles are used to evaluate samples. An advantage of sequential Monte Carlo is that the vast majority of the computation can be parallelized, unlike with standard MCMC. A major weakness is the potential for degeneracy [190], where many particles have extremely small weights, resulting in the effective use of only a few samples.

Another method is approximate Bayesian computation (ABC) [192, 193, 194]. ABC is a likelihood-free approach that compares model output to a set of summary statistics. ABC is therefore well-suited for models and residual structures which do not lend themselves to a computationally-tractable likelihood, but the resulting inferences are known to be biased if the set of summary statistics is not sufficient, which can be difficult to know a-priori.

A.6 The Critical First Step: How to Choose a Prior Distribution

Prior distributions play an important role in Bayesian uncertainty quantification, particularly when data is limited relative to the dimension of the model. Bayesian updating can be thought of as an information filter, where each additional datum is added to the information contained in the prior; eventually, the prior makes relatively little impact. In real world problems, it can be extremely difficult to assess how much data is required for the choice of prior to become less relevant. The choice of prior can also be influential when conducting SA prior to or without UQ. This is because a prior distribution for a sensitivity analysis for a given parameter which is much wider than the region where the model response surface is sensitive to the parameter value might cause the sensitivity calculation to underestimate the response in that potentially critical region. Similarly, a prior which is too narrow may miss regions where the model responds to the parameter altogether.

Ideally, prior distributions are constructed independently of any analysis of the new data considered. This is because using data to inform the prior as well as to compute the likelihood reuses information in a potentially inappropriate way, which can lead to overconfident inferences. Following Jaynes [195], Gelman *et al.* [196] refers to the ideal prior as one which encodes all available information about the model. For practical reasons (difficulty of construction or computational inconvenience), most priors fail to achieve this ideal. These compromises mean that priors should be transparently articulated and justified, so that the impact of the choice of prior can be fully understood. When there is ambiguity about an appropriate prior, such as how fat the tails should be, an analyst should examine how sensitive the UQ results are to the choice of prior.

Priors can also be classified in terms of the information encoded by them, demonstrated in Fig. 1.5. Non-informative priors (illustrated in Fig. 1.5 (a)) allegedly correspond to (and are frequently justified by) a position of ignorance. A classic example is the use of a uniform distribution. A uniform prior can, however, be problematic, as it can lead to improper inferences by giving extremely large values the same prior probability as values which may seem more likely [196, 197], and therefore does not really reflect a state of complete ignorance. In the extreme case of a uniform prior over the entire real line, every particular region has effectively a prior weight of zero, even though not all regions are a priori unlikely [197]. Moreover, a uniform prior which excludes possible parameter values is not actually noninformative, as it assigns zero probability to those values while jumping to a nonzero probability as soon as the boundary is crossed. While a uniform prior can be problematic for the task of uncertainty quantification, it may be useful for an initial sensitivity analysis to identify the boundary of any regions where the model is sensitive to the parameter.

Informative priors strongly bound the range of probable values (illustrated in Fig. 1.5 (c)). One example is a Gaussian distribution with a relatively small standard deviation, so that large values are assigned a close to null prior probability. Another example is the jump from zero to non-zero probability occurring at the truncation point of a truncated Gaussian, which could be justified based on information that the parameter cannot take on values beyond this point. Without this type of justification, however, priors may be too informative, failing to allow the information contained in the available data to update them.

Finally, weakly informative priors (illustrated in Fig. 1.5 (b)) fall in between [196]. They regularize better than non-informative priors, but allow for more inference flexibility than fully informative priors. An example might be a Gaus-

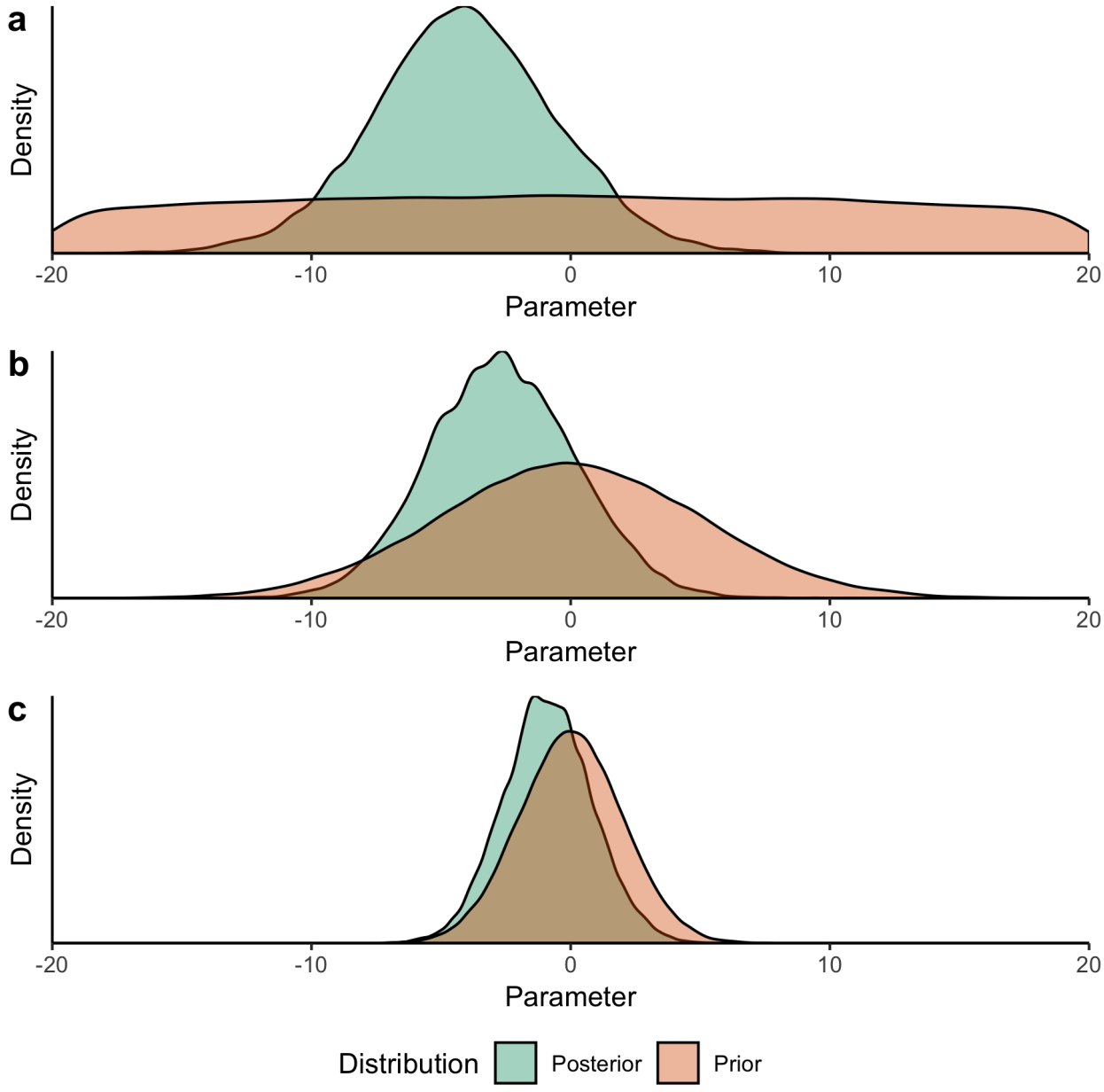


Fig. 1.5: Impact of priors on posterior inferences. These plots show the results of inference for a linear regression model with 15 data points. The true value of the parameter is equal to -3. All priors have mean 0. In panel (a), a non-informative prior allows the tails of the posterior to extend freely, which may result in unreasonably large parameter values. In panel (b), a weakly informative prior constrains the tails more, but allows them to extend without too much restriction. In panel (c), an informative prior strongly constrains the tails of the posterior and biases the inference closer towards the prior mean (the posterior mean is -0.89 in this case, and closer to -3 in the other two cases).

sian distribution with a moderate standard deviation, which still assigns negligible probability for values far away from the mean, but is less constrained than a narrow Gaussian for a reasonably large area. A key note is that it is not necessarily better to be more informative if this cannot be justified by the available information.

A.7 The Critical Final Step: Predictive Checks

Every UQ workflow requires a number of choices, potentially including selecting prior distributions, the likelihood specification, and any used numerical models. Checking the appropriateness of these choices is an essential step for sound inferences, as misspecification can produce biased results [179]. Model checking in this fashion is part of an iterative UQ process, as the results can reveal adjustments to the statistical model or the need to select a different numerical model [198, 199, 200].

A classic example is the need to check the structure of residuals for correlations. Many standard statistical models, such as linear regression, assume that the residuals are independent and identically distributed from the error distribution. The presence of correlations, including temporal autocorrelations and spatial correlations, indicates a structural mismatch between the likelihood and the data. In these cases, the likelihood should be adjusted to account for these correlations.

Checking residuals in this fashion is one example of a predictive check (or a posterior predictive check in the Bayesian setting). One way to view UQ is as a means to recover data-generating processes (associated with each parameter vector) consistent with the observations. Predictive checks compare the inferred data-generating process to the observations to determine whether the model is capable of appropriately capturing uncertainty. After conducting the UQ analysis, alternatively realized datasets are simulated from sampled parameters. These alternative datasets, or their summary statistics, can be tested against the observations to determine adequacy of the fit. Predictive checks are therefore a way of probing various model components to identify shortcomings that might result in biased inferences or poor projections, depending on the goal of the analysis.

One example of a graphical predictive check for time series models is hindcasting, where predictive intervals are constructed from the alternative datasets and plotted along with the data. Hindcasts demonstrate how well the model is capable of capturing the broader dynamics of the data, as well as whether the parameter distributions produce appropriate levels of output uncertainty. A related quantitative check is the surprise index, which calculates the percentage of data points located within a fixed predictive interval. For example, the 90% predictive interval should contain approximately 90% of the data. More uncertainty than this reflects underconfidence, while less uncertainty reflects overconfidence. This could be the result of priors that are not appropriately informative, or a likelihood that does not account for correlations between data points appropriately. It could also be the result of a numerical model that isn't sufficiently sensitive to the parameters that are treated as uncertain.

A.8 Key Take-Home Points

When appropriate, UQ is an important component of the exploratory modeling workflow. While a number of parameter sets could be consistent with observations, they may result in divergent model outputs when exposed to different future conditions. This can result in identifying risks which are not visible when selecting a "best fit" parameterization. Quantifying uncertainties also allows us to quantify the support for hypotheses, which is an essential part of the scientific process.

Due to the scale and complexity of the experiments taking place in IM3, UQ has not been extensively used. The tradeoff between the available number of function evaluations and the number of uncertain parameters illustrated in Fig. 1.1 is particularly challenging due to the increasing complexity of state-of-the-art models and the movement towards coupled, multisector models. This tradeoff can be addressed somewhat through the use of emulators and parallelizable methods. In particular, when attempting to navigate this tradeoff by limiting the number of uncertain parameters, it is important to carefully iterate with sensitivity analyses to ensure that critical parameters are identified.

Specifying prior distributions and likelihoods is an ongoing challenge. Prior distributions, in particular, should be treated as deeply uncertain when appropriate. One key advantage of the methods described in this chapter is that they have the potential for increased transparency. When it is not possible to conduct a sensitivity analysis on a number of critical priors due to limited computational budgets, fully specifying and providing a justification for the utilized distributions allows other researchers to identify key assumptions and build on existing work. The same is true for the specification of likelihoods—while likelihood-free methods avoid the need to specify a likelihood function, they require other assumptions or choices, which should be described and justified as transparently as possible.

We conclude this appendix with some key recommendations: 1. UQ analysis does not require full confidence in priors and likelihoods. Rather, UQ should be treated as part of an exploratory modeling workflow, where hypotheses related to model structures, prior distributions, and likelihoods can be tested. 2. For complex multisectoral models, UQ will typically require the use of a reduced set of parameters, either through emulation or by fixing the others to their best-fit values. These parameters should be selected through a careful sensitivity analysis. 3. Avoid the use of supposedly "non-informative" priors, such as uniform priors, whenever possible. In the absence of strong information about parameter values, the use of weakly informative priors, such as diffuse normals, is preferable. 4. Be cognizant of the limitations of conclusions that can be drawn by using each method. The bootstrap, for example, may result in overconfidence if the dataset is limited and is not truly representative of the underlying stochastic process. 5. When using MCMC, Markov chains can not be shown to have converged to the target distribution, but rather evidence can be collected to demonstrate that it is likely that they have. 6. Conduct predictive checks based on the assumptions underlying the choices made in the analysis, and iteratively update those choices if the assumptions prove to be ill-suited for the problem at hand.

APPENDIX

B

JUPYTER NOTEBOOK TUTORIALS

B.1 Fishery Dynamics Tutorial

Note:

Run the tutorial interactively: Fishery Dynamics Notebook.
Please be aware that notebooks can take a couple minutes to launch.
To run the notebooks yourself, download the files here and use these requirements.

B.1.1 Tutorial: Sensitivity Analysis (SA) to discover factors shaping consequential dynamics

This notebook demonstrates the application of sensitivity analysis to discover factors that shape the behavior modes of a socio-ecological system with dynamic human action.

The system of differential equations below represent a system of prey (defined in the equation below as x) and predator (defined as y) fish, with a human actor harvesting the prey fish. You can read more about this system at Hadjimichael et al. (2020).

$$\frac{\mathrm{d}x}{\mathrm{d}t} = bx\left(1 - \frac{x}{K}\right) - \frac{\alpha xy}{y^m + \alpha hx} - zx$$

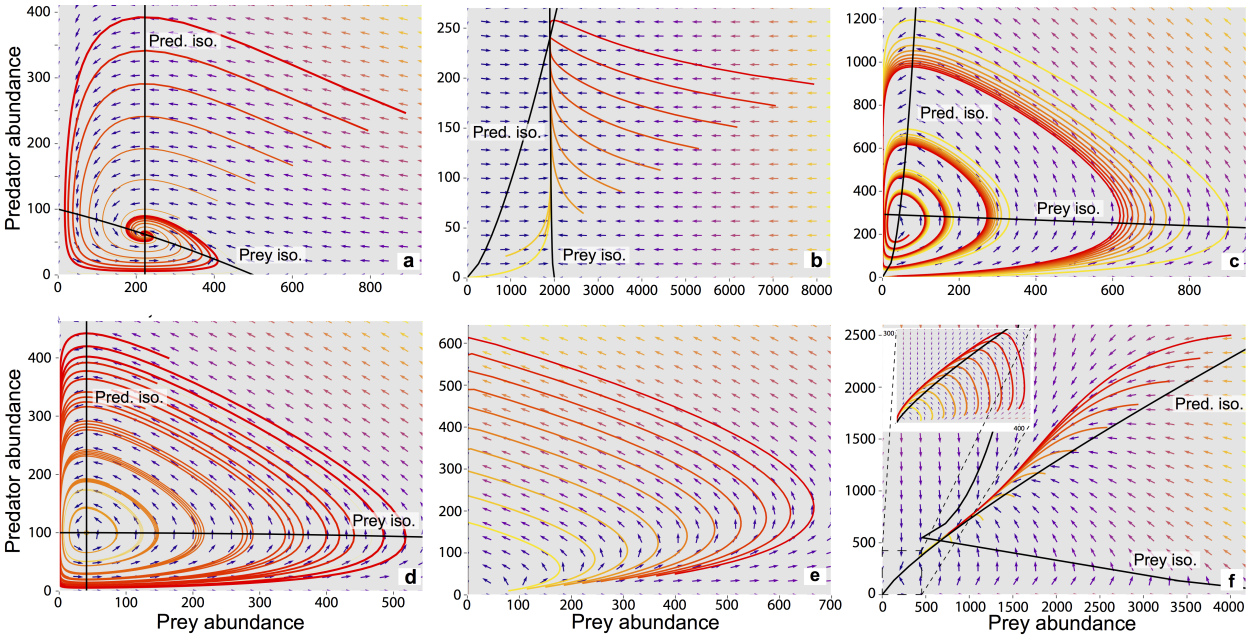
$$\frac{\mathrm{d}y}{\mathrm{d}t} = \frac{c\alpha xy}{y^m + \alpha hx} - dy$$

The table below defines the parameters in the system and also denotes the baseline and ranges associated with each uncertain parameter.

Parameter	Description	Unit	Base value	Minimum	Maximum
α	Rate at which the prey is available to the predator	1/time	0.005	0.002	2
b	Prey growth rate	1/time	0.5	0.005	1
c	Rate with which consumed prey is converted to predator abundance	mass/mass*	0.5	0.2	1
d	Predator death rate	1/time	0.1	0.05	0.2
h	Handling time (time each predator needs to consume the caught prey)	time	0.1	0.001	1
K	Prey carrying capacity given its environmental conditions	mass*	2000	100	5000
m	Predator interference parameter	mass/mass*	0.7	0.1	1.5
σ_x	Standard deviation of stochastic noise in prey population	mass^{2*}	0.004	0.001	0.01
σ_y	Standard deviation of stochastic noise in predator population	mass^{2*}	0.004	0.001	0.01

*The units of these parameters depend on the units used to measure prey (x) and predator (y) abundance. If prey and predator abundance is measured in volume then these units would equivalently change.

The system is simple but very rich in the dynamic behaviors it exhibits. This complexity is accompanied by the presence of several equilibria that come in and out of existence with different parameter values. The equilibria also change in their stability according to different parameter values, giving rise to different behavior modes as shown by the diverse predator and prey abundace trajectories in the figure below.



In the unharvested system (without the human actor) the stability of several of these equilibria can be derived analytically. The task becomes significantly more difficult when the adaptive human actor is introduced, deciding to harvest the system at different rates according to their objectives and preferences.

Sensitivity analysis methods can help us identify the factors that most control these dynamics by exploring the space of parameter values and seeing how system outputs change as a result.

Through previously conducted optimization, there already exists a set of potential harvesting strategies that were identified in pursuit of five objectives:

- Maximize Harvesting Discounted Profits (Net Present Value)
- Minimize Prey Population Deficit
- Minimize Longest Duration of Consecutive Low Harvest

- Maximize Worst Harvest Instance
- Minimize Harvest Variance

The identified harvesting strategies also meet the necessary constraint of not causing inadvertent predator collapse.

We will be examining the effects of parametric uncertainty on these identified strategies, particularly focusing on two strategies: one selected to maximize harvesting profits and one identified through previous analysis to perform 'well enough' for all objectives across a wide range of states of the world (referred to as the 'robust' harvesting policy).

Let's get started!

In this tutorial, we will be loading in data that has been produced in Hadjimichael et al. (2020). Before we start our analysis, we'll load the relevant Python libraries.

```
# Import necessary libraries

import msdbook
import numpy as np
import matplotlib.pyplot as plt
from SALib.sample import saltelli
from SALib.analyze import sobol
from matplotlib import patheffects as pe

# load example data
from msdbook.install_supplement import install_package_data
install_package_data()

%matplotlib inline
%config InlineBackend.print_figure_kwargs = {'bbox_inches':None}
```

```
Downloading example data for msdbook version 0.1.5...
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/uncertain_
↪params_bounds.txt
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/sa_metric_s1.
↪npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/sa_vary_
↪delta.npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/sa_by_mth_s1.
↪npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/solutions.
↪resultfile
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/3704614_
↪heatmap.npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/LHsamples_
↪original_1000.txt
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/3704614_
↪pseudo_r_scores.csv
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/param_values.
↪csv
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/sa_by_yr_s1.
↪npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/sa_by_mth_
↪delta.npy
```

(continues on next page)

(continued from previous page)

```
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/7000550_
↪pseudo_r_scores.csv
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/collapse_
↪days.csv
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/hymod_params_
↪256samples.npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/sa_vary_s1.
↪npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/7000550_
↪heatmap.npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/7200799_
↪heatmap.npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/sa_by_yr_
↪delta.npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/7200799_
↪pseudo_r_scores.csv
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/LeafCatch.csv
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/hymod_
↪simulations_256samples.csv
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/Robustness.
↪txt
```

Step 1: Load identified solutions and explore performance

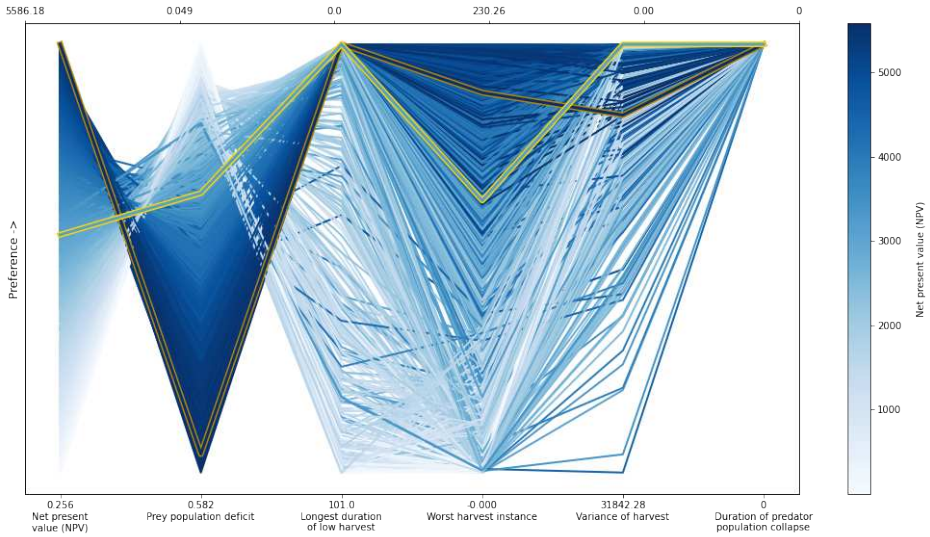
Here we load in the solution set obtained in Hadjimichael et al. (2020). The solution set contains the decision variables and objectives associated with a variety of harvesting policies. For this tutorial, we focus on comparing two policies: harvesting profits and one that performs robustly across all objectives. Below, we are reading in the decision variables and objectives from an external file that can be found within the msdbook package data.

```
robustness = msdbook.load_robustness_data()
results = msdbook.load_profit_maximization_data()

robust_solution = np.argmax(robustness[:,-1]) #pick robust solution
profit_solution = np.argmin(results[:,6]) #pick profitable solution
objective_performance = -results[:,6:] #Retain objective values

# Get decision variables for each of the policies
highprofitpolicy = results[profit_solution,0:6]
mostrobustpolicy = results[robust_solution,0:6]
```

Next we plot the identified solutions with regards to their objective performance in a parallel axis plot

Tip: View the source code used to create this plot here: plot_objective_performance

```
ax, ax1 = msdbook.plot_objective_performance(objective_performance, profit_solution,␣
↪robust_solution)
```



The solution set from the optimization in Hadjimichael et al. (2020) are presented in a parallel axis plot where each of the five objectives (and one constraint) are represented as an axis. Each solution on the Pareto front is represented as a line where the color of the line indicates the value of the NPV objective. The preference for objective values is in the upward direction. Therefore, the ideal solution would be a line straight across the top of the plot that satisfies every objective. However, no such line exists because there are tradeoffs when sets of objectives are prioritized over the others. When lines cross in between axes, this indicates a tradeoff between objectives (as seen in the first two axes).The solution that is most robust in the NPV objective has the highest value on the first axis and is outlined in dark gold. The solution that is most robust across all objectives is outlined in a brighter yellow. A parallel axis is an effective visual to characterize high-dimensional tradeoffs in the system and visualize differences in performance across policies.

Step 2: Use SALib to generate a sample for a Sobol sensitivity analysis

In Step 1, we showed how the optimized harvesting policies performed in the objective space, which utilized the baseline parameters outlined in the table above. Now, we are interested in understanding how sensitive our two policies are to alternative states of the world that may be characterized by different parameter values. To do so, we first need to define the problem dictionary that allows us to generate these alternative states of the world.

```
# Set up SALib problem
problem = {
  'num_vars': 9,
  'names': ['a', 'b', 'c', 'd', 'h', 'K', 'm', 'sigmaX', 'sigmaY'],
  'bounds': [[0.002, 2], [0.005, 1], [0.2, 1], [0.05, 0.2], [0.001, 1],
             [100, 5000], [0.1, 1.5], [0.001, 0.01], [0.001, 0.01]]
}
```

Then we use the following command to generate a Saltelli sample from these defined ranges:

```
param_values = saltelli.sample(problem, 1024, calc_second_order=False)
```

Generally, it is a good idea to save the result of the sample since it is often reused and regenerating it produces a different sample set. For this reason, we will load one from file that was previously generated.

```
# load previously generated Saltelli sample from our msdbook package data
param_values = msdbook.load_saltelli_param_values()
```

Step 3: Evaluate the system over all generated states of the world

Now we re-evaluate how well the policies do in the new states of the world. In order to characterize failure of a policy, we identify the states where the predator population collapses, as an inadvertent consequence of applying the harvesting strategy under a state of the world different from the one originally assumed. Due to how long this step takes to execute within the tutorial, we will read in the solutions from an external file. However, the block of code below shows how evaluation can be implemented.

```
# create array to store collapse values under both policies
collapse_days = np.zeros([len(param_values), 2])

# evaluate performance under every state
for i in range(len(param_values)):

    additional_inputs = np.append(['Previous_Prey'],
                                  [param_values[i,0],
                                   param_values[i,1],
                                   param_values[i,2],
                                   param_values[i,3],
                                   param_values[i,4],
                                   param_values[i,5],
                                   param_values[i,6],
                                   param_values[i,7],
                                   param_values[i,8]])

    collapse_days[i,0]=fish_game(highprofitpolicy, additional_inputs)[1][0]
    collapse_days[i,1]=fish_game(mostrobustpolicy, additional_inputs)[1][0]
```

```
# load the simulation data from our msdbook package data
collapse_days = msdbook.load_collapse_data()
```

Step 4: Calculate sensitivity indices

Now we use a Sobol sensitivity analysis to calculate first-order, second-order, and total-order sensitivity indices for each parameter and for each of the two policies. These indicies help determine which factors explain the most variability in the number of days of predator population collapse.

```
#Perform the Sobol SA for the profit-maximizing solution
Si_profit = sobol.analyze(problem, collapse_days[:, 0],
                          calc_second_order=False,
                          conf_level=0.95,
                          print_to_console=True)
```

```
#Perform the Sobol SA for the robust solution
Si_robustness = sobol.analyze(problem,
                              collapse_days[:, 1],
                              calc_second_order=False,
```

(continues on next page)

(continued from previous page)

```
                                conf_level=0.95,
                                print_to_console=True)
```

```
              ST   ST_conf
a       0.226402  0.036146
b       0.066819  0.013347
c       0.004395  0.004023
d       0.024509  0.006993
h       0.009765  0.005488
K       0.020625  0.009494
m       0.897971  0.066470
sigmaX  0.000136  0.000149
sigmaY  0.000739  0.001040
              S1   S1_conf
a       0.087936  0.044236
b       0.000554  0.021474
c      -0.002970  0.004590
d       0.001206  0.015881
h       0.004554  0.007998
K       0.003843  0.012661
m       0.751301  0.071862
sigmaX -0.000325  0.001245
sigmaY -0.001887  0.002768
```

Looking at the total-order indices, (ST) factors m, a, b, d and K explain a non-negligible amount of variance therefore have an effect on the stability of this system. Looking at the first-order indices (S1), we also see that besides factors m and a, all other factors are important in this system through their interactions, which make up the difference between their S1 and ST indices. This shows the danger of limiting sensitivity analyses to first order effects, as factor importance might be significantly misjudged.

These findings are supported by the analytical condition of equilibrium stability in this system:

$$\alpha(hK)^{1-m} < (b)^m$$

In an unharvested system, this condition is both necessary and sufficient for the equilibrium of the two species coexisting to be stable.

When adaptive human action is introduced however, this condition is still necessary, but no longer sufficient, as harvesting reduces the numbers of prey fish and as a result reduces the resources for the predator fish. Since this harvesting value is not constant, but can dynamically adapt according to the harvester's objectives, it cannot be introduced into this simple equation.

Step 5: Explore relationship between uncertain factors and performance

In the following steps, we will use the results of our sensitivity analysis to investigate the relationships between parametric uncertainty, equilibrium stability and the performance of the two policies.

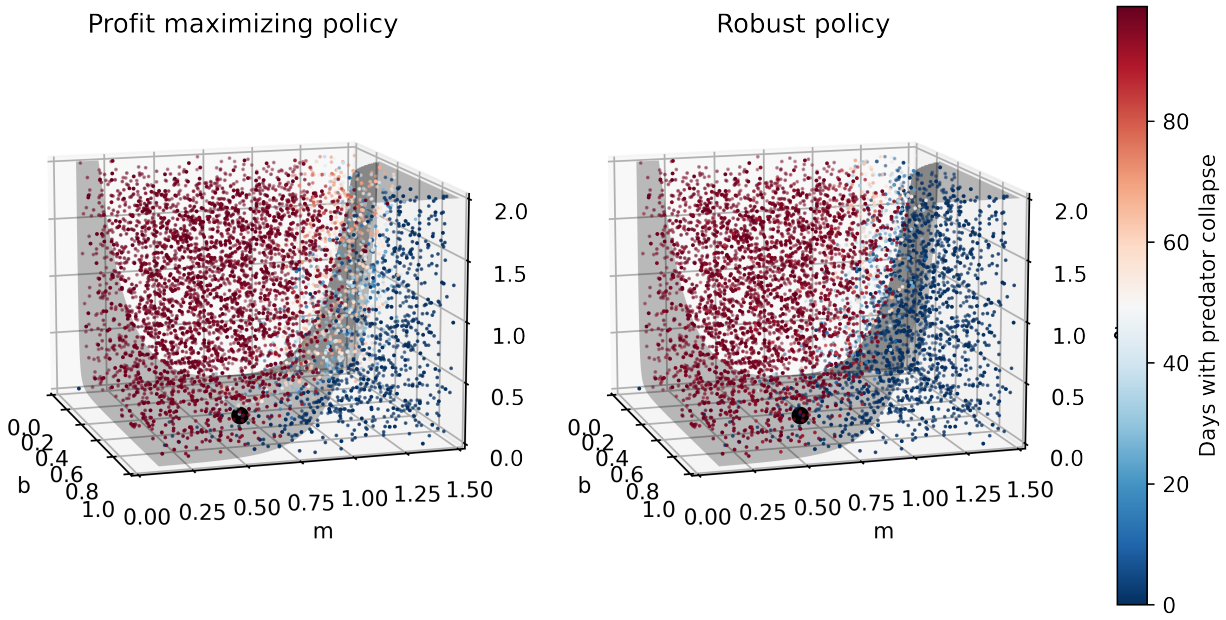
We can use the top three factors identified (m, a, and b) to visualize the performance of our policies in this three-dimensional parametric space.

We first define the stability condition, as a function of b and m, and calculate the corresponding values of a.

```
def inequality(b, m, h, K):
    return ((b**m)/(h*K)**(1-m))

# boundary interval that separates successful and failed states of the world
b = np.linspace(start=0.005, stop=1, num=1000)
m = np.linspace(start=0.1, stop=1.5, num=1000)
h = np.linspace(start=0.001, stop=1, num=1000)
K = np.linspace(start=100, stop=2000, num=1000)
b, m = np.meshgrid(b, m)
a = inequality(b, m, h, K)
a = a.clip(0,2)
```

Tip: View the source code used to create this plot here: plot_factor_performance

```
# generate plot
ax1, ax2 = msdbook.plot_factor_performance(param_values, collapse_days, b, m, a)
```



These figures show the combinations of factors that lead to success or failure in different states of the world for the profit-maximizing and robust policies. Each point is a state of the world, characterized by specific values of the parameters, and ideally, we would like the color of the point to be blue, to represent that there are a low number of days with a predator collapse in that world. The gray curve denotes the highly non-linear nature of the boundary, defined by the stability condition, that separates successful and failed states of the world. The figures demonstrate the following key points:

First, as asserted above, the policies interact with the system in different and complex ways. In the presence of human action, the stability condition is not sufficient in determining whether the policy will succeed, even though it clearly shapes the system in a fundamental manner.

Secondly, the robust policy manages to avoid collapse in many more of the sampled states of the world, indicated by the number of blue points. The robust policy avoids collapse in 31% of worlds versus 14% in the profit-maximizing policy.This presents a clear tradeoff between profit-maximizing performance androbustness against uncertainty.

Tips to Apply Sobol SA and Scenario Discovery to your Problem

In this tutorial, we demonstrated a Sobol SA to identify the most important factors driving the behavior of a system (i.e. the number of the collapse days). In order to apply this methodology to your problem, you will need to have a set of optimized policies for your system that you are interested in analyzing. The general workflow is as follows:

1. Choose sampling bounds for your parameters and set up the problem dictionary as in Step 2 above.
2. Generate samples, or alternative states of the world using the `saltelli.sample` function.
3. Evaluate your policies on the alternative states of the world. For your application, you will also need to develop a rule for determining success or failure of your policy in a new SOW. In this tutorial, success was denoted by a small number of collapse days. Ultimately, the rule will be specific to your application and can include various satisficing criteria.
4. Calculate the Sobol indices and discover the most important parameters driving success and failure.
5. Finally, use a similar plotting procedure as in step 5 to identify the combination of parameter values that lead to success and failure in the system.

B.2 Sobol SA Tutorial

Note:

Run the tutorial interactively: Sobol SA Tutorial.

Please be aware that notebooks can take a couple minutes to launch.

To run the notebooks yourself, download the files here and use these requirements.

B.2.1 Tutorial: Sensitivity Analysis (SA) using the Saltelli sampling scheme with Sobol SA

In this tutorial, we will set up a workflow to investigate how sensitive the output of a function is to its inputs. Why might you want to do this? Imagine that this function represents a complex system, such as the rainfall-runoff process of a watershed model, and that you, the researcher, want to investigate how your choice of input parameter values are affecting the model's characterization of runoff in the watershed. Your parameter values are likely uncertain and can take on any value in a pre-defined range. Using a Sobol SA will allow you to sample different values of your parameters and calculate how sensitive your output of interest is to certain parameters. Below, we demonstrate Sobol SA for a simple function to illustrate the method, but the workflow can be applied to your own problem of interest!

In order to conduct this analysis, we will use the popular Python Sensitivity Analysis Library (SALib) to:

1. Generate a problem set as a dictionary for our Ishigami function that has three inputs

2. Generate 2048 samples for our problem set using the Saltelli[1,2] sampling scheme
3. Execute the Ishigami function for each of our samples and gather the outputs
4. Compute the sensitivity analysis to generate first-order and total-order sensitivity indices using the Sobol[3] method
5. Interpret the meaning of our results

Let's get started!

NOTE: Content from this tutorial is taken directly from the SALib "Basics" walkthrough.

```
# Import relevant libraries
import matplotlib.pyplot as plt

import SALib.sample.sobol as sobol_sample
from SALib.analyze import sobol
from SALib.test_functions import Ishigami
```

Step 1: Generate the problem dictionary

The Ishigami function is of the form:

$$f(x_1, x_2, x_3) = sin(x_1) + asin^2(x_2) + bx_3^4 sin(x_1)$$

The function has three inputs, 1, 2, 3 where [,]. The constants a and b are defined as 7.0 and 0.1 respectively.

```
#Create a problem dictionary. Here we supply the number of variables, the names of each␣
↪variable, and the bounds of the variables.
problem = {
    'num_vars': 3,
    'names': ['x1', 'x2', 'x3'],
    'bounds': [[-3.14159265359, 3.14159265359],
               [-3.14159265359, 3.14159265359],
               [-3.14159265359, 3.14159265359]]
}
```

Step 2: Generate samples using the Saltelli sampling scheme

Sobol SA requires the use of the Saltelli sampling scheme. The output of the `saltelli.sample` function is a NumPy array that is of shape 2048 by 3. The sampler generates (2+2) samples, where in this example, N is 256 (the argument we supplied) and D is 3 (the number of model inputs), yielding 2048 samples. The keyword argument `calc_second_order=False` will exclude second-order indices, resulting in a smaller sample matrix with (+2) rows instead. Below, we plot the resulting Saltelli sample.

[1] Saltelli, A. (2002). "Making best use of model evaluations to compute sensitivity indices." Computer Physics Communications, 145(2):280-297, doi:10.1016/S0010-4655(02)00280-1.

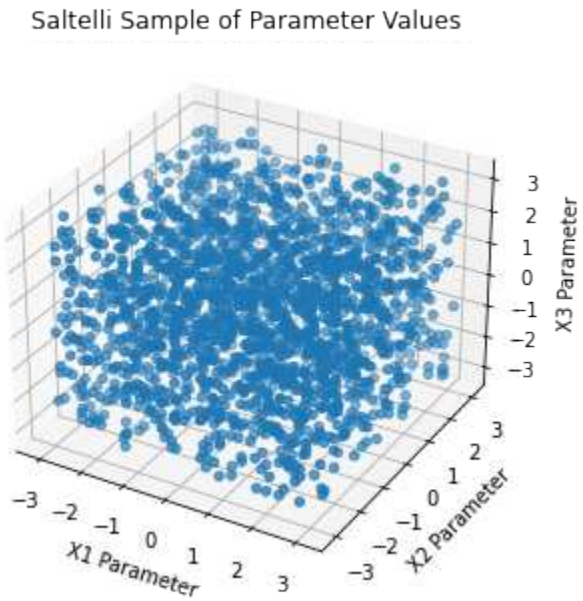
[2] Saltelli, A., P. Annoni, I. Azzini, F. Campolongo, M. Ratto, and S. Tarantola (2010). "Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index." Computer Physics Communications, 181(2):259-270, doi:10.1016/j.cpc.2009.09.018.

[3] 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, doi:10.1016/S0378-4754(00)00270-6.

```
# Generate parmeter values using the saltelli.sample function
param_values = sobol_sample.sample(problem, 256)


print(f"`param_values` shape:  {param_values.shape}")
```

param_values shape: (2048, 3)

```
# Plot the 2048 samples of the parameters

fig = plt.figure(figsize = (7, 5))
ax = plt.axes(projection ="3d")
ax.scatter3D(param_values[:,0], param_values[:,1], param_values[:,2])
ax.set_xlabel('X1 Parameter')
ax.set_ylabel('X2 Parameter')
ax.set_zlabel('X3 Parameter')
plt.title("Saltelli Sample of Parameter Values")

plt.show()
```



Step 3: Execute the Ishigami function over our sample set

SALib provides a nice wrapper to the Ishigami function that allows the user to directly pass the `param_values` array we just generated into the function directly.

```
Y = Ishigami.evaluate(param_values)
```

Step 4: Compute first-, second-, and total-order sensitivity indices using the Sobol method

The `sobol.analyze` function will use our problem dictionary and the result of the Ishigami runs (`Y`) to compute first-, second-, and total-order indicies.

```
Si = sobol.analyze(problem, Y)
```

`Si` is a Python dict with the keys "S1", "S2", "ST", "S1_conf", "S2_conf", and "ST_conf". The `_conf` keys store the corresponding confidence intervals, typically with a confidence level of 95%. Use the keyword argument `print_to_console=True` to print all indices. Or, we can print the individual values from `Si` as shown in the next step.

Step 5: Interpret our results

We execute the following code and take a look at our first-order indices (`S1`) for each of our three inputs. These indicies can be interpreted as the fraction of variance in the output that is explained by each input individually.

```
first_order = Si['S1']

print('First-order:')
print(f"x1: {first_order[0]}, x2: {first_order[1]}, x3: {first_order[2]}")
```

```
First-order:
x1: 0.3184242969763115, x2: 0.4303808201623416, x3: 0.022687722804980225
```

If we were to rank the importance of the inputs in how much they individually explain the variance in the output, we would rank them from greatest to least importance as follows: 2, 1 and then 3. Since 3 only explains 2% of the output variance, it does not explain output variability meaningfully. Thus, this indicates that there is contribution to the output variance by 2 and 1 independently, whereas 3 does not contribute to the output variance. Determining what inputs are most important or what index value is meaningful is a common question, but one for which there is no general rule or threshold. This question is problem and context-dependent, but procedures have been identified to rank order influential inputs and which can be used to identify the least influential factors. These factors can be fixed to simplify the model[4][5][6].

Next, we evaluate the total-order indices, which measure the contribution to the output variance caused by varying the model input, including both its first-order effects (the input varying alone) and all higher-order interactions across the input parameters.

[4] T. H. Andres, "Sampling methods and sensitivity analysis for large parameter sets," Journal of Statistical Computation and Simulation, vol. 57, no. 1–4, pp. 77–110, Apr. 1997, doi:10.1080/00949659708811804.

[5] Y. Tang, P. Reed, T. Wagener, and K. van Werkhoven, "Comparing sensitivity analysis methods to advance lumped watershed model identification and evaluation," Hydrology and Earth System Sciences, vol. 11, no. 2, pp. 793–817, Feb. 2007, doi:10.5194/hess-11-793-2007.

[6] J. Nossent, P. Elsen, and W. Bauwens, "Sobol' sensitivity analysis of a complex environmental model," Environmental Modelling & Software, vol. 26, no. 12, pp. 1515–1525, Dec. 2011, doi:10.1016/j.envsoft.2011.08.010.

```
total_order = Si['ST']

print('Total-order:')
print(f"x1: {total_order[0]}, x2: {total_order[1]}, x3: {total_order[2]}")
```

```
Total-order:
x1: 0.5184119098161343, x2: 0.41021260250026054, x3: 0.2299058431439953
```

The magnitude of the total order indices are substantially larger than the first-order indices, which reveals that higher-order interactions are occurring, i.e. that the interactions across inputs are also explaining some of the total variance in the output. Note that 3 has non-negligible total-order indices, which indicates that it is not a consequential parameter when considered in isolation, but becomes consequential and explains 25% of variance in the output through its interactions with 1 and 2.

Finally, we can investigate these higher order interactions by viewing the second-order indices. The second-order indicies measure the contribution to the output variance caused by the interaction between any two model inputs. Some computing error can appear in these sensitivity indices, such as negative values. Typically, these computing errors shrink as the number of samples increases.

```
second_order = Si['S2']

print("Second-order:")
print(f"x1-x2:  {second_order[0,1]}")
print(f"x1-x3:  {second_order[0,2]}")
print(f"x2-x3:  {second_order[1,2]}")
```

```
Second-order:
x1-x2:  -0.043237389723234154
x1-x3:  0.17506452088709862
x2-x3:  -0.03430682392607577
```

We can see that there are strong interactions between 1 and 3. Note that in the Ishigami function, these two variables are multiplied in the last term of the function, which leads to interactive effects. If we were considering first order indices alone, we would erroneously assume that 3 explains no variance in the output, but the second-order and total order indices reveal that this is not the case. It's easy to understand where we might see interactive effects in the case of the simple Ishigami function. However, it's important to remember that in more complex systems, there may be many higher-order interactions that are not apparent, but could be extremely consequential in explaining the variance of the output.

Tips to Apply Sobol SA to Your Own Problem

In this tutorial, we demonstrated how to apply an SA analysis to a simple mathematical test function. In order to apply a Sobol SA to your own problem, you will follow the same general workflow that we defined above. You will need to:

1. Choose sampling bounds for your parameters and set up the problem dictionary as in Step 1 above.
2. Generate samples using the `saltelli.sample` function. This step is problem-dependent and note that the Sobol method can be computationally intensive depending on the model being analyzed. For example, for a simple rainfall-runoff model such as HYMOD, it has been recommended to run a sample size of at least N = 10,000 (which translates to 60,000 model runs). More complex models will be slower to run and will also require more samples to calculate accurate estimates of Sobol indices. Once you complete this process, pay attention to the confidence bounds on your sensitivity indices to see whether you need to run more samples.

3. Run the parameter sets through your model. In the example above, the Ishigami function could be evaluated through SALib since it is a built in function. For your application, you will need to run these parameter sets through the problem externally and save the output. The output file should contain one row of output values for each model run.
4. Calculate the Sobol indices. Now, the Y will be a numpy array with your external model output and you will need to include the parameter samples as an additional argument.
5. Finally, we interpet the results. If the confidence intervals of your dominant indices are larger than roughly 10% of the value itself, you may want to consider increasing your sample size as computation permits. You should additionally read the references noted in Step 5 above to understand more about identifying important factors.

B.3 Logistic Regression Tutorial

Note:

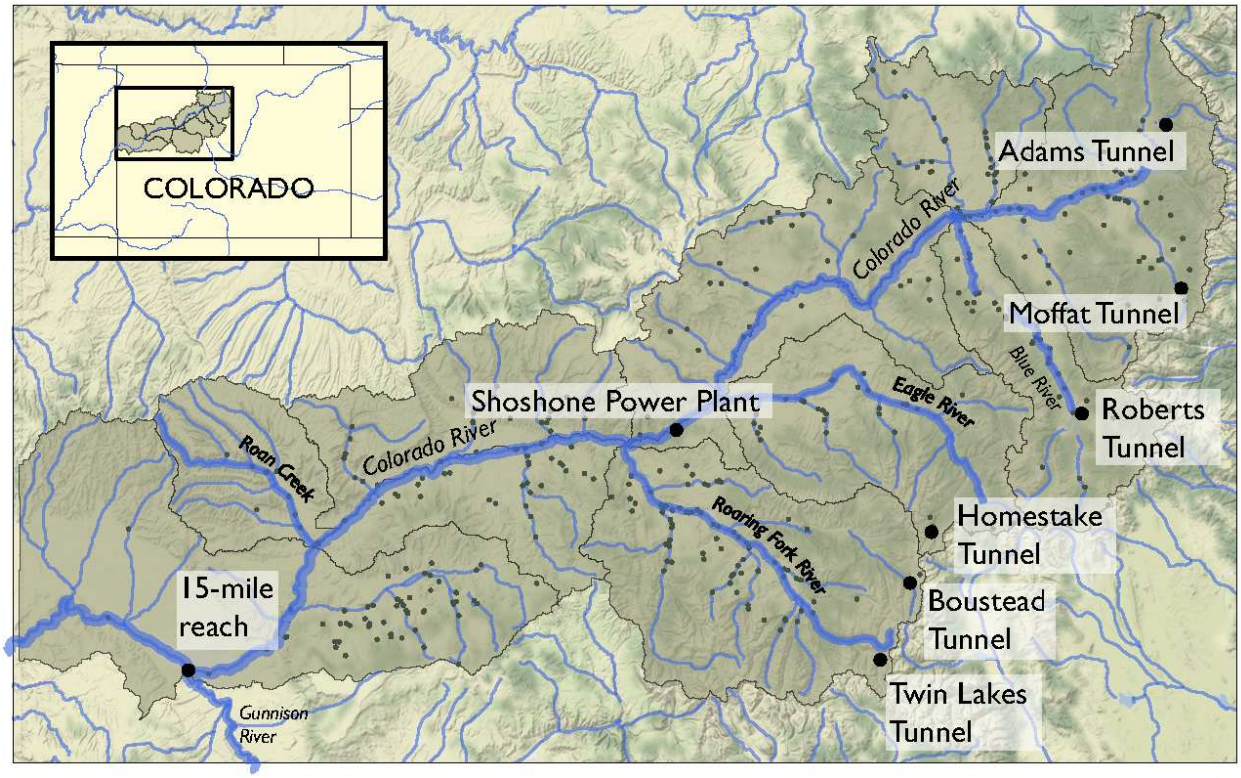
Run the tutorial interactively: Logistic Regression Tutorial.
Please be aware that notebooks can take a couple minutes to launch.
To run the notebooks yourself, download the files here and use these requirements.

B.3.1 Tutorial: Logistic Regression for Factor Mapping

This tutorial replicates a scenario discovery analysis performed in Hadjimichael et al. (2020).

Background

Planners in the the Upper Colorado River Basin (UCRB, shown in the figure below) are seeking to understand the vulnerability of water users to uncertainties stemming from climate change, population growth and water policy changes. The UCRB spans 25,682 km2 in western Colorado and is home to approximately 300,000 residents and 1,012 km2 of irrigated land. Several thousand irrigation ditches divert water from the main river and its tributaties for irrigation (shown as small black dots in the figure). Transmountain diversions of approximately 567,400,000 m3 per year are exported for irrigation, industrial and municipal uses in northern and eastern Colorado, serving the major population centers of Denver and Colorado Springs. These diversions are carried through tunnels, shown as large black dots in the figure.



An important planning consideration is the water rights of each user, defined by seniority across all water uses (irrigation diversions, transboundary diversions, power plants etc.) in the basin. To assess the vulnerability of users with varying degrees of water rights seniority, planners simulate the system across an ensemble of scenarios using the state of Colorado's StateMod platform. The model simulates streamflow, diversions, instream demands, and reservoir operations.

Hadjimichael et al. (2020) employ an exploratory analysis by simulating a large ensemble of plausible scenarios using StateMod and then identifying consequential decision-relevant combinations of uncertain factors, termed scenario discovery. Focusing on decision-relevant metrics (metrics that are important to the user, the scenario discovery is applied to the water shortages experienced by each individual user (i.e., not on a single basin-wide or sector-wide metric). For this training example, we'll be performing scenario discovery for three different water users: two irrigation users and one municipal user.

Let's get started!

In this tutorial, we will be loading in data that has been produced in Hadjimichael et al. (2020). Before we start our analysis, we'll load the relevant Python libraries, example data, and information for the three users.

```
# Import libraries
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt

from msdbook.package_data import load_basin_param_bounds, load_lhs_basin_sample, load_
↪user_heatmap_array, load_user_pseudo_scores
from msdbook.utils import fit_logit, plot_contour_map
```

(continues on next page)

(continued from previous page)

```
# Select the IDs for the three users that we will perform the analysis for
all_IDs = ['7000550','7200799','3704614']
usernames = ['Medium seniority irrigation',
             'Low seniority irrigation',
             'Transbasin municipal diversion']
nStructures = len(all_IDs)

# Install package data
from msdbook.install_supplement import install_package_data
install_package_data()
```

```
Downloading example data for msdbook version 0.1.5...
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/uncertain_
↪params_bounds.txt
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/sa_metric_s1.
↪npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/sa_vary_
↪delta.npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/sa_by_mth_s1.
↪npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/solutions.
↪resultfile
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/3704614_
↪heatmap.npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/LHsamples_
↪original_1000.txt
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/3704614_
↪pseudo_r_scores.csv
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/param_values.
↪csv
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/sa_by_yr_s1.
↪npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/sa_by_mth_
↪delta.npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/7000550_
↪pseudo_r_scores.csv
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/collapse_
↪days.csv
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/hymod_params_
↪256samples.npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/sa_vary_s1.
↪npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/7000550_
↪heatmap.npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/7200799_
↪heatmap.npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/sa_by_yr_
↪delta.npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/7200799_
↪pseudo_r_scores.csv
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/LeafCatch.csv
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/hymod_
```

(continues on next page)

(continued from previous page)

```
↪simulations_256samples.csv
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/Robustness.
↪txt
```

Step 1: Load Latin hypercube sample and set up problem

To examine regional vulnerability, we generate an ensemble of plausible future states of the world (SOWs) using Latin Hypercube Sampling. For this tutorial, we'll load a file containing 1,000 samples across 14 parameters. The sampled parameters encompass plausible changes to the future state of the basin, including changes to hydrology, water demands (irrigation, municipal & industry, transbasin), and institutional and environmental factors (environmental flows, reservoir storage, operation of the Shoshone Power Plant). These samples are taken from ranges identified in `param_bounds`. Below we load in the 1000 samples, the range of values that the samples can take for each parameter, and the parameter names. More information on what each parameter constitutes can be found in Table 1 of Hadjimichael et al., 2020.

```
# Identify the bounds for each of the 14 parameters
param_bounds = msdbook.load_basin_param_bounds()

#Load in the parameter samples
LHsamples = msdbook.load_lhs_basin_sample()

#Create an array of the parameter names
param_names=['Irrigation demand multiplier','Reservoir loss','Transbasin demand␣
↪multiplier',
             'Municipal & industrial multiplier', 'Shoshone','Environmental flows',
             'Evaporation change','Mean dry flow','Dry flow variance',
             'Mean wet flow','Wet flow variance','Dry-dry probability',
             'Wet-wet probability', 'Snowmelt shift']
```

Step 2: Define decision-relevant metrics for illustration

Scenario discovery attempts to identify parametric regions that lead to 'success' and 'failure'. For this demonstration we'll be defining 'success' as states of the world where a shortage level doesn't exceed its historical frequency.

Step 3: Run the logistic regression

Logistic regression estimates the probability that a future SOW will be classified as a success or failure given a set of performance criteria. A logistic regression model is defined by:

$$ln\left(\frac{p_i}{1-p_i}\right) = X_i^T\beta$$

where p_i is the probability the performance in the i^{th} SOW will be classified as a success, X_i is the vector of covariates describing the i^{th} SOW, and β is the vector of coefficients describing the relationship between the covariates and the response, which here will be estimated using maximum likelihood estimation.

A logistic regression model was fit to the ensemble of SOWs using the performance criteria defined in step 2. Logistic regression modeling was conducted using the Statsmodel Python package. The data required for the full analysis is too large to include in this tutorial, but results can be found in the data file loaded below.

The results files contain the occurrence of different shortage frequency and magnitude combinations under the experiment, in increments of 10, between 0 and 100. These combinations (100 for each user) are alternative decision-relevant metrics that can be used for scenario discovery.

```
# Set arrays for shortage frequencies and magnitudes
frequencies = np.arange(10, 110, 10)
magnitudes = np.arange(10, 110, 10)
realizations = 10

# Load performance and pseudo r scores for each of the users
results = [msdbook.load_user_heatmap_array(all_IDs[i]) / 100 for i in range(len(all_
↪IDs))]
```

Step 4: Factor ranking

To rank the importance of each uncertain factor, we utilize McFadden's psuedo-R2, a measure that quantifies the improvement of the model when utilizing each given predictor as compared to prediction using the mean of the data set:

$$R^2_{McFadden} = 1 - \frac{ln\hat{L}(M_{full})}{ln\hat{L}(M_{intercept})}$$

Here $ln\hat{L}(M_{full})$ is the log likelihood of the full model (including the predictor) and $ln\hat{L}(M_{intercept})$ is the log likelihood of the intercept model (which predicts the mean probability of success across all SOWs).

Higher values of McFadden's pseudo-R2 indicate higher factor importance (when the likelihood of the full model approaches one, the ratio of the likelihood of the full model compared to the intercept model will get very small).

```
# Load the pseudo-R^2 scores
scores = [msdbook.load_user_pseudo_scores(all_IDs[i]) for i in range(len(all_IDs))]

# Select indices of frequency and magnitudes that will be used for the visualization
freq = [1,0,0]
mag = [7,3,7]
```

Step 5: Draw factor maps

The McFadden's pseudo-R2 scores files contain preliminary logistic regression results on parameter importance for each of these combinations. Using these pseudo-R2 scores we will identify the two most important factors for each metric which we'll use to generate the final scenario discovery maps (note: there may be more than two important metrics for each user, but here we will demonstrate by mapping two).

```
# setup figure
fig, axes = plt.subplots(3,1, figsize=(6,18), tight_layout=True)
fig.patch.set_facecolor('white')

for i in range(len(axes.flat)):

    ax = axes.flat[i]

    allSOWsperformance = results[i]
    all_pseudo_r_scores = scores[i]
```

(continues on next page)

(continued from previous page)

```
    # construct dataframe
    dta = pd.DataFrame(data=np.repeat(LHsamples, realizations, axis = 0), columns=param_
↪names)
    dta['Success'] = allSOWsperformance[freq[i],mag[i],:]

    pseudo_r_scores = all_pseudo_r_scores[str(frequencies[freq[i]])+'yrs_
↪'+str(magnitudes[mag[i]])+'prc'].values
    top_predictors = np.argsort(pseudo_r_scores)[::-1][:2] #Sort scores and pick top 2␣
↪predictors

    # define color map for dots representing SOWs in which the policy
    # succeeds (light blue) and fails (dark red)
    dot_cmap = mpl.colors.ListedColormap(np.array([[227,26,28],[166,206,227]])/255.0)

    # define color map for probability contours
    contour_cmap = mpl.colormaps.get_cmap('RdBu')

    # define probability contours
    contour_levels = np.arange(0.0, 1.05,0.1)

    # define base values of the predictors
    SOW_values = np.array([1,1,1,1,0,0,1,1,1,1,1,0,0,0]) # default parameter values for␣
↪base SOW
    base = SOW_values[top_predictors]
    ranges = param_bounds[top_predictors]

    # define grid of x (1st predictor), and y (2nd predictor) dimensions
    # to plot contour map over
    xgrid = np.arange(param_bounds[top_predictors[0]][0],
                      param_bounds[top_predictors[0]][1], np.around((ranges[0][1]-
↪ranges[0][0])/500,decimals=4))
    ygrid = np.arange(param_bounds[top_predictors[1]][0],
                      param_bounds[top_predictors[1]][1], np.around((ranges[1][1]-
↪ranges[1][0])/500,decimals=4))
    all_predictors = [ dta.columns.tolist()[i] for i in top_predictors]
    dta['Interaction'] = dta[all_predictors[0]]*dta[all_predictors[1]]

    # logistic regression here
    predictor_list = [all_predictors[i] for i in [0,1]]
    result = msdbook.fit_logit(dta, predictor_list)

    # plot contour map
    contourset = msdbook.plot_contour_map(ax, result, dta, contour_cmap,
                                          dot_cmap, contour_levels, xgrid,
                                          ygrid, all_predictors[0], all_predictors[1],␣
↪base)

    ax.set_title(usernames[i])

# set up colorbar
cbar_ax = fig.add_axes([0.98, 0.15, 0.05, 0.7])
```

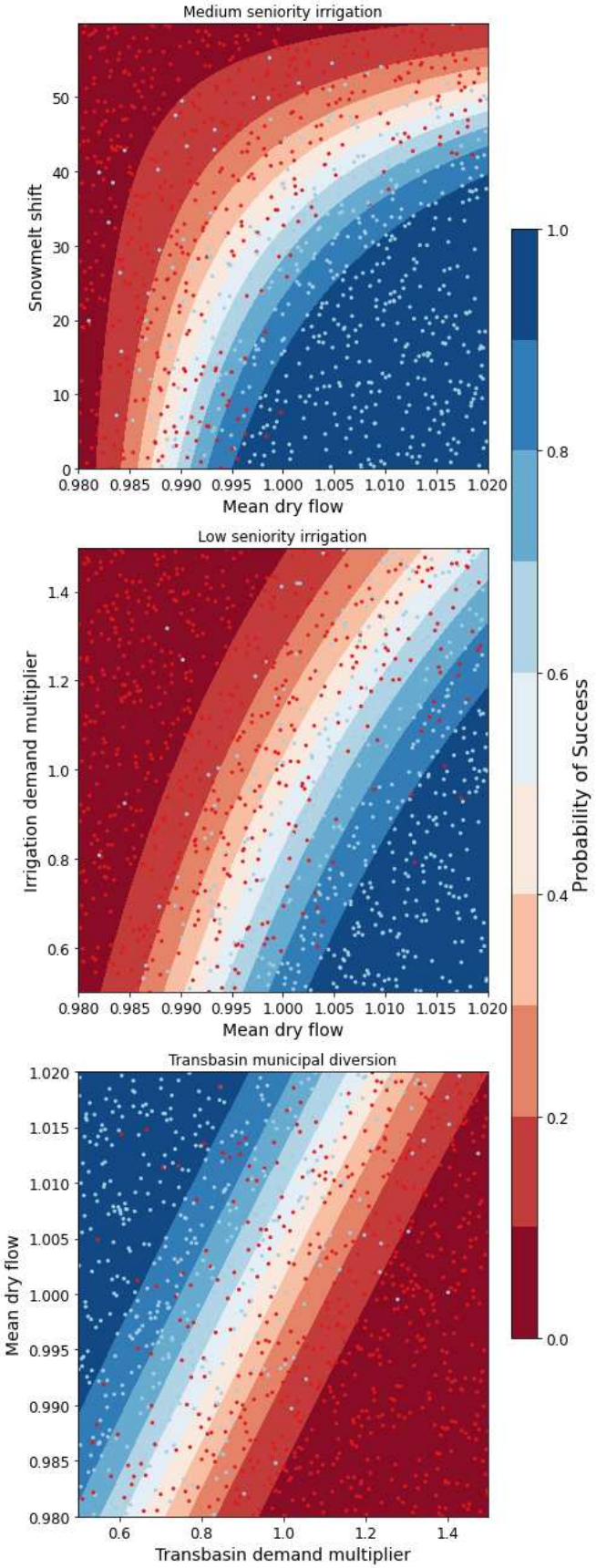
(continues on next page)

(continued from previous page)

```
cbar = fig.colorbar(contourset, cax=cbar_ax)
cbar_ax.set_ylabel('Probability of Success', fontsize=16)
cbar_ax.tick_params(axis='y', which='major', labelsize=12)
```

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/statsmodels/base/model.py:127:␣
↪ValueWarning: unknown kwargs ['disp']
  warnings.warn(msg, ValueWarning)
```

```
Optimization terminated successfully.
         Current function value: 0.378619
         Iterations 8
Optimization terminated successfully.
         Current function value: 0.397285
         Iterations 8
Optimization terminated successfully.
         Current function value: 0.377323
         Iterations 8
```


Medium seniority irrigation
Snowmelt shift
Mean dry flow
Low seniority irrigation
Irrigation demand multiplier
Mean dry flow
Transbasin municipal diversion
Mean dry flow
Transbasin demand multiplier
Probability of Success

The figure above demonstrates how different combinations of the uncertain factors lead to success or failure in different states of the world, which are denoted by the blue and red dots respectively. The probability of success and failure are further denoted by the contours in the figure. Several insights can be drawn from this figure.

First, using metrics chosen to be decision-relevant (specific to each user) causes different factors to be identified as most important by this scenario-discovery exercise (the x- and y-axes for each of the subplots). In other words, depending on what the decision makers of this system want to prioritize they might choose to monitor different uncertain factors to track performance.

Second, in the top panel, the two identified factors appear to also have an interactive effect on the metric used (shortages of a certain level and frequency in this example). In terms of scenario discovery, the Patient Rule Induction Method (PRIM) or Classification And Regression Trees (CART) would not be able to delineate this non-linear space and would therefore misclassify parameter combinations as 'desirable' when they were in fact undesirable, and vice versa.

Lastly, logistic regression also produces contours of probability of success, i.e. different factor-value combinations are assigned different probabilities that a shortage level will be exceeded. This allows the decision makers to evaluate these insights while considering their risk aversion.

Tips to Apply Scenario Discovery to Your Own Problem

In this tutorial, we demonstrated how to perform a scenario discovery analysis for three different users in the UCRB. The analysis allowed us to determine which parameters the users would be most affected by and to visualize how different ranges of these parameters lead to success and failure for different users. This framework can be applicable to any other application where it is of interest to characterize success and failure based on uncertain parameter ranges. In order to apply the same framework to your own problem:

1. Choose sampling bounds for your parameters of interest, which will represent uncertainties that characterize your system.
2. Generate samples for these parameters (this can be done using the `saltelli.sample` function or externally).
3. Define what constitutes success and failure in your problem. In this tutorial, success was defined based on not surpassing the historical drought frequency. Choose a metric that is relevant to your problem and decision-makers that might be involved. If your model involves an optimization, you can also define metrics based on meeting certain values of these objectives.
4. Run the parameter sets through your model and calculate success and failure based on your metrics and across different users if applicable. This step will allow you to create the scatter plot part of the final figure.
5. If it is of interest, the contours on the figure can be created by fitting the logistic regression model in a similar manner as denoted in Steps 3 and 5 of the tutorial.

B.4 HYMOD Dynamics Tutorial

Note:

Run the tutorial interactively: HYMOD Notebook.

Please be aware that notebooks can take a couple minutes to launch.

To run the notebooks yourself, download the files here and use these requirements.

B.4.1 Tutorial: Sensitivity Analysis of the HYMOD Model

The purpose of this tutorial is to demonstrate the global sensitivity analysis concepts and tools established in the Section 3.1 of the main text of this eBook. This demonstration will highlight the central role of design of experiments (Section 3.3), when implementing global sensitivity analysis tools described in Section 3.4.

We'll explore these tools and concepts using the HYdrological MODel (HYMOD), a rainfall-runoff model developed and used for river flow forecasting. HYMOD was chosen for demonstration because its purpose is to abstract highly complex and non-linear systems. The methods demonstrated in this tutorial can be applied to numerical models that simulate other complex non-linear systems.

This tutorial will first introduce HYMOD and use it to simulate streamflows in a river basin. Next, we'll employ sensitivity analysis concepts described in Section 3 of the main text to examine how values of HYMOD's parameters impact streamflow predictions. Finally, we'll explore how the effects of these parameters may change over time using time-varying sensitivity analysis.

The tutorial includes the following steps:

1.1 - Introduction to a simple hydrologic model (HYMOD)

1.2 - Input Data

1.3 - Running a basic simulation

1.4 - Model outputs

2.1 - Design of Experiments

2.2 - Sensitivity analysis for one output

2.3 - Sensitivity analysis across multiple outputs

2.4 - Time-varying sensitivity analysis

B.4.2 1 - Introduction to HYMOD

1.1 Overview

HYMOD is a hydrologic model (rainfall-runoff model) that simulates key hydrologic fluxes such as infiltration, streamflow and evapotranspiration. The model was originally developed and used for river flow forecasting, but it has also been been used to explore different sensitivity analysis (e.g., Herman et al., 2013), uncertainty quantification (e.g., Smith et al., 2008), and optimization (e.g., Ye et al., 2014) concepts.

HYMOD accepts two inputs - daily precipitation and daily potential evapotranspiration (PET)- and generates predictons of daily streamflow. HYMOD abstracts the highly non-linear process of runoff routing by dividing the flow into two components: quick flow, representing precipitation that quickly runs off the surface of the watershed into the stream, and slow flow, that moves through the soil and takes much longer to arrive at the stream.

To generate streamflow predictions, HYMOD first models vertical processes within the watershed to determine how much water infiltrates and evaporates from the soil at a given time step. It then determines how much water should be partitioned into quick flow and slow flow processes. Within each process it abstracts residence time (the time it takes a unit volume of water to move through the watershed and into the stream) using a series of "reservoirs" each with a calibrated residence time.

HYMOD's representation of hydrologic processes are shown Figure 1 below and controlled by the following parameters:

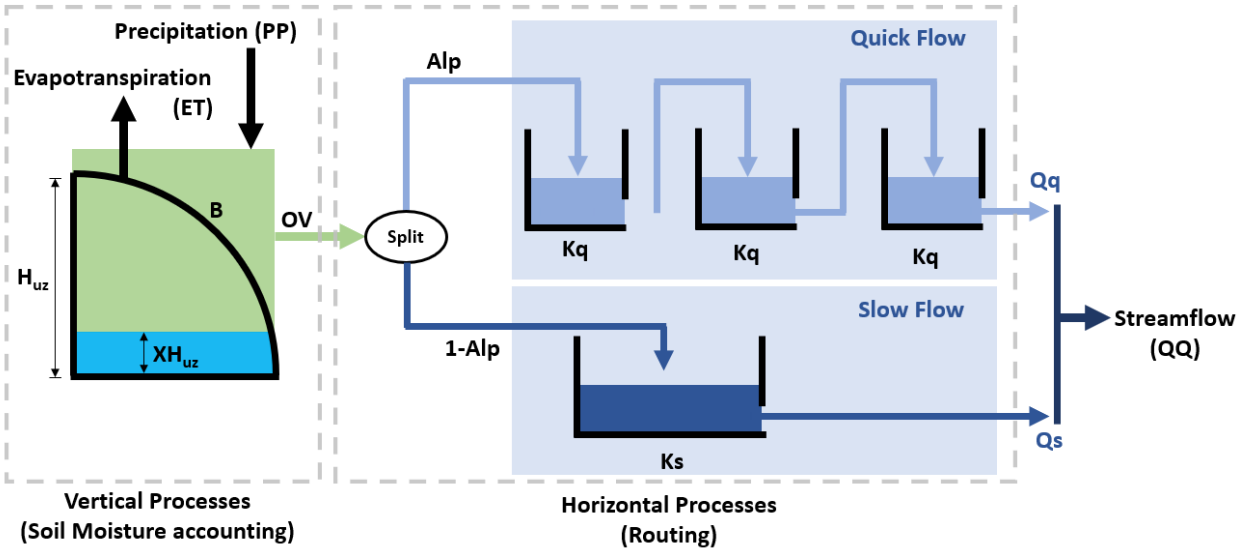
H_{uz}: the maximum water storage capacity of the soil (mm)

B_{exp}: parameters describing the degree of spatial variability within the basin between 0 and Huz

Alp: Fraction of runoff contributing to quick flow

K_q: Quick flow residence time of linear infinite reservoir (the Kq values of all three linear reservoirs are the same)

K_s: Slow flow residence time of linear infinite reservoir



HYMOD models the fraction of water that is stored in the soil ($F(XH_{uz})$) using the following relationship:

$$F(XH_{uz}) = 1 - (1 - \frac{XH_{uz}}{H_{uz}})^B$$

where XH_{uz} is the water storage capacity of the soil; H_{uz} is the parameter describing basin maximum water storage capacity (mm); and B is the parameter describing the degree of spatial variability within the basin.

The portion of precipitation that exceeds the water storage capacity is treated as runoff.

To route runoff to streamflow, the excess runoff from the vertical processes is split into quick flow and slow flow. The proportion of runoff partitioned into quick flow and slow flow is determined by a parameter Alp, which ranges between 0 and 1. Quick flow is routed through N identical quick flow tanks $Q1, Q2...QN$ (shown above as $N = 3$). The rate at which runoff moves through the quick flow system is described by the residence time of the quick flow tanks, Kq (day). Slow flow is routed through a parallel slow flow tank and the rate at which slow flow is routed is described by the slow flow residences time, Ks (day).

Citation: Wagener, T., Boyle, D. P., Lees, M. J., Wheater, H. S., Gupta, H. V., & Sorooshian, S. (2001). A framework for development and application of hydrological models. Hydrology and Earth System Sciences, 5(1), 13-26.

1.2 Input data

The HYMOD model only requires precipitation and potential evapotranspiration as inputs. For this example, we'll run HYMOD using data from the Leaf River, a humid catchment located north of Collins Mississippi that has been widely used to explore HYMOD. The dataset also includes daily observed runoff that we later use to evaluate the performance of each sensitivity analysis sample set.

In the following section of code, we'll load the necessary python libraries and read in the input file. For this exercise we'll only use the first eleven years of data. The first five rows of the input dataset are printed to show what they look like:

```
import itertools
import math
import msdbook
import numpy as np
```

(continues on next page)

(continued from previous page)

```
import pandas as pd
import seaborn as sns
import warnings

from matplotlib import pyplot as plt
import msdbook.hymod
from msdbook.package_data import load_hymod_input_file, load_hymod_params, load_hymod_
↪simulation, load_hymod_monthly_simulations, load_hymod_annual_simulations, load_hymod_
↪varying_simulations
from SALib.analyze import sobol
from sklearn import metrics

# load example data
from msdbook.install_supplement import install_package_data
install_package_data()

# load the Leaf River HYMOD input file
leaf_data = msdbook.load_hymod_input_file()

# extract the first eleven years of data
leaf_data = leaf_data.iloc[0:4015].copy()

print('Leaf River Data structure:')

# There are only three columns in the file including precipitation, potential␣
↪evapotranspiration and  streamflow
leaf_data.head()
```

```
Downloading example data for msdbook version 0.1.5...
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/uncertain_
↪params_bounds.txt
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/sa_metric_s1.
↪npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/sa_vary_
↪delta.npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/sa_by_mth_s1.
↪npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/solutions.
↪resultfile
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/3704614_
↪heatmap.npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/LHsamples_
↪original_1000.txt
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/3704614_
↪pseudo_r_scores.csv
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/param_values.
↪csv
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/sa_by_yr_s1.
↪npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/sa_by_mth_
↪delta.npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/7000550_
```

(continues on next page)

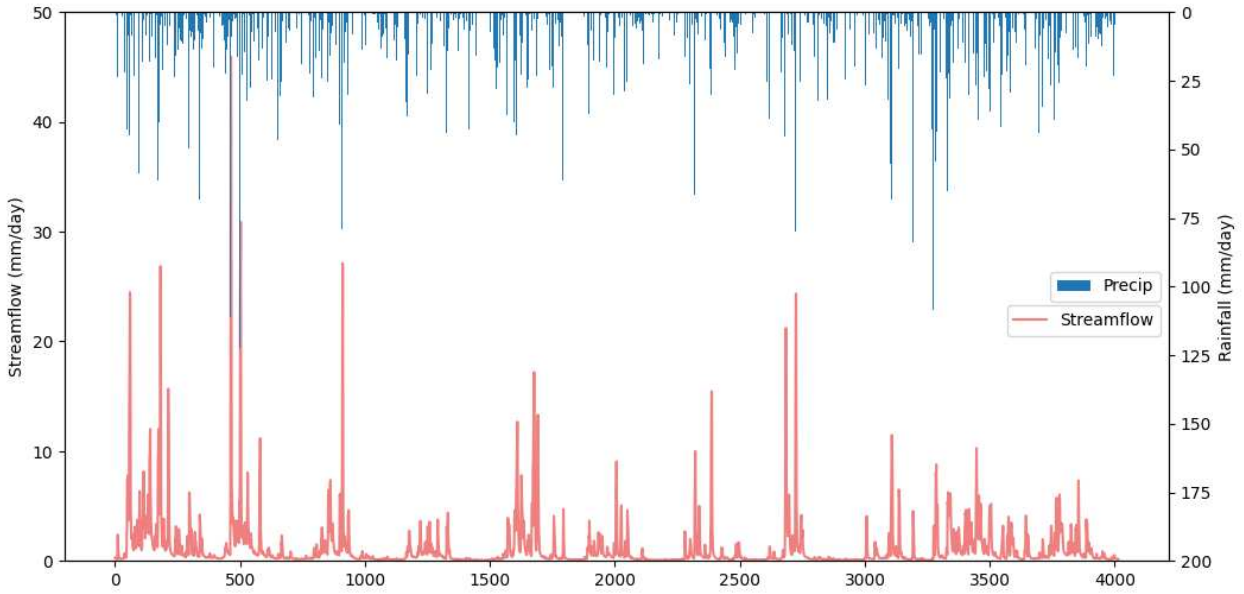
(continued from previous page)

```
↪pseudo_r_scores.csv
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/collapse_
↪days.csv
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/hymod_params_
↪256samples.npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/sa_vary_s1.
↪npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/7000550_
↪heatmap.npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/7200799_
↪heatmap.npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/sa_by_yr_
↪delta.npy
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/7200799_
↪pseudo_r_scores.csv
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/LeafCatch.csv
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/hymod_
↪simulations_256samples.csv
Unzipped: /srv/conda/envs/notebook/lib/python3.7/site-packages/msdbook/data/Robustness.
↪txt
Leaf River Data structure:
```

To visualize catchment hydrology, streamflow and precipitation data are usually plotted together as a combined hydrograph (streamflow) and hyetograph (rainfall, from Greek hyetos, "rain"). Streamflow is plotted as a time series, while rainfall is shown as an inverted bar plot along the top of the graph. Streamflow labels are shown on the left y-axis, while rainfall labels are shown on the right y-axis.

```
# make an axis for the hydrograph
fig, strmflw_ax = plt.subplots(figsize=[12,6])
strmflw_ax.set_ylim([0, 50])

#make a second y-axis for the hyetograph
precip_ax = strmflw_ax.twinx()
precip_ax.set_ylim([0, 200])
precip_ax.invert_yaxis()

precip = leaf_data['Precip']
strmflw_ax.plot(range(0, len(leaf_data['Precip'])), leaf_data['Strmflw'], color=
↪'lightcoral')
strmflw_ax.set_ylabel('Streamflow (mm/day)')

precip_ax.bar(range(0, len(leaf_data['Precip'])), leaf_data['Precip'], width=2)
precip_ax.set_ylabel('Rainfall (mm/day)')
precip_ax.legend(['Precip'], loc='center right')
strmflw_ax.legend(['Streamflow'],bbox_to_anchor=(1, 0.48))
```

```
<matplotlib.legend.Legend at 0x7f53b95c6850>
```



1.3 Running a Baseline Model Simulation

We'll start our experiment by running HYMOD using its default parameters.

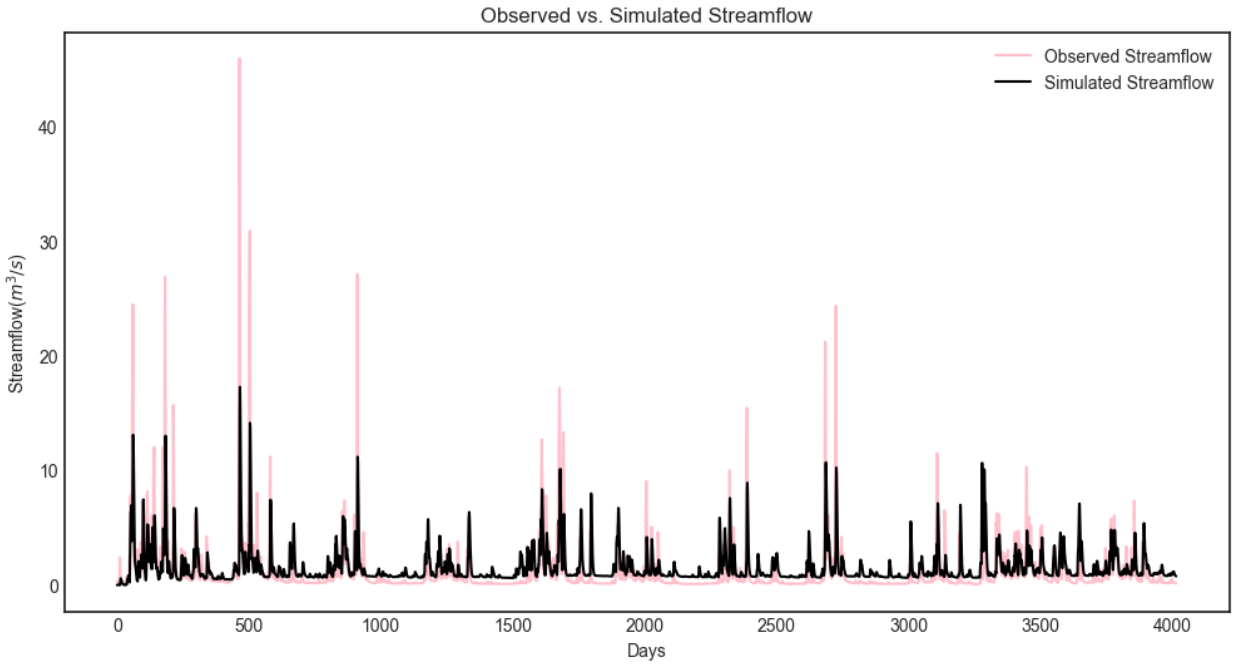
```
# assign input parameters to generate a baseline simulated streamflow
Nq = 3  # Number of quickflow routing tanks
Kq = 0.5 # Quickflow routing tanks' rate parameter
Ks =  0.001 # Slowflow routing tank's rate parameter
Alp = 0.5 # Quick/slow split parameter
Huz = 100 # Maximum height of soil moisture accounting tank
B = 1.0 # Scaled distribution function shape parameter

# Note that the number of years is 11. One year of model warm-up and ten years are used␣
↪for actual simulation
model = msdbook.hymod.hymod(Nq, Kq, Ks, Alp, Huz, B, leaf_data, ndays=4015)
```

1.4 Model Outputs

Model outputs include actual evapotranspiration, quick and fast streamflow, and combined runoff. In this tutorial we focus on the total daily runoff, QQ (m^3/s). We can use the following script to plot simulated streamflow against observed streamflow.

Tip: View the source code used to create this plot here: plot_observed_vs_simulated_streamflow

```
ax = msdbook.hymod.plot_observed_vs_simulated_streamflow(df=leaf_data, hymod_dict=model)
```



So how does our model perform? We can investigate model performance across several metrics:

1: Mean Absolute Error (MAE); MAE conveys how the model performs on average across the 10 year simulation period, with smaller values indicating better performance. The absolute value is taken so that positive and negative errors do not cancel each other out.

$$MAE = \frac{1}{N}\sum_{t=0}^{N}|Q_{sim,t} - Q_{obs,t}|$$

2: Root Mean Square Error (RMSE); RMSE is sum of square errors across the 10 year simulation period. RMSE is sensitive to large errors between the historical record and the simulated flows, and thus is useful for highlighting the model's ability to capture of extreme flood events.

$$RMSE = \sqrt{\frac{1}{N}\sum_{t=1}^{N}(Q_{sim,t} - Q_{obs,t})^2}$$

3: Log-Root Mean Square Error (Log(RMSE)) LOG(RMSE) focuses on model performance during low-flow events.

$$LOG(RMSE) = log(RMSE)$$

```
mae = np.mean(abs(leaf_data['Strmflw'] - model['Q']))
mse = metrics.mean_squared_error(model['Q'], leaf_data['Strmflw'])
rmse = mse**(1/2)

print('MAE: ' + str(mae) + '\nRMSE: ' + str(mse) + '\nLOG(RMSE): ' + str(rmse))
```

```
MAE: 1.0787471470460999
RMSE: 4.375695937555197
LOG(RMSE): 2.0918164206151544
```

The error metrics show that HYMOD performs reasonably well, the MAE is around 1 m^3/s, the RMSE is on the order of 10% of the largest observed streamflow and the LOG(RMSE) is fairly low.

B.4.3 2- Global Sensitivity Analysis

2.1 Experimental Design and Setup

Now we'll examine how sensitive streamflow simulations generated by HYMOD are to the model's input parameters. We'll perform global sensitivity analysis (see Section 3.1 of the main text) using the SALib Python library.

A first and critical step when conducting sensitivity analysis is determining the experimental design (see Design of Experiments, Section 3.4 of the main text). Our experimental design involves defining the uncertainties that we'll be examining, the output of interest, the ranges of each uncertainty that will be explored and the strategy for sampling the uncertainty space.

For this experiment we'll explore the five parameters highlighted in Figure 1. We'll draw their ranges from existing literature on the model (note Jon H. paper). We'll use a Sobol sampling an a quasi-random sampling with low sequences approach to sample the uncertainty space (Section 3.3.4).

In this demonstration we'll utilize Sobol Sensitivity Analysis, a variance based method (Section 3.4.5).

To explore HYMOD's behavoir, we'll examine the sensitivity of four model ouputs to input parameters: 1) predicted flow, 2) Mean Absolute Error (compared with the calibaration data set), 3) Root Mean Square Error and 1) Log Root Mean Square Error.

This analysis will employ SALib, a Python implementation also utilized in the other SA tutorial (make this more formal).

To start our analysis, we'll create a dictionary that describes our model uncertainties and their ranges, this dictionary is named "problem_hymod" (SALib refers to these dictionaries as "problems").

```
problem_hymod = {
    'num_vars': 5,
    'names': ['Kq', 'Ks', 'Alp', 'Huz', 'B'],
    'bounds': [[0.1, 1],  # Kq
               [0, 0.1],  # Ks
               [0, 1],    # Alp
               [0.1, 500],  # Huz
               [0, 1.9]]  # B
}
```

After defining our uncertainties and ranges, we'll use SALib to sample the uncertainty space and run the model for each of the sample sets. We will load a sample that has already been created `param_values_hymod` for demonstration purposes. For HYMOD, literature recommends running at least N = 10,000 samples, to keep this demonstration easy to run however, we utilize only 256 sobol samples of uncertainties. To generate accurate approximations of second order sensitivity indices SALib generates N*(2k+2) sets of samples, where N=256 and k=5 (number of uncertainties). For the math behind why this is needed, see (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).

The actual model simulation takes an extended period, so we also load the simulation data from a previous run. The following demonstrates how to conduct this analysis:

```
# generate 256 samples.
param_values_hymod = saltelli.sample(problem_hymod, 256)

# dictionary to store outputs in
d_outputs = {}

# run simulation for each parameter sample
for i in range(0, len(param_values_hymod)):
```

(continues on next page)

(continued from previous page)

```
    # run model for each sensitivity analysis parameter sets
    hymod_output = msdbook.hymod(Nq,
                                 param_values_hymod[i, 0],
                                 param_values_hymod[i, 1],
                                 param_values_hymod[i, 2],
                                 param_values_hymod[i, 3],
                                 param_values_hymod[i, 4],
                                 leaf_data,
                                 ndays=4015)

    # store the simulated total flow discharge
    d_outputs[f"Q{i}"] = hymod_output["Q"]


Q_df_bw = pd.DataFrame(d_outputs)
```

```
# load previously generated parameter values
param_values_hymod = load_hymod_params()

# number of samples
n_samples = len(param_values_hymod)

# load previously generated hymod simulated outputs
Q_df_bw = load_hymod_simulation()

# column names of each sample simulation number
sample_column_names = [i for i in Q_df_bw.columns if i[0] == 'Q']
```

A hydrological model such as HYMOD usually includes ordinary differential equations that are sensitive to their initial condition. They also have components in their underlying formulation that have long memory such that prior time steps can affect their current simulations. For example, soil moisture or groundwater can hold water for a long time and therefore they are often considered to exhibit a long memory. This can affect the partitioning of water to runoff and infiltration, while also controlling the generation of base flow. Therefore, it is important to have a reasonable initial value for them. To achieve this, hydrologists usually extend their simulation period and after the simulations, they remove that extended time period that has unreasonable groundwater or surface water values. This time period is called the warm-up time period.

Here we extended our simulation for one year (from 10 years to 11 years) and we removed the first year of simulation, therefore our warm-up period is one year.

```
# exclude the first year of simulation from the simulations and reset the index
Q_df = Q_df_bw.iloc[365:4015].copy().reset_index(drop=True)

# exclude the first year of the input data and reset the index
leaf_data = leaf_data.iloc[365:4015].copy().reset_index(drop=True)
```

Now that HYMOD has been warmed up, we'll examine how HYMOD's streamflow outputs vary under different sample sets, and compare them with the observed streamflow.

```
# add date columns to our simulation data frame; for this data our start date is 1/1/2000
date_ts = pd.date_range(start='1/1/2000', periods=3650, freq='D')
Q_df['date'] = date_ts
```

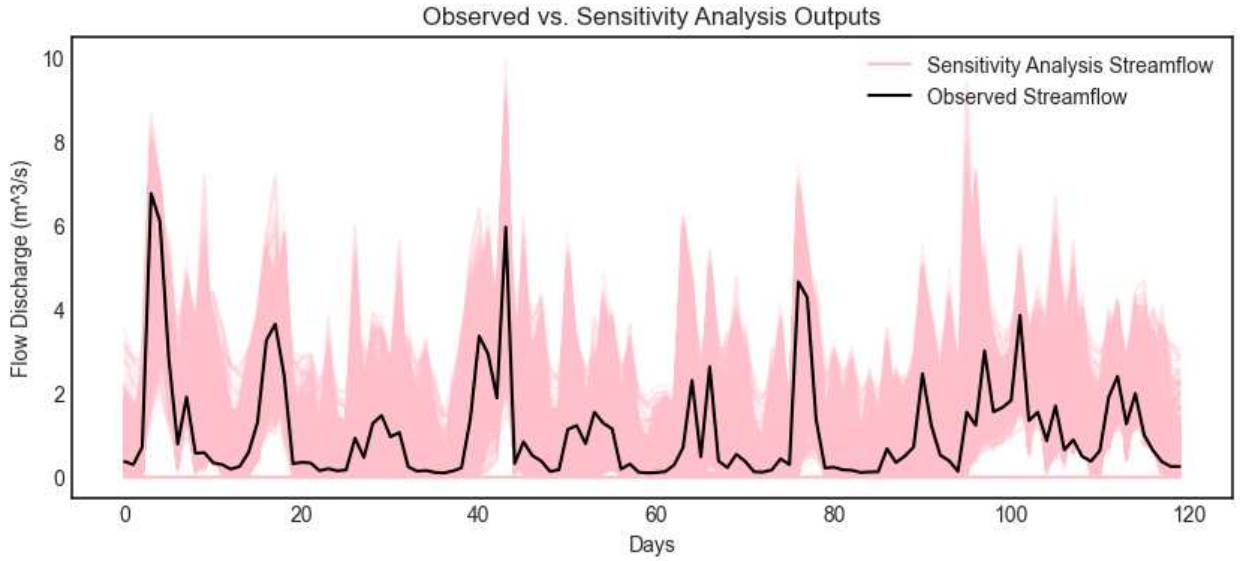
(continues on next page)

(continued from previous page)

```
Q_df['year'] = date_ts.year
Q_df['month'] = date_ts.month
Q_df['day'] = date_ts.day

# aggregate the simulated observed streamflow to monthly mean
df_sim_mth_mean = Q_df.groupby(['year', 'month'])[sample_column_names].mean()

# do the same for the observed data
date_ts = pd.date_range(start='1/1/2000', periods=len(leaf_data), freq='D')
leaf_data['date'] = date_ts
leaf_data['year'] = date_ts.year
leaf_data['month'] = date_ts.month
leaf_data['day'] = date_ts.day

# aggregate the daily observed streamflow to monthly mean
df_obs_mth_mean = leaf_data.groupby(['year', 'month']).mean()
```

Tip: View the source code used to create this plot here: plot_observed_vs_sensitivity_streamflow

```
ax = msdbook.hymod.plot_observed_vs_sensitivity_streamflow(df_obs=df_obs_mth_mean,
                                                           df_sim=df_sim_mth_mean)
```



2.2 Sensitivity of streamflows to model parameters

Now we'll examine how each of HYMOD's parameters impact the variance of simulated streamflows. Using SALib we'll calculate the first order and total order sensitivity indices of each model parameter. The first order sensitivity index measure's the individual impact that a given parameter has on the variance of the simulated streamflows. The total order index measures the impact of a given parameter along with all interactions that other parameters have with the given parameter on simulated streamflows.

We'll start with an matrix, Y, which contains our simulated streamflows for every uncertainty sample. We'll then use the sobol.analyze function from SALib to calculate the sensitivity indices (Si). The arguments for this function are the problem dictionary defined in part 2.2 of this tutorial, and the matrix of simulated streamflows, Y.

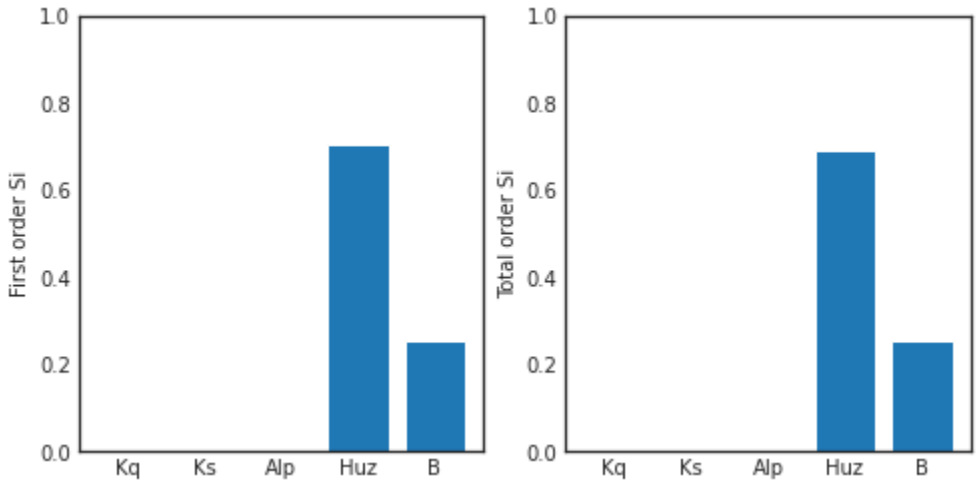
```
# overall aggregated indices
Y = Q_df[sample_column_names].mean().to_numpy()

# Perform analysis
Si = sobol.analyze(problem_hymod, Y)
```

Now we can examine our results, we'll print the first order and total order Si's for each parameter, then visualize the results with bar plots

```
print('First order indices = ', Si['S1'])

print('Total order indices = ', Si['ST'])

sns.set_style('white')
fig = plt.figure(figsize=(8,4))
ax1 = fig.add_subplot(121)
ax1.bar(np.arange(5), Si['S1'])
ax1.set_xticklabels(['','Kq', 'Ks', 'Alp', 'Huz', 'B'])
ax1.set_ylabel('First order Si')
ax1.set_ylim([0,1])

ax2 = fig.add_subplot(122)
ax2.bar(np.arange(5), Si['ST'])
ax2.set_xticklabels(['','Kq', 'Ks', 'Alp', 'Huz', 'B'])
ax2.set_ylabel('Total order Si')
ax2.set_ylim([0,1])
```

```
First order indices =  [9.55550001e-05 7.49249463e-04 5.62386413e-04 7.03327551e-01
 2.53701895e-01]
Total order indicies =  [1.76174200e-06 1.63288175e-03 3.41378460e-04 6.88983864e-01
 2.53922146e-01]
```

```
(0.0, 1.0)
```



Our findings indicate that in this instance, the streamflow estimate from HYMOD is highly sensitive to soil moisture parameters Huz and B and hardly affected by the routing parameters. Notably, there are very little interactions between parameters causing the total order indices to be nearly identical to the first order indices.

2.3 How do different performance metrics affect the results of our sensitivity analysis?

Streamflow has many different properties. In this section, we discuss how the selection of metrics can lead to fundamentally different sensitivity analysis results. For example, one can only focus on aggregated streamflow metrics such as mean (what has been presented so far), or only on extreme events such as drought or floods.

Here we compare three different metrics: 1- Mean error (ME) 2- Root Mean Square Error (RMSE) 3- Log-Root Mean Square Error (Log(RMSE))

Each of these metrics focuses on a specific attribute of streamflow. For example, RMSE highlights the impacts of extreme flood events, while LOG(RMSE) focuses on model performance during low-flow events.

```
# calculate error metrics
mae = Q_df[sample_column_names].apply(lambda x: abs(x-leaf_data["Strmflw"]), axis=0)
mse = Q_df[sample_column_names].apply(lambda x: metrics.mean_squared_error(x, leaf_data[
↪"Strmflw"]), axis=0)
rmse = mse**(1/2)

# add error metrics to a dictionary
d_metrics = {'MAE': mae.mean().values,
             'RMSE': rmse.values,
             'LOG[RMSE]': np.log10(rmse.values)}

# convert to a dataframe
df_metrics_SA = pd.DataFrame(d_metrics)
```

We can use the following to calculate the SA indices for each metric and visualize it.

```
df_metric_s1_result = pd.DataFrame(np.zeros((3, 5)), columns=['Kq', 'Ks', 'Alp', 'Huz',
↪'B'])
df_metric_sT_result = pd.DataFrame(np.zeros((3, 5)), columns=['Kq', 'Ks', 'Alp', 'Huz',
↪'B'])

# conduct sensitivity analysis for each metric
for index, i in enumerate(d_metrics.keys()):

    # get the data as a numpy array for the target metric
    Y = d_metrics[i]

    # use the metric to conduct SA
    Si = sobol.analyze(problem_hymod, Y, print_to_console=False)

    # add the sensitivity indices to the output data frame
    df_metric_s1_result.iloc[index, :] = Si['S1']
    df_metric_sT_result.iloc[index, :] = Si['ST']
```

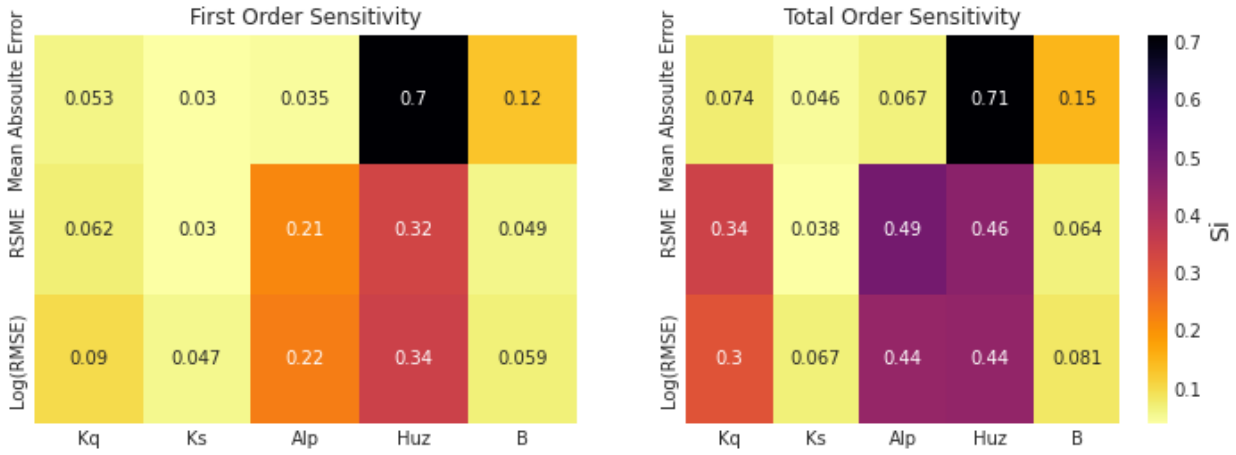
```
# create seaborn heatmap with required labels
fig = plt.figure(figsize=(12,4))
ax1 = fig.add_subplot(121)
# labels for y-axis
y_axis_labels = ['Mean Absolute Error', 'RSME', 'Log(RMSE)']

# plot heatmap
ax1 = sns.heatmap(df_metric_s1_result, yticklabels=y_axis_labels, annot=True,  cmap=
↪'inferno_r', cbar_kws={'label': 'Si'}, cbar=False)
```

(continues on next page)

(continued from previous page)

```
ax1.figure.axes[-1].yaxis.label.set_size(14)
ax1.set_title('First Order Sensitivity')

ax2 = fig.add_subplot(122)
ax2 = sns.heatmap(df_metric_sT_result, yticklabels=y_axis_labels, annot=True,  cmap=
↪'inferno_r', cbar_kws={'label': 'Si'})
ax2.figure.axes[-1].yaxis.label.set_size(14)
ax2.set_title('Total Order Sensitivity')
```

```
Text(0.5, 1.0, 'Total Order Sensitivity')
```



The first order sensitivity indices indicate that HYMOD's sensitivity to its parameters is different depending on how its output is measured. Unsurprisingly, the mean absolute error is highly sensitive to the soil moisture accounting parameters Huz and B, just like the overall streamflow predictions above. However, when we examine RMSE and log(RMSE), the routing parameters Alp become sensitive, and the sensitivity to parameter B is reduced. As described above, RMSE and LOG(RMSE) respond to model performance in high-flow and low flow periods respectively. Our results indicate that for these flow regimes Alp, the parameter that governs the split between quick and slow flow is an important factor. While still the parameter with the highest most effect on all three measures, Huz is much less influential for RMSE and LOG(RMSE) than it is for MAE.

The total order sensitivity indices review a different, more complex story. While the MAE sensitivity is relatively governed by first order effects (like the streamflow predictions above), the RMSE and LOG(RMSE) error metrics show significant interactions. Alp has the highest total order sensitivity for RMSE and is equal to Huz for Log(RMSE). Kq, which has a relatively low first order sensitivity index, shows strong contribution to variance when interactions are taken into account.

Radial convergence plots are a helpful way to visualize the interactions between parameters. These plots array the model parameters in a circle and plot the first order, total order and second order Sobol sensitivity indices for each parameter. The first order sensitivity is shown as the size of a closed circle, the total order as the size of a larger open circle and the second order as the thickness of a line connecting two parameters. Below is an example of a radial convergence plot for the LOG(RMSE) measure. The plot indicates strong interactions between the Huz and Alp parameters, as well as Alp and Kq. There is also an interaction between Alp and Ks.

```
sns.set_style('whitegrid', {'axes_linewidth': 0, 'axes.edgecolor': 'white'})

def is_significant(value, confidence_interval, threshold="conf"):
    if threshold == "conf":
        return value - abs(confidence_interval) > 0
```

(continues on next page)

(continued from previous page)

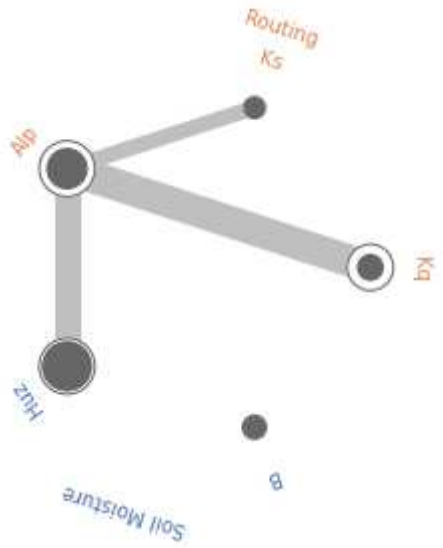
```
    else:
        return value - abs(float(threshold)) > 0


def grouped_radial(SAresults, parameters, radSc=2.0, scaling=1, widthSc=0.5, STthick=1,␣
↪varNameMult=1.3, colors=None, groups=None, gpNameMult=1.5, threshold="conf"):
    # Derived from https://github.com/calvinwhealton/SensitivityAnalysisPlots
    fig, ax = plt.subplots(1, 1)
    color_map = {}

    # initialize parameters and colors
    if groups is None:

        if colors is None:
            colors = ["k"]

        for i, parameter in enumerate(parameters):
            color_map[parameter] = colors[i % len(colors)]
    else:
        if colors is None:
            colors = sns.color_palette("deep", max(3, len(groups)))

        for i, key in enumerate(groups.keys()):
            for parameter in groups[key]:
                color_map[parameter] = colors[i % len(colors)]

    n = len(parameters)
    angles = radSc*math.pi*np.arange(0, n)/n
    x = radSc*np.cos(angles)
    y = radSc*np.sin(angles)

    # plot second-order indices
    for i, j in itertools.combinations(range(n), 2):
        if is_significant(SAresults["S2"][i][j], SAresults["S2_conf"][i][j], threshold):
            angle = math.atan((y[j]-y[i])/(x[j]-x[i]))

            if y[j]-y[i] < 0:
                angle += math.pi

            line_hw = scaling*(max(0, SAresults["S2"][i][j])**widthSc)/2

            coords = np.empty((4, 2))
            coords[0, 0] = x[i] - line_hw*math.sin(angle)
            coords[1, 0] = x[i] + line_hw*math.sin(angle)
            coords[2, 0] = x[j] + line_hw*math.sin(angle)
            coords[3, 0] = x[j] - line_hw*math.sin(angle)
            coords[0, 1] = y[i] + line_hw*math.cos(angle)
            coords[1, 1] = y[i] - line_hw*math.cos(angle)
            coords[2, 1] = y[j] - line_hw*math.cos(angle)
            coords[3, 1] = y[j] + line_hw*math.cos(angle)

            ax.add_artist(plt.Polygon(coords, color="0.75"))
```

(continues on next page)

(continued from previous page)

```
    # plot total order indices
    for i, key in enumerate(parameters):
        if is_significant(SAresults["ST"][i], SAresults["ST_conf"][i], threshold):
            ax.add_artist(plt.Circle((x[i], y[i]), scaling*(SAresults["ST"][i]**widthSc)/
↪2, color='w'))
            ax.add_artist(plt.Circle((x[i], y[i]), scaling*(SAresults["ST"][i]**widthSc)/
↪2, lw=STthick, color='0.4', fill=False))

    # plot first-order indices
    for i, key in enumerate(parameters):
        if is_significant(SAresults["S1"][i], SAresults["S1_conf"][i], threshold):
            ax.add_artist(plt.Circle((x[i], y[i]), scaling*(SAresults["S1"][i]**widthSc)/
↪2, color='0.4'))

    # add labels
    for i, key in enumerate(parameters):
        ax.text(varNameMult*x[i], varNameMult*y[i], key, ha='center', va='center',
                rotation=angles[i]*360/(2*math.pi) - 90,
                color=color_map[key])

    if groups is not None:
        for i, group in enumerate(groups.keys()):
            print(group)
            group_angle = np.mean([angles[j] for j in range(n) if parameters[j] in␣
↪groups[group]])

            ax.text(gpNameMult*radSc*math.cos(group_angle), gpNameMult*radSc*math.
↪sin(group_angle), group, ha='center', va='center',
                rotation=group_angle*360/(2*math.pi) - 90,
                color=colors[i % len(colors)])

    ax.set_facecolor('white')
    ax.set_xticks([])
    ax.set_yticks([])
    plt.axis('equal')
    plt.axis([-2*radSc, 2*radSc, -2*radSc, 2*radSc])

    return fig

# define groups for parameter uncertainties
groups={"Soil Moisture" : ["Huz", "B"],
        "Routing" : ["Alp", "Kq", "Ks"]}


fig = grouped_radial(Si, ['Kq', 'Ks', 'Alp', 'Huz', 'B'], groups=groups, threshold=0.025)
```

```
Soil Moisture
Routing
```



2.4 Time-Varying Sensitivity Analysis

In section 2.5 we saw how performing sensitivity analysis on different measurements of model output can yield in different results on the importance of each uncertain input. In this section we'll examine how performing this analysis over time can yield additional insight into the performance of HYMOD. We'll first examine how model sensitivities vary by month, then examine how they change across each year of the simulation.

For this demonstration, we'll focus only on the monthly streamflow predictions generated by HYMOD.

```
# aggregate simulated streamflow data to monthly time series
df_sim_by_mth_mean = Q_df.groupby('month')[sample_column_names].mean()

# aggregate observed streamflow data to monthly time series
df_obs_by_mth_mean = leaf_data.groupby('month').mean()
```

We can use the following to calculate the SA indices for each month and visualize it. Results are pre-loaded for efficiency.

```
# set up dataframes to store outputs
df_mth_s1 = pd.DataFrame(np.zeros((12,5)), columns=['Kq', 'Ks', 'Alp', 'Huz', 'B'])
df_mth_delta = df_mth_s1.copy()

# iterate through each month
for i in range(0, 12):

    # generate the simulation data
    Y = df_sim_by_mth_mean.iloc[i, :].to_numpy()

    # run SA
    Si = delta.analyze(problem_hymod, param_values_hymod, Y, print_to_console=False)

    # add to output dataframes
    df_mth_s1.iloc[i, :] = np.maximum(Si['S1'], 0)
    df_mth_delta.iloc[i, :] = np.maximum(Si['delta'], 0)

# convert to arrays
```

(continues on next page)

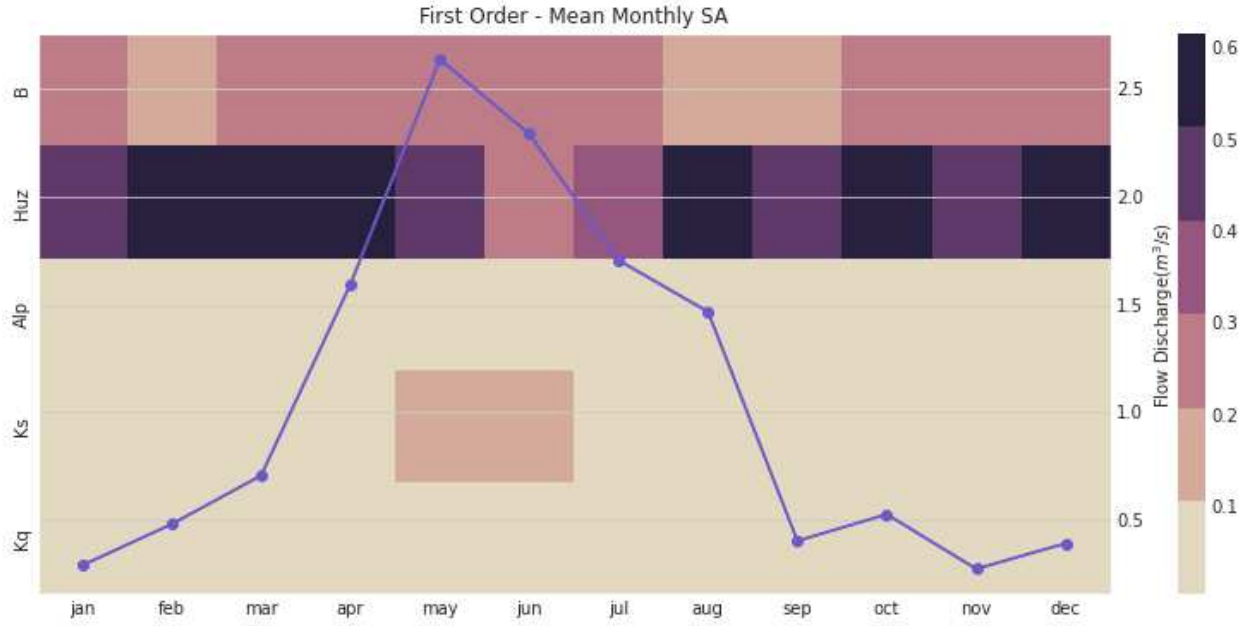
(continued from previous page)

```
arr_mth_s1 = df_mth_s1.values
arr_mth_delta = df_mth_delta.values
```

The following can be used to visualize the time-varying first-order indices. The first order represents the direct impacts of a specific parameter on model outputs.

Tip: View the source code used to create this plot here: plot_monthly_heatmap

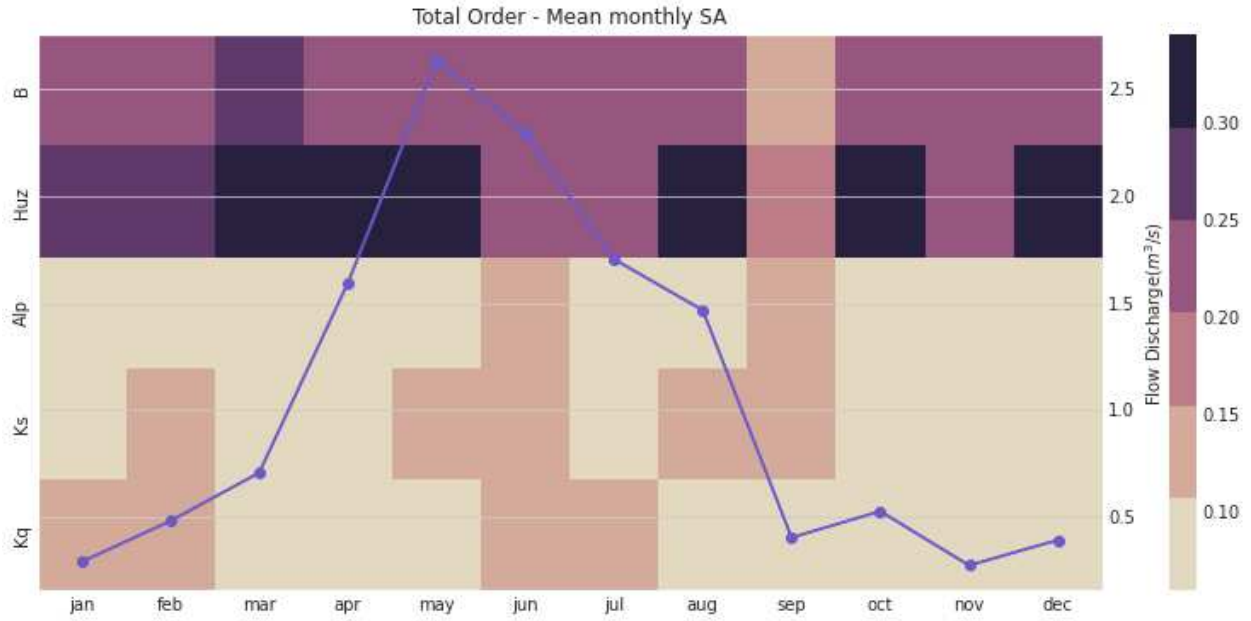
```
# load previously ran data
arr_mth_delta, arr_mth_s1 = msdbook.load_hymod_monthly_simulations()

# plot figure
ax, ax2 = msdbook.hymod.plot_monthly_heatmap(arr_sim=arr_mth_s1.T,
                                             df_obs=df_obs_by_mth_mean,
                                             title='First Order - Mean Monthly SA')
```



This figure demonstrates the first order sensitivity indices when the streamflow data are aggregated by month. The purple line represents the observed monthly discharge. The figure indicates that the first order indices are highest for B and Huz across all months and lowest for Alp, Ks, and Kq. Note that in the months with the highest flow, Ks becomes an influential parameter.

We can also focus on the total order sensitivity index that includes first-order SA indices and interactions between parameters

```
# plot figure
ax, ax2 = msdbook.hymod.plot_monthly_heatmap(arr_sim=arr_mth_delta.T,
                                             df_obs=df_obs_by_mth_mean,
                                             title='Total Order - Mean monthly SA')
```



Notably, the total order sensitivity results are different than the first order sensitivity results, which indicates that interactions between the parameters (particularly in regards to routing parameters Kq, Ks, and Alp) contribute to changes in HYMOD output.

```
# group by year and get mean
df_sim_by_yr_mean = Q_df.groupby(['year'])[sample_column_names].mean()

# group input data and get mean
df_obs_by_yr_mean = leaf_data.groupby(['year']).mean()
```

We can also calculate the sensitivity analysis indices for each individual year. This will allow us to understand if model control changes during different years. The following code first aggregates the outputs to annual time steps, and then calculates the SA indices.

```
# set up dataframes to store outputs
df_yr_s1 = pd.DataFrame(np.zeros((10, 5)), columns=['Kq', 'Ks', 'Alp', 'Huz', 'B'])
df_yr_delta = df_yr_s1.copy()

# iterate through each year
for i in range(0, 10):

    # generate the simulation data
    Y = df_sim_by_yr_mean.iloc[i, :].to_numpy()

    # run SA
    Si = delta.analyze(problem_hymod, param_values_hymod, Y, print_to_console=False)

    # add to output dataframes
    df_yr_s1.iloc[i, :] = np.maximum(Si['S1'], 0)
    df_yr_delta.iloc[i, :] = np.maximum(Si['delta'], 0)

# convert to arrays
arr_yr_s1 = df_mth_s1.values
```

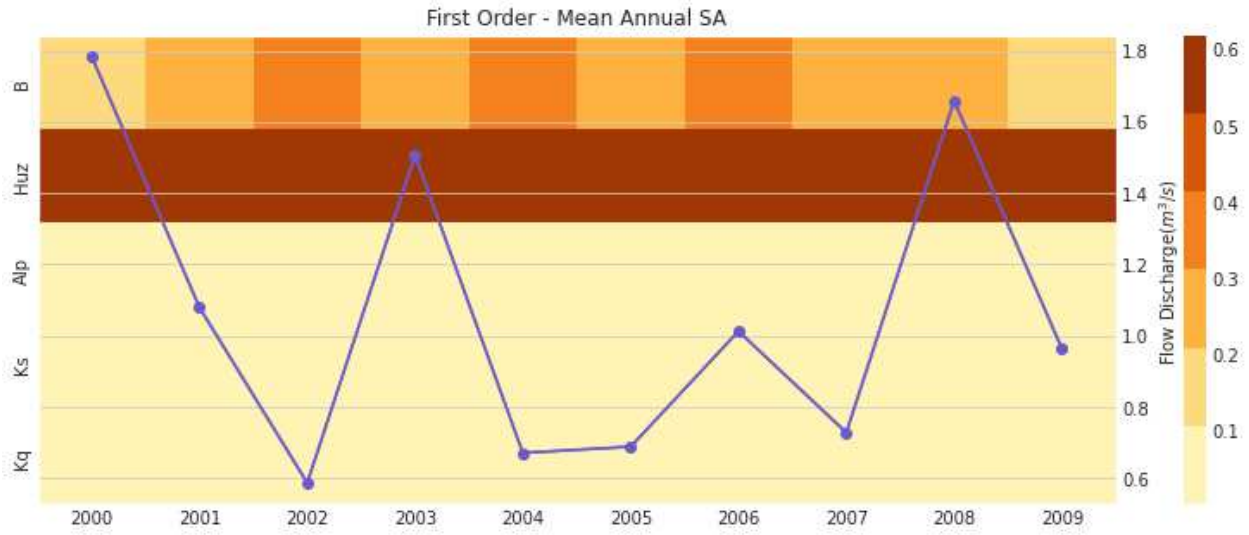
(continues on next page)

(continued from previous page)

```
arr_yr_delta = df_mth_delta.values
```

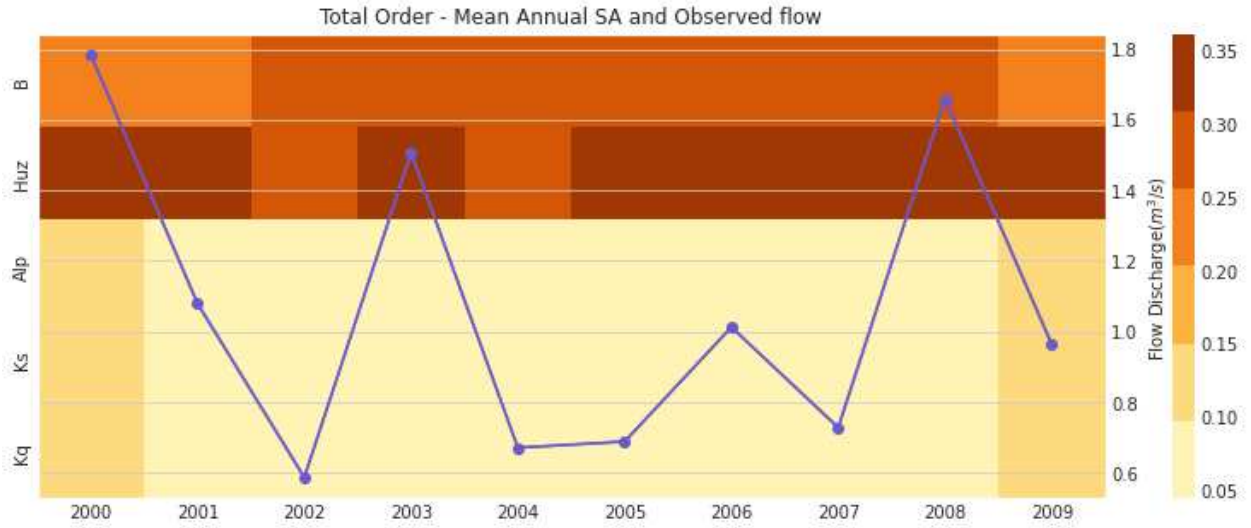
Tip: View the source code used to create this plot here: plot_annual_heatmap

```
# load previously ran data
arr_yr_delta, arr_yr_s1 = load_hymod_annual_simulations()

# plot figure
ax, ax2 = msdbook.hymod.plot_annual_heatmap(arr_sim=arr_yr_s1.T,
                                            df_obs=df_obs_by_yr_mean,
                                            title='First Order - Mean Annual SA')
```



The first order sensitivities at the annual scale are not unlike the first order monthly sensitivities. Once again, sensitivities vary across year and Huz and B are the most consequential parameters.

```
# plot figure
ax, ax2 = msdbook.hymod.plot_annual_heatmap(arr_sim=arr_yr_delta.T,
                                            df_obs=df_obs_by_yr_mean,
                                            title='Total Order - Mean Annual SA and␣
↪Observed flow')
```



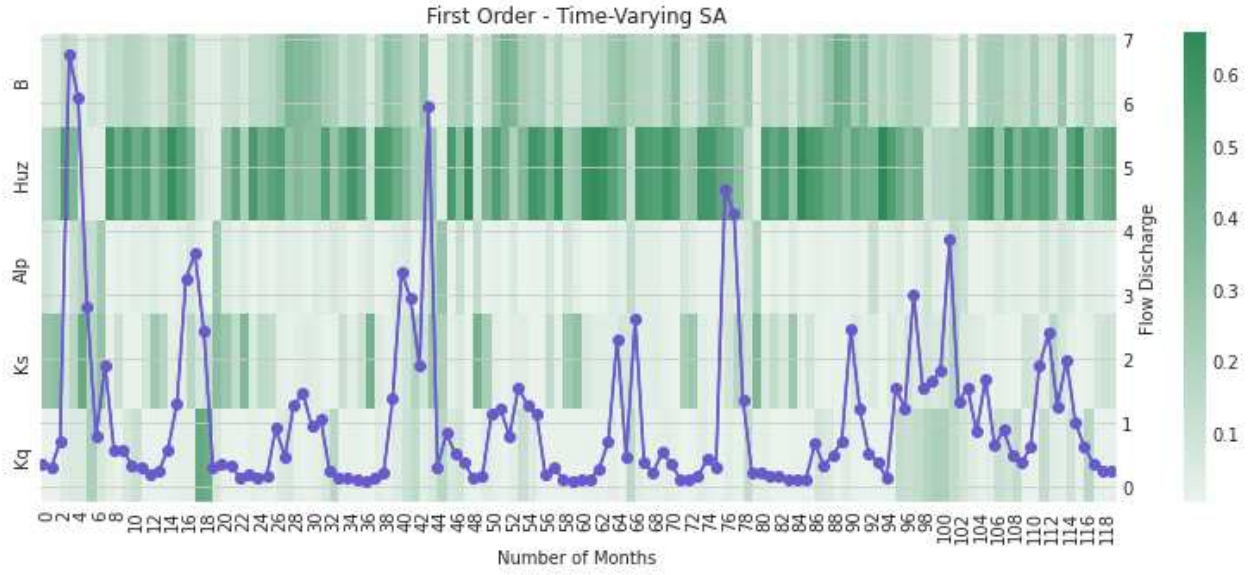
Our results indicate that sensitivity analysis indices vary in different years and now that interactions are included, the Kq, Ks, and Alp variables impact the sensitivity of the streamflow output.

Although time-varying sensitivity analysis (TVSA) at average monthly and average annual temporal resolutions is informative, TVSA is susceptible to the aggregation issue that we discussed earlier in section 3-2. To avoid that we can further discretize our time domain to zoom into individual months. This will provide us with even more information about model behavior and the sensitivity of different parameters in different states of the system. The block of code demonstrates how to implement the monthly TVSA.

```
# set up dataframes to store outputs
df_vary_s1 = pd.DataFrame(np.zeros((df_obs_mth_mean.shape[0], 5)),
                          columns=['Kq', 'Ks', 'Alp', 'Huz', 'B'])

df_vary_delta = df_vary_s1.copy()

# iterate through each month
for i in range(0, df_obs_mth_mean.shape[0]):

    # generate the simulation data
    Y = df_sim_mth_mean.iloc[i, :].to_numpy()

    # run SA
    Si = delta.analyze(problem_hymod, param_values_hymod, Y, print_to_console=False)

    # add to output dataframes
    df_vary_s1.iloc[i, :] = np.maximum(Si['S1'], 0)
    df_vary_delta.iloc[i, :] = np.maximum(Si['delta'], 0)

# convert to arrays
arr_vary_s1 = df_vary_s1.values
arr_vary_delta = df_vary_delta.values
```

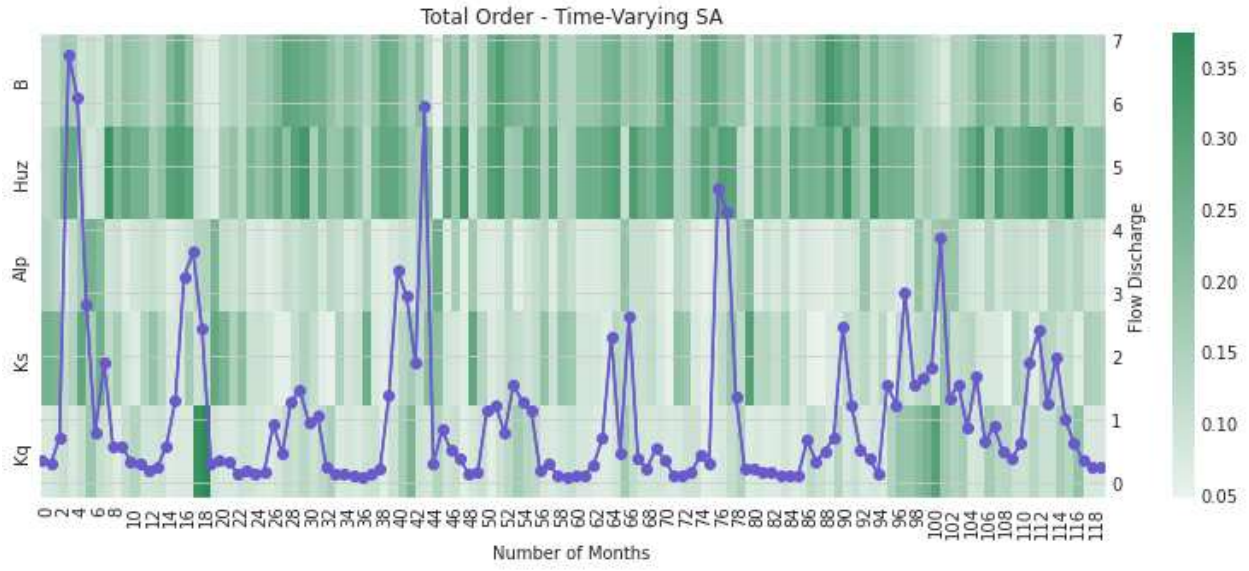
Tip: View the source code used to create this plot here: plot_varying_heatmap

```
# load in previously ran data
arr_vary_delta, arr_vary_s1 = load_hymod_varying_simulations()

# plot figure
ax, ax2 = msdbook.hymod.plot_varying_heatmap(arr_sim=arr_vary_s1.T,
                                             df_obs=df_obs_mth_mean,
                                             title='First Order - Time-Varying SA')
```



Compared to the TVSA when streamflow was aggregated, this figure suggests that Kq is indeed a relevant parameter for influencing streamflow output when individual months are considered.

```
# plot figure
ax, ax2 = msdbook.hymod.plot_varying_heatmap(arr_sim=arr_vary_delta.T,
                                             df_obs=df_obs_mth_mean,
                                             title='Total Order - Time-Varying SA')
```



As above, the total order sensitivities further indicate the importance of Kq that is not apparent if aggregation is utilized.

B.4.4 Tips to Apply this methodology to your own problem

In this tutorial, we demonstrated how to use global sensitivtiy analysis to explore a complex, non-linear model. We showed how measuring sensitivity across multiple measures of model performance and temporal aggregations yielding differing results about model sensitivity/behavoir. While these results may seem contraditory, they provide useful insight into the behavoir of HYMOD. Would we expect the same parameters to control high flow and low flow regimes within the model? Maybe, depending on the system, but also, maybe not. This analysis can provide insight into how the model responds to its input parameters, allowing us to compare the results to our expectaions. This may allow us to find problems with our intial assumptions, or call attention to model features that can be improved or expanded upon. Depending on the model and context, it may also yield insight into the workings of the underlying system.

To run this tutorial on your own model you will need to:

1. Design your experiment by choosing sampling bounds for your parameters and setting up the problem dictionary as in step 2-2
2. Choose the parameters of interest
3. Generate samples using the saltelli.sample function. This step is problem-dependent and note that the Sobol method can be computationally intensive depending on the model being analyzed. More complex models will be slower to run and will also require more samples to calculate accurate estimates of Sobol indices. Once you complete this process, pay attention to the confidence bounds on your sensitivity indices to see whether you need to run more samples.
4. Run the parameter sets through your model and record each of the desired model outputs.
5. Calculate the Sobol indices for each performance criteria. Now, the Y will be a numpy array with your external model output and you will need to include the parameter samples as an additional argument.
6. Follow the procedure in step 2.6 to disaggregate performance across time

B.5 Time-evolving scenario discovery for infrastructure pathways

Note:

Run the tutorial interactively: Scenario Discovery Notebook.
Please be aware that notebooks can take a couple minutes to launch.
To run the notebooks yourself, download the files here and use these requirements.

B.5.1 Time-evolving scenario discovery for infrastructure pathways

The purpose of this tutorial is to explore time-evolving vulnerability for systems that dynamically adapt to changing conditions. Using an example from water supply planning, we'll first examine how performance of a dynamic infrastructure pathway policy changes over time, then use factor mapping (main text Chapter 4.3) to understand which combinations of uncertainties generate vulnerability for two water utilities. Next, we'll perform factor prioritization (main text Chapter 4.3) to determine which uncertainties have the most influence on water supply performance. Finally, we'll provide an open platform to explore vulnerability across multiple measures of performance and different combinations of uncertainties.

Background

The Bedford-Greene metropolitan area (Figure 1) is a stylized water resources test case where two urban water utilities seek to develop an infrastructure and investment and management strategy to confront growing demands and changing climate. The utilities have agreed to jointly construct a new water treatment plant on Lake Classon, a large regional resource. Both utilities have also identified a set of individual infrastructure options to construct if necessary.

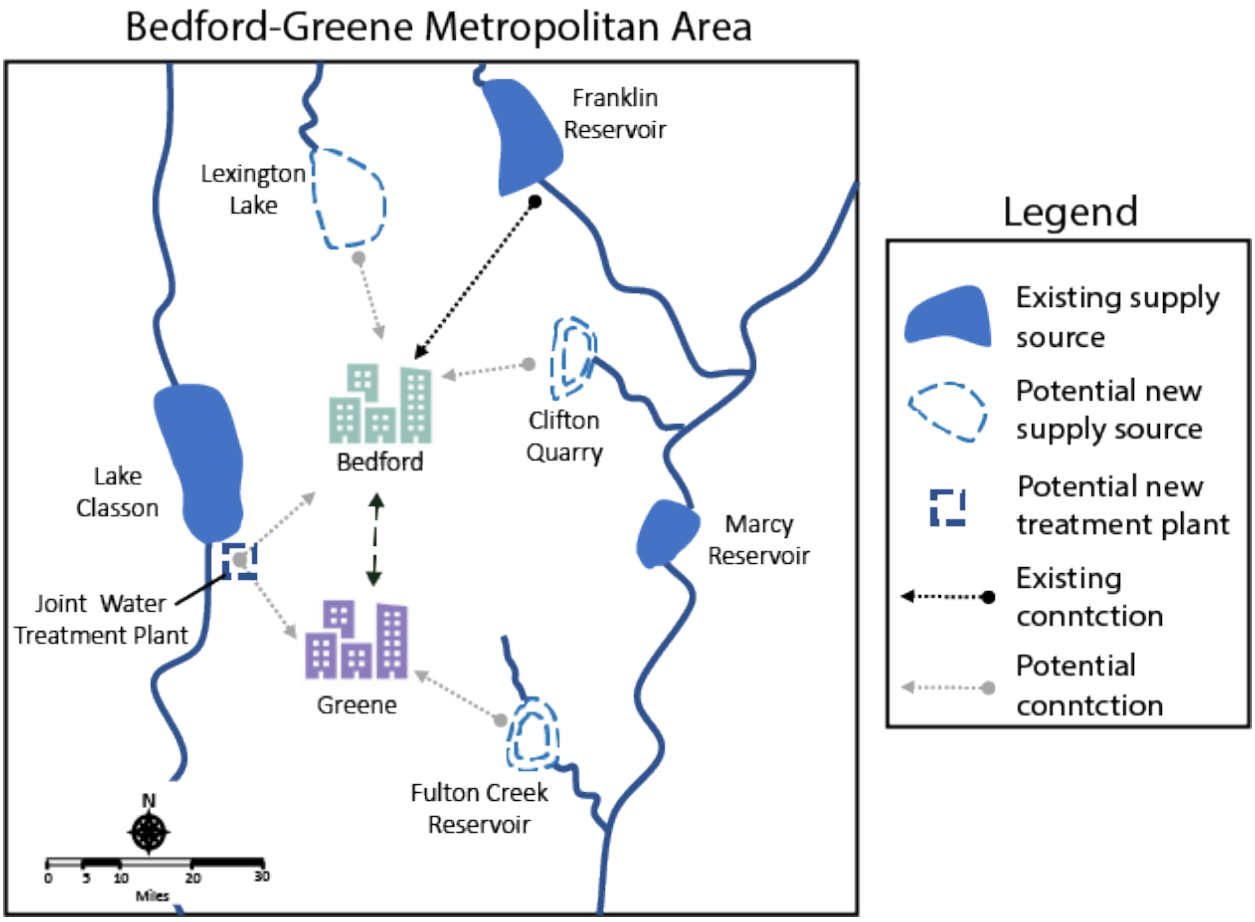


Fig. 2.1: Figure 1

The utilities are formulating a cooperative and adaptive regional management strategy that uses a risk-of-failure (ROF) metric to trigger both short term drought mitigation actions (water use restrictions and treated transfers between utilities) and long-term infrastructure investment decisions (shown in Figure 2a). Both utilities have specified a set of risk triggers and developed a construction order for available infrastructure options.

The utilities have run a Monte Carlo simulation to evaluate how these policies respond to a wide array of future States Of the World (SOWs). Each SOW represents a different combinations of thirteen uncertain system inputs including demand growth rates, changes to streamflows and financial variables. In this context, a fully specified SOW is composed of one sample of uncertain model inputs (e.g. one projection of demand growth rate coupled with one future streamflow scenario and one projection of future financial conditions). The water utilities used Latin Hypercube sampling (Chapter 3.3 of the main text) to develop an ensemble of 1,000 plausible future SOWs. The Monte Carlo simulation evaluates each candidate water supply infrastructure investment and management policy across all 1,000 SOWs, as shown in Figure 2b. For more details on the Monte Carlo sampling for this type of analysis, see Trindade et al., (2019).

The ROF-based policies respond to each SOW by generating a unique infrastructure pathway - a sequence of infrastructure investment decisions over time. Infrastructure pathways over a set of 1,000 future SOWs are shown in Figure

2c. Infrastructure options are arrayed along the vertical axis and the sequence of infrastructure investments triggered by the policy is plotted as pathways over time. Since the adaptive rule system generates a unique infrastructure sequence for each scenario, Figure 2c summarizes the ensemble of pathways by clustering SOWs according to infrastructure intensity. Dark green lines represent SOWs where the utilities heavily invest in new infrastructure, light green lines represent SOWs with low infrastructure investment and medium shaded lines represent moderate investment. The shading behind each pathway represents the frequency that each infrastructure option was triggered over time across sampled scenarios

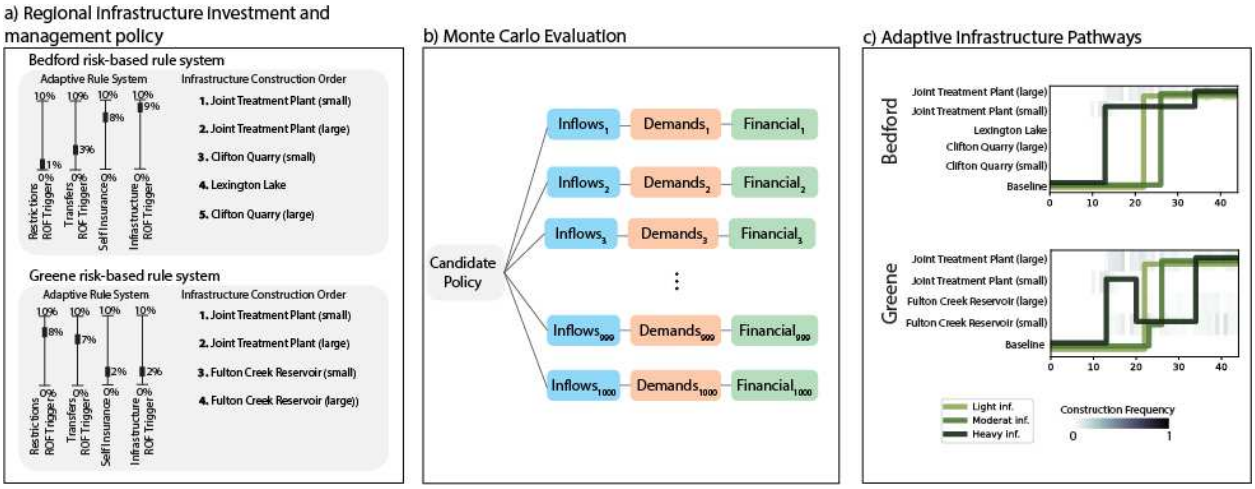


Fig. 2.2: Figure 2

Evaluating Robustness over time

The two water utilities are interested in maintaining both supply reliability and financial stability across the broadest set of plausible future SOWs. To measure the performance of the infrastructure pathway policy, they've defined five critical performance criteria:

- Reliability > 99%
- Restriction Frequency < 20%
- Peak Financial Cost < 80% of annual revenue (a measure of debt service spending)
- Worst-case drought management cost < 10% of annual revenue (a measure of unexpected drought costs)
- Unit Cost of Expansion < 5 dollars/kgal

To assess the robustness of the infrastructure pathway policy, the two utilities apply a satisficing metric, which measures the percentage of sampled SOWs where the pathway policy meets the performance criteria:

$R = \frac{1}{N} \sum_{j=1}^{N} \Lambda_{\theta,j}$

Where, $\Lambda\_\theta, j =$

$$\begin{cases} 1, & \text{if } F(\theta)_j \leq \Phi_j \\ 0, & \text{otherwise} \end{cases}$$

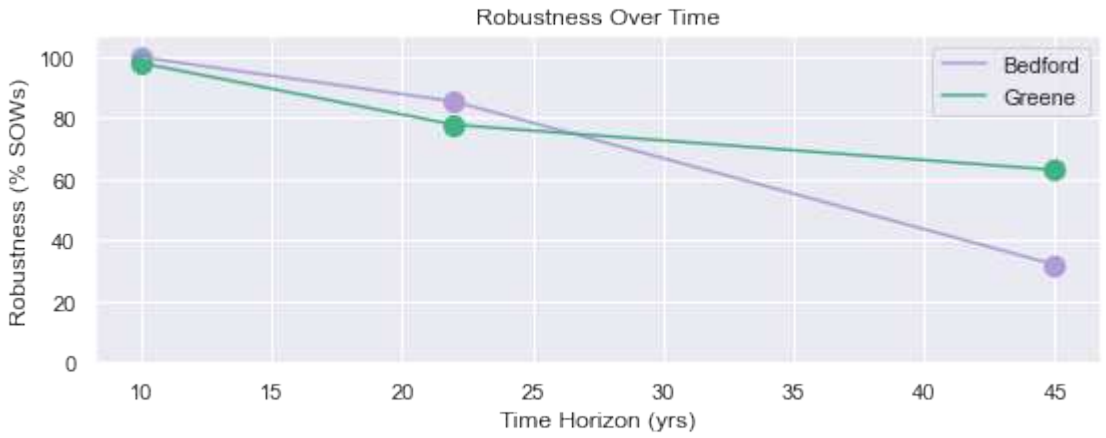
And Φ is a vector of performance criteria for utility j, θ is the portfolio and N is the total number of sampled SOWs.

Below, we'll visualize how robustness for the two utilities evolves over the 45-year planning horizon. We'll assess robustness across three time periods, near-term (first 10 years), mid-term (22 years) and long term (45 years).

We start by loading robustness values for the both utilities. These values are calculated by applying the robustness metric above across 2,000 simulated SOWs. To make this exercise computationally tractable, we've precomputed these values, which can be found in the files "short_term_robustness.csv", "mid_term_robustness.csv" and "long_term_robustness.csv". These values are calculated using the function "check_rdm_meet_criteria" within the helper functions.

```
import numpy as np
from matplotlib import pyplot as plt
from functions.eBook_SD_helpers import check_rdm_meet_criteria, create_sd_input, plot_
↪selected_tree_maps, get_factor_importances, open_exploration
import seaborn as sns

# load Deeply uncertain factors
rdm_factors = np.loadtxt('data/DU_Factors.csv', delimiter= ',')

sns.set()
short_term_robustness = np.loadtxt('data/short_term_robustness.csv', delimiter= ',')
mid_term_robustness = np.loadtxt('data/mid_term_robustness.csv', delimiter = ',')
long_term_robustness = np.loadtxt('data/long_term_robustness.csv', delimiter = ',')

# plot robustness over time
fig =plt.figure(figsize=(9,3))
plt.plot([10,22,45], [short_term_robustness[5]*100, mid_term_robustness[5]*100,long_term_
↪robustness[5]*100], c='#B19CD9')
plt.plot([10, 22, 45], [short_term_robustness[11]*100, mid_term_robustness[11]*100, long_
↪term_robustness[11]*100], c= '#43b284')
plt.scatter([10,22,45], [short_term_robustness[5]*100, mid_term_robustness[5]*100,long_
↪term_robustness[5]*100], s=100, c='#B19CD9')
plt.scatter([10, 22, 45], [short_term_robustness[11]*100, mid_term_robustness[11]*100,␣
↪long_term_robustness[11]*100], s=100, c='#43b284')
plt.xlabel('Time Horizon (yrs)')
plt.ylabel('Robustness (% SOWs)')
plt.legend(['Bedford', 'Greene'])
plt.title('Robustness Over Time')
plt.ylim([0, 107])
```

```
(0.0, 107.0)
```



Exploring performance evolution

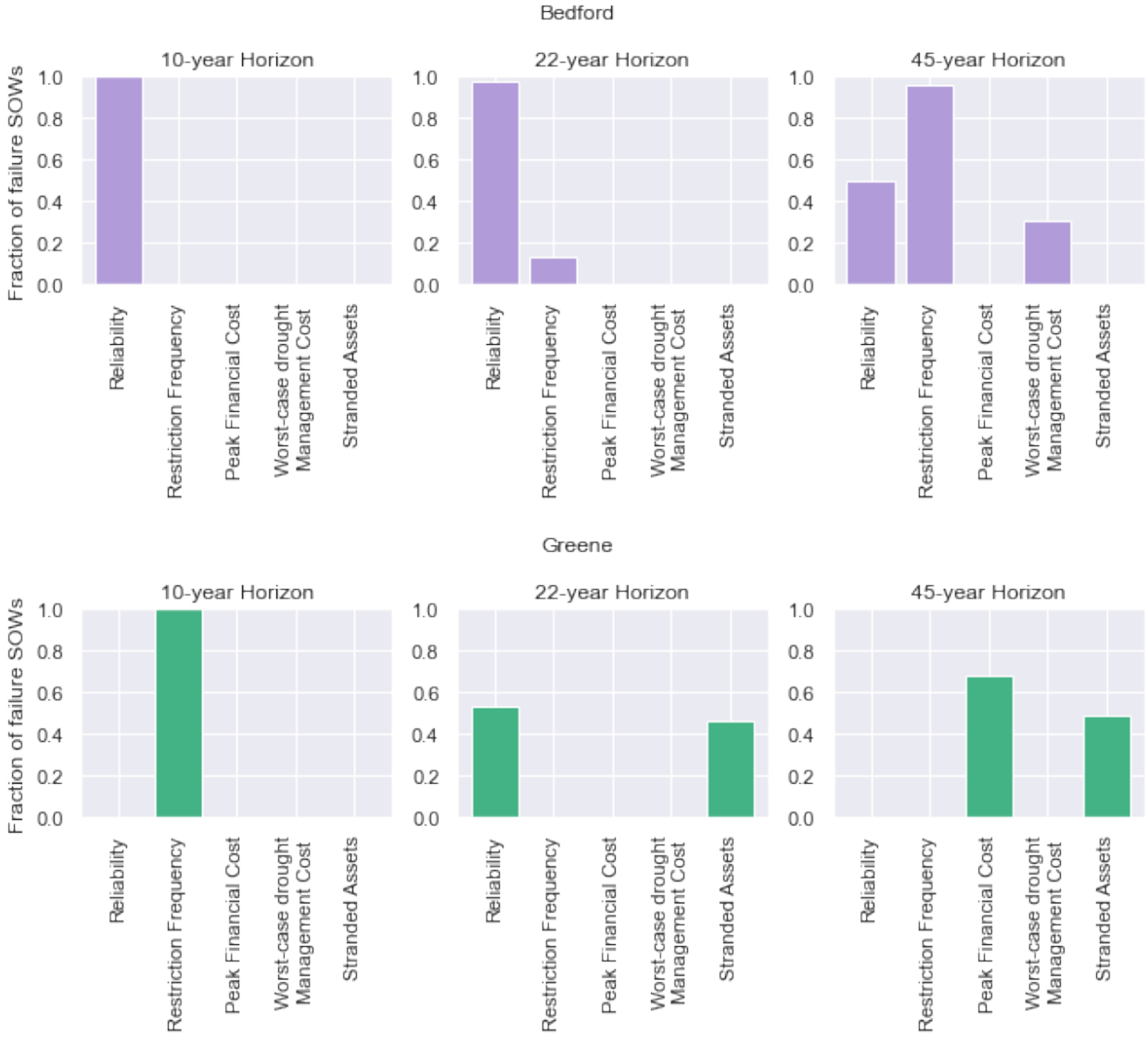
The figure above reveals that the robustness of both water utilities degrades over time, with Bedford's robustness declining further than Greene. This suggests that the proposed pathway policy is likely insufficient to meet the long-term needs of the two utilities. But how is the current policy insufficient? To answer that question we examine the performance measures that fail to meet performance criteria for each utility across the three planning horizons.

```
# Plot the type of vulnerability over time

### Bedford ###
plot_robustness_1 = np.zeros([3,5])
# Determine the percentage of failure SOWs that violate each criterion (note some SOWS␣
↪fail multiple criteria, so this may some to >1)
criteria = ['Reliability', 'Restriction Frequency', 'Peak Financial Cost', 'Worst-case␣
↪drought\nManagement Cost', 'Stranded Assets']
plot_robustness_1[0,:] = (1 - short_term_robustness[0:5])/(1-short_term_robustness[5])
plot_robustness_1[1,:] = (1 - mid_term_robustness[0:5])/(1-mid_term_robustness[5])
plot_robustness_1[2,:] = (1 - long_term_robustness[0:5])/(1-long_term_robustness[5])

# Plot over time
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(9,4))
axes[0].bar(np.arange(5), plot_robustness_1[0,:], color='#B19CD9')
axes[0].set_xticks(np.arange(5))
axes[0].set_xticklabels(criteria, rotation='vertical')
axes[0].set_ylim([0,1])
axes[0].set_title('10-year Horizon')
axes[0].set_ylabel('Fraction of failure SOWs')
axes[1].bar(np.arange(5), plot_robustness_1[1,:], color='#B19CD9')
axes[1].set_xticks(np.arange(5))
axes[1].set_xticklabels(criteria, rotation='vertical')
axes[1].set_ylim([0,1])
axes[1].set_title('22-year Horizon')
axes[2].bar(np.arange(5), plot_robustness_1[2,:], color='#B19CD9')
axes[2].set_xticks(np.arange(5))
axes[2].set_xticklabels(criteria, rotation='vertical')
axes[2].set_title('45-year Horizon')
axes[2].set_ylim([0,1])
fig.suptitle('Bedford')
plt.tight_layout()

### Greene ###
# Determine the percentage of failure SOWs that violate each criterion (note some SOWS␣
↪fail multiple criteria, so this may some to >1)
plot_robustness_2 = np.zeros([3, 5])
plot_robustness_2[0, :] = (1 - short_term_robustness[6:11]) / (1 - short_term_
↪robustness[11])
plot_robustness_2[1, :] = (1 - mid_term_robustness[6:11]) / (1 - mid_term_robustness[11])
plot_robustness_2[2, :] = (1 - long_term_robustness[6:11]) / (1 - long_term_
↪robustness[11])

fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(9, 4))
axes[0].bar(np.arange(5), plot_robustness_2[0, :], color='#43b284')
axes[0].set_xticks(np.arange(5))
```

(continues on next page)

(continued from previous page)

```
axes[0].set_xticklabels(criteria, rotation='vertical')
axes[0].set_title('10-year Horizon')
axes[0].set_ylim([0,1])
axes[0].set_ylabel('Fraction of failure SOWs')
axes[1].bar(np.arange(5), plot_robustness_2[1, :], color='#43b284')
axes[1].set_xticks(np.arange(5))
axes[1].set_xticklabels(criteria, rotation='vertical')
axes[1].set_title('22-year Horizon')
axes[1].set_ylim([0,1])
axes[2].bar(np.arange(5), plot_robustness_2[2, :], color='#43b284')
axes[2].set_xticks(np.arange(5))
axes[2].set_xticklabels(criteria, rotation='vertical')
axes[2].set_title('45-year Horizon')
axes[2].set_ylim([0,1])
fig.suptitle('Greene')
plt.tight_layout()
```



In the figures above, we observe that the vulnerability of both utilities changes in different ways. Early in the simulation period, Bedford is vulnerable to failures in reliability (though the robustness figure created in step B5.2 reveals that these failures are very rare). As the simulation period progresses, Bedford's vulnerability expands to include failures in restriction frequency and worst-case cost. These failures indicate that the utility has an overall inability to manage drought conditions and future conditions progress.

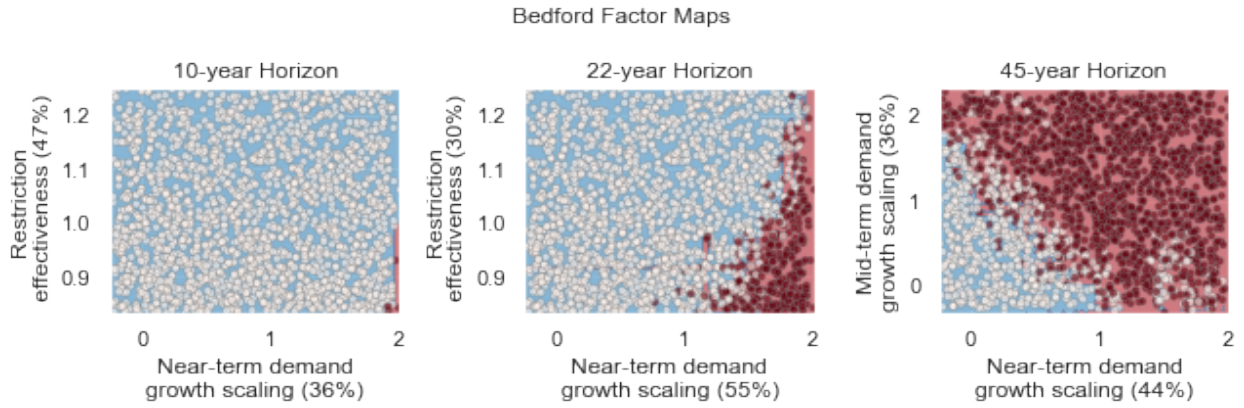
Greene shows a very different evolution in vulnerability. Early in the simulation period, failures manifest in the restriction frequency objective, suggesting that the utility must rely on water use restrictions to maintain supply reliability. As the simulation progresses however, the vulnerability evolves. When evaluated across the 45-year planning horizon, a new failure modes emerges - financial failure manifesting in peak financial cost and stranded assets. This suggests that the proposed pathway policy may be over-investing in new infrastructure, straining the utility's budget with large debt payments that are unnecessary to maintain supply reliability.

How do deep uncertainties generate vulnerability

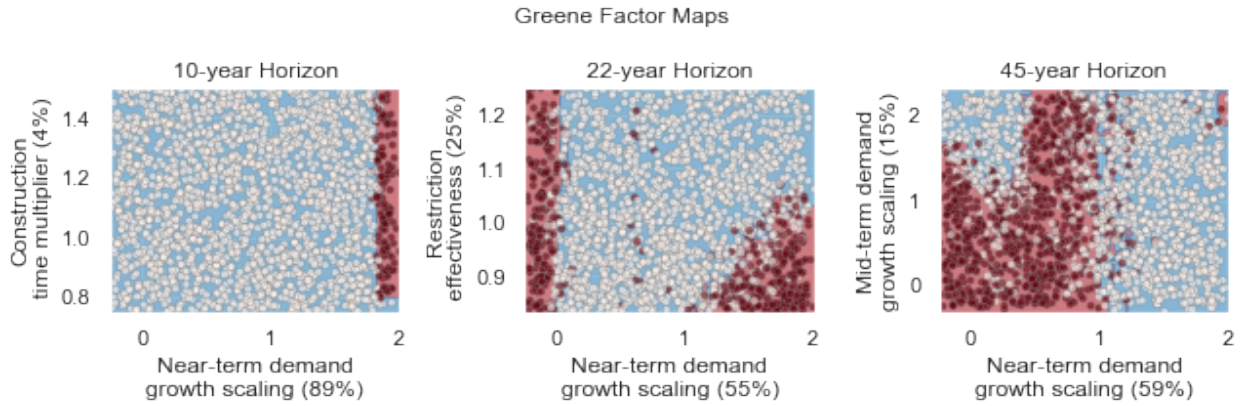
While the evolution of robustness provides insight into how the system evolves over time, it does not reveal *why* each utility is vulnerable. To examine how deep uncertainties generate vulnerability over time for the two utilities, we perform scenario discovery (factor mapping, Chapter 4.3). Here we'll utilize gradient boosted trees to identify regions of the uncertainty space that cause the utilities to fail to meet performance criteria.

```
# import the performance data across 2000 SOWs for three time periods
short_term_performance = np.loadtxt('data/short_term_performance.csv', delimiter= ',')
mid_term_performance = np.loadtxt('data/mid_term_performance.csv', delimiter = ',')
long_term_performance = np.loadtxt('data/long_term_performance.csv', delimiter = ',')

satisficing_criteria = [.98, .2, .8, .1, 5]

# transform into scenario discovery input
short_term_SD_input = create_sd_input(short_term_performance, satisficing_criteria)
mid_term_SD_input = create_sd_input(mid_term_performance, satisficing_criteria)
long_term_SD_input = create_sd_input(long_term_performance, satisficing_criteria)

# factor mapping Bedford
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(9,3))
plot_selected_tree_maps(5, 'short_term', 0, 6, satisficing_criteria, 0, axes[0])
axes[0].set_title('10-year Horizon')
plot_selected_tree_maps(5, 'mid_term', 0, 6, satisficing_criteria, 0, axes[1])
axes[1].set_title('22-year Horizon')
plot_selected_tree_maps(5, 'long_term', 0, 1, satisficing_criteria, 0, axes[2])
axes[2].set_title('45-year Horizon')
fig.suptitle('Bedford Factor Maps')
plt.tight_layout()

# factor mapping Greene
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(9,3))
plot_selected_tree_maps(11, 'short_term', 0, 8, satisficing_criteria, 0, axes[0])
axes[0].set_title('10-year Horizon')
plot_selected_tree_maps(11, 'mid_term', 0, 6, satisficing_criteria, 0, axes[1])
axes[1].set_title('22-year Horizon')
plot_selected_tree_maps(11, 'long_term', 0, 1, satisficing_criteria, 0, axes[2])
axes[2].set_title('45-year Horizon')
fig.suptitle('Greene Factor Maps')
plt.tight_layout()
```

```
Factor map for Bedford
Factor map for Bedford
Factor map for Bedford
```




```
Factor map for Greene
Factor map for Greene
Factor map for Greene
```



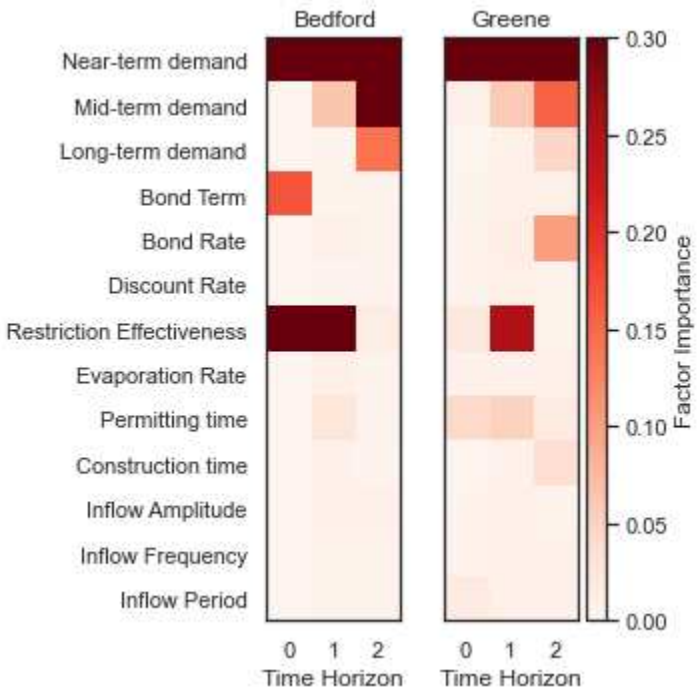
In the figures above, we learn more about how the vulnerability of the two utilities evolves over time. Bedford begins with very few possible failures but appears vulnerable to high demand growth scenarios under future scenarios with high demands. When evaluated across a 22-year planning horizon, Bedford is vulnerable when the near-term demand growth is high and water use restrictions are less effective than predicted. Under the full 45-year planning horizon, Bedford is vulnerable to sustained high levels of demand growth, failing if either near-term or mid-term demand growth exceeds expected levels.

Greene's vulnerability evolves differently. It begins with vulnerability to high demand growth, but as the simulation progresses (and infrastructure is constructed), the utility becomes vulnerable to low-demand growth futures which cause the failures in financial criteria shown in section B.5.3. This indicates that the pathway policy over-builds in many SOWs, and becomes financially unstable if demand does not grow sufficiently to provide revenue to cover debt service payments.

Which uncertainties have the most influence on time-evolving performance?

The factor maps generated in B.5.4 present the vulnerability generated by the two most important deep uncertainties as determined by Gradient Boosted Trees. Yet the factor prioritization shows that more than two uncertainties are influential to regional performance. Further, we can observe that individual uncertainties have different impacts on each performance objective, and these impacts may change over time. In the cells below, explore the impact of deep uncertainty by generating factor maps for different combinations of deep uncertain factors, objectives and time horizons.

```
sns.set_style('white')
uncertainties = ['D1', 'D2', 'D3', 'BT', 'BM', 'DR', 'RE', 'EV', 'PM', 'CT', 'IA', 'IF',
↪'IP']
uncertainties = ['Near-term demand', 'Mid-term demand', 'Long-term demand', 'Bond Term',
↪'Bond Rate', 'Discount Rate', 'Restriction Effectiveness', 'Evaporation Rate',
↪'Permitting time', 'Construction time', 'Inflow Amplitude', 'Inflow Frequency',
↪'Inflow Period']

u1_st_FI = get_factor_importances(short_term_SD_input, rdm_factors, 250, 4, 5)
u1_mt_FI = get_factor_importances(mid_term_SD_input, rdm_factors, 250, 4, 5)
u1_lt_FI = get_factor_importances(long_term_SD_input, rdm_factors, 250, 4, 5)

u1_all = np.vstack([u1_st_FI,u1_mt_FI, u1_lt_FI])
u1_all = np.transpose(u1_all)

# factor ranking -- utility 2
u2_st_FI = get_factor_importances(short_term_SD_input, rdm_factors, 250, 4, 11)
u2_mt_FI = get_factor_importances(mid_term_SD_input, rdm_factors, 250, 4, 11)
u2_lt_FI = get_factor_importances(long_term_SD_input, rdm_factors, 250, 4, 11)
u2_all = np.vstack([u2_st_FI,u2_mt_FI, u2_lt_FI])
u2_all = np.transpose(u2_all)

fig, (ax, ax2, cax) = plt.subplots(ncols=3,figsize=(5,5),
                  gridspec_kw={"width_ratios":[1,1, 0.1]})
fig.subplots_adjust(wspace=0.3)
im = ax.imshow(u1_all, cmap='Reds', vmin=0, vmax=.3)
ax.set_yticks(np.arange(13))
ax.set_yticklabels(uncertainties)
ax.set_xticks(np.arange(3))
ax.set_xlabel('Time Horizon')
ax.set_title('Bedford')

im1 = ax2.imshow(u2_all, cmap='Reds', vmin=0, vmax=.3)
ax2.set_yticks(np.arange(13))
ax2.set_yticklabels([])
ax2.set_xticks(np.arange(3))
ax2.set_xlabel('Time Horizon')
ax2.set_title('Greene')
fig.colorbar(im, cax=cax, label='Factor Importance')
plt.tight_layout()
```



The Figure above shows the factor importance as determined by gradient boosted trees for both utilities across the three planning horizons. While near-term demand growth is important for both utilities under all three planning horizons, the importance of other factors evolves over time. For example, restriction effectiveness plays an important role for Greene under the 22-year planning horizon but disappears under the 45-year planning horizon. In contrast, the bond interest rate is important for predicting success over the 45-year planning horizon, but does not appear important over the 10- or 22-year planning horizons. These findings highlight how assumptions about the planning period can have a large impact on modeling outcomes.

Open exploration

In the cell below, use the function to explore how factor maps change for the two utilities based upon the uncertainties plotted, the objectives of interest and the time horizon.

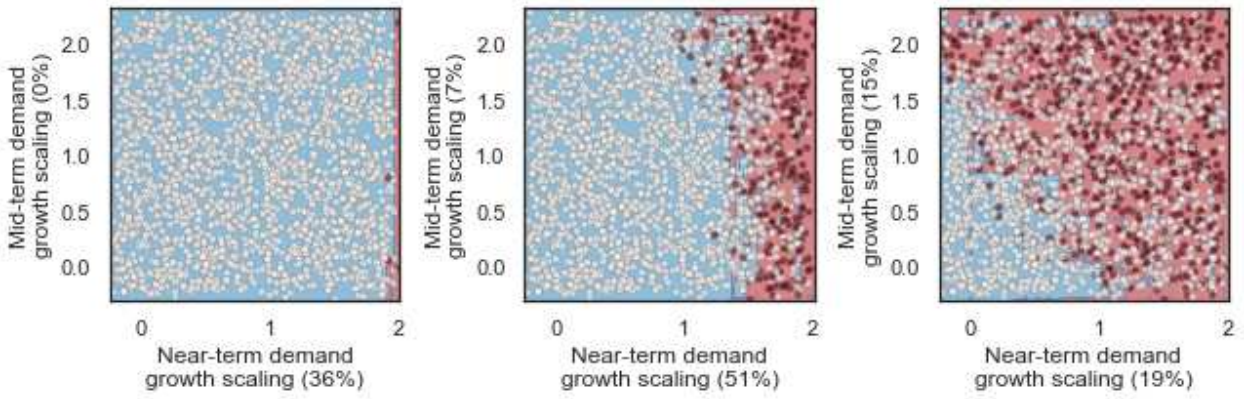
```
# specify the utility ("Bedford" or "Greene")
utility = "Bedford"

# specify which performance objectives to investigate (note that not all performance␣
↪objectives have failures, which may result in a blank factor map)
# set this to one of the following: "Reliability", "Restriction Frequency", "Peak␣
↪Financial Cost", "Worst Case Cost" or "Unit Cost"
objective = "Reliability"

# select uncertainties from the following list: 'D1', 'D2', 'D3', 'BT', 'BM', 'DR', 'RE', 'EV', 'PM',
↪ 'CT', 'IA', 'IF', 'IP'
uncertainty_1 = 'D1'
uncertainty_2 = 'D2'

# The code below will plot factor maps over the three planning horizons for the␣
↪information above
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(9,3))
```

(continues on next page)

(continued from previous page)

```
open_exploration(utility, objective, 'short_term', uncertainty_1, uncertainty_2, axes[0])
open_exploration(utility, objective, 'mid_term', uncertainty_1, uncertainty_2, axes[1])
open_exploration(utility, objective, 'long_term', uncertainty_1, uncertainty_2, axes[2])
plt.tight_layout()
```

```
Factor map for Bedford, reliability
Factor map for Bedford, reliability
Factor map for Bedford, reliability
```



Tips to apply this methodology to your own problem

In this tutorial, we demonstrated time-evolving scenario discovery for a cooperative water supply system. To apply this workflow to your own problem:

1. Choose sampling bounds for your parameters of interest, which will represent uncertainties that characterize your system.
2. Generate samples for these parameters (this can be done using the saltelli.sample function as in B.2 or done with another package).
3. Define performance criteria for your problem
4. Evaluate parameter sets through your model, and save performance measures across multiple time horizons
5. Draw from the supporting code for this tutorial to perform scneario discovery and visualize results

References

Trindade, B. C., Reed, P. M., & Characklis, G. W. (2019). Deeply uncertain pathways: Integrated multi-city regional water supply infrastructure investment and portfolio management. Advances in Water Resources, 134, 103442.

B.6 A Hidden-Markov Modeling Approach to Creating Synthetic Streamflow Scenarios Tutorial

Note:

Run the tutorial interactively: HMM Notebook.

Please be aware that notebooks can take a couple minutes to launch.

To run the notebooks yourself, download the files here and use these requirements.

B.6.1 A Hidden-Markov Modeling Approach to Creating Synthetic Streamflow Scenarios

In this notebook, we will be covering the basics of fitting a Hidden Markov Model-based synthetic streamflow generator for a single site in the Upper Colorado River Basin. First, we will characterize the observed historical flow in the basin from 1909-2013. Then, we will fit a synthetic streamflow generator to the observed flows in the basin in order to create stationary synthetic flows. Finally, we will create a non-stationary version of the generator to create flows that could be representative of plausible future climate in the region. We ultimately show how to place the synthetically generated flows in the context of physically-informed CMIP5 projections to compare the two methods.

Background

In the Western United States (US), and particularly the Colorado River Basin, a recent study used tree-ring reconstructions to suggest that the megadrought that has been occurring in the Southwest over the past 22 years is the region's worst drought since about 800 AD (Williams et al., 2022). The study's lead author, UCLA climatologist Park Williams, suggested that had the sequence of wet-dry years occurred without anthropogenic forcing, the 2000s would have likely still been dry, but not on the same level as the worst of the last millennium's megadroughts.

The recent trend of warming and reduced soil moisture in the Southwest US is highly challenging from a water systems planning and management perspective for the Colorado River Basin. Given the wide recognition that the river is over-allocated, the most recent drought highlights the difficulty of sustaining the flow requirements as dictated by the Colorado Compact. Thus, there has been an increasing focus in exploratory modeling efforts to clarify how vulnerable water systems in this region are to plausible drought streamflow scenarios for the future. In this tutorial, we'll discuss how to create these scenarios using a Hidden Markov Model (HMM)- based streamflow synthetic generator. As discussed in Section 2.1 and 4.2 of the eBook, future climate conditions in the basin represent a deep uncertainty that can lead to highly consequential water scarcity outcomes. It is advantageous to create a model such as the HMM-based generator in order to facilitate the creation of many ensembles of streamflow that can ultimately be used to force regional water systems models to understand how variability and drought extremes affect regional water shortages, operations, and policies.

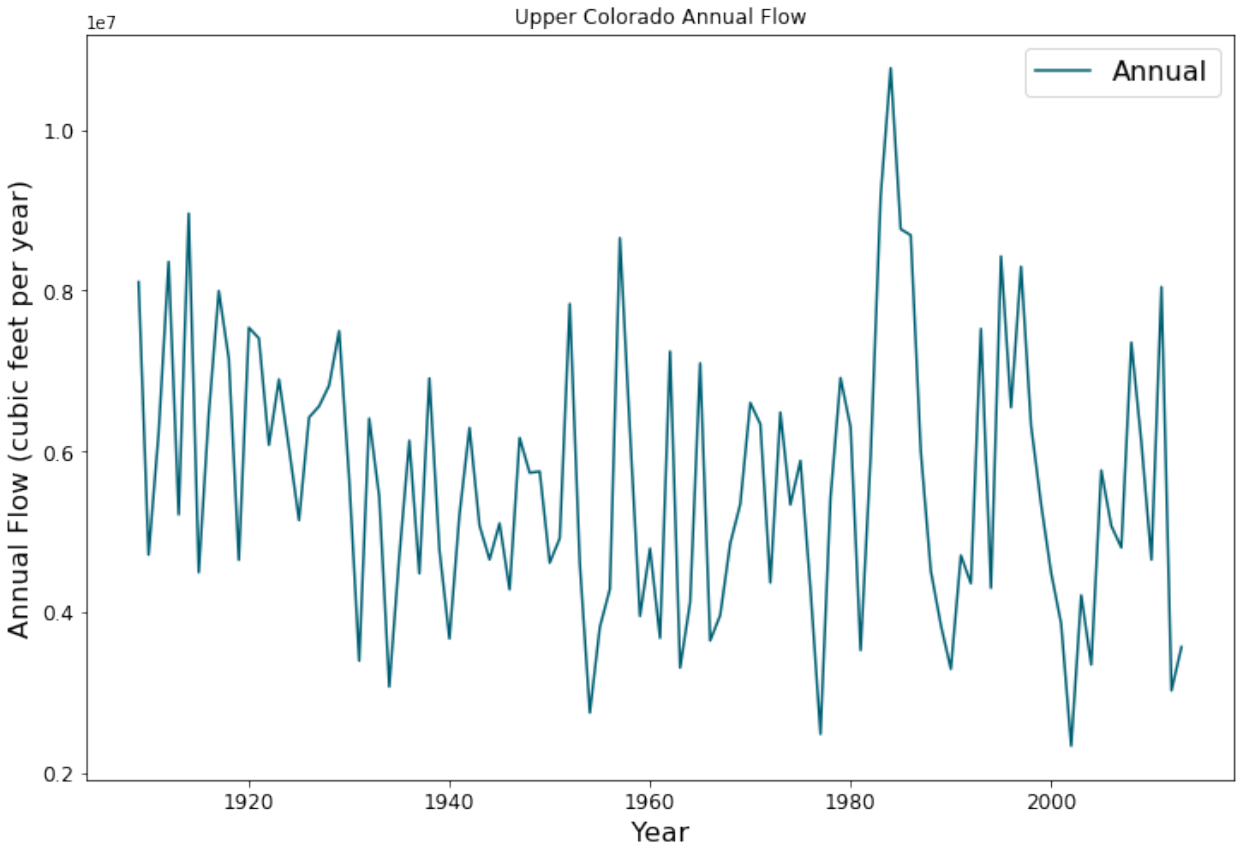
B.6.2 Let's Get Started!

Observed Record

First, let's take a look at the observed data from 1909-2013 for a specific site. In this example, we use the outlet gauge of the Upper Colorado River (USGS Gauge 09163500 at the Colorado-Utah state line). Below, we create a plot of the annual streamflow.

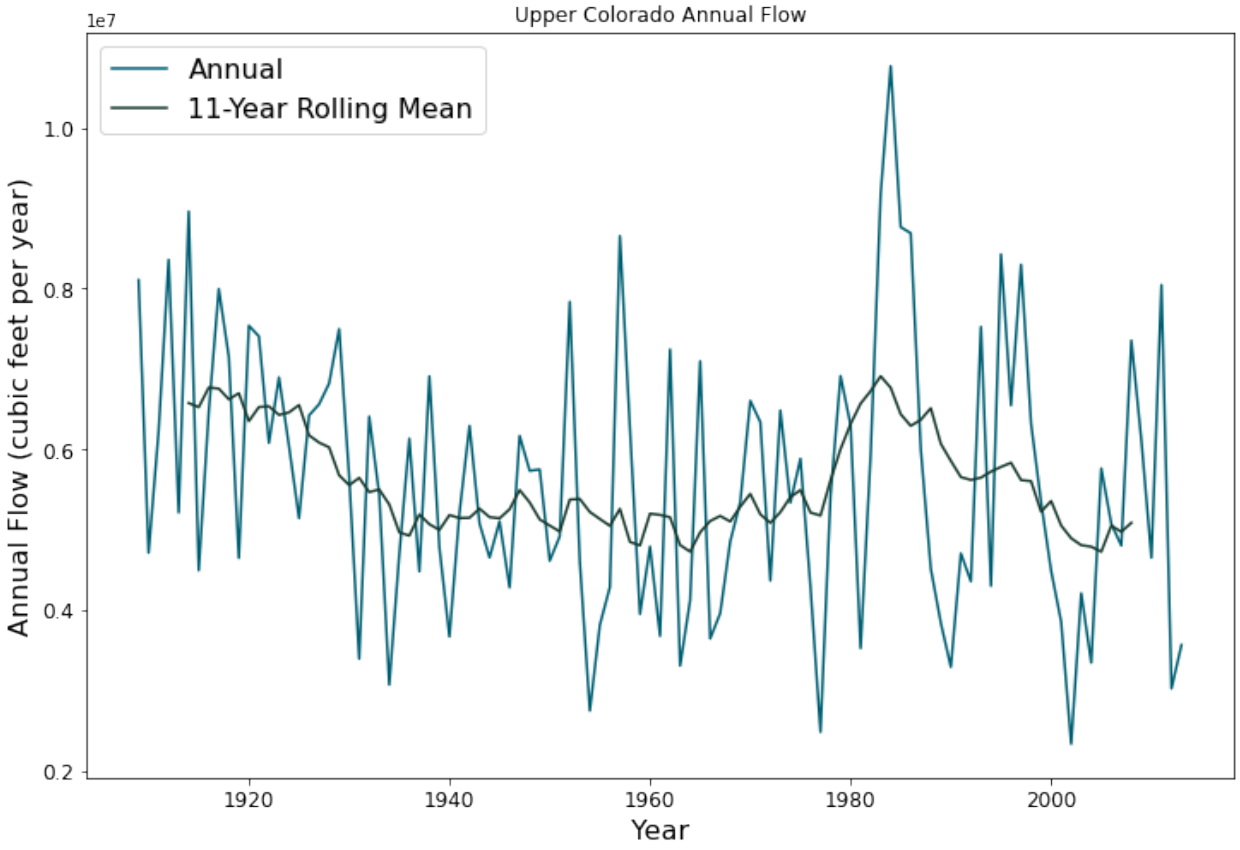
```
# Import libraries
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import numpy as np
import pandas as pd
from random import random
from SALib.sample import latin
from scipy import stats as ss
import statistics
import statsmodels.api as sm

# Import helper functions from local package
from functions import fitmodel
from functions import plotstates
from functions import plotdist
```

```
# Read in annual historical data
AnnualQ = pd.read_csv('data/uc_historical.csv')
AnnualQ['Year'] = list(range(1909, 2014))

# Plot a line graph
fig, ax = plt.subplots(figsize=(12, 8))
ax.plot(AnnualQ.iloc[:, 1],
        AnnualQ.iloc[:, 0],
        color='#005F73',
        label='Annual')

# Add labels and title
ax.set_title("Upper Colorado Annual Flow")
ax.set_xlabel("Year", fontsize=16)
ax.set_ylabel("Annual Flow (cubic feet per year)", fontsize=16)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
mpl.rc('legend', fontsize=16)
legend = plt.legend(loc="upper right")
plt.show()
plt.close()
```



Let's calculate an 11-year rolling mean of the same data to get a sense of long-term trends.

```
fig, ax = plt.subplots(figsize=(12, 8))

# Plot the original line graph
plt.plot(AnnualQ.iloc[:,1],
         AnnualQ.iloc[:,0],
         color='#005F73',
         label='Annual')

# Plot an 11-year rolling mean
plt.plot(AnnualQ.iloc[:, 1].rolling(11).mean(),
         AnnualQ.iloc[:, 0].rolling(11).mean(),
         color='#183A2E',
         label='11-Year Rolling Mean')

# Add labels and title
plt.title("Upper Colorado Annual Flow")
ax.set_xlabel("Year",fontsize=16)
ax.set_ylabel("Annual Flow (cubic feet per year)", fontsize=16)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
mpl.rc('legend', fontsize=16)
legend = plt.legend()
plt.show()
plt.close()
```



The Colorado Compact, which prescribes flows between the Upper and Lower Colorado Basins, was negotiated using data prior to 1922, a time period revealed by the above figure to be one of the consistently wetter periods on record. It's clear today that since the 1980s, the Southwest US has been experiencing aridification (Overpeck et al., 2020) and that this observed record alone isn't an accurate representation of what future climate might look like in this region.

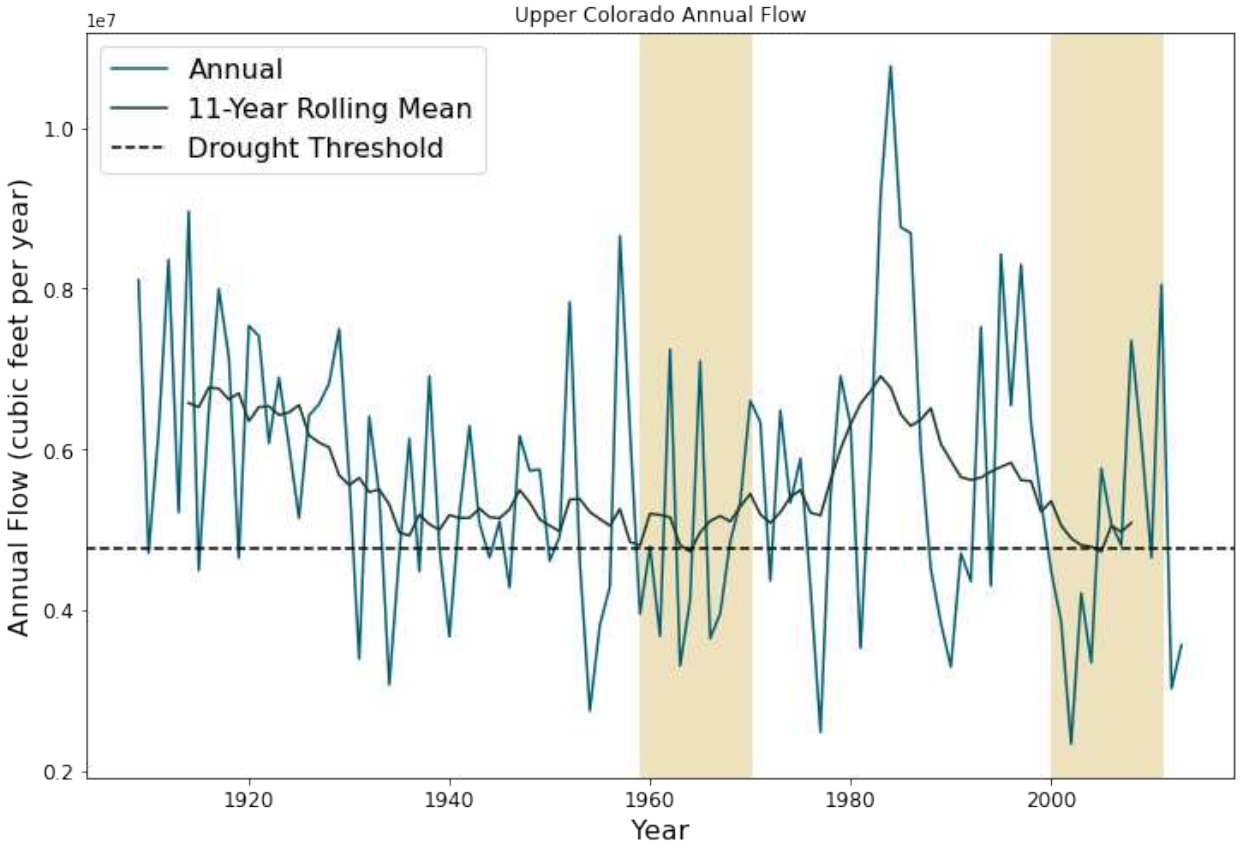
Let's get a little more specific and formally quantify decadal droughts that have occurred in the observed period. We use a metric proposed in Ault et al. (2014). The authors define a decadal drought as when the 11-year rolling mean falls below a threshold that is 1/2 a standard deviation below the overall mean of the record. We can then highlight the block of years that fall in a decadal drought using yellow rectangles below.

```
# Define drought threshold
std = statistics.stdev(AnnualQ.iloc[:, 0])
threshold = np.mean(AnnualQ.iloc[:, 0] - (0.5 * std))

# Find where the rolling mean dip below the threshold?
drought_instances = [i for i, v in enumerate(AnnualQ.iloc[:,0].rolling(11).mean()) if v
↪< threshold]
drought_years = AnnualQ.iloc[:, 1].rolling(11).mean()[drought_instances]

# Add labels and title
fig, ax = plt.subplots(figsize=(12, 8))
ax.plot(AnnualQ.iloc[:,1],
        AnnualQ.iloc[:,0],
        color='#005F73',
        label='Annual')
```

(continues on next page)

(continued from previous page)

```
ax.plot(AnnualQ.iloc[:,1].rolling(11,center=True).mean(),
        AnnualQ.iloc[:,0].rolling(11,center=True).mean(),
        color='#183A2E',
        label='11-Year Rolling Mean')

ax.axhline(y=threshold,
           color='black',
           linestyle='--',
           label='Drought Threshold')

# Visualize the drought periods as yellow rectangles
for i in drought_years:

    # Plot a box centered around those values and with 5 years on either side.
    rect = patches.Rectangle((i-5,0), 11,2e7, linewidth=1, edgecolor='#EFE2BE',␣
↪facecolor='#EFE2BE')

    # Add the patch to the Axes
    ax.add_patch(rect)


plt.title("Upper Colorado Annual Flow")
ax.set_xlabel("Year", fontsize=16)
ax.set_ylabel("Annual Flow (cubic feet per year)", fontsize=16)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
mpl.rc('legend', fontsize=16)
legend = plt.legend()
plt.show()
plt.close()
```



By this metric, the Upper Colorado Basin region has experienced two decadal droughts over the last century.

Synthetic Stationary Generator to Better Quantify Natural Variability

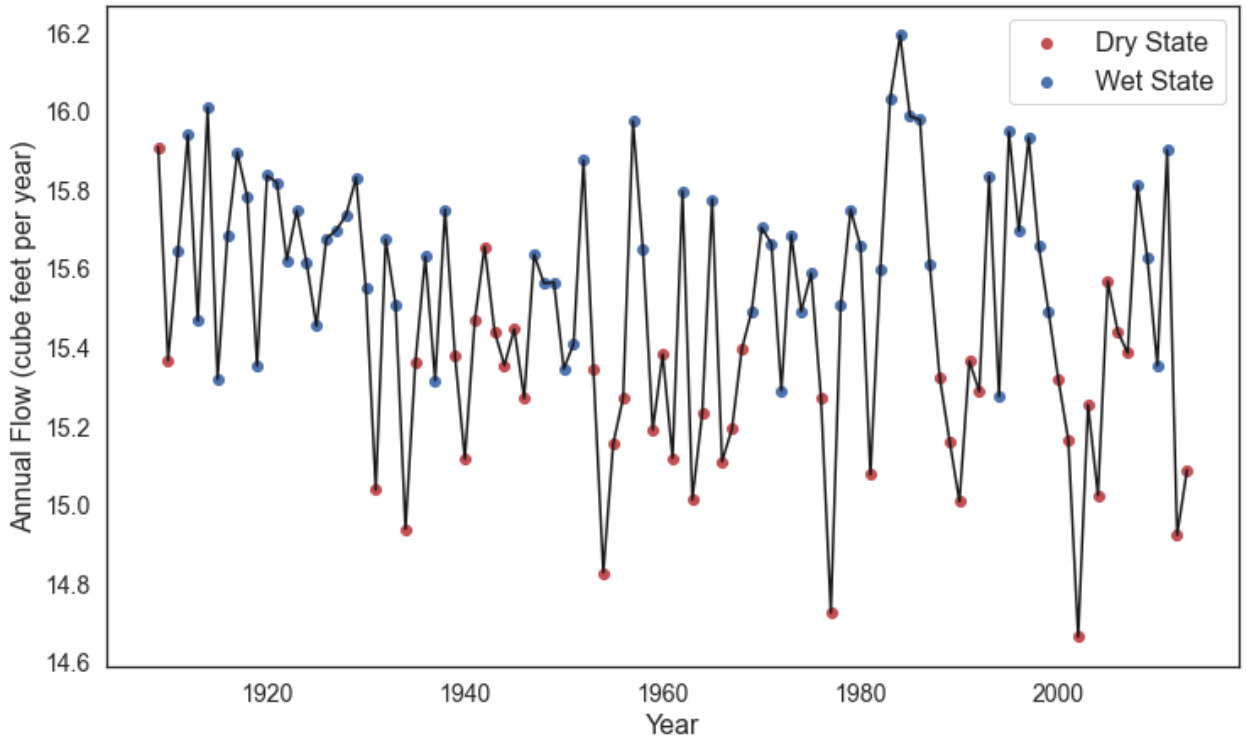
It is important to remember that the streamflow that we have observed in the region over the last century is only one instance of the hydrology that could occur since the atmosphere is an inherently stochastic system. Thus, we require a tool that will allow us to see multiple plausible realizations of the streamflow record to understand the internal variability that characterizes the historical period. One observed realization of historical streamflow is limited in its ability to capture rare extremes; plausible (but not observed) alternative instances of streamflow records can help to fill this gap. The tool that we use to develop synthetic flows for the region is a Gaussian Hidden Markov Model (HMM). If a system follows a Markov process, it switches between a number of "hidden states" dictated by a transition matrix. Each state has its own Gaussian probability distribution (defined by a mean and standard deviation) and one can draw from this distribution to create synthetic flows that fit the properties of the historical distribution. HMMs are an attractive choice for this region because they can simulate persistence (i.e., long duration droughts), which is a characteristic of the region's hydro-climatology. The figure below shows an example of a 2-state Gaussian HMM that we will be fitting for this example.

Below is the code that fits the HMM model to the last 2/3 of the historical record of log annual flows at the CO-UT stateline gauge and creates an alternative trace of 105 years. A subset of the dataset is chosen in order to minimize overfitting and to retain a set of data for validation of the model. When we fit our model, we utilize the Baum-Welch algorithm (a special version of the expectation-maximization algorithm) to find the optimal parameters that maximize the likelihood of seeing the observed flows. Ultimately, the algorithm will return a mean and standard deviation associated with each state (mus and sigmas defined below) and a 2x2 transition probability matrix that captures the likelihood of transitioning between states (P). We can also retrieve the annual hidden states across the observed series, also known as the Viterbi sequence of states, which classifies each year in a "wet" or "dry" state.

```
# Number of years for alternative trace
n_years = 105

# Import historical data that it used to fit HMM model
AnnualQ_h = pd.read_csv('data/uc_historical.csv')

# Fit the model and pull out relevant parameters and samples
logQ = np.log(AnnualQ_h)
hidden_states, mus, sigmas, P, logProb, samples, model = fitmodel.fitHMM(logQ, n_years)
```

We've fit our HMM, but what does the model look like? Let's plot the annual time series of hidden states, or the Viterbi sequence. In the code, above, we have defined that the drier state is always represented by state 0. Thus, we know that hidden_states = 0 corresponds to the dry state and hidden_states = 1 to the wet state.

```
# Plot Vitebi sequence
plotstates.plotTimeSeries(np.log(AnnualQ.iloc[:,0]), hidden_states, 'Annual Flow (cube␣
↪feet per year)')
```



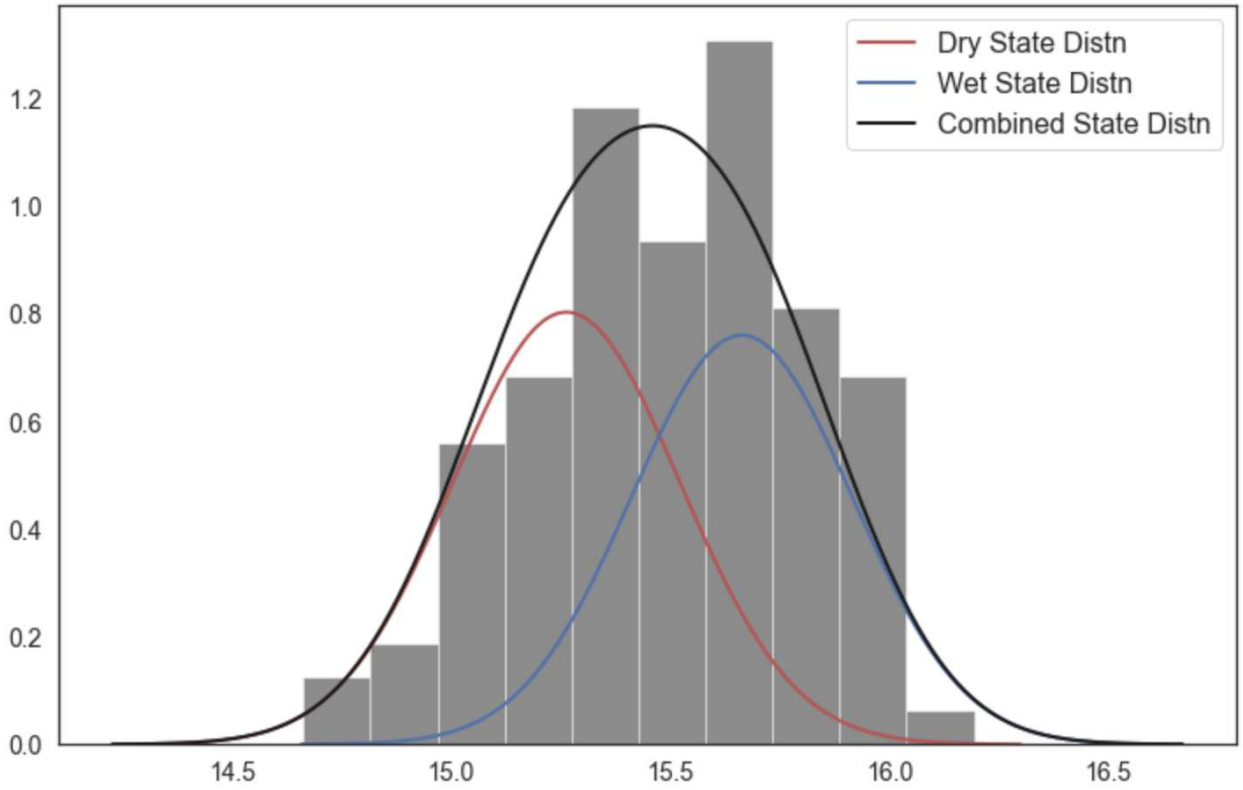
In the figure above, we see that the years with the higher log flows tend to be classified in a "wet" state and the opposite is true of the "dry" state. We can also print the transition matrix, which shows the likelihood of transitioning between states. Note that the system has a high likelihood of persisting in the same state.

```
print(model.transmat_)
```

```
[[0.65095026 0.34904974]
 [0.3205531  0.6794469 ]]
```

Let's also plot the distribution of log annual flows associated with the wet and dry states.

```
# Plot wet and dry state distributions
plotdist.plotDistribution(logQ, mus, sigmas, P)
```



The wet state distribution is characterized by a greater mean flow, but note that there is significant overlap in the tails of the distributions below which demonstrates why years with similiar flows can be classified in different states.

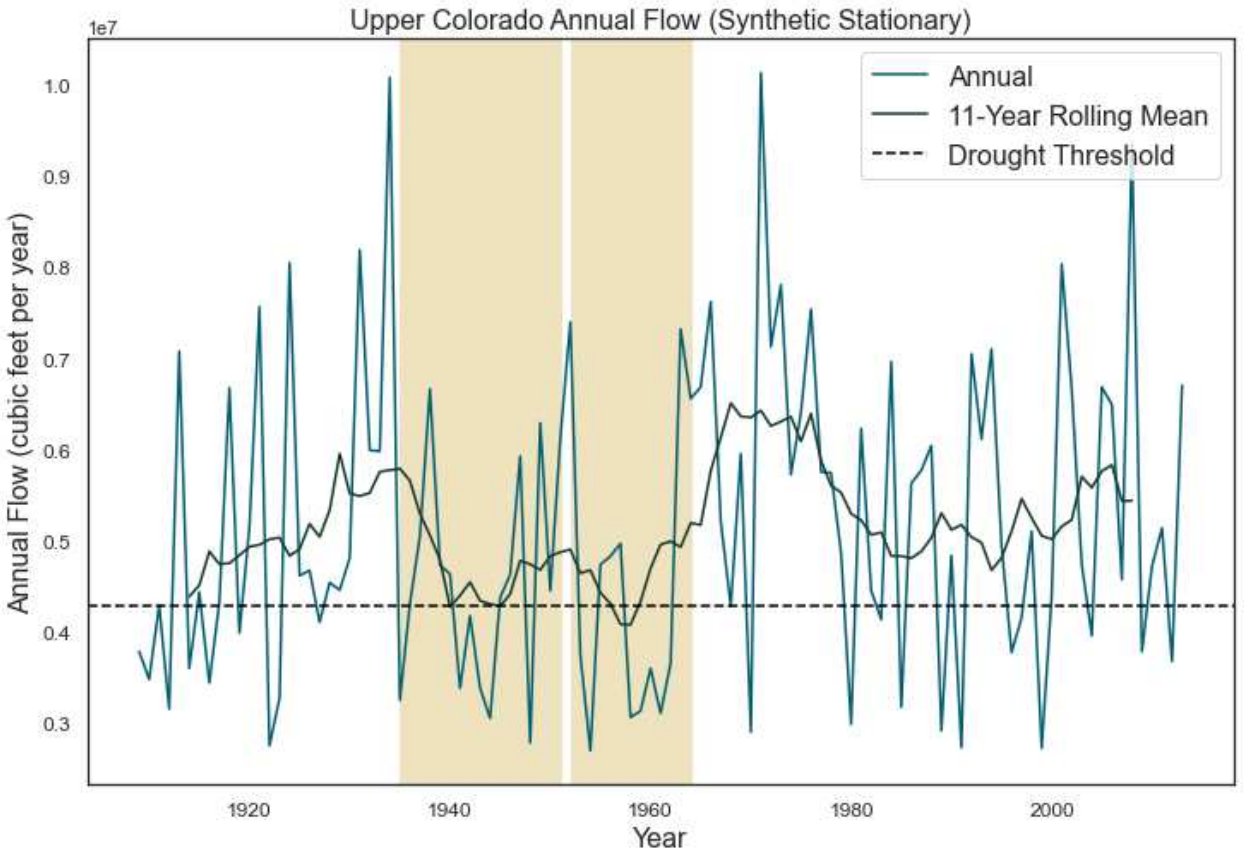
Now let's see what the drought dynamics look like in the synthetic scenario that we created using the same definition that we had used for the historical period.

```
# Retrieve samples and back-transform out of log space
AnnualQ_s = np.exp(samples[0])
AnnualQ_s = pd.DataFrame(AnnualQ_s)
AnnualQ_s['Year'] = list(range(1909, 2014))

# Define drought threshold
std=statistics.stdev(AnnualQ_s.iloc[:, 0])
threshold=np.mean(AnnualQ_s.iloc[:, 0] - (0.5 * std))

# Where does the rolling mean dip below the threshold
drought_instances = [i for i,v in enumerate(AnnualQ_s.iloc[:, 0].rolling(11).mean()) if␣
↪v < threshold]
drought_years = AnnualQ_s.iloc[:, 1].rolling(11).mean()[drought_instances]

# Visualize the streamflow scenario
fig, ax = plt.subplots(figsize=(12, 8))

# Plot the original line graph
ax.plot(AnnualQ_s.iloc[:,1],
```

(continues on next page)

(continued from previous page)

```
        AnnualQ_s.iloc[:,0],
        color='#005F73',
        label='Annual')

# Plot a 11-year rolling mean
ax.plot(AnnualQ_s.iloc[:,1],
        AnnualQ_s.iloc[:,0].rolling(11, center=True).mean(),
        color='#183A2E',
        label='11-Year Rolling Mean')

# Add labels and title
ax.axhline(y=threshold,
           color='black',
           linestyle='--',
           label='Drought Threshold')


for i in drought_years:

    # Plot a box centered around those values and with 5 years on either side.
    rect = patches.Rectangle((i - 5,
                              0),
                              11,
                              2e7,
                              linewidth=1,
                              edgecolor='#EFE2BE',
                              facecolor='#EFE2BE')

    # Add the patch to the Axes
    ax.add_patch(rect)


plt.title("Upper Colorado Annual Flow (Synthetic Stationary)",fontsize=16)
plt.xlabel("Year", fontsize=16)
plt.ylabel("Annual Flow (cubic feet per year)", fontsize=16)
mpl.rc('legend', fontsize=16)
plt.legend()
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()
plt.close()
```



You can sample from the model and create more 105-year traces and note how the location and number of decadal droughts changes. This demonstrates how different the historical record can look just within the range of natural variability. It's also important to remember that when droughts occur can also define the ultimate effect of the drought (i.e. is it a time when there is a large population growth or a time when humans can adapt by conserving or building more infrastructure?). A hydrologic drought need not manifest into an agricultural or operational drought of the same magnitude if stored surface water is available.

We externally run the HMM many times to create a dataset of 100 instances of the 105-year traces and 1000 instances of the 105-year traces that are available in the package ("synthetic_stationary_small_sample_100.csv", "synthetic_stationary_large_sample_1000"). The shaded green lines correspond to the flow duration curves (FDCs) for the generated streamflow traces in comparison with the FDC of the historical record in beige.

As expected, the stationary synthetic FDCs envelope the historical FDC and particularly, the synthetic traces offer many more instances of low flow conditions that could lead to more extreme drought conditions than what has been observed historically. It is also useful to check for convergence of samples and to determine how many samples are needed to fully represent internal variability. Above we see that the extension to 1000 instances of 105-year traces fills out regions of the FDC, including creating some more extreme drought conditions, but that additional samples will likely not fill out the FDC substantially more.

Non-Stationary Synthetic Generator to Impose Climate Changes

Now, we create flows under non-stationary conditions to get a better understanding of what flows can look like under climate changes. In order to create flows under non-stationary conditions, we can toggle the parameters of the HMM model in order to create systematic changes to the model that can represent a changing climate. The HMM has 6 parameters that define it. When we fit the historical model, the parameters that are fit represent a baseline parameter value. In this non-stationary generator, we define a range to sample these parameters from.

Parameter	Current Value	Lower Bound	Upper Bound
Log-Space Wet State Mean Multiplier	1.00	0.98	1.02
Log-Space Dry State Mean Multiplier	1.00	0.98	1.02
Log-Space Wet State Standard Deviation Multiplier	1.00	0.75	1.25
Log-Space Dry State Standard Deviation Multiplier	1.00	0.75	1.25
Change in Dry-Dry Transition Probability	0.00	-0.30	+0.30
Change in Wet-Wet Transition Probability	0.00	-0.30	+0.30

Now let's sample 1000 times from these bounds to create 1000 new parameterizations of the model. Here we use SALib and the Latin Hypercube sample function.

```
# Create problem structure with parameters that we want to sample
problem = {
    'num_vars': 6,
    'names': ['wet_mu', 'dry_mu', 'wet_std','dry_std','dry_tp',"wet_tp"],
    'bounds': [[0.98, 1.02],
               [0.98, 1.02],
               [0.75,1.25],
               [0.75,1.25],
               [-0.3,0.3],
               [-0.3,0.3]]
}

# generate 1000 parameterizations
n_samples = 1000

# set random seed for reproducibility
seed_value = 123

# Generate our samples
LHsamples = latin.sample(problem, n_samples, seed_value)
```

Now let's look at what some of the traces look like in our non-stationary generator. Let's choose a random instance from the 1000-member space and adjust the parameters accordingly.

```
# Define static parameters
n_years = 105

# Sample parameter; Adjust to any sample number from 0-999
sample = 215

# Create empty arrays to store the new Gaussian HMM parameters for each SOW
Pnew = np.empty([2,2])
piNew = np.empty([2])
```

(continues on next page)

(continued from previous page)

```
musNew_HMM = np.empty([2])
sigmasNew_HMM = np.empty([2])
logAnnualQ_s = np.empty([n_years])

# Calculate new transition matrix and stationary distribution of SOW at last node as␣
↪well as new means and standard deviations
Pnew[0, 0] = max(0.0, min(1.0, P[0, 0] + LHsamples[sample][4]))
Pnew[1, 1] = max(0.0, min(1.0, P[1, 1] + LHsamples[sample][5]))
Pnew[0, 1] = 1 - Pnew[0, 0]
Pnew[1, 0] = 1 - Pnew[1, 1]
eigenvals, eigenvecs = np.linalg.eig(np.transpose(Pnew))
one_eigval = np.argmin(np.abs(eigenvals - 1))
piNew = np.divide(np.dot(np.transpose(Pnew), eigenvecs[:, one_eigval]),
                  np.sum(np.dot(np.transpose(Pnew), eigenvecs[:,one_eigval])))

musNew_HMM[0] = mus[0] * LHsamples[sample][1]
musNew_HMM[1] = mus[1] * LHsamples[sample][0]
sigmasNew_HMM[0] = sigmas[0] * LHsamples[sample][3]
sigmasNew_HMM[1] = sigmas[1] * LHsamples[sample][2]

# Generate first state and log-space annual flow at last node
states = np.empty([n_years])
if random() <= piNew[0]:
    states[0] = 0
    logAnnualQ_s[0] = ss.norm.rvs(musNew_HMM[0], sigmasNew_HMM[0])
else:
    states[0] = 1
    logAnnualQ_s[0] = ss.norm.rvs(musNew_HMM[1], sigmasNew_HMM[1])

# Generate remaining state trajectory and log space flows at last node
for j in range(1, n_years):
    if random() <= Pnew[int(states[j-1]), int(states[j-1])]:
        states[j] = states[j-1]
    else:
        states[j] = 1 - states[j-1]

    if states[j] == 0:
        logAnnualQ_s[j] = ss.norm.rvs(musNew_HMM[0], sigmasNew_HMM[0])
    else:
        logAnnualQ_s[j] = ss.norm.rvs(musNew_HMM[1], sigmasNew_HMM[1])

# Convert log-space flows to real-space flows
AnnualQ_s = np.exp(logAnnualQ_s)-1
```

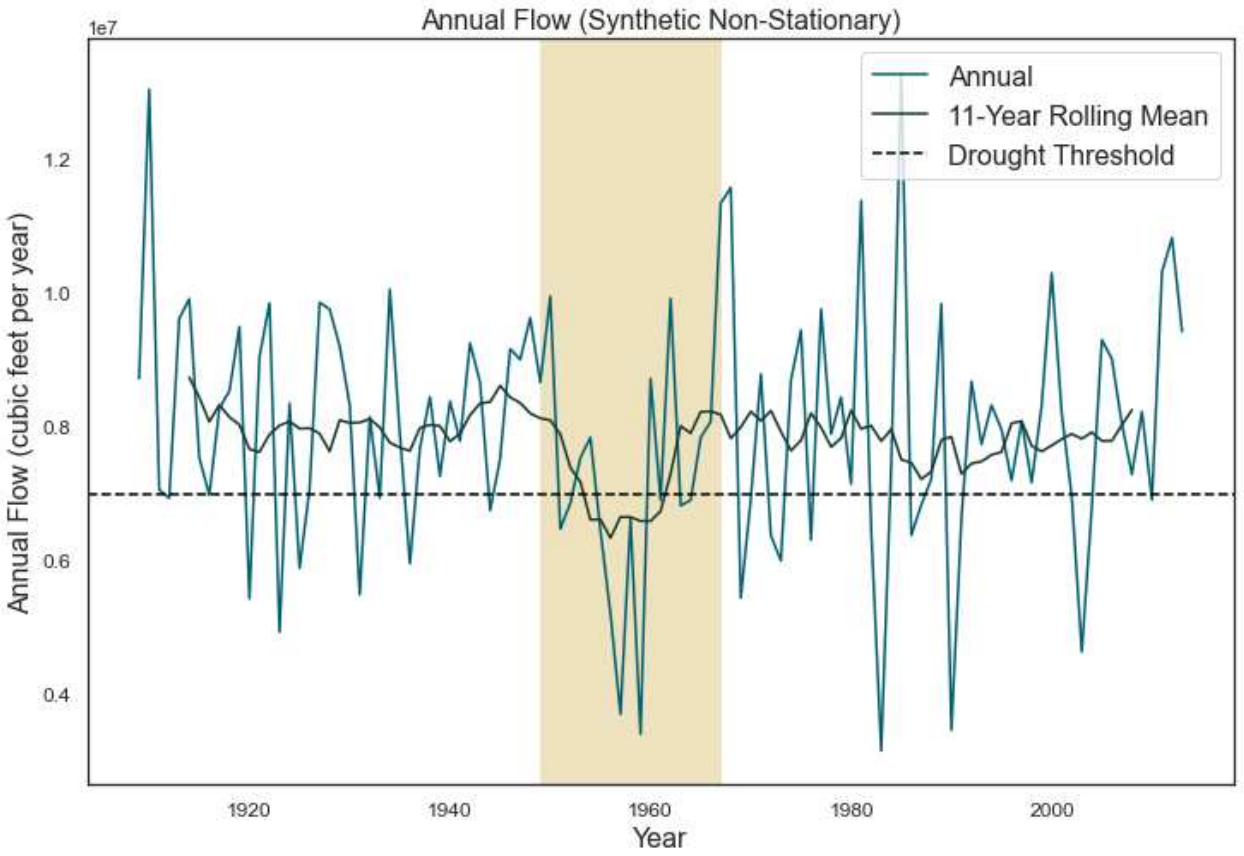
Now let's see what this synthetic trace looks like.

```
# Retrieve samples and back-transform out of log space
AnnualQ_s = pd.DataFrame(AnnualQ_s)
AnnualQ_s['Year'] = list(range(1909, 2014))

# Define drought threshold
std = statistics.stdev(AnnualQ_s.iloc[:, 0])
```

(continues on next page)

(continued from previous page)

```
threshold = np.mean(AnnualQ_s.iloc[:, 0] - (0.5 * std))

# Where does the rolling mean dip below the threshold
drought_instances = [i for i, v in enumerate(AnnualQ_s.iloc[:, 0].rolling(11).mean()) if␣
↪v < threshold]
drought_years = AnnualQ_s.iloc[:, 1].rolling(11).mean()[drought_instances]

# Visualize the streamflow scenario
fig, ax = plt.subplots(figsize=(12, 8))

# Plot the original line graph
ax.plot(AnnualQ_s.iloc[:,1],
        AnnualQ_s.iloc[:,0],
        color='#005F73',
        label='Annual')

# Plot a 11-year rolling mean
ax.plot(AnnualQ_s.iloc[:, 1],
        AnnualQ_s.iloc[:, 0].rolling(11, center=True).mean(),
        color='#183A2E',
        label='11-Year Rolling Mean')

# Add labels and title
ax.axhline(y=threshold,
           color='black',
           linestyle='--',
           label='Drought Threshold')


for i in drought_years:

    # Plot a box centered around those values and with 5 years on either side.
    rect = patches.Rectangle((i - 5,0),
                             11,
                             2e7,
                             linewidth=1,
                             edgecolor='#EFE2BE',
                             facecolor='#EFE2BE')

    # Add the patch to the Axes
    ax.add_patch(rect)


plt.title("Annual Flow (Synthetic Non-Stationary)", fontsize=16)
plt.xlabel("Year", fontsize=16)
plt.ylabel("Annual Flow (cubic feet per year)", fontsize=16)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
mpl.rc('legend', fontsize=16)
legend = plt.legend(loc="upper right")
plt.show()
plt.close()
```



Above is the example trace from the new non-stationary model. You may see fewer or more decadal drought instances. We can further summarize overall decadal drought characteristics across the samples. Let's plot a histogram of the total number of times we go below the drought threshold across these realizations.

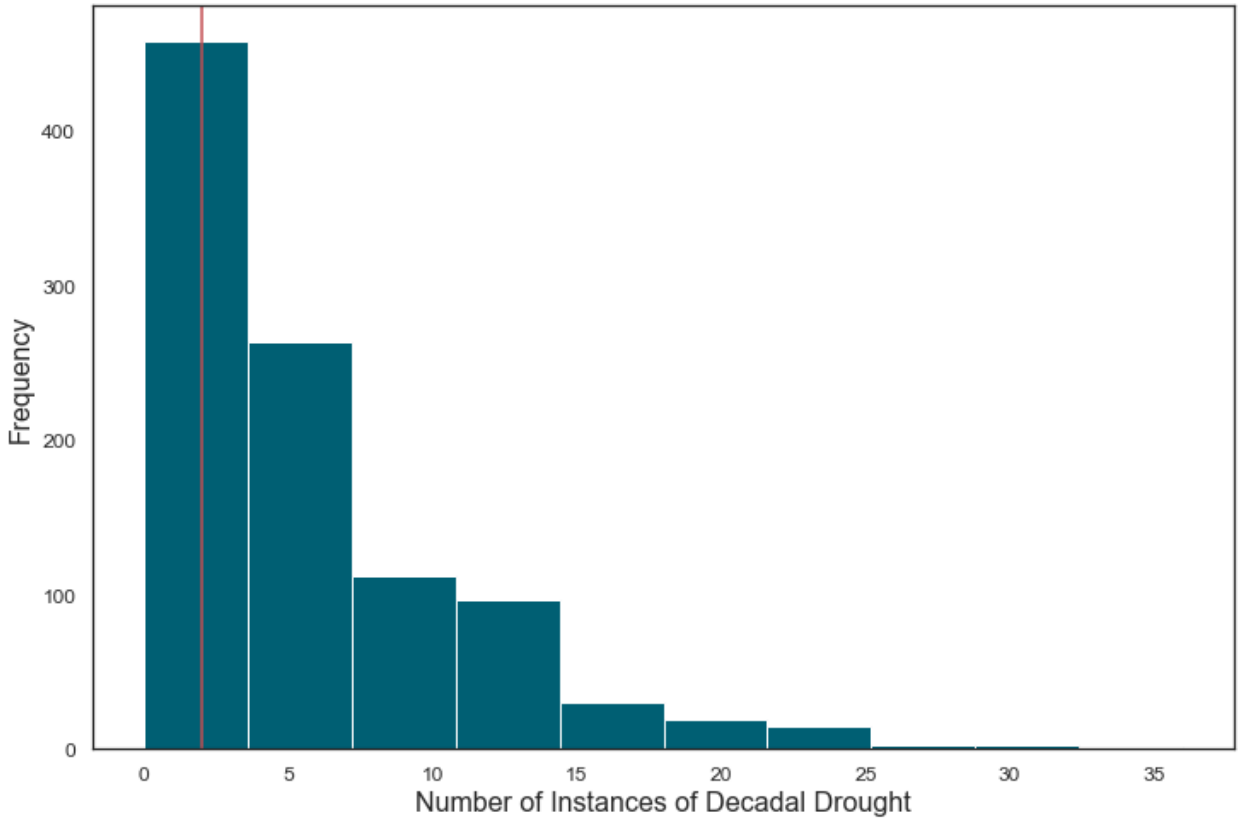
```
decadal_drought_occurrence = np.empty([1000])

for y in range(1000):

    # Create empty arrays to store the new Gaussian HMM parameters for each SOW
    Pnew = np.empty([2, 2])
    piNew = np.empty([2])
    musNew_HMM = np.empty([2])
    sigmasNew_HMM = np.empty([2])
    logAnnualQ_s = np.empty([n_years])

    # Calculate new transition matrix and stationary distribution of SOW at last node
    # as well as new means and standard deviations

    Pnew[0, 0] = max(0.0,min(1.0, P[0, 0] + LHsamples[y][4]))
    Pnew[1, 1] = max(0.0,min(1.0, P[1, 1] + LHsamples[y][5]))
    Pnew[0, 1] = 1 - Pnew[0, 0]
    Pnew[1, 0] = 1 - Pnew[1, 1]
    eigenvals, eigenvecs = np.linalg.eig(np.transpose(Pnew))
    one_eigval = np.argmin(np.abs(eigenvals - 1))
    piNew = np.divide(np.dot(np.transpose(Pnew), eigenvecs[:, one_eigval]),
                      np.sum(np.dot(np.transpose(Pnew), eigenvecs[:, one_eigval])))
```

(continues on next page)

(continued from previous page)

```
    musNew_HMM[0] = mus[0][0] * LHsamples[y][1]
    musNew_HMM[1] = mus[1][0] * LHsamples[y][0]
    sigmasNew_HMM[0] = sigmas[0][0] * LHsamples[y][3]
    sigmasNew_HMM[1] = sigmas[1][0] * LHsamples[y][2]

    # Generate first state and log-space annual flow at last node
    states = np.empty([n_years])
    if random() <= piNew[0]:
        states[0] = 0
        logAnnualQ_s[0] = ss.norm.rvs(musNew_HMM[0], sigmasNew_HMM[0])
    else:
        states[0] = 1
        logAnnualQ_s[0] = ss.norm.rvs(musNew_HMM[1], sigmasNew_HMM[1])

    # generate remaining state trajectory and log space flows at last node
    for j in range(1, n_years):
        if random() <= Pnew[int(states[j-1]), int(states[j-1])]:
            states[j] = states[j-1]
        else:
            states[j] = 1 - states[j-1]

        if states[j] == 0:
            logAnnualQ_s[j] = ss.norm.rvs(musNew_HMM[0], sigmasNew_HMM[0])
        else:
            logAnnualQ_s[j] = ss.norm.rvs(musNew_HMM[1], sigmasNew_HMM[1])

    # Convert log-space flows to real-space flows
    AnnualQ_s = np.exp(logAnnualQ_s) - 1
    AnnualQ_s = pd.DataFrame(AnnualQ_s)
    AnnualQ_s['Year'] = list(range(1909, 2014))

    # Define drought threshold
    std = statistics.stdev(AnnualQ_s.iloc[:, 0])
    threshold = np.mean(AnnualQ_s.iloc[:, 0] - (0.5 * std))

    # Where does the rolling mean dip below the threshold
    drought_instances = [i for i, v in enumerate(AnnualQ_s.iloc[:, 0].rolling(11).
↪mean()) if v < threshold]
    decadal_drought_occurrence[y] = len(drought_instances)
```

```
fig, ax = plt.subplots(figsize=(12, 8))
ax.hist(decadal_drought_occurrence,label='Non-Stationary generator',color="#005F73")
ax.set_xlabel('Number of Instances of Decadal Drought',fontsize=16)
ax.set_ylabel('Frequency',fontsize=16)
ax.axvline(x=2, color='r', linestyle='-',label='Observed')
mpl.rc('legend', fontsize = 16)
plt.xticks(fontsize = 12)
plt.yticks(fontsize = 12)
plt.show()
plt.close()
```



Note how many more instances of the decadal droughts we are creating with the non-stationary generator than our observed 105-year trace which creates a rich space in which we can test our models. Just as we did with the stationary generator, we can externally run the non-stationary generator to create 10,000 instances of the 105-year traces that are available in the package ("synthetic_nonstationary_large_sample_10000.csv"). The shaded green and blue lines correspond to the FDCs for the stationary and non-stationary generated streamflow traces in comparison with the FDC of the historical record in beige. Note how the non-stationary generator produces even more drought extremes than the stationary non-synthetic traces.

Placing CMIP5 Projections in the Context of Non-Stationary Flows

We have broadened the drought conditions that we are creating which that can be very useful to understand how our water systems model performs under potentially extreme scenarios. However, it's useful to compare our bottom-up synthetically generated flows in the context of global physically-driven CMIP5 projections to get a better understanding of how the two approaches compare. We first aquire 97 CMIP5 projections from the Colorado River Water Availability Study (CWCB, 2012). In each of these projections, monthly precipitation factor changes and temperature delta changes were computed between mean projected 2035–2065 climate statistics and mean historical climate statistics from 1950–2013. These 97 different combinations of 12 monthly precipitation multipliers and 12 monthly temperature delta shifts were applied to historical precipitation and temperature time series from 1950–2013. The resulting climate time series were run through a Variable Infiltration Capacity (VIC) model of the UCRB, resulting in 97 time series of projected future streamflows at the Colorado-Utah state line.

We fit an HMM to each trace of projected streamflow and get a set of corresponding HMM parameters. Then we take the ratio between these parameters and the baseline HMM parameters that we calculated earlier in the notebook in order to calculate the multipliers associated with each CMIP5 projection. This is all done externally, so we import the resulting multipliers in the next line.

```
# Read in CMIP5 and paleo multipliers
CMIP5_multipliers = pd.read_csv('data/CMIP5_SOWs.txt', header=None, sep=" ")
```

Let's plot a response surface that will allow us to see how combinations of HMM parameters tend to influence decadal drought. In order to get a continuous surface, we'll fit a non-linear regression to the parameter values and then predict the decadal drought over a set of grid points. We fit the response surface for two parameters that should have an affect on decadal drought: the dry distribution mean and the dry-dry transition probabilites.

```
# Choose two parameters to fit the response surface for
mu_dry=[i[1] for i in LHsamples]
tp_dry=[i[4] for i in LHsamples]

# Create an interpolation grid
xgrid = np.arange(np.min(mu_dry),
                  np.max(mu_dry),
                  (np.max(mu_dry) - np.min(mu_dry)) / 100)

ygrid = np.arange(np.min(tp_dry),
                  np.max(tp_dry),
                  (np.max(tp_dry) - np.min(tp_dry)) / 100)

# Fit regression
d = {'Dry_Tp': tp_dry,
     'Dry_Mu': mu_dry,
     'Drought_Occurrence':decadal_drought_occurrence}

df = pd.DataFrame(d)
df['Intercept'] = np.ones(np.shape(df)[0])
df['Interaction'] = df['Dry_Tp'] * df['Dry_Mu']
cols = ['Intercept'] + ['Dry_Mu'] + ['Dry_Tp'] + ['Interaction']
ols = sm.OLS(df['Drought_Occurrence'], df[cols])
result = ols.fit()

# Calculate drought occurrence for each grid point
X, Y = np.meshgrid(xgrid, ygrid)
x = X.flatten()
y = Y.flatten()
grid = np.column_stack([np.ones(len(x)), x, y, x * y])
z = result.predict(grid)
z[z < 0.0] = 0.0 # replace negative shortage predictions with 0
```

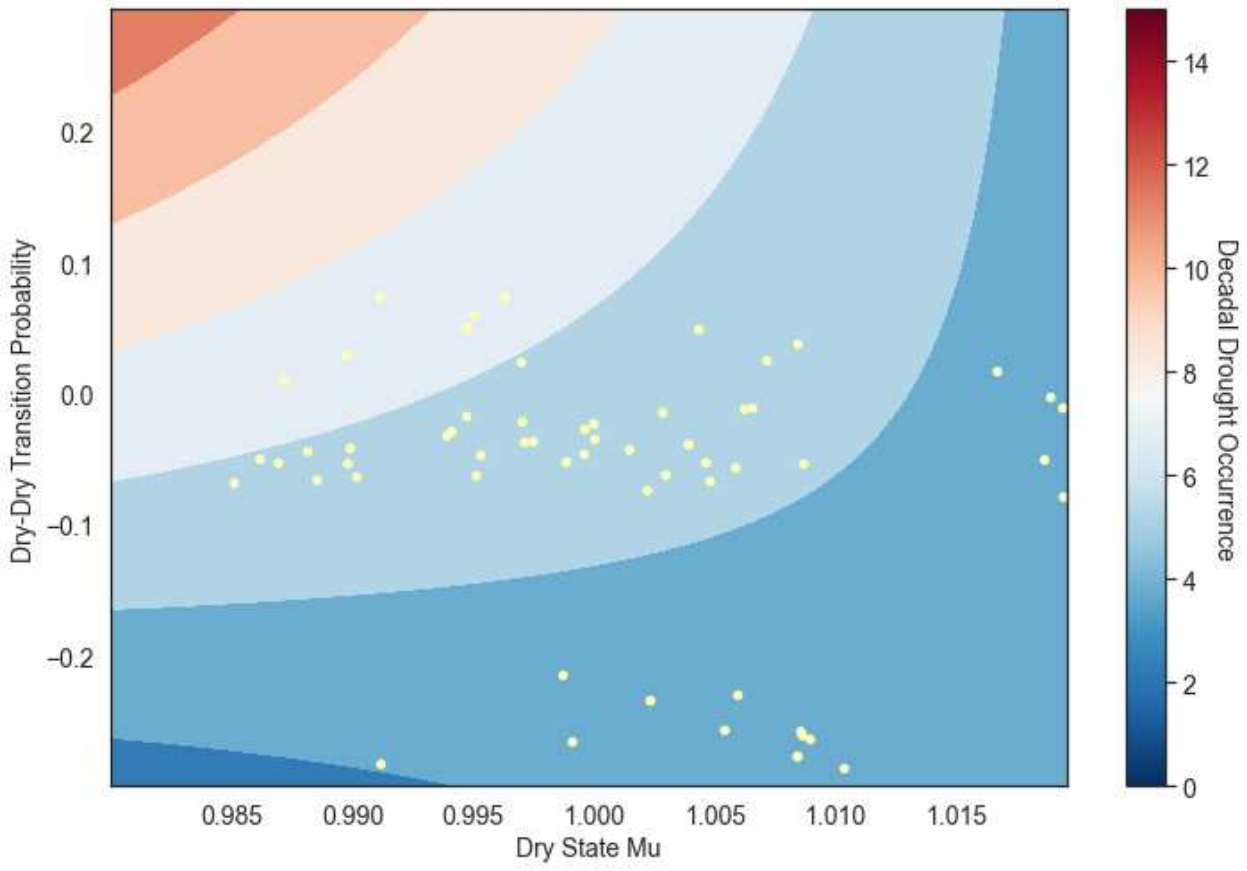
Let's plot our results:

```
# Set color gradient for response surface
drought_map = mpl.colormaps.get_cmap('RdBu_r')

# Reshape our predicted drought occurrence and define bounds of colors
Z = np.reshape(z, np.shape(X))
vmin = np.min([np.min(z), np.min(df['Drought_Occurrence'].values)])
vmax = 15
norm = mpl.colors.Normalize(vmin, vmax)

# Plot response surface and CMIP5 projections
```

(continues on next page)

(continued from previous page)

```
fig, ax = plt.subplots(figsize=(12, 8))
ax.contourf(X, Y, Z, cmap=drought_map, norm=norm)
ax.scatter(CMIP5_multipliers.iloc[:,7],
           CMIP5_multipliers.iloc[:,12],
           c='#ffffb3',
           edgecolor='none',
           s=30)
cbar = ax.figure.colorbar(mpl.cm.ScalarMappable(norm=norm, cmap=drought_map), ax=ax)
ax.set_xlim(np.nanmin(X), np.nanmax(X))
ax.set_ylim(np.nanmin(Y), np.nanmax(Y))
ax.set_xlabel('Dry State Mu', fontsize=14)
ax.set_ylabel('Dry-Dry Transition Probability', fontsize=14)
ax.tick_params(axis='both', labelsize=14)
cbar.ax.set_ylabel('Decadal Drought Occurrence', rotation=-90, fontsize=14, labelpad=15)
cbar.ax.tick_params(axis='y',labelsize=14)
plt.show()
plt.close()
```



We see the influence of the dry state mean and dry-dry transition parameters. We're likely to see more decadal droughts when we (1) increase the dry-dry transition probability, which inherently will increase persistence of the dry state, and (2) when we make the dry state log mean drier. Note that the CMIP5 scenarios tend to span the extent of the dry mean sample space, but are less representative of the dry transition probability sample space, which suggests that the types of hydrological droughts represented in the projections tend to only be wetter to slightly drier than our baseline. Both methods of producing these scenarios are valid, though studies have suggested that globally-resolved GCMs may be inappropriate to represent regional extremes. Ultimately, if your goal is to produce a variety of ensembles that are

characterized by many different drought characteristics, you will likely find that a generator approach will serve this purpose better.

Tips to Create an HMM-Based Generator for your System

In this tutorial, we demonstrated how to fit an HMM-based generator for a single gauge located in the Upper Colorado River Basin. In order to apply this methodology to your problem, you will need to first ask:

(1) Is this model appropriate for my location of interest? We have applied this style of generator to locations where persistent wet and dry states are characteristic, which tends to be in the Western US. Ultimately the best way to judge if an HMM is useful for your application is to fit the model and explore the resulting distributions. Are there two (or more) distinct states that emerge? If not, then your location may not exhibit the type of persistence that an HMM-based generator is useful for. You can consider exploring other styles of generators such as the Kirsch-Nowak generator (Kirsch et al., 2013).

(2) Do I have the right datasets? We use annual data for our location of interest. In this notebook, the HMM is fit to log annual flows. Ultimately, it can be disaggregated to daily flows (using a reference historical daily dataset) to be useful in water resources operational applications. You could also disaggregate to a finer resolution than daily if the historical dataset exists.

If you meet these requirements, feel free to proceed through fitting the model using the code available in the notebook. Be sure to consider the appropirate number of samples to generate (both in a stationary and non-stationary case). Make sure that you test multiple sample sizes and continue to increase your sample size until you converge to a consistent representation of extremes. What is the appropriate number of LHS samples of the parameters to use? In this experiment we used 1,000 samples of parameters due to extensive stability tests described in Quinn et al. (2020).

Finally, to learn more about this test case refer to Hadmichael et al. (2020a) and Hadmichael et al. (2020b). For another study on synthetic drought generation to support vulnerability assessments in the Research Triangle region of North Carolina, please refer to Herman et al. (2016)

References

Ault, T. R., Cole, J. E., Overpeck, J. T., Pederson, G. T., & Meko, D. M. (2014). Assessing the risk of persistent drought using climate model simulations and paleoclimate data. Journal of Climate, 27(20), 7529-7549.

CWCB (2012).Colorado River Water Availability Study Phase I Report. Colorado Water Conservation Board

Hadjimichael, A., Quinn, J., Wilson, E., Reed, P., Basdekas, L., Yates, D., & Garrison, M. (2020a). Defining robustness, vulnerabilities, and consequential scenarios for diverse stakeholder interests in institutionally complex river basins. Earth's Future, 8(7), e2020EF001503.

Hadjimichael, A., Quinn, J., & Reed, P. (2020). Advancing diagnostic model evaluation to better understand water shortage mechanisms in institutionally complex river basins. Water Resources Research, 56(10), e2020WR028079.

Herman, J. D., Zeff, H. B., Lamontagne, J. R., Reed, P. M., & Characklis, G. W. (2016). Synthetic drought scenario generation to support bottom-up water supply vulnerability assessments. Journal of Water Resources Planning and Management, (11), 04016050.

Kirsch, B. R., Characklis, G. W., & Zeff, H. B. (2013). Evaluating the impact of alternative hydro-climate scenarios on transfer agreements: Practical improvement for generating synthetic streamflows. Journal of Water Resources Planning and Management, 139(4), 396-406.

Overpeck, J.T. & Udall, B. (2020) "Climate change and the aridification of North America." Proceedings of the national academy of sciences 117.22 11856-11858.

Quinn, J. D., Hadjimichael, A., Reed,P. M., & Steinschneider, S. (2020). Canexploratory modeling of water scarcity vulnerabilities and robustness bescenario neutral?Earth's Future,8,e2020EF001650. https://doi.org/10.1029/2020EF001650Received

Williams, A. P., Cook, B. I., & Smerdon, J. E. (2022). Rapid intensification of the emerging southwestern North American megadrought in 2020–2021. Nature Climate Change, 12(3), 232-234.

B.7 Model Calibration with Markov chain Monte Carlo Tutorial

Note:

Run the tutorial interactively: MCMC Notebook.

Please be aware that notebooks can take a couple minutes to launch.

To run the notebooks yourself, download the files here and use these requirements.

Community Contribution

This tutorial was contributed by the community. Use the citation below in additional to the main citation when referencing this code:

Srikrishnan, V. (2025). Model Calibration with Markov Chain Monte Carlo Tutorial (v1.0.0). MSD-LIVE Data Repository. https://doi.org/10.57931/2565322

B.7.1 Model Calibration with Markov chain Monte Carlo

The purpose of this tutorial is to demonstrate how to use Markov chain Monte Carlo (MCMC) to calibrate a model. By calibration, we mean the selection of model parameters (and, when relevant, structures). This tutorial notebook will build on the *HYMOD sensitivity analysis notebook*.

```
import math
from matplotlib import pyplot as plt
import numpy as np
import random
from scipy import stats

import msdbook
import msdbook.hymod
from msdbook.package_data import load_hymod_input_file
```

```
# Set seed for reproducibility
random.seed(1)
```

```
# from msdbook.install_supplement import install_package_data
# install_package_data()
```

MCMC for Model Calibration and Uncertainty Quantification

Bayesian Uncertainty Quantification

A common goal in model development and diagnostics is *calibration*, or the identification of model structures and parameters which are consistent with data. While models can be calibrated through hand-tuning parameters or minimizing simple error metrics such as root-mean-square-error (RMSE), these approaches can underrepresent the probabilistic nature of the data-generating process, as well as the potential for multiple model configurations to be consistent with the data. Probabilistic uncertainty quantification, which is the topic of this notebook, can address these concerns.

The notion that different model configurations can be consistent with data to different degrees is related to the Bayesian interpretation of probability as representing the degree of belief in an outcome. Bayesian uncertainty quantification has two characteristics:

1. Obtaining probability distributions over model structures and/or parameter values θ reflecting consistency with prior beliefs $p(\theta)$ and data y. These probabilities represent the *posterior* probability emerging from Bayes' Rule,

$$p(\theta|y) \propto p(y|\theta)p(\theta),$$

   where $p(y|\theta)$ is the *likelihood* of seeing the data given the parameterization θ. The likelihood captures the probability model by which the data is observed and can include bias terms, observation errors, or other influences.

2. The use of prior distributions. Priors are classically thought of as means to express beliefs about admissible or plausible values, but they can also be used to limit the degree of overfitting by requiring more data to force more extreme parameter estimates.

The fundamental challenge is that of sampling from the posterior probability distribution, which may not have a nice representation. MCMC is one family of approaches to solving that challenge.

Rejection Sampling

Many methods for drawing samples (or simulating) from "non-standard" distributions rely on an accept-reject approach, where samples are generated from an easier-to-simulate *proposal* distribution and are probabilistically accepted or rejected based on the relative probability density between the proposal and the target distribution.

To illustrate this accept-reject concept, we can use an algorithm called *rejection sampling*. Rejection sampling involves simulation of independent and identically-distributed (i.i.d.) samples from a proposal distribution which "covers" the target distribution. More specifically, let $\pi(x)$ be the (known) density of the target distribution, and let $g(x)$ be a density which satisfies $\pi(x) < Mg(x)$ where $1 < M < \infty$; in other words. This covering property is essential to ensure that values are proposed across the entire range of π with positive probability (called the *support*). We can see an example of this in the figure below.

```
# Define the target mixture model pdf.
# This represents a 50/50 mixture of N(-1, 0.75) and N(1, 0.4).
def mixture_pdf(x):
    return 0.5 * stats.norm.pdf(x, loc=-1, scale=0.75) + 0.5 * stats.norm.pdf(x, loc=1,␣
↪scale=0.4)

# Create an array of x values from -5 to 5 with a step of 0.01.
x = np.arange(-5, 5, 0.01)

# Set the number of samples and the constant M for rejection sampling.
nsamp = 10000
M = 2.5
```

(continues on next page)

(continued from previous page)

```
# Draw nsamp samples from the proposal distribution (Normal(0, 1.5)).
u = np.random.uniform(0, 1, nsamp)
y = np.random.normal(0, 1.5, nsamp)

# Calculate the proposal density g and target density f at y.
g = stats.norm.pdf(y, loc=0, scale=1.5)
f = mixture_pdf(y)

# Acceptance criterion: u < f / (M * g)
keep_samp = u < (f / (M * g))
accepted = y[keep_samp]

# Estimate density using Gaussian KDE.
kde = stats.gaussian_kde(accepted)
y_vals = np.linspace(accepted.min(), accepted.max(), 200)

# Plot the target mixture model and the proposal distribution.
fig, axs = plt.subplots(1, 2, figsize=(14, 6))

axs[0].plot(x, mixture_pdf(x), lw=2, color='red', label='Target')
axs[0].plot(x, 2.5 * stats.norm.pdf(x, loc=0, scale=1.5), lw=2, color='blue', label=
↪'Proposal (M=2.5)')
axs[0].set_xlabel(r'$x$', size=16)
axs[0].set_ylabel('Density', size=16)

axs[0].legend()

axs[1].hist(accepted, bins=30, density=True, alpha=0.7, edgecolor='black', label='Kept␣
↪Samples')
axs[1].plot(x, mixture_pdf(x), lw=2, color='black', label='True Target')

axs[1].plot(y_vals, kde(y_vals), color='red', label='Sampled Density')
axs[1].set_xlabel(r'$x$', size=16)
axs[1].set_ylabel('Density', size=16)
axs[1].legend(loc='upper left')

plt.show()
```

The rejection sampling algorithm is then:

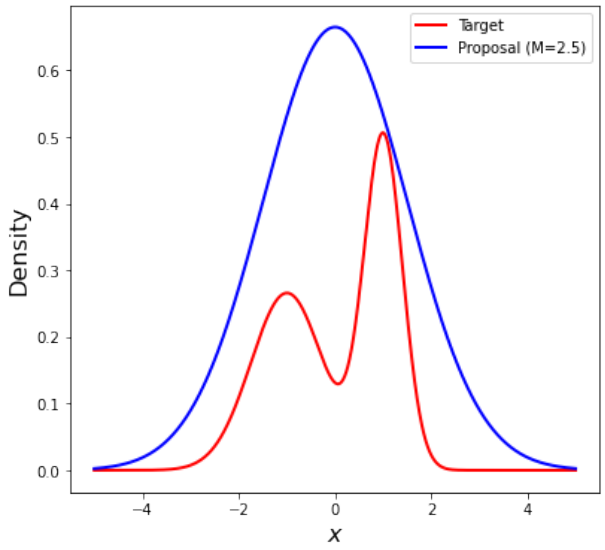
1. Simulate $Y_i \sim g(x)$;
2. Simulate $U_i \sim \text{Uniform}(0, 1)$.
3. Accept Y_i if $U_i <= \pi(Y_i)/Mg(Y_i)$.

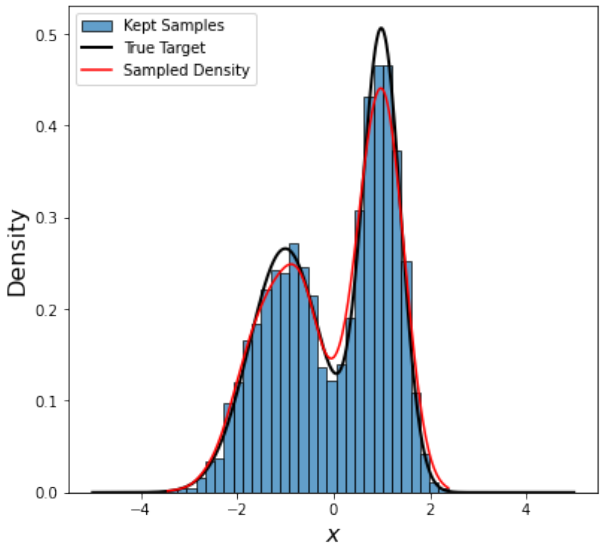
In other words, Y is accepted as a sample from $\pi(x)$ with probability $\rho = \pi(x)/Mg(x)$. As a result of this procedure, the proposals (Y_i, U_i) are uniformly distributed over the area under the curve of $g(x)$, and the rejection procedure results in the accepted samples being uniformly distributed over the area under the curve of $\pi(x)$, as desired.

An illustration of rejection sampling can be seen below.

```
# Set parameters
nsamp = 500
M = 3.5
```

(continues on next page)





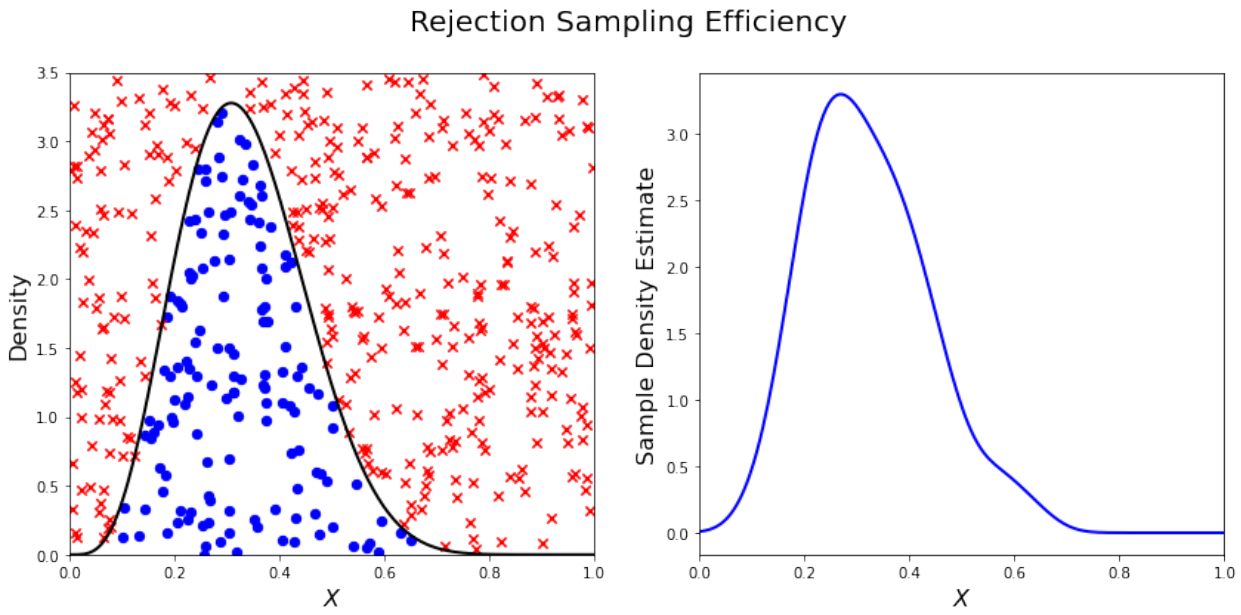
(continued from previous page)

```
# Generate nsamp samples from Uniform(0, 1) for u and y
u = np.random.uniform(0, 1, nsamp)
y = np.random.uniform(0, 1, nsamp)

# Compute the Beta(5, 10) pdf at each y value
f = stats.beta.pdf(y, 5, 10)

# Determine which samples to keep: condition (M * u) < f
keep_samp = (M * u) < f

# Create the figure with the desired size.
fig, axs = plt.subplots(1, 2, figsize=(14, 6))
fig.suptitle("Rejection Sampling Efficiency", fontsize=20)

# First plot
axs[0].set_xlim(0, 1)
axs[0].set_ylim(0, 3.5)
axs[0].set_xlabel(r'$X$', fontsize=16)
axs[0].set_ylabel('Density', fontsize=16)

# Plot the Beta(5, 10) density line
x_vals = np.linspace(0, 1, 200)
axs[0].plot(x_vals, stats.beta.pdf(x_vals, 5, 10), color='black', lw=2, label='Beta(5,10)
↪')

for i in range(len(y)):
    if keep_samp[i]:
        marker = 'o'
        color = 'blue'
    else:
        marker = 'x'
        color = 'red'
```

(continues on next page)

(continued from previous page)

```
    axs[0].scatter(y[i], M * u[i], color=color, marker=marker, s=40)

# Second plot
accepted = y[keep_samp]
# Create a density estimate using Gaussian KDE.
kde = stats.gaussian_kde(accepted)

# Define x values for the density plot.
x1_vals = np.linspace(0, 1, 200)
density_vals = kde(x1_vals)

axs[1].plot(x1_vals, density_vals, linewidth=2, color='blue')
axs[1].set_xlabel(r'$X$', fontsize=16)
axs[1].set_ylabel("Sample Density Estimate", fontsize=16)
axs[1].set_xlim(0, 1)

plt.show()
```



There are several downsides and practical challenges associated with rejection sampling, which helps motivate the use of Markov chain Monte Carlo methods, such as the Metropolis-Hastings algorithm. In particular, the expected value of the acceptance rate is approximately $1/M$, which means choosing a proposal density that minimizes M while still covering π is valuable. However, this can be challenging for complex target distributions or, in particular, high-dimensional distributions.

Markov chain Monte Carlo

Markov chain Monte Carlo (MCMC) is a family of algorithms to sample from (almost) arbitrary probability distributions. The underlying idea is to construct a Markov chain of samples whose stationary distribution is the same as the target distribution π. That the target distribution is the *stationary* distribution of the constructed chain is important for *diagnostics*.

While there are many MCMC algorithms, the most fundamental is the **Metropolis-Hastings algorithm**. We will focus on the Metropolis-Hastings algorithm in this tutorial, as it makes the MCMC procedure and the impacts of choices transparent, though *other approaches* can scale better.

The Metropolis-Hastings algorithm relies on an accept-reject step to ensure that the resulting Markov transition probabilities have the right properties to ensure convergence to the target distribution π. This requires the specification of a *proposal distribution* q.

0. Start from an initial parameter value

$$x_0.$$

Given

$$X_t = x_t :$$

1. Generate

$$Y_t \sim q(y|x_t);$$

2. Set

$$X_{t+1} = Y_t$$

with probability

$$\rho(x_t, Y_t)$$

where

$$\rho(x, y) = \min\left\{\frac{\pi(y)}{\pi(x)}\frac{q(x|y)}{q(y|x)}, 1\right\},$$

else set

$$X_{t+1} = x_t.$$

Often the proposal distribution is chosen to be symmetric, $q(y|x) = q(x|y)$, so the accept-reject probability $\rho(x, y) = \min\{\pi(y)/\pi(x), 1\}$. We will look later at the impact of choices of q and some adaptive approaches.

We can visualize how the algorithm works in practice with the figure below. The impact of the accept-reject step is that proposals which increase the target probability relative to the current value ($\pi(Y_t) > \pi(X_t)$, as in the top panel) will always be accepted, while proposals which decrease the target probability (as in the bottom panel) will be accepted based on the ratio of $\pi(Y_t)/\pi(X_t)$. In this case, the probability of accepting the proposal of y is approximately 0.3. If the proposal is accepted, $X_{t+1} = Y_t$ and the new proposal is centered on Y_t, while if it is rejected, $X_{t+1} = x_t$ and the value is repeated in the resulting Markov chain.

The sequential accept-reject step and the localization of the proposal density on the current sample X_t is what results in the autocorrelation of the Markov chain, which has implications for the use of the resulting samples for Monte Carlo estimation and simulation. Namely, the *effective sample size*

$$N_{\text{eff}} = \frac{N}{1 + 2\sum_{i=1}^{\infty}\rho_i},$$

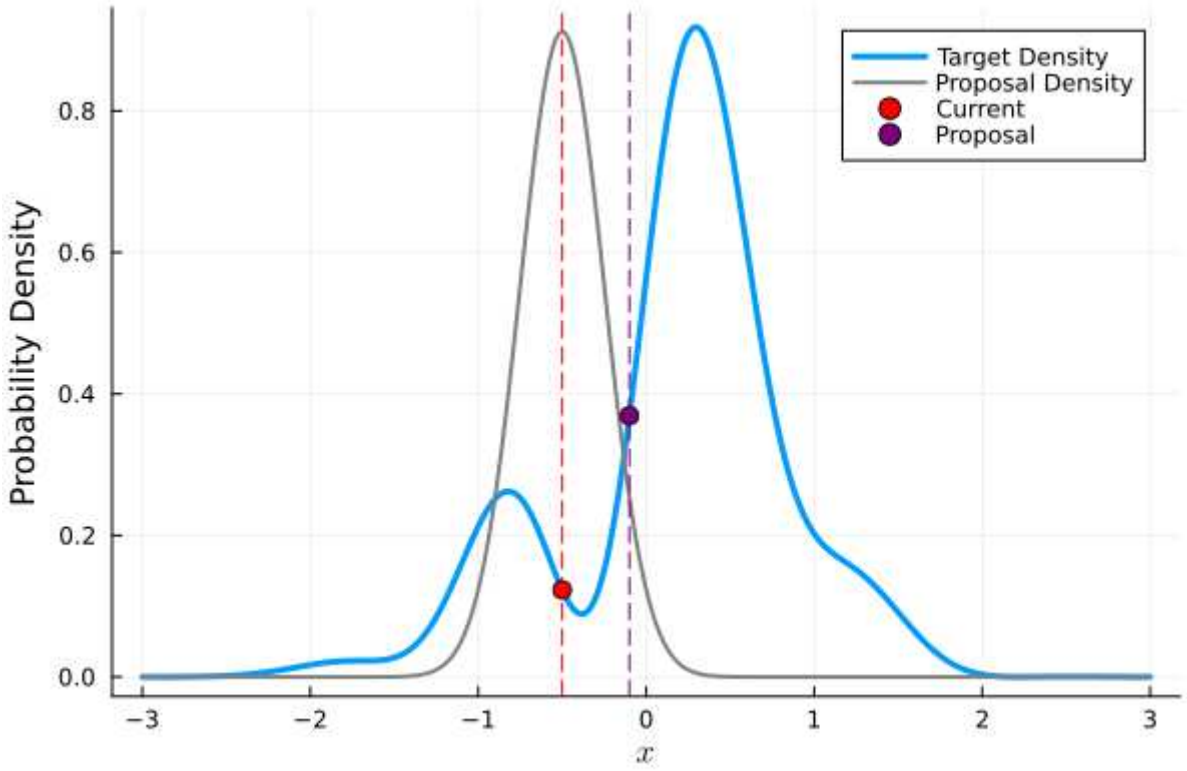


Fig. 2.3: Metropolis-Hastings step where the proposal is always accepted as it has higher probability according to the target density π than the current value

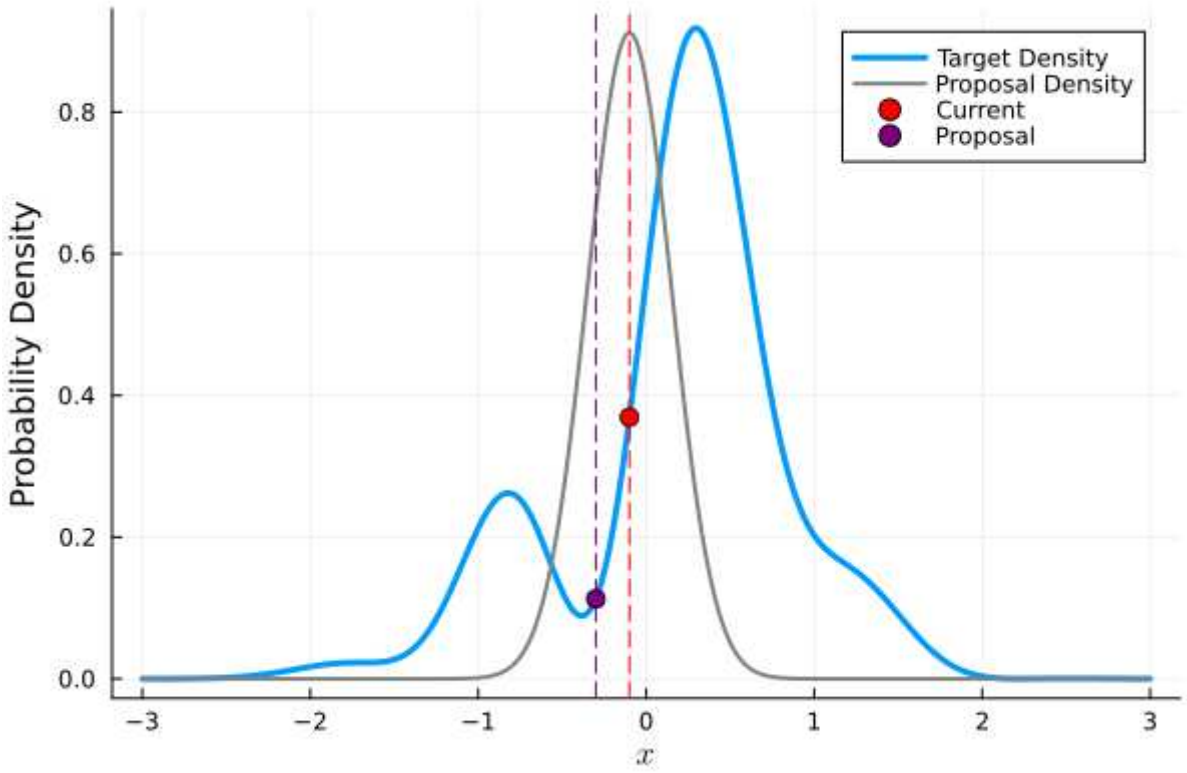


Fig. 2.4: Metropolis-Hastings step where the proposal may not be accepted as it has lower probability according to the target density π than the current value. In this case, $\pi(y)/\pi(x) \approx 0.30$, so the proposal will be accepted with probability 30%.

is always less than N, and can be dramatically smaller if the resulting chain has very high autocorrelation. N_{eff} is the value that should be used to estimate the Monte Carlo standard error for any resulting estimatation.

However, this autocorrelation across the samples is a potentially small price to pay for the flexibility of MCMC. The local proposals mean that there is no need to find a "general" covering distribution, as in rejection sampling, which allows the Metropolis-Hastings algorithm to be practical in higher dimensions and for distributions with unexpected features such as multi-modality.

In code form, the Metropolis-Hastings algorithm looks like this.

```
# Inputs:
#   - num_iter: Int, number of iterations to run Metropolis_Hastings algorithm
#   - proposal_sd: List or vector of proposal standard deviations, corresponding to each␣
↪parameter
#   - p0: initial parameter vector
#   - logposterior: function to calculate the log-posterior for a given parameter vector
# Outputs:
#   - parameters: matrix of sampled parameters, num_iter x num_parameters
#   - lp: vector of log-posterior values for the sampled parameters
#   - accept_rate: Float of the percentage of proposals which were accepted.

def metropolis(num_iter, proposal_sd, p0, logposterior):
    # Initialize our lists for sampled parameters and log-posterior values
    # Create empty array
    parameters = np.zeros((num_iter+1, np.size(p0)))
    lp = np.zeros(num_iter+1)
    # Set initial values
    parameters[0, :] = p0
    lp[0] = logposterior(p0)
    # Set up proposal covariance matrix
    cov = stats.Covariance.from_diagonal(np.square(proposal_sd))
    acceptances = 0
    for i in range(1, num_iter + 1):
        # Propose a new state
        proposal = stats.multivariate_normal.rvs(mean=parameters[i-1, :], cov=cov)
        # Calculate the acceptance probability
        lp_proposal = logposterior(proposal)
        p_accept = lp_proposal - lp[i-1]
        p_accept = np.min([p_accept, 0])
        u = stats.uniform.rvs()
        # Accept with probability p_accept
        if u < np.exp(p_accept):
            # Add the proposed parameter to the end of the list `parameters`
            parameters[i, :] = proposal
            # Add the corresponding posterior score to the end of that list too
            acceptances += 1
            lp[i] = lp_proposal
        # Reject with probability 1-p_accept
        else:
            # Add another copy of the current parameter value to the end of the list␣
↪`parameters`
            parameters[i, :] = parameters[i-1, :]
            # Add the corresponding posterior score to the end of that list too
            lp[i] = lp[i-1]
```

(continues on next page)

(continued from previous page)

```
    # Calculate the acceptance rate; this is a useful diagnostic
    accept_rate = acceptances / num_iter
    # Leave off the initial value but return the rest
    return parameters[1:], lp[1:], accept_rate
```

HYMOD Calibration

Let's look at how well HYMOD with some default parameters explain the streamflow data. This example may take a while to converge; HYMOD is sufficiently complex (both computationally and in terms of dynamics) that this "naive" approach to MCMC is relatively slow on a local machine. We will discuss some alternative approaches for this category of models in Section 3 (Diagnostics).

```
# load the Leaf River HYMOD input file
leaf_data = load_hymod_input_file()

# extract the first eleven years of data
leaf_data = leaf_data.iloc[0:4015].copy()

print('Leaf River Data structure:')

# There are only three columns in the file including precipitation, potential␣
↪evapotranspiration, and streamflow
leaf_data.head()
```

```
Leaf River Data structure:
```

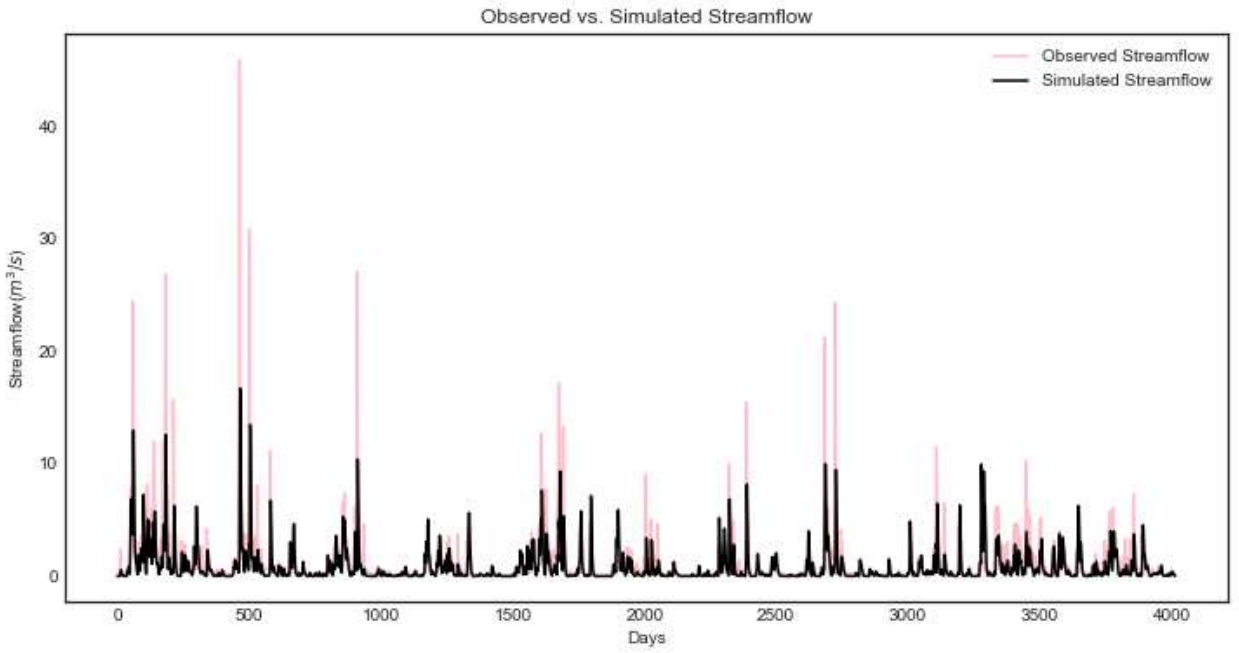
Let's look at how well the model performs with some default parameter values.

```
# assign input parameters to generate a baseline simulated streamflow
Nq = 3  # Number of quickflow routing tanks
Kq = 0.5 # Quickflow routing tanks' rate parameter
Ks =  0.001 # Slowflow routing tank's rate parameter
Alp = 0.5 # Quick/slow split parameter
Huz = 100 # Maximum height of soil moisture accounting tank
B = 1.0 # Scaled distribution function shape parameter

# Note that the number of years is 11. One year of model warm-up and ten years are used␣
↪for actual simulation
model = msdbook.hymod.hymod(Nq, Kq, Ks, Alp, Huz, B, leaf_data, ndays=4015)
ax = msdbook.hymod.plot_observed_vs_simulated_streamflow(df=leaf_data, hymod_dict=model)
```

We can see that this HYMOD parameterization generally does well, but tends to underestimate the peak streamflows. Can we do better?

First, we need to specify a probability model for the data. To do this, we can write the data y_t as the sum of the model output $F(\theta_F; \mathbf{x}_t)$ (where θ_F is the parameter vector and $\mathbf{x}_t$ are the exogenous model forcings) and the residuals $\mathbf{z}_t(\theta_z)$, where θ_z are the statistical parameters used to describe the residual distribution. The residual probability model can be relatively simple, such as the common assumption that $\mathbf{z}_t$ are independently distributed according to a Gaussian distribution, or can be more complex, including auto-correlations, cross-correlations, and/or combinations of systematic *model data-discrepancy* and independent observation errors.

In this example, we will assume that the residuals are normally distributed (on the log scale, since HYMOD predictions and streamflow are non-negative), though in practice we would check this assumption by fitting the model and looking



at residual diagnostics, such as partial autocorrelation and Q-Q plots. Since HYMOD can simulate zero streamflow, which is not in the data, we will also include a strictly positive bias term β. As a result, our probability model is

This means that we need the following model and statistical parameters:

1. `Nq`: the number of quickflow routing tanks;
2. `Kq`: the quickflow routing tanks' rate parameter;
3. `Ks`: The slowflow routing tanks' rate parameter;
4. `Alp`: The quick/slow split parameter;
5. `Huz`: The maximum height of soil moisture accounting tank;
6. `B`: The scaled distribution function scale parameter;
7. `beta`: Positive bias term, since HYMOD can produce zero simulated streamflow;
8. `sigma`: Standard deviation of the log-residual normal distribution.

Prior Distributions

MCMC lets us sample from arbitrary probability distributions, including Bayesian posterior distributions. One advantage of a Bayesian approach to model calibration is that it lets us include prior information for parameter values, which can help guide inferences towards mechanistically reasonable values. In the absence of firm prior information about parameter values, we can check that prior distributions result in reasonable simulations with a *prior predictive check*. Let's start with the following priors, which we assume are independent across parameter.

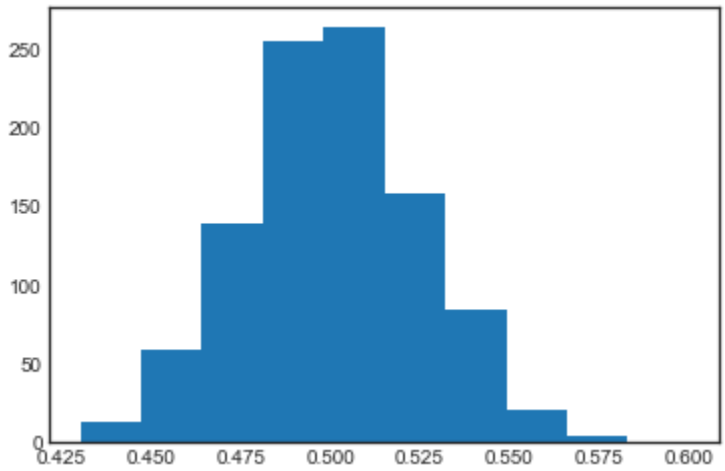
1. `Kq`: LogNormal$(0.25, 0.5)$;
2. `Ks`: LogNormal$(0.95, 0.003)$;
3. `Alp`: Beta$(2, 2)$;

4. `Huz`: $\mathcal{N}(100, 20)$;
5. `B`: LogNormal$(0.1, 1)$;
6. `beta`: LogNormal$(0.05, 0.5)$;
7. `sigma`: LogNormal$(0.5, 0.5)$.

To conduct a prior predictive check, we will generate samples from these distributions, evaluate the model (and add residuals), and then look at the distribution of output (or output summary statistics) about which we have some intuition about what are reasonable values. Note that we will not explicitly compare these results to the data, we do not want to overfit.

```
plt.hist(stats.lognorm(s=0.05, scale=0.5).rvs(1000))
```

```
(array([ 13.,  59., 139., 255., 264., 159.,  85.,  21.,   4.,   1.]),
 array([0.43029764, 0.44725484, 0.46421203, 0.48116923, 0.49812642,
        0.51508362, 0.53204081, 0.54899801, 0.5659552 , 0.5829124 ,
        0.59986959]),
 <BarContainer object of 10 artists>)
```



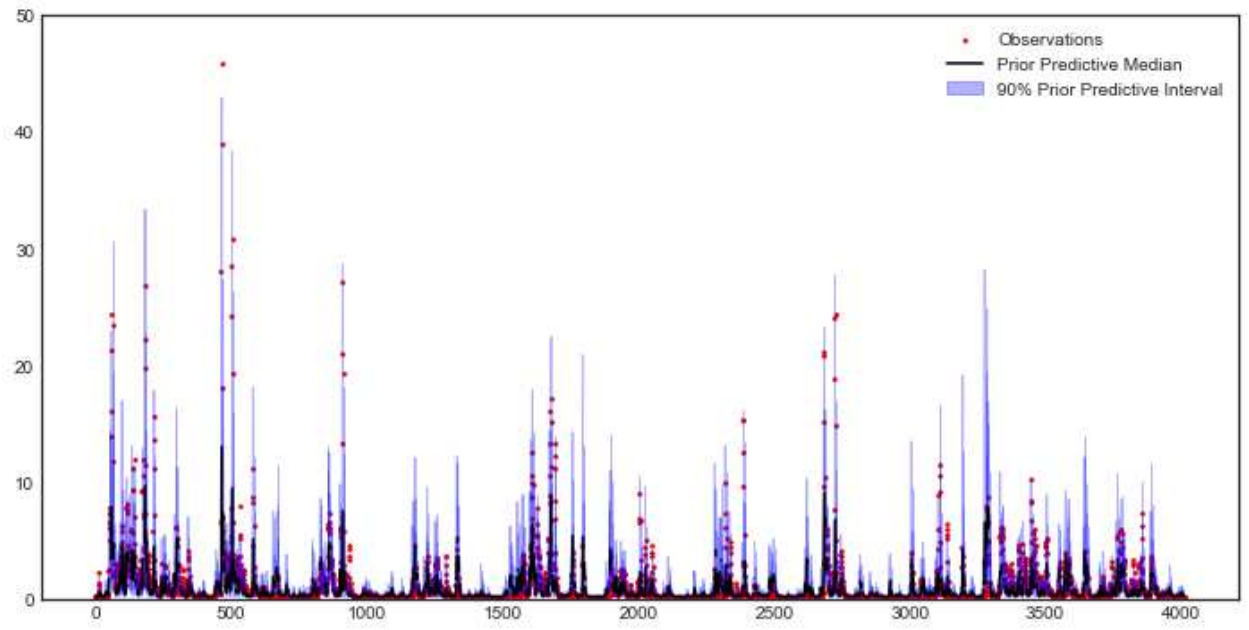

```
ndays = 4015
nsamples = 1000

# generate prior samples
Kq_prior = stats.lognorm(s=0.25, scale=0.5)
Ks_prior = stats.lognorm(s=0.95, scale=0.003)
Alp_prior = stats.beta(2, 2)
Huz_prior = stats.norm(100, 20)
B_prior = stats.lognorm(s=0.1, scale=1)
beta_prior = stats.lognorm(s=0.05, scale=0.25)
sigma_prior = stats.lognorm(s=0.25, scale=0.25)

Kq = Kq_prior.rvs(nsamples)
Ks = Ks_prior.rvs(nsamples)
Alp = Alp_prior.rvs(nsamples)
Huz = Huz_prior.rvs(nsamples)
B = B_prior.rvs(nsamples)
```

(continues on next page)

(continued from previous page)

```
beta = beta_prior.rvs(nsamples)
sigma = sigma_prior.rvs(nsamples)

# preallocate output storage
prior_out = np.zeros((ndays, nsamples))

# note that we include the error/noise in these simulations
for i in range(nsamples):
    prior_out[:, i] = np.exp(np.log(msdbook.hymod.hymod(3, Kq[i], Ks[i], Alp[i], Huz[i],␣
↪B[i], leaf_data, ndays=ndays)['Q'] + beta[i]) + stats.norm(0, sigma[i]).rvs(ndays))
```

```
# compute 90% prediction interval for each time step
prior_q90 = np.quantile(prior_out, [0.05, 0.5, 0.95], axis=1)
fig, strmflw_ax = plt.subplots(figsize=[12,6])
strmflw_ax.set_ylim([0, 50])
strmflw_ax.scatter(range(0, ndays), leaf_data.Strmflw, color="red", s=3)
strmflw_ax.plot(range(0, ndays), prior_q90[1, :], color="black")
strmflw_ax.fill_between(range(0, ndays), prior_q90[0, :], prior_q90[2, :], color="blue",␣
↪alpha=0.3)
strmflw_ax.legend(['Observations', 'Prior Predictive Median', '90% Prior Predictive␣
↪Interval'], loc='upper right')
```



This looks reasonable as a starting point; we may not be capturing the most extreme data in our 90% interval, but we also wouldn't expect to, and as none of our priors are uniform, we are not closing off the possibility that the posteriors could be wider.

Metropolis-Hastings

To implement the Metropolis-Hastings algorithm, we'll start by writing functions to compute the log-posterior of the probability model.

```
def log_prior(Kq, Ks, Alp, Huz, B, beta, sigma):
    lp = 0
    lp += stats.lognorm.logpdf(Kq, s=0.25, scale=0.5)
    lp += stats.lognorm.logpdf(Ks, s=0.95, scale=0.003)
    lp += stats.beta.logpdf(Alp, 2, 2)
    lp += stats.norm.logpdf(Huz, 100, 20)
    lp += stats.lognorm.logpdf(B, s=0.1, scale=1)
    lp += stats.lognorm.logpdf(beta, s=0.05, scale=0.25)
    lp += stats.lognorm.logpdf(sigma, s=0.5, scale=0.25)
    return lp


def log_likelihood(Kq, Ks, Alp, Huz, B, beta, sigma, leaf_data, ndays):
    hymod_out = msdbook.hymod.hymod(3, Kq, Ks, Alp, Huz, B, leaf_data, ndays=ndays)['Q']
    residuals = np.log(leaf_data['Strmflw']) - np.log(hymod_out + beta) # compute␣
↪residuals
    ll = np.sum(stats.norm.logpdf(residuals, scale=sigma))
    return ll


def log_posterior(params, leaf_data=leaf_data, ndays=4015):
    Kq, Ks, Alp, Huz, B, beta, sigma = tuple(params[0])
    lp = log_prior(Kq, Ks, Alp, Huz, B, beta, sigma)
    # only evaluate the model if the log-prior > -Inf
    if not (math.isinf(lp) and lp < 0):
        ll = log_likelihood(Kq, Ks, Alp, Huz, B, beta, sigma, leaf_data, ndays)
        lp += ll
    return lp
```

Next, we'll implement the Metropolis-Hastings algorithm. The number of iterations is set to 100,000, which is needed for convergence. The `metropolis()` function may take a long time to run (75-290 min), to speed this up, reduce the `niter` parameter (ex. `niter = 1000`).

```
niter = 100000

init_params = np.array([[1.0, 0.5, 0.5, 100, 1.0, 0.1, 0.5]])
proposal_sd = [0.005, 0.001, 0.005, 1.0, 0.005, 0.001, 0.005]
```

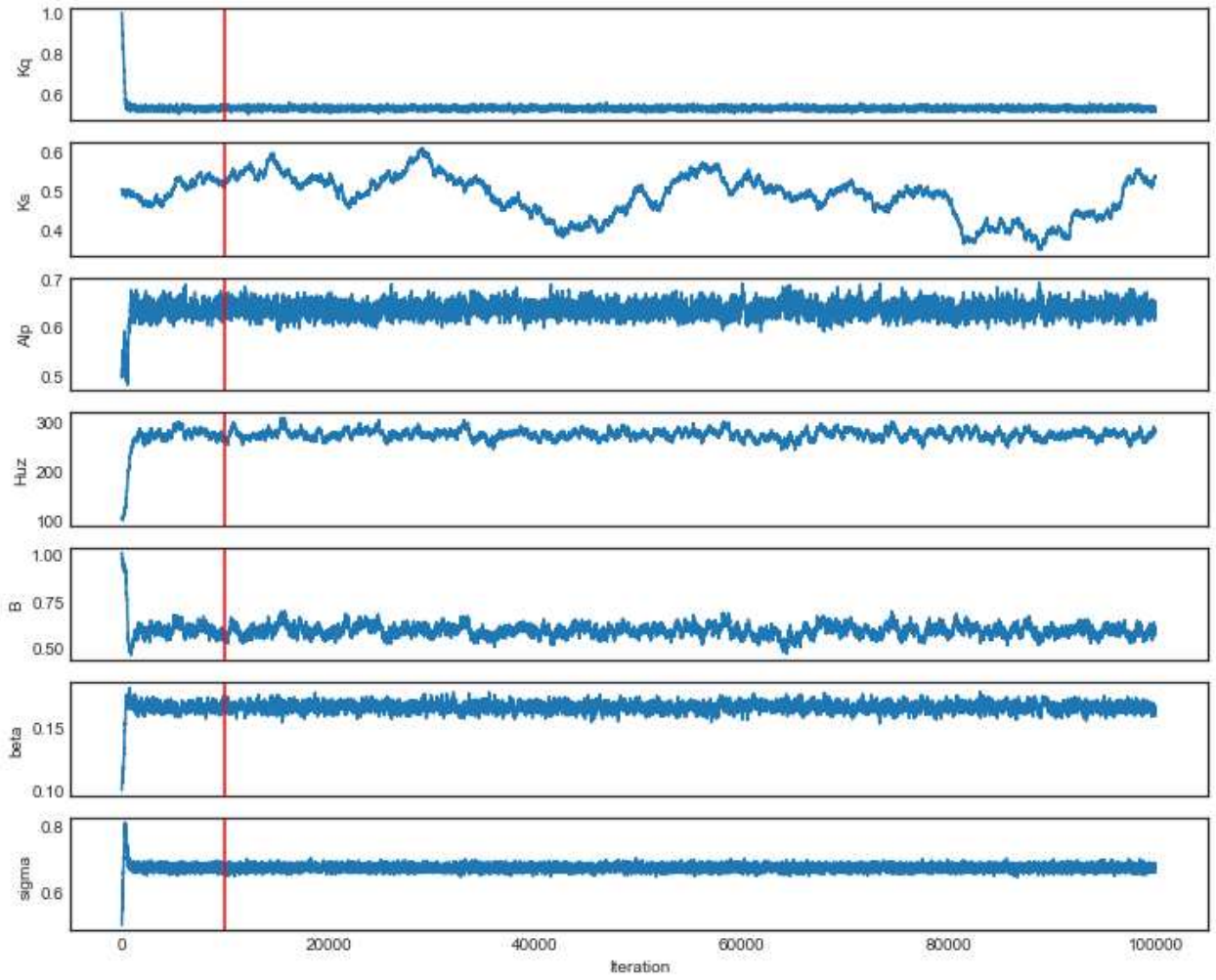
```
out = metropolis(niter, proposal_sd, init_params, log_posterior)
```

What is the acceptance rate? Both too high and too low of an acceptance rate suggest something is off with how our sampler is balancing exploration and exploitation. The theoretical "ideal" is between 24-45%.

```
0.27565
```

To provide some evidence for convergence, let's look at the traceplots. We'll look at a burn-in of 1/10 the number of iterations; this may need to change depending on the number of iterations you run (e.g. if the traceplot after the red vertical line appears to shift versus appearing roughly stationary for the rest of the chain).

```
parnames = ["Kq", "Ks", "Alp", "Huz", "B", "beta", "sigma"]
nburn = int(niter / 10)
fig, axs = plt.subplots(7, 1, sharex=True, figsize=[12, 10])
for i in range(0, 7):
    axs[i].plot(out[0][:, i])
    axs[i].axvline(x=nburn, color="red") # modify x to look at other burnin lengths
    axs[i].set_ylabel(parnames[i])
axs[6].set_xlabel("Iteration");
```

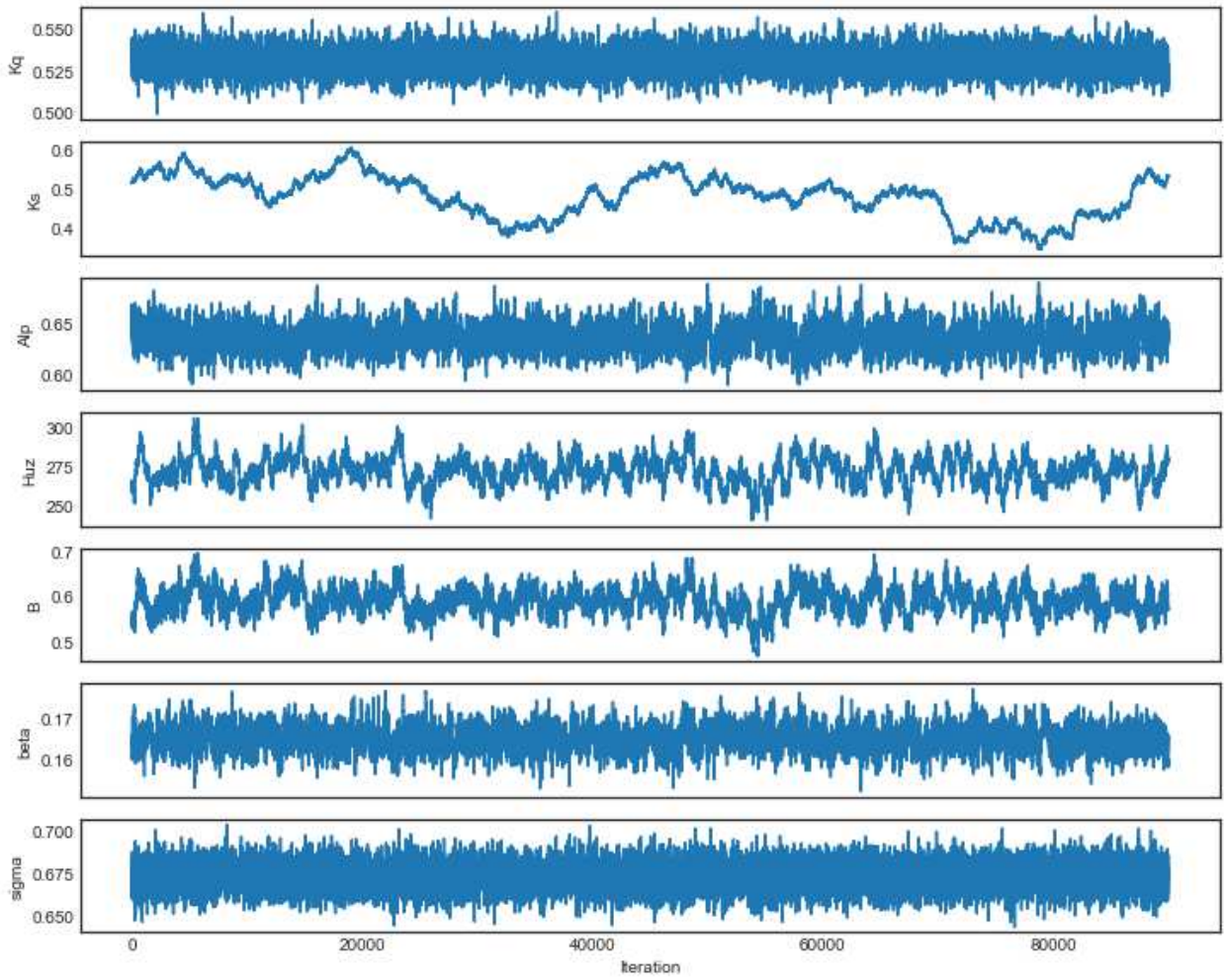


We can see that we might have converged by 10,000 iterations (or possibly earlier). We will discard the samples from before this point as burn-in since they have an unrepresentative probability in the sampled chain.

Let's zoom in on the samples from after this point.

```
fig, axs = plt.subplots(7, 1, sharex=True, figsize=[12, 10])
for i in range(0, 7):
    axs[i].plot(out[0][(nburn+1):niter, i])
    axs[i].set_ylabel(parnames[i])
axs[6].set_xlabel("Iteration");
```

These chains look like a "hairy caterpillar", which is the ideal pattern for the chain to mix well and sample systematically


Kq
0.550
0.525
0.500
Kg
0.6
0.5
0.4
Alp
0.65
0.60
Huz
300
275
250
B
0.7
0.6
0.5
beta
0.17
0.16
sigma
0.700
0.675
0.650
0
20000
40000
60000
80000
Iteration

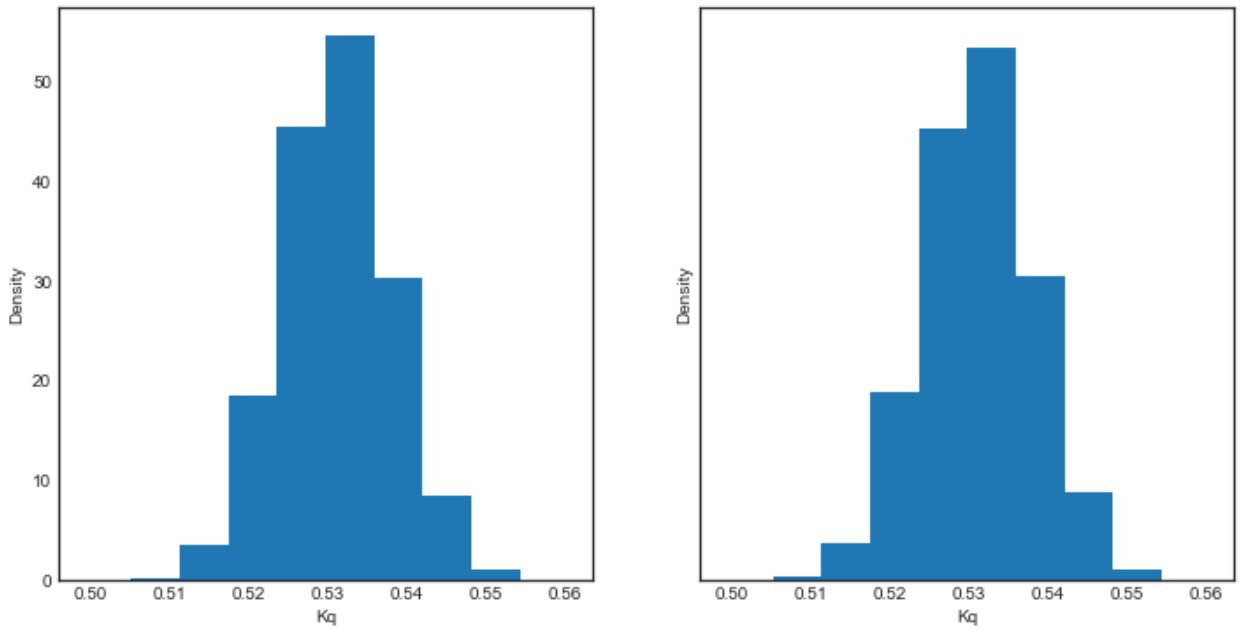
throughout the posterior distribution. If our proposal distribution had been too narrow, we would have accepted many more samples, but the traceplot above would look like a narrow line "dragging" slowly, instead of bouncing around (the chain for K_s looks closest to this type fo behavior). If it had been too wide, we would have rejected many more samples, and the traceplot would have looked more like a city skyline, as the sampler would have gotten stuck at the same value for a long time.

The chains shown above *look* roughly stationary: there is no visual evidence of large shifts in the distribution, such as jumps or changes in the variance. However, the only guarantee that the Markov chain produced by the Metropolis-Hastings algorithm will converge to the target distribution is asymptotic (as the number of iterations $n \to \infty$), and there is no mathematically-guaranteed rate of convergence to guide our decision-making. Instead, we generally want to be skeptical that our chain has converged to the target distribution and to accumulate evidence contradicting our skepticism.

One quick check for convergence is to look at whether the distribution of samples change between the first half of the post-burn-in chain and its entirety. If the second half of the samples do not materially change the distribution, that is evidence for convergence, as it suggests that the later samples are drawn from the same distribution as the earlier ones. On the other hand, if the two distributions differ, the later samples are clearly not drawn from the same distribution as the first samples, and it would be unclear that the chain has converged.

Let's implement this check for K_q as an example. We can see from the figure below that the two histograms look roughly similar, which passes this convergence check.

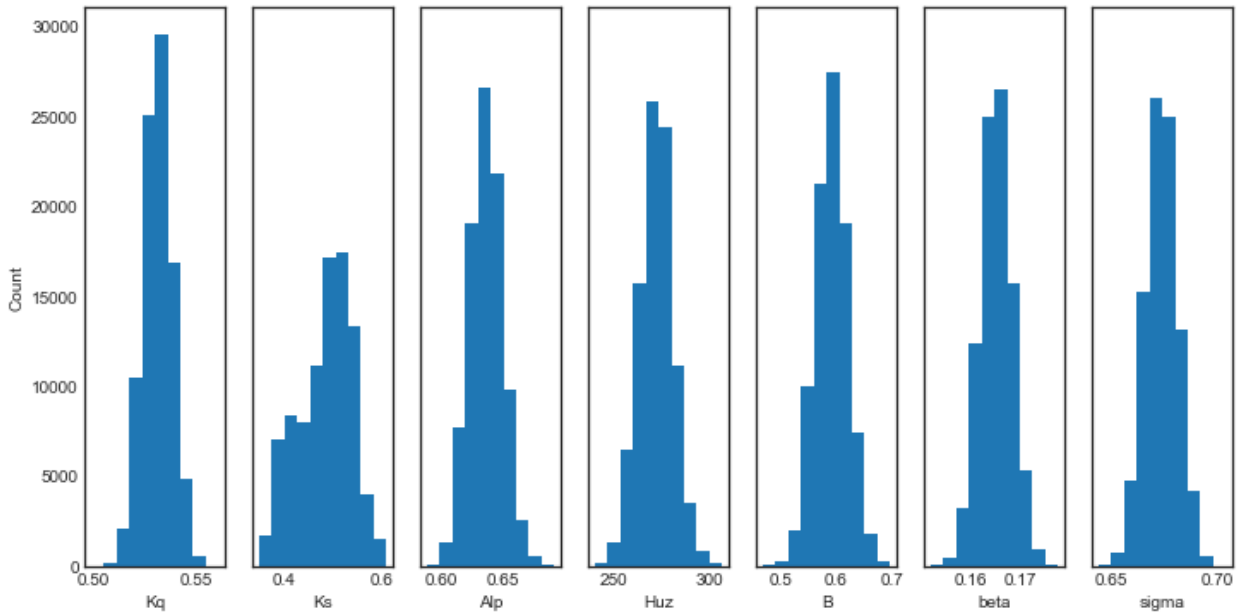
```
fig, axs = plt.subplots(1, 2, figsize=[12,6], sharey=True)
axs[0].hist(out[0][(nburn+1):int(niter/2), i], density=True)
axs[1].hist(out[0][(nburn+1):niter, i], density=True)
axs[0].set_xlabel("Kq");
axs[0].set_ylabel("Density");
axs[1].set_xlabel("Kq");
axs[1].set_ylabel("Density");
```



A more systematic generalization of this convergence check would involve generating multiple chains starting at different initial conditions to check that the chains reach roughly the same distribution, but we will skip that for now.

Let's look at the resulting parameter distributions.

```
fig, axs = plt.subplots(1, 7, figsize=[12,6], sharey=True)
for i in range(0, 7):
    axs[i].hist(out[0][(nburn+1):niter, i])
    axs[i].set_xlabel(parnames[i])
axs[0].set_ylabel("Count");
```

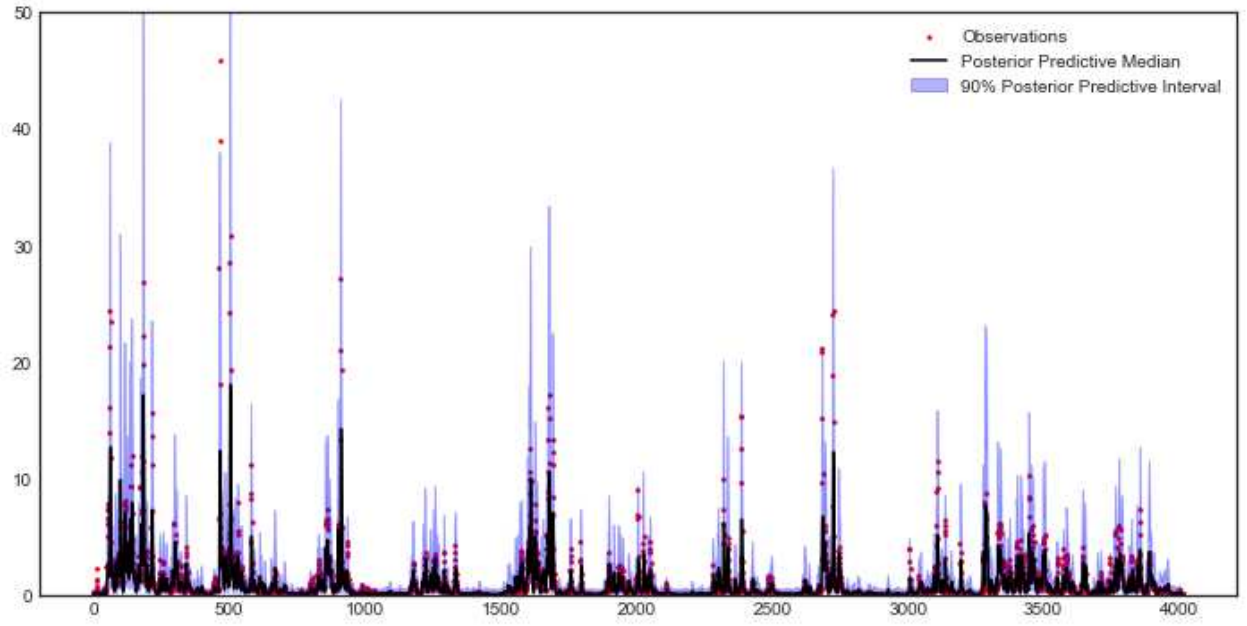


Now, let's simulate from the posterior distribution to see how well we capture the observed streamflow.

```
nsamp = 2000
idx = random.choices(range((nburn+1), niter), k=nsamp)

# simulate
hymod_sim = np.zeros((ndays, nsamp))
for index, i in enumerate(idx):
    hymod_sim[:, index] = np.exp(np.log(msdbook.hymod.hymod(3, out[0][i, 0], out[0][i,␣
↪1], out[0][i, 2], out[0][i, 3], out[0][i, 4], leaf_data, ndays=ndays)['Q'] + out[0][i,␣
↪5]) + stats.norm(0, out[0][i, 6]).rvs(ndays))

# compute quantiles
hymod_q = np.quantile(hymod_sim, [0.05, 0.5, 0.95], axis=1)
```

```
fig, strmflw_ax = plt.subplots(figsize=[12,6])
strmflw_ax.set_ylim([0, 50])
strmflw_ax.scatter(range(0, ndays), leaf_data.Strmflw, color="red", s=3)
strmflw_ax.plot(range(0, ndays), hymod_q[1, :], color="black")
strmflw_ax.fill_between(range(0, ndays), hymod_q[0, :], hymod_q[2, :], color="blue",␣
↪alpha=0.3)
strmflw_ax.legend(['Observations', 'Posterior Predictive Median', '90% Posterior␣
↪Predictive Interval'], loc='upper right');
```

We can visually see that we fail to capture some of the extremes in the 90% projection interval. This is ok; we would expect about 10% of the data to be outside of the interval if the model were well-calibrated. To check, we can compute the *surprise index*, which is the fraction of points outside of the projection interval.




```
si = 1 - (sum([hymod_q[0, i] <= leaf_data.Strmflw[i] <= hymod_q[2, i] for i in range(0,␣
↪ndays)]) / ndays)
si
```

```
0.09464508094645085
```

The surprise index is 9.4%, when we would expect it to be 10%. That's not bad (actually, it's quite good), and means that the model is well calibrated. If we wanted to dial the calibration in further (or if the surprise index were far off, like 20% or 2%), we could change the priors to be more or less restrictive as appropriate. This is somewhat of a judgement call; there is no objectively acceptable threshold for deviation from the target calibration level, but in general, being within a few percentage points is acceptable.

Challenges and Next Steps

Two of the main challenges in implementing MCMC are:

1. The complexity of the model. As MCMC can take hundreds of thousands of model evaluations, small increases in computational expense can be the difference in whether MCMC is feasible or not. Increasing number of un- or weakly-correlated parameters (model or statistical) can also pose problems, as these require more samples to fully explore and capture the distribution. Since the Metropolis-Hastings algorithm in particular is fundamentally serial (the need to burn in every chain means there is only a weak benefit to parallelization), these challenges are to some degree unavoidable without the use of a more sophisticated algorithm.
2. Selection of the proposal distribution. The efficiency of the sampler makes a big difference in the number of needed samples and the *effective sample size* of the resulting chain. This can require a lot of tuning and gets more complex as the number of parameters increases.
3. Specification of the likelihood/probability model. We used a fairly simple model for the HYMOD residuals, but for more complex settings, the residuals may exhibit a high degree of spatial or temporal autocorrelation or may be highly nonstationary. Developing the model and writing down the likelihood function for the error process may be intractable for some classes of models.

The first two challenges can be addressed with more advanced methods than those used here. Adaptive Metropolis-Hastings algorithms (such as those included in the `adaptMCMC` R package or `AdaptiveMCMC` in Julia) automatically tune the proposal distribution based on the acceptance rate. Much more powerful algorithms such as Hamiltonian Monte Carlo (used in the Stan family of packages, `pyMC3` in Python, and `Turing` in Julia) use information about the gradient of the posterior to sample very efficiently, though this often requires the ability to automatically differentiate external simulation models, which may or may not always be possible.

The third challenge is more fundamental (and general) for uncertainty quantifican. When writing down a likelihood function is intractable, Approximate Bayesian Computation (ABC) is a likelihood-free approach which is based on comparing summary statistics, rather than computing the posterior density.

Tips for Using MCMC

In this tutorial, we saw how to implement the Metropolis-Hastings algorithm for HYMOD. In order to use Metropolis-Hastings or other MCMC algorithms to your problem, you will need to answer the following questions:

1. Do you have a probability model for the data-generating process? This could be a statistical model for the data or a model for the discrepancy between a simulation model and the data. We often begin with a relatively simple model (*e.g.* normally-distributed residuals) and add complexity based on whether residual diagnostics suggest that the probability model was appropriate. If you do not or cannot write down an appropriate probability model, you could look at likelihood-free methods such as Approximate Bayesian Computation (ABC).
2. How complex is your inference problem? The more computationally complex your model or the higher the dimensionality of the parameter space, the longer MCMC will need to run to fully sample from the posterior distribution. If your model is too complex, you could begin with initial uncertainty characterization or sensitivity analyses to evaluate the extent to which dimension reduction is possible, and you could look into emulation or surrogate modeling methods. Using Hamiltonian Monte Carlo methods are also an option if your model is amenable to automatic differentiation.
3. How important is parametric uncertainty for your problem? If you're only interested in a point estimate of parameters, you could more directly optimize the posterior density to find the maximum *a posteriori* estimate instead of sampling from the posterior distribution.

If your answers to these questions suggest that MCMC is tractable and useful for your problem, you should feel free to experiment with the HYMOD example, including the number of iterations, the probability model specification, and the proposal distribution. Just be aware that increasing the number of iterations or making the probability model more complex might make the notebook take longer to run.

APPENDIX

C

PLOTTING CODE SAMPLES

C.1 hymod.ipynb

The following are the plotting functions as described in the `hymod.ipynb` Jupyter notebook tutorial.

The following are the necessary package imports to run these functions:

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from matplotlib.lines import Line2D
```

C.1.1 plot_observed_vs_simulated_streamflow()

```
def plot_observed_vs_simulated_streamflow(df, hymod_dict, figsize=[12, 6]):
    """Plot observed versus simulated streamflow.

    :param df:              Dataframe of hymod input data including columns for precip,␣
↪potential evapotranspiration,
                            and streamflow

    :param hymod_dict:      A dictionary of hymod outputs
    :type hymod_dict:       dict

    :param figsize:         Matplotlib figure size
    :type figsize:          list

    """

    # set plot style
    plt.style.use("seaborn-v0_8-white")

    # set up figure
    fig, ax = plt.subplots(figsize=figsize)

    # plot observed streamflow
    ax.plot(range(0, len(df["Strmflw"])), df["Strmflw"], color="pink", label="Observed␣
↪Streamflow")
```

(continues on next page)

(continued from previous page)

```
    # plot simulated streamflow
    ax.plot(range(0, len(df["Strmflw"])), hymod_dict["Q"], color="black", label=
↪"Simulated Streamflow")

    # set axis labels
    ax.set_ylabel("Streamflow($m^3/s$)")
    ax.set_xlabel("Days")
    ax.legend()

    # set plot title
    plt.title("Observed vs. Simulated Streamflow")

    return ax
```

C.1.2 plot_observed_vs_sensitivity_streamflow()

```
def plot_observed_vs_sensitivity_streamflow(df_obs, df_sim, figsize=[10, 4]):
    """Plot observed streamflow versus simulations generated from sensitivity analysis.

    :param df_obs:          Dataframe of mean monthly hymod input data including columns␣
↪for precip,
                            potential evapotranspiration, and streamflow

    :param df_sim:          Dataframe of mean monthly simulation data from sensitivity␣
↪analysis

    :param figsize:         Matplotlib figure size
    :type figsize:          list

    """

    month_list = range(len(df_sim))

    # set up figure
    fig, ax = plt.subplots(figsize=figsize)

    # set labels
    ax.set_xlabel("Days")
    ax.set_ylabel("Flow Discharge (m^3/s)")

    # plots all simulated streamflow cases under different sample sets
    for i in df_sim.columns:
        plt.plot(month_list, df_sim[i], color="pink", alpha=0.2)
    ax.plot([], [], label="Sensitivity Analysis Streamflow", color="pink")

    # plot observed streamflow
    plt.plot(month_list, df_obs["Strmflw"], color="black", label="Observed Streamflow")

    plt.title("Observed vs. Sensitivity Analysis Outputs")
```

(continues on next page)

(continued from previous page)

```
    ax.legend(loc="upper right")

    return ax
```

C.1.3 plot_monthly_heatmap()

```
def plot_monthly_heatmap(arr_sim, df_obs, title='', figsize=[14, 6]):
    """Plot a sensitivity metric overlain by observed flow.

    :param arr_sim:         Numpy array of simulated metrics

    :param df_obs:          Dataframe of mean monthly observed data from sensitivity␣
↪analysis

    :param title:           Title of plot
    :type title:            str

    :param figsize:         Matplotlib figure size
    :type figsize:          list

    """

    # set up figure
    fig, ax = plt.subplots(figsize=figsize)

    # plot heatmap
    sns.heatmap(arr_sim,
                ax=ax,
                yticklabels=['Kq', 'Ks', 'Alp', 'Huz', 'B'],
                cmap=sns.color_palette("ch:s=-.2,r=.6"))

    # setup overlay axis
    ax2 = ax.twinx()

    # plot line
    ax2.plot(np.arange(0.5, 12.5), df_obs['Strmflw'], color='slateblue')

    # plot points on line
    ax2.plot(np.arange(0.5, 12.5), df_obs['Strmflw'], color='slateblue', marker='o')

    # set axis limits and labels
    ax.set_ylim(0, 5)
    ax.set_xlim(0, 12)
    ax.set_xticklabels(['jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep',
↪'oct', 'nov', 'dec'])
    ax2.set_ylabel('Flow Discharge($m^3/s$)')

    plt.title(title)

    plt.show()
```

(continues on next page)

(continued from previous page)

```
    return ax, ax2
```

C.1.4 plot_annual_heatmap()

```
def plot_annual_heatmap(arr_sim, df_obs, title='', figsize=[14,5]):
    """Plot a sensitivity metric overlain by observed flow..

    :param arr_sim:         Numpy array of simulated metrics

    :param df_obs:          Dataframe of mean monthly observed data from sensitivity␣
↪analysis

    :param title:           Title of plot
    :type title:            str

    :param figsize:         Matplotlib figure size
    :type figsize:          list

    """

    # set up figure
    fig, ax = plt.subplots(figsize=figsize)

    # plot heatmap
    sns.heatmap(arr_sim, ax=ax, cmap=sns.color_palette("YlOrBr"))

    # setup overlay axis
    ax2 = ax.twinx()

    # plot line
    ax2.plot(np.arange(0.5, 10.5), df_obs['Strmflw'], color='slateblue')

    # plot points on line
    ax2.plot(np.arange(0.5, 10.5), df_obs['Strmflw'], color='slateblue', marker='o')

    # set up axis lables and limits
    ax.set_ylim(0, 5)
    ax.set_xlim(0, 10)
    ax.set_yticklabels(['Kq', 'Ks', 'Alp', 'Huz', 'B'])
    ax.set_xticklabels(range(2000, 2010))
    ax2.set_ylabel('Flow Discharge($m^3/s$)')

    plt.title(title)

    return ax, ax2
```

C.1.5 plot_varying_heatmap()

```
def plot_varying_heatmap(arr_sim, df_obs, title='', figsize=[14,5]):
    """Plot a sensitivity metric overlain by observed flow..

    :param arr_sim:         Numpy array of simulated metrics

    :param df_obs:          Dataframe of mean monthly observed data from sensitivity␣
↪analysis

    :param title:           Title of plot
    :type title:            str

    :param figsize:         Matplotlib figure size
    :type figsize:          list

    """

    # set up figure
    fig, ax = plt.subplots(figsize=figsize)

    # plot heatmap
    sns.heatmap(arr_sim,
                ax=ax,
                yticklabels=['Kq', 'Ks', 'Alp', 'Huz', 'B'],
                cmap=sns.light_palette("seagreen", as_cmap=True))

    n_years = df_obs.shape[0]

    # setup overlay axis
    ax2 = ax.twinx()

    # plot line
    ax2.plot(range(0, n_years), df_obs['Strmflw'], color='slateblue')

    # plot points on line
    ax2.plot(range(0, n_years), df_obs['Strmflw'], color='slateblue', marker='o')

    # set up axis lables and limits
    ax.set_ylim(0, 5)
    ax.set_xlim(-0.5, 119.5)
    ax2.set_ylabel('Flow Discharge')
    ax.set_xlabel('Number of Months')

    plt.title(title)

    return ax, ax2
```

C.1.6 plot_precalibration_flow()

```
def plot_precalibration_flow(df_sim, df_obs, figsize=[10, 4]):
    """Plot flow discharge provided by the ensemble of parameters sets from Pre-
↪Calibration versus the observed
    flow data.

    :param df_sim:          Dataframe of simulated metrics

    :param df_obs:          Dataframe of mean monthly observed data from sensitivity␣
↪analysis

    :param figsize:         Matplotlib figure size
    :type figsize:          list

    """

    # set up figure
    fig, ax = plt.subplots(figsize=figsize)

    # set axis labels
    ax.set_xlabel('Days')
    ax.set_ylabel('Flow Discharge')

    # plot pre-calibration results
    for i in range(df_sim.shape[1]):
        plt.plot(range(len(df_sim)), df_sim.iloc[:, i],  color="lightgreen", alpha=0.2)

    # plot observed
    plt.plot(range(len(df_sim)), df_obs['Strmflw'],  color="black")

    plt.title('Observed vs. Pre-Calibration Outputs')

    # customize legend
    custom_lines = [Line2D([0], [0],  color="lightgreen", lw=4),
                    Line2D([0], [0], color="black", lw=4)]
    plt.legend(custom_lines, ['Pre-Calibration', 'Observed'])

    return ax
```

C.1.7 plot_precalibration_glue()

```
def plot_precalibration_glue(df_precal, df_glue, df_obs, figsize=[10, 4]):
    """Plot flow discharge provided by the ensemble of parameters sets from Pre-
↪Calibration versus the observed
    flow data.

    :param df_sim:          Dataframe of simulated metrics

    :param df_obs:          Dataframe of mean monthly observed data from sensitivity␣
↪analysis
```

(continues on next page)

(continued from previous page)

```
    :param figsize:         Matplotlib figure size
    :type figsize:          list

    """

    # set up figure
    fig, ax = plt.subplots(figsize=figsize)

    # set axis labels
    ax.set_xlabel('Days')
    ax.set_ylabel('Flow Discharge')

    # plot pre-calibration results
    for i in range(df_precal.shape[1]):
        plt.plot(range(len(df_precal)), df_precal.iloc[:, i],  color="lightgreen",␣
↪alpha=0.2)

    # plot glue
    for i in range(df_glue.shape[1]):
        plt.plot(range(len(df_glue)), df_glue.iloc[:, i], color="lightblue", alpha=0.2)

    # plot observed
    plt.plot(range(len(df_precal)), df_obs['Strmflw'],  color="black")

    plt.title('Observed vs. Sensitivity Analysis Outputs across GLUE/Pre-Calibration')

    # customize legend
    custom_lines = [Line2D([0], [0],  color="lightgreen", lw=4),
                    Line2D([0], [0], color="lightblue", lw=4),
                    Line2D([0], [0], color="black", lw=4)]
    plt.legend(custom_lines, ['Pre-Calibration', 'GLUE', 'Observed'])

    return ax
```

C.2 fishery_dynamics.ipynb

The following are the plotting functions as described in the `fishery_dynamics.ipynb` Jupyter notebook tutorial.

The following are the necessary package imports to run these functions:

```
import numpy as np
import matplotlib.pyplot as plt

from matplotlib import patheffects as pe
```

C.2.1 plot_objective_performance()

```
def plot_objective_performance(objective_performance, profit_solution, robust_solution,␣
↪figsize=(18, 9)):
    """Plot the identified solutions with regards to their objective performance
    in a parallel axis plot

    :param objective_performance:               Objective performance array
    :param profit_solution:                     Profitable solutions array
    :param robust_solution:                     Robust solutions array
    :param figsize:                             Figure size
    :type figsize:                              tuple

    """

    # create the figure object
    fig = plt.figure(figsize=figsize)

    # set up subplot axis object
    ax = fig.add_subplot(1, 1, 1)

    # labels where constraint is always 0
    objs_labels = ['Net present\nvalue (NPV)',
                   'Prey population deficit',
                   'Longest duration\nof low harvest',
                   'Worst harvest instance',
                   'Variance of harvest',
                   'Duration of predator\npopulation collapse']

    # normalization across objectives
    mins = objective_performance.min(axis=0)
    maxs = objective_performance.max(axis=0)
    norm_reference = objective_performance.copy()

    for i in range(5):
        mm = objective_performance[:, i].min()
        mx = objective_performance[:, i].max()
        if mm != mx:
            norm_reference[:, i] = (objective_performance[:, i] - mm) / (mx - mm)
        else:
            norm_reference[:, i] = 1

    # colormap from matplotlib
    cmap = plt.cm.get_cmap("Blues")

    # plot all solutions
    for i in range(len(norm_reference[:, 0])):
        ys = np.append(norm_reference[i, :], 1.0)
        xs = range(len(ys))
        ax.plot(xs, ys, c=cmap(ys[0]), linewidth=2)

    # to highlight robust solutions
    ys = np.append(norm_reference[profit_solution, :], 1.0)  # Most profitable
```

(continues on next page)

(continued from previous page)

```
    xs = range(len(ys))
    l1 = ax.plot(xs[0:6],
                 ys[0:6],
                 c=cmap(ys[0]),
                 linewidth=3,
                 label='Most robust in NPV',
                 path_effects=[pe.Stroke(linewidth=6, foreground='darkgoldenrod'), pe.
↪Normal()])

    ys = np.append(norm_reference[robust_solution, :], 1.0)  # Most robust in all␣
↪criteria
    xs = range(len(ys))
    l2 = ax.plot(xs[0:6],
                 ys[0:6],
                 c=cmap(ys[0]),
                 linewidth=3,
                 label='Most robust across criteria',
                 path_effects=[pe.Stroke(linewidth=6, foreground='gold'), pe.Normal()])

    # build colorbar
    sm = plt.cm.ScalarMappable(cmap=cmap)
    sm.set_array([objective_performance[:, 0].min(), objective_performance[:, 0].max()])
    cbar = fig.colorbar(sm)
    cbar.ax.set_ylabel("\nNet present value (NPV)")

    # tick values
    minvalues = ["{0:.3f}".format(mins[0]),
                 "{0:.3f}".format(-mins[1]),
                 str(-mins[2]),
                 "{0:.3f}".format(-mins[3]),
                 "{0:.2f}".format(-mins[4]),
                 str(0)]

    maxvalues = ["{0:.2f}".format(maxs[0]),
                 "{0:.3f}".format(-maxs[1]),
                 str(-maxs[2]),
                 "{0:.2f}".format(maxs[3]),
                 "{0:.2f}".format(-maxs[4]),
                 str(0)]

    ax.set_ylabel("Preference ->", size=12)
    ax.set_yticks([])
    ax.set_xticks([0, 1, 2, 3, 4, 5])
    ax.set_xticklabels([minvalues[i] + '\n' + objs_labels[i] for i in range(len(objs_
↪labels))])

    # make a twin axis for toplabels
    ax1 = ax.twiny()
    ax1.set_yticks([])
    ax1.set_xticks([0, 1, 2, 3, 4, 5])
    ax1.set_xticklabels([maxvalues[i] for i in range(len(maxs) + 1)])
```

(continues on next page)

(continued from previous page)

```
    return ax, ax1
```

C.2.2 plot_factor_performance()

```
def plot_factor_performance(param_values, collapse_days, b, m, a):
    """Visualize the performance of our policies in three-dimensional
    parametric space.

    :param param_values:                Saltelli sample array
    :param collapse_days:               Simulation array
    :param b:                           b parameter boundary interval
    :param m:                           m parameter boundary interval
    :param a:                           a parameter boundary interval

    """

    # set colormap
    cmap = plt.cm.get_cmap("RdBu_r")

    # build figure object
    fig = plt.figure(figsize=plt.figaspect(0.5), dpi=600, constrained_layout=True)

    # set up scalable colormap
    sm = plt.cm.ScalarMappable(cmap=cmap)

    # set up subplot for profit maximizing policy
    ax1 = fig.add_subplot(1, 2, 1, projection='3d')

    # add point data for profit plot
    sows = ax1.scatter(param_values[:,1],
                       param_values[:,6],
                       param_values[:,0],
                       c=collapse_days[:,0],
                       cmap=cmap,
                       s=0.5)

    # add surface data for boundary separating successful and failed states of the world
    pts_ineq = ax1.plot_surface(b, m, a, color='black', alpha=0.25, zorder=1)

    # add reference point to plot
    pt_ref = ax1.scatter(0.5, 0.7, 0.005, c='black', s=50, zorder=0)

    # set up plot aesthetics and labels
    ax1.set_xlabel("b")
    ax1.set_ylabel("m")
    ax1.set_zlabel("a")
    ax1.set_zlim([0.0, 2.0])
    ax1.set_xlim([0.0, 1.0])
    ax1.set_ylim([0.0, 1.5])
    ax1.xaxis.set_view_interval(0,  0.5)
```

(continues on next page)

(continued from previous page)

```
    ax1.set_facecolor('white')
    ax1.view_init(12, -17)
    ax1.set_title('Profit maximizing policy')

    # set up subplot for robust policy
    ax2 = fig.add_subplot(1, 2, 2, projection='3d')

    # add point data for robust plot
    sows = ax2.scatter(param_values[:,1],
                       param_values[:,6],
                       param_values[:,0],
                       c=collapse_days[:,1],
                       cmap=cmap,
                       s=0.5)

    # add surface data for boundary separating successful and failed states of the world
    pts_ineq = ax2.plot_surface(b, m, a, color='black', alpha=0.25, zorder=1)

    # add reference point to plot
    pt_ref = ax2.scatter(0.5, 0.7, 0.005, c='black', s=50, zorder=0)

    # set up plot aesthetics and labels
    ax2.set_xlabel("b")
    ax2.set_ylabel("m")
    ax2.set_zlabel("a")
    ax2.set_zlim([0.0, 2.0])
    ax2.set_xlim([0.0, 1.0])
    ax2.set_ylim([0.0, 1.5])
    ax2.xaxis.set_view_interval(0, 0.5)
    ax2.set_facecolor('white')
    ax2.view_init(12, -17)
    ax2.set_title('Robust policy')

    # set up colorbar
    sm.set_array([collapse_days.min(), collapse_days.max()])
    cbar = fig.colorbar(sm)
    cbar.set_label('Days with predator collapse')

    return ax1, ax2
```

GLOSSARY

Design of experiment: Provides a framework for the extraction of all plausible information about the impact of each factor on the output of the numerical model

Exploratory modeling: Use of large ensembles of uncertain conditions to discover decision-relevant combinations of uncertain factors

Factor: Any model component that can affect model outputs: inputs, resolution levels, coupling relationships, model relationships and parameters. In models with acceptable model fidelity these factors may represent elements of the real-world system under study.

Factor mapping: A technique to identify which uncertain model factors lead to certain model behavior

Factor prioritization: A technique to identify the uncertain factors which, when fixed to their true value, would lead to the greatest reduction in output variability

Factor screening: A technique to identify model components that have a negligible effect or make no significant contributions to the variability of the outputs or metrics of interest

First-, second-, total-order effects: First-order effects indicate the percent of model output variance contributed by a factor individually. Second-order effects capture how interactions between a pair of parameter input variables can lead to change in model output. Total-order effects consider all the effects a factor has, individually and in interaction with other factors.

Hindcasting: A type of predictive check that uses the model to estimate output for past events to see how well the output matches the known results.

Pre-calibration: A hybrid uncertainty assessment method that involves identifying a plausible set of parameters using some prespecified screening criterion, such as the distance from the model results to the observations.

Prior: The best assessment of the probability of an event based on existing knowledge before a new experiment is conducted

Posterior: The revised or updated probability of an event after taking into account new information

Probabilistic inversion: Uses additional information, for instance, a probabilistic expert assessment or survey result, to update an existing prior distribution

Return level: A value that is expected to be equaled or exceeded on average once every interval of time (T) (with a probability of 1/T)

Return period: The estimated time interval between events of a similar size or intensity/

Sampling: The process of selecting model parameters or inputs that characterize the model uncertainty space.

Scenario discovery: Use of large ensembles of uncertain conditions to discover decision-relevant combinations of uncertain factors

Sensitivity analysis: Conducted to understand the factors and processes that most (or least) control a model's outputs

> *Local sensitivity analysis*: Model evaluation performed by varying uncertain factors around specific reference values
>
> *Global sensitivity analysis*: Model evaluation performed by varying uncertain factors throughout their entire feasible value space

Uncertainty

Deep uncertainty: Refers to situations where expert opinions consulted on a decision do not know or cannot agree on system boundaries, the outcomes of interest and their relative importance, or the prior probability distribution for the various uncertain factors present

Epistemic uncertainty: Systematic uncertainty that comes about due to the lack of knowledge or data to choose the best model

Ontological uncertainty: Uncertainties due to processes, interactions, or futures, that are not contained within current conceptual models

Aleatory uncertainty: Uncertainty due to natural randomness in processes

Uncertainty characterization: Model evaluation under alternative factor hypotheses to explore their implications for model output uncertainty

Uncertainty quantification: Representation of model output uncertainty using probability distributions

Variance decomposition: A technique to partition how much of the variability in a model's output is due to different explanatory variables.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] G. E. P. Box. Science and Statistics. *Journal of the American Statistical Association*, 71(356):791–799, 1976. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1976.10480949, doi:10.1080/01621459.1976.10480949.

[2] National Research Council and others. *Convergence: Facilitating transdisciplinary integration of life sciences, physical sciences, engineering, and beyond*. National Academies Press, 2014.

[3] Sondoss Elsawah, Tatiana Filatova, Anthony J Jakeman, Albert J Kettner, Moira L Zellner, Ioannis N Athanasiadis, Serena H Hamilton, Robert L Axtell, Daniel G Brown, Jonathan M Gilligan, and others. Eight grand challenges in socio-environmental systems modeling. *Socio-Environmental Systems Modelling*, 2:16226–16226, 2020.

[4] Yacov Y Haimes. Risk modeling of interdependent complex systems of systems: theory and practice. *Risk analysis*, 38(1):84–98, 2018.

[5] Dirk Helbing. Globally networked risks and how to respond. *Nature*, 497(7447):51–59, 2013.

[6] Andrea Saltelli, Ksenia Aleksankina, William Becker, Pamela Fennell, Federico Ferretti, Niels Holst, Sushan Li, and Qiongli Wu. Why so many published sensitivity analyses are false: a systematic review of sensitivity analysis practices. *Environmental modelling & software*, 114:29–39, 2019.

[7] Daniel Wirtz and Wolfgang Nowak. The rocky road to extended simulation frameworks covering uncertainty, inversion, optimization and control. *Environmental Modelling & Software*, 93:180–192, 2017.

[8] Roger Cooke and others. *Experts in uncertainty: opinion and subjective probability in science*. Oxford University Press on Demand, 1991.

[9] Enayat A Moallemi, Jan Kwakkel, Fjalar J de Haan, and Brett A Bryan. Exploratory modeling for analyzing coupled human-natural systems under uncertainty. *Global Environmental Change*, 65:102186, 2020.

[10] Warren E Walker, Poul Harremoës, Jan Rotmans, Jeroen P Van Der Sluijs, Marjolein BA Van Asselt, Peter Janssen, and Martin P Krayer von Krauss. Defining uncertainty: a conceptual basis for uncertainty management in model-based decision support. *Integrated assessment*, 4(1):5–17, 2003.

[11] Andrea Saltelli, Philip B Stark, William Becker, and Pawel Stano. Climate models as economic guides scientific challenge or quixotic quest? *Issues in Science and Technology*, 31(3):79–84, 2015.

[12] Hoshin V. Gupta, Thorsten Wagener, and Yuqiong Liu. Reconciling theory with observations: elements of a diagnostic approach to model evaluation. *Hydrological Processes: An International Journal*, 22(18):3802–3813, 2008. Publisher: Wiley Online Library.

[13] Antonia Hadjimichael, Julianne Quinn, and Patrick Reed. Advancing Diagnostic Model Evaluation to Better Understand Water Shortage Mechanisms in Institutionally Complex River Basins. *Water Resources Research*, 56(10):e2020WR028079, 2020. URL: http://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2020WR028079 (visited on 2020-10-16), doi:10.1029/2020WR028079.

[14] Andrea Saltelli, Marco Ratto, Terry Andres, Francesca Campolongo, Jessica Cariboni, Debora Gatelli, Michaela Saisana, and Stefano Tarantola. *Global Sensitivity Analysis: The Primer*. Wiley-Interscience, Chichester, England ; Hoboken, NJ, 1 edition edition, February 2008. ISBN 978-0-470-05997-5.

[15] Keith Beven. Towards a coherent philosophy for modelling the environment. *Proceedings of the royal society of London. Series A: mathematical, physical and engineering sciences*, 458(2026):2465–2484, 2002.

[16] Naomi Oreskes, Kristin Shrader-Frechette, and Kenneth Belitz. Verification, Validation, and Confirmation of Numerical Models in the Earth Sciences. *Science*, 263(5147):641–646, February 1994. URL: https://science.sciencemag.org/content/263/5147/641 (visited on 2020-04-15), doi:10.1126/science.263.5147.641.

[17] Keith Beven. Prophecy, reality and uncertainty in distributed hydrological modelling. *Advances in water resources*, 16(1):41–51, 1993.

[18] Keith Beven and Andrew Binley. The future of distributed models: Model calibration and uncertainty prediction. *Hydrological Processes*, 6(3):279–298, 1992. doi:10.1002/hyp.3360060305.

[19] Yaman Barlas and Stanley Carpenter. Philosophical roots of model validation: Two paradigms. *System Dynamics Review*, 6(2):148–166, 1990. doi:10.1002/sdr.4260060203.

[20] Stephen Toulmin. From form to function: philosophy and history of science in the 1950s and now. *Daedalus*, pages 143–162, 1977. Publisher: JSTOR.

[21] George B. Kleindorfer, Liam O'Neill, and Ram Ganeshan. Validation in simulation: Various positions in the philosophy of science. *Management Science*, 44(8):1087–1099, 1998. Publisher: INFORMS.

[22] Sibel Eker, Elena Rovenskaya, Michael Obersteiner, and Simon Langan. Practice and perspectives in the validation of resource management models. *Nature communications*, 9(1):1–10, 2018.

[23] Yaman Barlas. Formal aspects of model validity and validation in system dynamics. *System Dynamics Review: The Journal of the System Dynamics Society*, 12(3):183–210, 1996. Publisher: Wiley Online Library.

[24] Thomas H. Naylor and Joseph Michael Finger. Verification of computer simulation models. *Management science*, 14(2):B–92, 1967. Publisher: INFORMS.

[25] Keith J Beven. On hypothesis testing in hydrology: why falsification of models is still a really good idea. *Wiley Interdisciplinary Reviews: Water*, 5(3):e1278, 2018.

[26] Hoshin V Gupta, Martyn P Clark, Jasper A Vrugt, Gab Abramowitz, and Ming Ye. Towards a comprehensive assessment of model structural adequacy. *Water Resources Research*, 2012.

[27] Praveen Kumar. Typology of hydrologic predictability. *Water Resources Research*, 2011. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2010WR009769 (visited on 2020-04-15), doi:10.1029/2010WR009769.

[28] Grey S. Nearing, Benjamin L. Ruddell, Andrew R. Bennett, Cristina Prieto, and Hoshin V. Gupta. Does Information Theory Provide a New Paradigm for Earth Science? Hypothesis Testing. *Water Resources Research*, 56(2):e2019WR024918, 2020. doi:10.1029/2019WR024918.

[29] Hoshin Vijai Gupta, Soroosh Sorooshian, and Patrice Ogou Yapo. Toward improved calibration of hydrologic models: Multiple and noncommensurable measures of information. *Water Resources Research*, 34(4):751–763, 1998. URL: http://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/97WR03495 (visited on 2020-04-07), doi:10.1029/97WR03495.

[30] Francesca Pianosi and Thorsten Wagener. Understanding the time-varying importance of different uncertainty sources in hydrological modelling using global sensitivity analysis. *Hydrological Processes*, pages 3991–4003, November 2017. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/hyp.10968%4010.1111/%28ISSN%291099-1085.Kieth-Beven, doi:10.1002/hyp.10968@10.1111/(ISSN)1099-1085.Kieth-Beven.

[31] Charles Rougé, Patrick M. Reed, Danielle S. Grogan, Shan Zuidema, Alexander Prusevich, Stanley Glidden, Jonathan R. Lamontagne, and Richard B. Lammers. Coordination and Control: Limits in Standard Representations of Multi-Reservoir Operations in Hydrological Modeling. *Hydrology and Earth System Sciences Dis-*

cussions, pages 1–37, November 2019. URL: https://www.hydrol-earth-syst-sci-discuss.net/hess-2019-589/, doi:https://doi.org/10.5194/hess-2019-589.

[32] David W. Cash, William C. Clark, Frank Alcock, Nancy M. Dickson, Noelle Eckley, David H. Guston, Jill Jäger, and Ronald B. Mitchell. Knowledge systems for sustainable development. *Proceedings of the national academy of sciences*, 100(14):8086–8091, 2003. Publisher: National Acad Sciences.

[33] Dave D. White, Amber Wutich, Kelli L. Larson, Patricia Gober, Timothy Lant, and Clea Senneville. Credibility, salience, and legitimacy of boundary objects: water managers' assessment of a simulation model in an immersive decision theater. *Science and Public Policy*, 37(3):219–232, April 2010. Publisher: Oxford Academic. URL: https://academic.oup.com/spp/article/37/3/219/1626552 (visited on 2020-05-12), doi:10.3152/030234210X497726.

[34] Andrea Saltelli and Silvio Funtowicz. When all models are wrong. *Issues in Science and Technology*, 30(2):79–85, 2014. Publisher: JSTOR.

[35] Thorsten Wagener and Francesca Pianosi. 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, July 2019. URL: https://www.sciencedirect.com/science/article/pii/S0012825218300990 (visited on 2021-08-30), doi:10.1016/j.earscirev.2019.04.006.

[36] Steve Bankes. Exploratory Modeling for Policy Analysis. *Operations Research*, 41(3):435–449, June 1993. URL: https://pubsonline.informs.org/doi/abs/10.1287/opre.41.3.435 (visited on 2018-09-11), doi:10.1287/opre.41.3.435.

[37] Christopher P. Weaver, Robert J. Lempert, Casey Brown, John A. Hall, David Revell, and Daniel Sarewitz. Improving the contribution of climate model information to decision making: the value and demands of robust decision frameworks. *Wiley Interdisciplinary Reviews: Climate Change*, 4(1):39–60, 2013. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/wcc.202 (visited on 2019-10-01), doi:10.1002/wcc.202.

[38] Andrea Saltelli, Stefano Tarantola, Francesca Campolongo, and Marco Ratto. *Sensitivity analysis in practice: a guide to assessing scientific models*. Volume 1. Wiley Online Library, 2004.

[39] Emanuele Borgonovo and Elmar Plischke. Sensitivity analysis: a review of recent advances. *European Journal of Operational Research*, 248(3):869–887, 2016.

[40] O Rakovec, Mary C Hill, MP Clark, AH Weerts, AJ Teuling, and R Uijlenhoet. Distributed evaluation of local sensitivity analysis (delsa), with application to hydrologic models. *Water Resources Research*, 50(1):409–426, 2014.

[41] Andrea Saltelli and Paola Annoni. How to avoid a perfunctory sensitivity analysis. *Environmental Modelling & Software*, 25(12):1508–1517, 2010.

[42] Yong Tang, Patrick Reed, Thibaut Wagener, and K van Werkhoven. Comparing sensitivity analysis methods to advance lumped watershed model identification and evaluation. *Hydrology and Earth System Sciences*, 11(2):793–817, 2007.

[43] Nicholas AS Hamm, Jim W Hall, and MG Anderson. Variance-based sensitivity analysis of the probability of hydrologically induced slope instability. *Computers & geosciences*, 32(6):803–817, 2006.

[44] Andrea Saltelli, Ksenia Aleksankina, William Becker, Pamela Fennell, Federico Ferretti, Niels Holst, Sushan Li, and Qiongli Wu. Why so many published sensitivity analyses are false: a systematic review of sensitivity analysis practices. *Environmental modelling & software*, 114:29–39, 2019.

[45] Benjamin P Bryant and Robert J Lempert. Thinking inside the box: a participatory, computer-assisted approach to scenario discovery. *Technological Forecasting and Social Change*, 77(1):34–49, 2010.

[46] Andrea Saltelli and Stefano Tarantola. On the relative importance of input factors in mathematical models: safety assessment for nuclear waste disposal. *Journal of the American Statistical Association*, 97(459):702–709, 2002.

[47] Barry Anderson, Emanuele Borgonovo, Marzio Galeotti, and Roberto Roson. Uncertainty in climate change modeling: can global sensitivity analysis be of help? *Risk analysis*, 34(2):271–293, 2014.

[48] Emanuele Borgonovo. Sensitivity analysis with finite changes: an application to modified eoq models. *European Journal of Operational Research*, 200(1):127–138, 2010.

[49] Andrea Saltelli, Marco Ratto, Terry Andres, Francesca Campolongo, Jessica Cariboni, Debora Gatelli, Michaela Saisana, and Stefano Tarantola. *Global sensitivity analysis: the primer*. John Wiley & Sons, 2008.

[50] Neil R Edwards, David Cameron, and Jonathan Rougier. Precalibrating an intermediate complexity climate model. *Climate dynamics*, 37(7):1469–1482, 2011.

[51] Francesca Pianosi, Keith Beven, Jim Freer, Jim W Hall, Jonathan Rougier, David B Stephenson, and Thorsten Wagener. Sensitivity analysis of environmental models: a systematic review with practical workflow. *Environmental Modelling & Software*, 79:214–232, 2016.

[52] RC Spear and GM Hornberger. Eutrophication in peel inlet—ii. identification of critical uncertainties via generalized sensitivity analysis. *Water research*, 14(1):43–49, 1980.

[53] Jon C Helton, Jay Dean Johnson, Cedric J Sallaberry, and Curt B Storlie. Survey of sampling-based methods for uncertainty and sensitivity analysis. *Reliability Engineering & System Safety*, 91(10-11):1175–1209, 2006.

[54] Ronald Aylmer Fisher. Design of experiments. *Br Med J*, 1(3923):554–554, 1936.

[55] JD Herman, PM Reed, and T Wagener. Time-varying sensitivity analysis clarifies the effects of watershed model formulation on model behavior. *Water Resources Research*, 49(3):1400–1414, 2013.

[56] Carolina Massmann, Thorsten Wagener, and Hubert Holzmann. A new approach to visualizing time-varying sensitivity indices for environmental model diagnostics across evaluation time-scales. *Environmental modelling & software*, 51:190–194, 2014.

[57] An Van Schepdael, Aurélie Carlier, and Liesbet Geris. Sensitivity analysis by design of experiments. In *Uncertainty in Biology*, pages 327–366. Springer, 2016.

[58] Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.

[59] John Norton. An introduction to sensitivity assessment of simulation models. *Environmental Modelling & Software*, 69:166–174, 2015.

[60] Douglas C Montgomery. *Design and analysis of experiments*. John wiley & sons, 2017.

[61] George EP Box and J Stuart Hunter. The 2 k—p fractional factorial designs. *Technometrics*, 3(3):311–351, 1961.

[62] Izabella Surowiec, Ludvig Vikstrom, Gustaf Hector, Erik Johansson, Conny Vikstrom, and Johan Trygg. Generalized subset designs in analytical chemistry. *Analytical chemistry*, 89(12):6491–6497, 2017.

[63] Michael D McKay, Richard J Beckman, and William J Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(1):239–2451, 1979.

[64] Boxin Tang. Orthogonal array-based latin hypercubes. *Journal of the American statistical association*, 88(424):1392–1397, 1993.

[65] Ishaan L Dalal, Deian Stefan, and Jared Harwayne-Gidansky. Low discrepancy sequences for monte carlo simulations on reconfigurable platforms. In *2008 International Conference on Application-Specific Systems, Architectures and Processors*, 108–113. IEEE, 2008.

[66] SK Zaremba. The mathematical basis of monte carlo and quasi-monte carlo methods. *SIAM review*, 10(3):303–314, 1968.

[67] Sergei Kucherenko, Daniel Albrecht, and Andrea Saltelli. Exploring multi-dimensional spaces: a comparison of latin hypercube and quasi monte carlo sampling techniques. *arXiv preprint arXiv:1505.02350*, 2015.

[68] Bertrand Iooss, Loïc Boussouf, Vincent Feuillard, and Amandine Marrel. Numerical studies of the metamodel fitting and validation processes. *arXiv preprint arXiv:1001.1049*, 2010.

[69] Ruichen Jin, Wei Chen, and Agus Sudjianto. An efficient algorithm for constructing optimal design of computer experiments. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 37009, 545–554. 2003.

[70] Max D Morris and Toby J Mitchell. Exploratory designs for computational experiments. *Journal of statistical planning and inference*, 43(3):381–402, 1995.

[71] Jeong-Soo Park. Optimal latin-hypercube designs for computer experiments. *Journal of statistical planning and inference*, 39(1):95–111, 1994.

[72] Ilya M Sobol. Uniformly distributed sequences with an additional uniform property. *USSR Computational Mathematics and Mathematical Physics*, 16(5):236–242, 1976.

[73] Il'ya Meerovich Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 7(4):784–802, 1967.

[74] Max D Morris. Factorial sampling plans for preliminary computational experiments. *Technometrics*, 33(2):161–174, 1991.

[75] RI Cukier, CM Fortuin, Kurt E Shuler, AG Petschek, and JH Schaibly. Study of the sensitivity of coupled reaction systems to uncertainties in rate coefficients. i theory. *The Journal of chemical physics*, 59(8):3873–3878, 1973.

[76] Andrea Saltelli, Stefano Tarantola, and KP-S Chan. A quantitative model-independent method for global sensitivity analysis of model output. *Technometrics*, 41(1):39–56, 1999.

[77] Ilya M Sobol. Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates. *Mathematics and computers in simulation*, 55(1-3):271–280, 2001.

[78] Jonathan D Herman, Harrison B Zeff, Jonathan R Lamontagne, Patrick M Reed, and Gregory W Characklis. Synthetic drought scenario generation to support bottom-up water supply vulnerability assessments. *Journal of Water Resources Planning and Management*, 142(11):04016050, 2016.

[79] PCD Milly, Julio Betancourt, Malin Falkenmark, Robert M Hirsch, Zbigniew W Kundzewicz, Dennis P Lettenmaier, and Ronald J Stouffer. Stationarity is dead: whither water management? *Earth*, 4:20, 2008.

[80] Edoardo Borgomeo, Christopher L Farmer, and Jim W Hall. Numerical rivers: a synthetic streamflow generator for water resources vulnerability assessments. *Water Resources Research*, 51(7):5382–5405, 2015.

[81] Manuel Herrera, Sukumar Natarajan, David A Coley, Tristan Kershaw, Alfonso P Ramallo-González, Matthew Eames, Daniel Fosas, and Michael Wood. A review of current and future weather data for building simulation. *Building Services Engineering Research and Technology*, 38(5):602–627, 2017.

[82] Daniel S Wilks and Robert L Wilby. The weather generation game: a review of stochastic weather models. *Progress in physical geography*, 23(3):329–357, 1999.

[83] JR Lamontagne and JR Stedinger. Generating synthetic streamflow forecasts with specified precision. *Journal of Water Resources Planning and Management*, 144(4):04018007, 2018.

[84] Sanghamitra Medda and Kalyan Kumar Bhar. Comparison of single-site and multi-site stochastic models for streamflow generation. *Applied Water Science*, 9(3):67, 2019.

[85] Brian R Kirsch, Gregory W Characklis, and Harrison B Zeff. Evaluating the impact of alternative hydro-climate scenarios on transfer agreements: practical improvement for generating synthetic streamflows. *Journal of Water Resources Planning and Management*, 139(4):396–406, 2013.

[86] Daniel P Loucks and Eelco Van Beek. *Water resource systems planning and management: An introduction to methods, models, and applications*. Springer, 2017.

[87] Scott Steinschneider, Sungwook Wi, and Casey Brown. The integrated effects of climate and hydrologic uncertainty on future flood risk assessments. *Hydrological Processes*, 29(12):2823–2839, 2015.

[88] Richard M Vogel. Stochastic watershed models for hydrologic risk management. *Water Security*, 1:28–35, 2017.

[89] Richard M Vogel and Jery R Stedinger. The value of stochastic streamflow models in overyear reservoir design applications. *Water Resources Research*, 24(9):1483–1490, 1988.

[90] Emanuele Borgonovo. Sensitivity analysis of model output with input constraints: a generalized rationale for local methods. *Risk Analysis: An International Journal*, 28(3):667–680, 2008.

[91] Bertrand Iooss and Paul Lemaître. A review on global sensitivity analysis methods. In *Uncertainty management in simulation-optimization of complex systems*, pages 101–122. Springer, 2015.

[92] Francesca Campolongo and Roger Braddock. The use of graph theory in the sensitivity analysis of the model output: a second order screening method. *Reliability Engineering & System Safety*, 64(1):1–12, 1999.

[93] Roger A Cropp and Roger D Braddock. The new morris method: an efficient second-order screening method. *Reliability Engineering & System Safety*, 78(1):77–83, 2002.

[94] Jon C Helton. Uncertainty and sensitivity analysis techniques for use in performance assessment for radioactive waste disposal. *Reliability Engineering & System Safety*, 42(2-3):327–367, 1993.

[95] Gemma Manache and Charles S Melching. Identification of reliable regression-and correlation-based sensitivity measures for importance ranking of water-quality model parameters. *Environmental Modelling & Software*, 23(5):549–562, 2008.

[96] F Pappenberger and Keith J Beven. Ignorance is bliss: or seven reasons not to use uncertainty analysis. *Water resources research*, 2006.

[97] Jerome H Friedman and Nicholas I Fisher. Bump hunting in high-dimensional data. *Statistics and Computing*, 9(2):123–143, 1999.

[98] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[99] Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[100] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[101] George M Hornberger and Robert C Spear. Approach to the preliminary analysis of environmental systems. *J. Environ. Mgmt.*, 12(1):7–18, 1981.

[102] Robert J Lempert, David G Groves, Steven W Popper, and Steve C Bankes. A general, analytic method for generating robust strategies and narrative scenarios. *Management science*, 52(4):514–528, 2006.

[103] David G Groves and Robert J Lempert. A new analytic method for finding policy-relevant scenarios. *Global Environmental Change*, 17(1):73–85, 2007.

[104] Keith Beven and Andrew Binley. Glue: 20 years on. *Hydrological processes*, 28(24):5897–5918, 2014.

[105] Roberta-Serena Blasone, Jasper A Vrugt, Henrik Madsen, Dan Rosbjerg, Bruce A Robinson, and George A Zyvoloski. Generalized likelihood uncertainty estimation (glue) using adaptive markov chain monte carlo sampling. *Advances in Water Resources*, 31(4):630–648, 2008.

[106] SA Cryer and PL Havens. Regional sensitivity analysis using a fractional factorial method for the usda model gleams. *Environmental modelling & software*, 14(6):613–624, 1999.

[107] Pengfei Wei, Zhenzhou Lu, and Xiukai Yuan. Monte carlo simulation for moment-independent sensitivity analysis. *Reliability Engineering & System Safety*, 110:60–67, 2013.

[108] Peter Young. Data-based mechanistic modelling, generalised sensitivity and dominant mode analysis. *Computer Physics Communications*, 117(1-2):113–129, 1999.

[109] Andrea Saltelli. Making best use of model evaluations to compute sensitivity indices. *Computer physics communications*, 145(2):280–297, 2002.

[110] Andrea Saltelli. Sensitivity analysis for importance assessment. *Risk analysis*, 22(3):579–590, 2002.

[111] Toshimitsu Homma and Andrea Saltelli. Importance measures in global sensitivity analysis of nonlinear models. *Reliability Engineering & System Safety*, 52(1):1–17, 1996.

[112] Gregory J McRae, William R Goodin, and John H Seinfeld. Development of a second-generation mathematical model for urban air pollution—i. model formulation. *Atmospheric Environment (1967)*, 16(4):679–696, 1982.

[113] Andrea Saltelli and Ricardo Bolado. An alternative way to compute fourier amplitude sensitivity test (fast). *Computational Statistics & Data Analysis*, 26(4):445–460, 1998.

[114] MA Vazquez-Cruz, R Guzman-Cruz, IL Lopez-Cruz, O Cornejo-Perez, I Torres-Pacheco, and RG Guevara-Gonzalez. Global sensitivity analysis by means of efast and sobol'methods and calibration of reduced state-variable tomgro model using genetic algorithms. *Computers and Electronics in Agriculture*, 100:1–12, 2014.

[115] Benjamin Auder and Bertrand Iooss. Global sensitivity analysis based on entropy. In *Safety, reliability and risk analysis-Proceedings of the ESREL 2008 Conference*, 2107–2115. 2008.

[116] Farkhondeh Khorashadi Zadeh, Jiri Nossent, Fanny Sarrazin, Francesca Pianosi, Ann van Griensven, Thorsten Wagener, and Willy Bauwens. Comparison of variance-based and moment-independent global sensitivity analysis approaches by application to the swat model. *Environmental Modelling & Software*, 91:210–222, 2017.

[117] Francesca Pianosi and Thorsten Wagener. A simple and efficient method for global sensitivity analysis based on cumulative distribution functions. *Environmental Modelling & Software*, 67:1–11, 2015.

[118] Ronald Aylmer Fisher and others. Statistical methods for research workers. *Statistical methods for research workers.*, 1934.

[119] Loıc Brevault, Mathieu Balesdent, Nicolas Bérend, and Rodolphe Le Riche. Comparison of different global sensitivity analysis methods for aerospace vehicle optimal design. In *10th World Congress on Structural and Multidisciplinary Optimization, WCSMO-10*. 2013.

[120] GEB Archer, Andrea Saltelli, and IM Sobol. Sensitivity measures, anova-like techniques and the use of bootstrap. *Journal of Statistical Computation and Simulation*, 58(2):99–120, 1997.

[121] Art B Owen. Variance components and generalized sobol'indices. *SIAM/ASA Journal on Uncertainty Quantification*, 1(1):19–41, 2013.

[122] Emanuele Borgonovo. Measuring uncertainty importance: investigation and comparison of alternative approaches. *Risk analysis*, 26(5):1349–1361, 2006.

[123] Emanuele Borgonovo. A new uncertainty importance measure. *Reliability Engineering & System Safety*, 92(6):771–784, 2007.

[124] Elmar Plischke, Emanuele Borgonovo, and Curtis L Smith. Global sensitivity measures from given data. *European Journal of Operational Research*, 226(3):536–550, 2013.

[125] Jiri Nossent, Pieter Elsen, and Willy Bauwens. Sobol' sensitivity analysis of a complex environmental model. *Environmental Modelling & Software*, 26(12):1515–1525, 2011. URL: https://www.sciencedirect.com/science/article/pii/S1364815211001939, doi:https://doi.org/10.1016/j.envsoft.2011.08.010.

[126] Jon Herman and Will Usher. Salib: an open-source python library for sensitivity analysis. *Journal of Open Source Software*, 2(9):97, 2017.

[127] Hoshin V Gupta, Thorsten Wagener, and Yuqiong Liu. Reconciling theory with observations: elements of a diagnostic approach to model evaluation. *Hydrological Processes: An International Journal*, 22(18):3802–3813, 2008.

[128] Keith Beven and Jim Freer. Equifinality, data assimilation, and uncertainty estimation in mechanistic modelling of complex environmental systems using the glue methodology. *Journal of hydrology*, 249(1-4):11–29, 2001.

[129] Cameron McPhail, HR Maier, JH Kwakkel, M Giuliani, A Castelletti, and S Westra. Robustness metrics: how are they calculated, when should they be used and why do they give different results? *Earth's Future*, 6(2):169–191, 2018.

[130] John D Sterman. System dynamics modeling: tools for learning in a complex world. *California management review*, 43(4):8–25, 2001.

[131] John M Anderies, Jean-Denis Mathias, and Marco A Janssen. Knowledge infrastructure and safe operating spaces in social–ecological systems. *Proceedings of the National Academy of Sciences*, 116(12):5277–5284, 2019.

[132] Rachata Muneepeerakul and John M Anderies. The emergence and resilience of self-organized governance in coupled infrastructure systems. *Proceedings of the National Academy of Sciences*, 117(9):4617–4622, 2020.

[133] Antonia Hadjimichael, Patrick M Reed, and Julianne D Quinn. Navigating deeply uncertain tradeoffs in harvested predator-prey systems. *Complexity*, 2020.

[134] Julianne D Quinn, Patrick M Reed, and Klaus Keller. Direct policy search for robust multi-objective management of deeply uncertain socio-ecological tipping points. *Environmental modelling & software*, 92:125–141, 2017.

[135] S. R. Carpenter, D. Ludwig, and W. A. Brock. Management of Eutrophication for Lakes Subject to Potentially Irreversible Change. *Ecological Applications*, 9(3):751–771, August 1999. URL: http://onlinelibrary.wiley.com/doi/10.1890/1051-0761(1999)009{[}0751:MOEFLS{]}2.0.CO;2/abstract, doi:10.1890/1051-0761(1999)009[0751:MOEFLS]2.0.CO;2.

[136] Julianne Quinn. Julianneq/Lake_problem_dps. December 2017. original-date: 2017-02-06T18:33:54Z. URL: https://github.com/julianneq/Lake_Problem_DPS (visited on 2021-06-14).

[137] David Hadka. Project-Platypus/Rhodium. 2017. original-date: 2015-10-29T18:08:43Z. URL: https://github.com/Project-Platypus/Rhodium (visited on 2021-06-14).

[138] Stephen R. Carpenter, William A. Brock, Carl Folke, Egbert H. van Nes, and Marten Scheffer. Allowing variance may enlarge the safe operating space for exploited ecosystems. *Proceedings of the National Academy of Sciences*, 112(46):14384–14389, November 2015. URL: http://www.pnas.org/content/112/46/14384 (visited on 2017-08-18), doi:10.1073/pnas.1511804112.

[139] Mustafa Hekimoğlu and Yaman Barlas. Sensitivity analysis for models with multiple behavior modes: a method based on behavior pattern measures. *System Dynamics Review*, 32(3-4):332–362, 2016.

[140] Patrick Steinmann, Willem L Auping, and Jan H Kwakkel. Behavior-based scenario discovery using time series clustering. *Technological Forecasting and Social Change*, 156:120052, 2020.

[141] Steven C Bankes, Robert J Lempert, and Steven W Popper. Computer-assisted reasoning. *Computing in Science & Engineering*, 3(2):71–77, 2001.

[142] Robert J. Lempert, Steven W. Popper, and Steven C. Bankes. *Shaping the Next One Hundred Years*. RAND Corporation, 2003. URL: https://www.rand.org/pubs/monograph_reports/MR1626.html (visited on 2017-09-14).

[143] Jonathan R Lamontagne, Patrick M Reed, Robert Link, Katherine V Calvin, Leon E Clarke, and James A Edmonds. Large ensemble analytic framework for consequence-driven discovery of climate change scenarios. *Earth's Future*, 6(3):488–504, 2018.

[144] Brian C O'Neill, Elmar Kriegler, Keywan Riahi, Kristie L Ebi, Stephane Hallegatte, Timothy R Carter, Ritu Mathur, and Detlef P van Vuuren. A new scenario framework for climate change research: the concept of shared socioeconomic pathways. *Climatic change*, 122(3):387–400, 2014.

[145] Warren E Walker, Marjolijn Haasnoot, and Jan H Kwakkel. Adapt or perish: a review of planning approaches for adaptation under deep uncertainty. *Sustainability*, 5(3):955–979, 2013.

[146] Suraje Dessai, Mike Hulme, Robert Lempert, and Roger Pielke Jr. Climate prediction: a limit to adaptation. *Adapting to climate change: thresholds, values, governance*, 64:78, 2009.

[147] Jonathan D Herman, Patrick M Reed, Harrison B Zeff, and Gregory W Characklis. How should robustness be defined for water systems planning under change? *Journal of Water Resources Planning and Management*, 141(10):04015012, 2015.

[148] Robert J Lempert. Robust decision making (rdm). In *Decision making under deep uncertainty*, pages 23–51. Springer, Cham, 2019.

[149] Jan H Kwakkel and Marjolijn Haasnoot. Supporting dmdu: a taxonomy of approaches and tools. In *Decision Making under Deep Uncertainty*, pages 355–374. Springer, Cham, 2019.

[150] BC Trindade, PM Reed, and GW Characklis. Deeply uncertain pathways: integrated multi-city regional water supply infrastructure investment and portfolio management. *Advances in Water Resources*, 134:103442, 2019.

[151] Julianne D Quinn, Patrick M Reed, Matteo Giuliani, Andrea Castelletti, Jared W Oyler, and Robert E Nicholas. Exploring how changing monsoonal dynamics and human pressures challenge multireservoir management for flood protection, hydropower production, and agricultural water supply. *Water Resources Research*, 54(7):4638–4662, 2018.

[152] DF Gold, PM Reed, BC Trindade, and GW Characklis. Identifying actionable compromises: navigating multi-city robustness conflicts to discover cooperative safe operating spaces for regional water supply portfolios. *Water Resources Research*, 55(11):9024–9050, 2019.

[153] JR Lamontagne, PM Reed, G Marangoni, K Keller, and GG Garner. Robust abatement pathways to tolerable climate futures require immediate global action. *Nature Climate Change*, 9(4):290–294, 2019.

[154] Antonia Hadjimichael, Julianne Quinn, Erin Wilson, Patrick Reed, Leon Basdekas, David Yates, and Michelle Garrison. Defining Robustness, Vulnerabilities, and Consequential Scenarios for Diverse Stakeholder Interests in Institutionally Complex River Basins. *Earth's Future*, 8(7):e2020EF001503, 2020. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2020EF001503 (visited on 2020-07-13), doi:10.1029/2020EF001503.

[155] Harris Drucker and Corinna Cortes. Boosting decision trees. *Advances in neural information processing systems*, pages 479–485, 1996.

[156] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[157] Kelsey L. Ruckert, Gary Shaffer, David Pollard, Yawen Guan, Tony E. Wong, Chris E. Forest, and Klaus Keller. Assessing the Impact of Retreat Mechanisms in a Simple Antarctic Ice Sheet Model Using Bayesian Calibration. *PLOS ONE*, 12(1):e0170052, January 2017. doi:10.1371/journal.pone.0170052.

[158] B. Efron and R. Tibshirani. Bootstrap Methods for Standard Errors, Confidence Intervals, and Other Measures of Statistical Accuracy. *Statistical Science*, 1(1):54–75, February 1986. doi:10.1214/ss/1177013815.

[159] Ryan L. Sriver, Robert J. Lempert, Per Wikman-Svahn, and Klaus Keller. Characterizing uncertain sea-level rise projections to support investment decisions. *PLOS ONE*, 13(2):e0190641, February 2018. URL: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0190641 (visited on 2021-06-09), doi:10.1371/journal.pone.0190641.

[160] Kelsey L. Ruckert, Yawen Guan, Alexander M. R. Bakker, Chris E. Forest, and Klaus Keller. The effects of time-varying observation errors on semi-empirical sea-level projections. *Climatic Change*, 140(3):349–360, February 2017. URL: https://doi.org/10.1007/s10584-016-1858-z (visited on 2021-06-09), doi:10.1007/s10584-016-1858-z.

[161] Neil R Edwards, David Cameron, and Jonathan Rougier. Precalibrating an intermediate complexity climate model. *Clim. Dyn.*, 37(7-8):1469–1482, 2011. URL: http://dx.doi.org/10.1007/s00382-010-0921-0, doi:10.1007/s00382-010-0921-0.

[162] Alexis Boukouvalas, Pete Sykes, Dan Cornford, and Hugo Maruri-Aguilar. Bayesian Precalibration of a Large Stochastic Microsimulation Model. *IEEE Transactions on Intelligent Transportation Systems*, 15(3):1337–1347, June 2014. doi:10.1109/TITS.2014.2304394.

[163] David Makowski, Daniel Wallach, and Marie Tremblay. Using a Bayesian approach to parameter estimation; comparison of the GLUE and MCMC methods. *Agronomie*, 22(2):191–203, 2002. Publisher: EDP Sciences.

[164] Mahyar Shafii, Bryan Tolson, and Loren Shawn Matott. Uncertainty-based multi-criteria calibration of rainfall-runoff models: a comparative study. *Stochastic Environmental Research and Risk Assessment*, 28(6):1493–1510, August 2014. doi:10.1007/s00477-014-0855-x.

[165] Keith Beven and Jim Freer. Equifinality, data assimilation, and uncertainty estimation in mechanistic modelling of complex environmental systems using the GLUE methodology. *Journal of hydrology*, 249(1-4):11–29, 2001. Publisher: Elsevier.

[166] Jasper A. Vrugt and Keith J. Beven. Embracing equifinality with efficiency: Limits of Acceptability sampling using the DREAM(LOA) algorithm. *Journal of Hydrology*, 559:954–971, April 2018. doi:10.1016/j.jhydrol.2018.02.026.

[167] Jery R. Stedinger, Richard M. Vogel, Seung Uk Lee, and Rebecca Batchelder. Appraisal of the generalized likelihood uncertainty estimation (GLUE) method. *Water Resources Research*, 2008. doi:10.1029/2008WR006822.

[168] Christian Robert and George Casella. *Monte Carlo Statistical Methods*. Springer Science & Business Media, March 2013. ISBN 978-1-4757-3071-5.

[169] Christian P. Robert. The Metropolis–Hastings Algorithm. In *Wiley StatsRef: Statistics Reference Online*, pages 1–15. American Cancer Society, 2015. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118445112.stat07834 (visited on 2021-06-14), doi:10.1002/9781118445112.stat07834.

[170] James M. Flegal, Murali Haran, and Galin L. Jones. Markov Chain Monte Carlo: Can We Trust the Third Significant Figure? *Statistical Science*, 23(2):250–260, May 2008. Publisher: Institute of Mathematical Statistics. URL: https://projecteuclid.org/journals/statistical-science/volume-23/issue-2/Markov-Chain-Monte-Carlo--Can-We-Trust-the-Third/10.1214/08-STS257.full (visited on 2021-06-14), doi:10.1214/08-STS257.

[171] Carla Currin, Toby Mitchell, Max Morris, and Don Ylvisaker. Bayesian Prediction of Deterministic Functions, with Applications to the Design and Analysis of Computer Experiments. *Journal of the American Statistical Association*, 86(416):953–963, December 1991. Publisher: Taylor & Francis _eprint: https://www.tandfonline.com/doi/pdf/10.1080/01621459.1991.10475138. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1991.10475138 (visited on 2021-06-14), doi:10.1080/01621459.1991.10475138.

[172] Jerome Sacks, William J. Welch, Toby J. Mitchell, and Henry P. Wynn. Design and Analysis of Computer Experiments. *Statistical Science*, 4(4):409–423, 1989. Publisher: Institute of Mathematical Statistics. URL: https://www.jstor.org/stable/2245858 (visited on 2021-06-14).

[173] Roger G. Ghanem and Pol D. Spanos. Spectral Stochastic Finite-Element Formulation for Reliability Analysis. *Journal of Engineering Mechanics*, 117(10):2351–2372, October 1991. Publisher: American Society of Civil Engineers. URL: https://ascelibrary.org/doi/abs/10.1061/%28ASCE%290733-9399%281991%29117%3A10%282351%29 (visited on 2021-06-14), doi:10.1061/(ASCE)0733-9399(1991)117:10(2351).

[174] Dongbin Xiu and George Em Karniadakis. The Wiener–Askey Polynomial Chaos for Stochastic Differential Equations. *SIAM Journal on Scientific Computing*, 24(2):619–644, January 2002. Publisher: Society for Industrial and Applied Mathematics. URL: https://epubs.siam.org/doi/abs/10.1137/S1064827501387826 (visited on 2021-06-14), doi:10.1137/S1064827501387826.

[175] Angelo Ciccazzo, Gianni Di Pillo, and Vittorio Latorre. A SVM Surrogate Model-Based Method for Parametric Yield Optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(7):1224–1228, July 2016. Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. doi:10.1109/TCAD.2015.2501307.

[176] W. Andrew Pruett and Robert L. Hester. The Creation of Surrogate Models for Fast Estimation of Complex Model Outcomes. *PLOS ONE*, 11(6):e0156574, June 2016. Publisher: Public Library of Science. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0156574 (visited on 2021-06-14), doi:10.1371/journal.pone.0156574.

[177] John Eason and Selen Cremaschi. Adaptive sequential sampling for surrogate model generation with artificial neural networks. *Computers & Chemical Engineering*, 68:220–232, September 2014. URL: https://www.sciencedirect.com/science/article/pii/S0098135414001719 (visited on 2021-06-14), doi:10.1016/j.compchemeng.2014.05.021.

[178] Dirk Gorissen, Luciano De Tommasi, Karel Crombecq, and Tom Dhaene. Sequential modeling of a low noise amplifier with neural networks and active learning. *Neural Computing and Applications*, 18(5):485–494, June 2009. URL: https://doi.org/10.1007/s00521-008-0223-1 (visited on 2021-06-14), doi:10.1007/s00521-008-0223-1.

[179] Jenný Brynjarsdóttir and Anthony O'Hagan. Learning about physical parameters: the importance of model discrepancy. *Inverse Problems*, 30(11):114007, October 2014. Publisher: IOP Publishing. URL: https://doi.org/10.1088/0266-5611/30/11/114007 (visited on 2021-06-14), doi:10.1088/0266-5611/30/11/114007.

[180] Michael Betancourt. A conceptual introduction to Hamiltonian Monte Carlo. *arXiv preprint arXiv:1701.02434*, 2017.

[181] Radford M. Neal. MCMC using Hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

[182] Matti Vihola. Robust adaptive Metropolis algorithm with coerced acceptance rate. *Statistics and Computing*, 22(5):997–1008, September 2012. URL: https://doi.org/10.1007/s11222-011-9269-5 (visited on 2021-06-14), doi:10.1007/s11222-011-9269-5.

[183] Perry de Valpine, Daniel Turek, Christopher J. Paciorek, Clifford Anderson-Bergman, Duncan Temple Lang, and Rastislav Bodik. Programming With Models: Writing Statistical Algorithms for General Model Structures With NIMBLE. *Journal of Computational and Graphical Statistics*, 26(2):403–413, April 2017. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/10618600.2016.1172487. URL: https://doi.org/10.1080/10618600.2016.1172487 (visited on 2021-06-14), doi:10.1080/10618600.2016.1172487.

[184] NIMBLE Development Team. NIMBLE: MCMC, Particle Filtering, and Programmable Hierarchical Modeling. May 2021. URL: https://zenodo.org/record/4829693 (visited on 2021-06-14), doi:10.5281/zenodo.4829693.

[185] Stan Development Team. Stan Modeling Language Users Guide and Reference Manual. 2021. URL: https://mc-stan.org/docs/2_27/stan-users-guide/index.html (visited on 2021-06-14).

[186] John Salvatier, Thomas V. Wiecki, and Christopher Fonnesbeck. Probabilistic programming in Python using PyMC3. *PeerJ Computer Science*, 2:e55, April 2016. Publisher: PeerJ Inc. URL: https://peerj.com/articles/cs-55 (visited on 2021-06-14), doi:10.7717/peerj-cs.55.

[187] Hong Ge, Kai Xu, and Zoubin Ghahramani. Turing: A Language for Flexible Probabilistic Inference. In *International Conference on Artificial Intelligence and Statistics*, 1682–1690. PMLR, March 2018. ISSN: 2640-3498. URL: http://proceedings.mlr.press/v84/ge18b.html (visited on 2021-06-14).

[188] Andrew Gelman and Donald B. Rubin. Inference from Iterative Simulation Using Multiple Sequences. *Statistical Science*, 7(4):457–472, November 1992. Publisher: Institute of Mathematical Statistics. URL: https://projecteuclid.org/journals/statistical-science/volume-7/issue-4/Inference-from-Iterative-Simulation-Using-Multiple-Sequences/10.1214/ss/1177011136.full (visited on 2021-06-14), doi:10.1214/ss/1177011136.

[189] Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. Sequential Monte Carlo samplers. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(3):411–436, 2006. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-9868.2006.00553.x. URL: http://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9868.2006.00553.x (visited on 2021-06-14), doi:10.1111/j.1467-9868.2006.00553.x.

[190] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10(3):197–208, July 2000. URL: https://doi.org/10.1023/A:1008935410038 (visited on 2021-06-14), doi:10.1023/A:1008935410038.

[191] Jane Liu and Mike West. Combined Parameter and State Estimation in Simulation-Based Filtering. In Arnaud Doucet, Nando de Freitas, and Neil Gordon, editors, *Sequential Monte Carlo Methods in Practice*, Statistics for Engineering and Information Science, pages 197–223. Springer, New York, NY, 2001. URL: https://doi.org/10.1007/978-1-4757-3437-9_10 (visited on 2021-06-14), doi:10.1007/978-1-4757-3437-9_10.

[192] Stefano Cabras, Maria Eugenia Castellanos Nueda, and Erlis Ruli. Approximate Bayesian Computation by Modelling Summary Statistics in a Quasi-likelihood Framework. *Bayesian Analysis*, 10(2):411–439, June 2015. Pub-

lisher: International Society for Bayesian Analysis. URL: https://projecteuclid.org/journals/bayesian-analysis/volume-10/issue-2/Approximate-Bayesian-Computation-by-Modelling-Summary-Statistics-in-a-Quasi/10.1214/14-BA921.full (visited on 2021-06-14), doi:10.1214/14-BA921.

[193] Jarno Lintusaari, Michael U. Gutmann, Ritabrata Dutta, Samuel Kaski, and Jukka Corander. Fundamentals and Recent Developments in Approximate Bayesian Computation. *Systematic Biology*, 66(1):e66–e82, January 2017. URL: https://doi.org/10.1093/sysbio/syw077 (visited on 2021-06-14), doi:10.1093/sysbio/syw077.

[194] Mikael Sunnåker, Alberto Giovanni Busetto, Elina Numminen, Jukka Corander, Matthieu Foll, and Christophe Dessimoz. Approximate Bayesian Computation. *PLOS Computational Biology*, 9(1):e1002803, January 2013. Publisher: Public Library of Science. URL: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1002803 (visited on 2021-06-14), doi:10.1371/journal.pcbi.1002803.

[195] Edwin T. Jaynes. *Probability theory: the logic of science*. Washington University St. Louis, MO, 1996.

[196] Andrew Gelman, Daniel Simpson, and Michael Betancourt. The Prior Can Often Only Be Understood in the Context of the Likelihood. *Entropy*, 19(10):555, October 2017. Number: 10 Publisher: Multidisciplinary Digital Publishing Institute. URL: https://www.mdpi.com/1099-4300/19/10/555 (visited on 2021-06-14), doi:10.3390/e19100555.

[197] Christian Robert. *The Bayesian choice: from decision-theoretic foundations to computational implementation*. Springer Science & Business Media, 2007.

[198] Andrew Gelman, Xiao-Li Meng, and Hal Stern. Posterior Predictive Assessment of Model Fitness via Realized Discrepancies. *Statistica Sinica*, 6(4):733–760, 1996. Publisher: Institute of Statistical Science, Academia Sinica. URL: https://www.jstor.org/stable/24306036 (visited on 2021-06-14).

[199] Andrew Gelman, Aki Vehtari, Daniel Simpson, Charles C. Margossian, Bob Carpenter, Yuling Yao, Lauren Kennedy, Jonah Gabry, Paul-Christian Bürkner, and Martin Modrák. Bayesian Workflow. *arXiv:2011.01808 [stat]*, November 2020. arXiv: 2011.01808. URL: http://arxiv.org/abs/2011.01808 (visited on 2021-06-14).

[200] Andrew Gelman and Cosma Rohilla Shalizi. Philosophy and the practice of Bayesian statistics. *British Journal of Mathematical and Statistical Psychology*, 66(1):8–38, 2013. _eprint: https://bpspsychub.onlinelibrary.wiley.com/doi/pdf/10.1111/j.2044-8317.2011.02037.x. URL: https://bpspsychub.onlinelibrary.wiley.com/doi/abs/10.1111/j.2044-8317.2011.02037.x (visited on 2021-06-14), doi:10.1111/j.2044-8317.2011.02037.x.