INCOSE

ASIA-OCEANIA SYSTEMS ENGINEERING CONFERENCE

Improving Model Validity with Sensitivity Analysis

Method and Implementation

INCOSE AOSEC · 2025

AUTHORS

Thomas C. Zimmermann, Benjamin Futász, and Johan Cederbladh

PUBLICATION DATE

July, 2026

SUBMISSION #

83

INCOSE PUBLICATIONS

Improving Model Validity with Sensitivity Analysis: Method and Implementation

Thomas C. Zimmermann
Fraunhofer IPK
Pascalstr. 8-9, 10587 Berlin, Germany
thomas.zimmermann@ipk.fraunhofer.de

Benjamin Futász
Fraunhofer IPK
Pascalstr. 8-9, 10587 Berlin, Germany
Benjamin.futasz@ipk.fraunhofer.de

Johan Cederbladt
Mälardalen University, Sweden
Box 883, 721 23 Västerås, Sweden
johan.cederbladh@mdu.se

Abstract

Model-Based Systems Engineering (MBSE) increasingly relies on simulation as a core capability throughout the system life cycle to support trade-off studies, risk assessment, and evidence-based decision-making. However, ensuring the validity and robustness of simulation results—especially for complex, high-dimensional, non-linear cyber-physical systems—remains a critical challenge. In this paper, we introduce a lightweight, open-source Python library that seamlessly integrates global sensitivity analysis into Functional Mock-up Interface (FMI)–based co-simulations. Building on the widely adopted FMI standard, our tool employs a two-adapter pattern, one adapter for FMU simulation and another for sensitivity computation, to deliver a thin orchestration layer over both simulators and analysis engines. Users specify the sensitivity method (Morris or Sobol), the FMU to test, parameter bounds, sampling size, and stop time. The library then validates these settings, generates parameter samples, runs the FMU for each sample, computes the chosen sen-sitivity indices, and applies user-defined acceptance criteria. By automating sampling, simulation, and index calculation, the tool replaces ad-hoc scripts, ensures full traceability through GUIDs and metadata, and fits seamlessly into MBSE and digital engineering workflows, thereby boosting confidence in simulation results and guiding model refinement.

Keywords

Sensitivity Analysis, MBSE, Morris Method, Sobol Method, Model Validity.

Introduction

In systems development, decisions are often based on predictions derived from modeling and simulation of partial or entire systems of interest (Horber, Pickel, Goetz, & Wartzack, 2024). Sensitivity analysis of a system model can help to identify potential problem areas which may give rise to high dependencies of outputs on specific sub-system elements (Murray-Smith, 2015). Therefore, sensitivity analysis is of great importance in modeling and simulation to improve model reliability, ensure the validity of models and support effective decision-making. Under-standing the robustness of outcomes enables more informed and confident decision-making. With the increasing integration of Model-Based Systems Engineering (MBSE), the role of simulation is further made central to system development. Supporting valid integration

across a system's life cycle is therefore of utmost importance to ensure valid decision-making.

The Functional Mockup Interface (FMI) is a tool independent standard for the exchange of dynamic models and for Co-Simulation (Hällqvist, Munjulury, Braun, Eek, & Krus, 2022). The Functional Mockup Interface (FMI) defines a standardized interface to be used in co-simulations of complex cyber-physical systems. The original FMI standard was developed by many software companies and research centers in the collaborative project MODELISAR led by a European consortium led by Dassault Systèmes of France (Blockwitz et al., 2012). Co-simulation proves to be beneficial in validating multi-domain and cyber-physical systems, as it provides a flexible solution that allows simultaneous consideration of multiple domains with different time steps. Since the computational load is shared between simulators, co-simulation also offers the possi-bility of large-scale system evaluation (Zimmermann, Manoury, & Lindow, 2024).

Despite these advantages, practitioners still face significant hurdles when applying global sensitivity methods to FMU-based simulations. Current workflows often rely on ad-hoc scripts, lack seamless integration with co-simulation environments, and require specialized expertise to handle parameter sampling and result aggregation. This fragmentation slows down the analysis process, introduces potential errors, and limits reproducibility, impeding the widespread adoption of robust sensitivity analysis in industrial settings. This paper introduces a structured workflow that employs both the Morris and Sobol methods to conduct sensitivity analysis, enabling re-searchers to efficiently identify key parameters and their interactions. We then present an im-plemented library designed to facilitate the application of these methods, enhancing accessibility for model developers and analysts. Our Python library seamlessly integrates Morris and Sobol sensitivity workflows into FMU-based co-simulations, eliminating ad-hoc scripts, manual con-figuration steps, and reproducibility issues. This makes the sensitivity analysis of complex system models markedly more efficient, scalable, and user-friendly. In conclusion, we reflect on the insights gained from this analysis and outline future directions for research, highlighting the potential for sensitivity analysis to significantly improve the robustness and accuracy of FMU-based simulations.

Problem Statement and Background

Complex cyber-physical systems (CPS) are characterized by various interconnected subsystems and components (Khaitan & McCalley, 2015). As per the FMI Standard, a network of system elements to be co-simulated consists of various functional-mockup units, which are essentially wrappers for domain-specific models.

In industry, those models have often been created prior, as part of domain-specific development activities, without co-simulation use cases in mind (Cederbladh, Cicchetti, & Jongeling, 2025). FMI enables data exchange between subsystems represented by Functional Mockup Units (FMU) through discrete communication points that represent external interfaces of subsystems. The FMI standard defines two main types, FMI for Co-Simulation (CS) and FMI for Model Exchange (ME). FMUs for CS integrate a solver. FMUs for ME require an external solver. Regardless of internal or external solvers, FMUs are solved independently from each other by their individual solver. Master algorithms control the data exchange between subsystems and the synchronization of individual sub-system model solvers (Blockwitz et al., 2012). FMI supports advanced master algorithms, including the usage of variable communication step sizes, higher order signal extrapolation, and error control. Sensitivity analysis of FMUs faces several interrelated challenges: the simulation of multiple parameters that interact in complex, non-linear ways makes it difficult to isolate individual effects; the high dimensionality of many models leads to computationally intensive analyses that are hard to interpret; the non-linear behavior of dynamic systems means small parameter changes can produce disproportionately large output variations; tight coupling between subsystems complicates the attribution of overall system behavior to any single component; and limited traceability hinders the ability to maintain a clear audit trail from parameter perturbations through to their

impact on model outputs, reducing reproducibility, error tracking, and informed decision-making.

In systems engineering and simulation, robust and efficient analysis methods are crucial for quantifying risk, managing uncertainty, and underpinning informed decision support amid the inherent complexity of interconnected system elements. These systems often include a variety of subsystems, each represented by different models that capture specific behaviors and interactions. Particularly, as MBSE is often a collaborative venture between engineering domains, mechanisms to integrate models on both a synthetic and semantic level become necessary. In this context, sensitivity analysis plays a crucial role in enhancing model validity within MBSE frameworks. By systematically examining how variations in input parameters affect model outputs, sensitivity analysis helps identify critical parameters that drive system behavior. This understanding is essential for refining models, ensuring that they accurately represent the complexities of interconnected subsystems. Moreover, the iterative nature of MBSE aligns well with the principles of sensitivity analysis, where insights gained from one analysis can inform subsequent model revisions. The integration of sensitivity analysis into MBSE not only supports robust model development but also enhances confidence in decision-making processes. As systems engineering increasingly relies on data-driven approaches, the ability to quantify the impact of uncertainties and interactions among parameters will be vital in achieving reliable and valid models consistent between system, subsystem and component level representations. As these subsystems interact, their collective behavior can lead to emergent properties that are not easily predictable from individual component models alone. The interconnected nature of these elements means that changes or faults in one part of the system can significantly affect others, necessitating comprehensive analysis methods to understand these interdependencies. Robust analytical methods are essential to ensure that the system meets performance, reliability, and safety requirements, especially in complex applications such as aerospace, automotive, and industrial automation. The presence of multiple models for each subsystem adds another layer of complexity, as these models may vary in fidelity, abstraction, and the assumptions they are based on. Efficient analysis methods are therefore crucial to manage this complexity, allowing engineers to evaluate various design alternatives, optimize performance, and conduct sensitivity analyses to identify critical parameters. In addition, as systems become increasingly dynamic and operate in uncertain environments, robust analytical methods become vital for risk assessment and decision-making. They help engineers anticipate potential issues, validate system behavior against requirements, and ensure that the design is resilient to variations in operating conditions. Overall, the need for robust and efficient methods in systems engineering and simulation arises from the complexity of interconnected subsystems, the necessity to manage multiple models, and the imperative to ensure system performance and reliability in the face of uncertainty.

Category	Example Method	Type	Purpose	Computational Cost	Notes
Local Sensitivity	One-at-a-Time (OAT)	Local	Derivative-based analysis	Low	Evaluates sensitivity at a specific point in the input space.
Screening Methods	Morris Method	Global	Identifying influential factors	Moderate	Efficient for models with many inputs; provides qualitative insights.
Quantitative Methods	Sobol Method	Global	Variance-based sensitivity indices	High	Provides detailed quantitative measures of input contributions.

Table 1: Categories of Sensitivity Analysis Methods

Sensitivity analysis methods can be divided into three categories, as shown in Table 1, namely local sensitivity analysis methods, as well as global screening, and quantitative analysis methods (Li, Jiang, Hu, & Yan, 2023). Local sensitivity analysis examines how small changes in inputs impact outputs around a specific point, while global sensitivity analysis via quantitative methods evaluates the effects of varying inputs across their entire range (Li et al., 2023). While local sensitivity analyses focus on small perturbations around specific input values, screening methods aim to evaluate the effects of input variations across a broader range but usually with a reduced computational effort compared to full-scale global sensitivity analysis. Screening methods, such as the Morris method,

are designed to quickly determine which parameters warrant further in-vestigation in a more comprehensive global sensitivity analysis (Li et al., 2023).

Knowing the extent of uncertainty related to input variables is crucial for reliable model pre-dictions and effective risk assessment. The applications of sensitivity analysis are wide-ranging, covering areas like finance, engineering, and environmental science. In finance, it is utilized to assess the risks linked to investment strategies. In environmental science, it is instrumental in evaluating how different factors influence ecological models. In engineering, sensitivity analysis plays a vital role in optimizing designs and improving system performance. (Reed et al., 2025).

Although it can be demonstrated that a model is invalid for certain conditions, it cannot be proven that it is universally valid. At best, it can be shown that the model is regarded as an ac-ceptable representation of the real system for the specific purposes of the intended application. The primary performance dimensions (relatively independent of external quality factors) of simulation models are fidelity, accuracy and resolution. Model fidelity reflects the level of physical and behavioral detail represented in a simulation and depends on the assumptions, scope, and intended use of the model. Accuracy is determined through validation and verification against empirical or analytical references and is influenced by both fidelity and resolution.

Uncertainty and sensitivity analyses are critical for assessing confidence in model predictions and identifying key parameters that impact output reliability. Fernández-Godino (2023) highlights that “establishing the superiority of one fidelity over another can be a complex endeavor” (Giselle Fernández-Godino, 2023) and that the comparative value of different fidelity levels is con-text-dependent, varying with objectives, constraints, and specific simulation tasks. In literature approaches to score fidelity can be found, but explicitly notes that these frameworks have limited practical applicability for measurement due to the inherent complexity and subjective compo-nents involved in defining fidelity in real-world modelling contexts (Liu, Macchiarella, & Vin-cenzi, 2008).

Morris screening, a global approach for high-dimensional models, efficiently identifies non-influential inputs and yields a qualitative ranking of input importance with relatively low computational cost. Sobol analysis performs a global, quantitative decomposition of output variance, attributing contributions to individual inputs and their interactions and delivering de-tailed sensitivity indices at the expense of a larger number of model evaluations. (Iooss & Le-maître, 2015)

When conducting sensitivity analysis, it is generally advisable to apply the Morris method first followed by Sobol, because the Morris method focused on identifying which parameters have a significant impact on the output of a model.

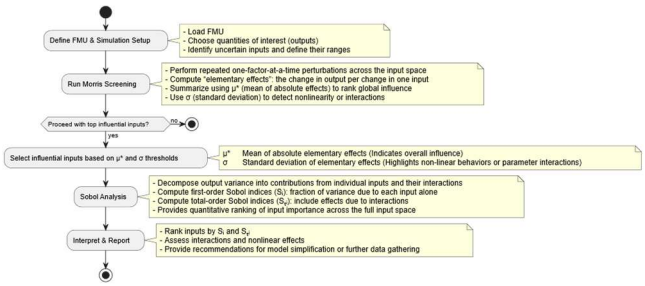


Figure 1: Sensitivity Analysis: Morris screening and Sobol analysis

The suggested approach illustrated in Fig.1 follows the same workflow of first applying a relatively fast screening followed by an in-depth analysis, leveraging the versatility of the developed sensitivity analysis library presented in the next section. It provides a quick way to evaluate many parameters at once and helps to reduce the dimensionality of the problem by identifying the most influential parameters. This is particularly useful when dealing with complex models with many inputs. Once the most influential parameters have been identified using the Morris method, the Sobol method can then be employed for a more detailed quantification of the sensitivity. Sobol sensitivity analysis can provide not only the main effects of parameters but also their interactions, giving a comprehensive view of how input variables affect the output. Using these two methods in succession allows for a complementary approach to sensitivity analysis. The Morris method can guide the application of the Sobol method by focusing computational resources on the most impactful parameters, making the overall analysis more efficient and effective.

Sensitivity Analysis with FMU Test Library

This subsection presents a lightweight Python library that turns sensitivity analysis of Functional Mock-up Units (FMUs) into repeatable, configuration-driven unit tests within MBSE workflows. Targeted at FMI 3.0 co-simulation, yet compatible with FMI 1.0/2.0 through the chosen simulator, the library provides a thin orchestration layer over two pluggable components: an FMU simulator and a sensitivity analysis engine. (source code available at https://gitlab.cc-asp.fraunhofer.de/ipk-oe100public/mbsesensitivity) The requirements focus on enabling Morris and Sobol sensitivity methods for

FMU-based co-simulation in Python while keeping dependencies minimal and integration straightforward. To that end, the core relies only on ubiquitous scientific Python packages and delegates numerical work to adapters for a simulator (default: FMPy (Kleine, 2020)) and a sensitivity analyzer (default: SALib (Herman & Usher, 2017; Iwanaga, Usher, & Herman, 2022)). Interoperability is achieved through a dictionary structure for both experiment configuration and results, which eases connection to parameter databases. Crucially, users retain freedom of choice: either the simulator or the analysis implementation, or both, can be replaced without changing the public API or the result schema.

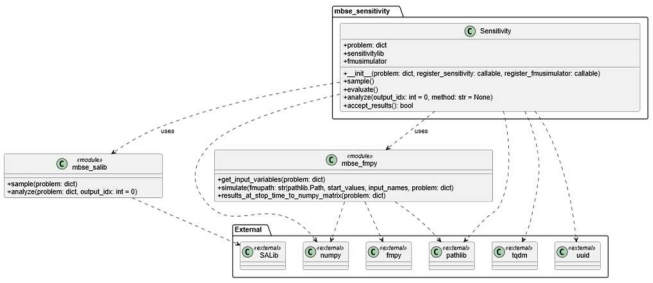


Figure 2: Class and Module overview of the Sensitivity Analysis Library

The architecture centers on a single orchestrator, the Sensitivity class, which manages the full life cycle from configuration to pass/fail decision. At construction, users provide a configuration object, register a simulator adapter and an analyzer adapter. The configuration, typically sourced from a JSON document, declares the sensitivity method, FMU file path, parameter bounds, and method-specific options. The library then prescribes a five-stage pipeline. First, it validates and loads the configuration, ensuring required fields are present and that options are consistent with the selected method. The following stages are repeatable. Second, it triggers sampling, delegating to the analyzer to generate designs such as Morris trajectories or Sobol sequences within declared bounds, which will be start parameters for the FMU. Third, it evaluates the FMU across the generated parameters, relying on the simulator adapter to manage FMI life cycle calls, co-simulation stepping, and output collection. Fourth, it computes sensitivity indices, again delegating to the analyzer to derive Morris statistics (e.g., μ^*, σ) or Sobol indices (e.g., first-order, total-order). Finally, it applies user-defined acceptance criteria, which maps sensitivity claims to a Boolean outcome. The metamodel remains deliberately simple to keep the orchestration thin and implementation-agnostic. The Sensitivity class encapsulates configuration state, registered adapters, sampled designs, raw evaluation results, derived indices, and the final acceptance outcome. Alignment to FMI standards is achieved through the simulator adapter rather than the core: the default FMPy-based adapter supports FMI 1.0, 2.0, and 3.0 and handles both Co-Simulation and Model Exchange. The library adopts a two-adapter pattern—mbse_fmpy.py for simulation and mbse_salib.py for analysis in the default distribution—exposed through a main orchestration script (mbse_sensitivity.py). This separation keeps the core stable and minimal while allowing users to substitute either adapter to integrate vendor simulators, in-house co-simulation masters, or proprietary sensitivity methods. JSON is leveraged for configuration and results to support reproducibility, versioning, and auditability; it also facilitates ingestion into databases and comparison across runs. Furthermore, by decoupling orchestration from implementation, the framework integrates seamlessly into MBSE workflows and continuous integration pipelines, automatically validating model changes against sensitivity criteria. Community-driven extensions can introduce new adapter interfaces, alternative analysis engines, and richer reporting formats. This open, modular design not only future-proofs the tool but also fosters collaboration across teams and domains.

Method	Role	Notes
__init__(config, register_sensitivity, register_fmusimulator)	Constructor	Config from dict (e.g., JSON-sourced); Optional custom dependency injection of implementations
sample()	Generate parameter samples	Delegates to analyzer
evaluate()	Run FMU across samples	Delegates to simulator; collects outputs and stop-time snapshots
analyze()	Compute sensitivity indices	Delegates to analyzer; Morris and/or Sobol
accept_result() → bool	Decide pass/fail	Evaluates user-defined acceptance rules

Table 2: Methods available in the Sensitivity class

The public API only exposes the Sensitivity class methods listed in Table 2. The constructor accepts

a configuration and adapter registrations; sample() generates parameter sets; evaluate() executes simulations and captures both time series and stop-time values for named outputs; analyze() computes indices for the selected method; and accept_result() collects a pass/fail decision for the previously computed indices.

Config Key	Required	Description
method	mandatory	"morris" or "sobol"
fmu_path	mandatory	Path to FMU under test
seed	optional	Reproducible samples in sensitivity analysis
N	optional	Base sample size
num_resamples	optional	Number of bootstrap resamples used to estimate uncertainty
num_levels	optional	Discretization for Morris
optimal_trajectories	optional	Trajectory optimization for Morris
bounds	optional	Per-parameter lower/upper bounds
stop_time	optional	Simulation horizon (falls back to FMU/experiment defaults)
guid	optional	Globally Unique Identifier (defaults to self-generated)

Table 3: Configuration Schema of the Sensitivity Analysis FMU Tests Library

The schema in Table 3 defines all inputs required to run a sensitivity analysis with the FMU Tests Library. It requires exactly two keys. "method" selects either " morris" for elementary effect screening or "sobol" for variance - based decomposition. and "fmu_path" to specify the file system path to the target FMU. All other entries are optional but offer more detailed control over sampling, analysis rigor, and traceability: User can apply an integer seed for the random number generator, ensuring that repeated runs produce identical sample sets. Another way of adapting the library to specific use cases is the "optimal_trajectories" configuration. This is a boolean toggle that, when true, applies trajectory - optimization heuristics to reduce redundancy in Morris designs, improving estimator efficiency.

"bounds" is a dictionary to mapp each parameter name to a [lower, upper] interval, establishing the permissible range for sampling. Parameters without explicit bounds revert to FMU defaults if available

Result Key	Description
results	Time series per run and output
results_at_stop_time	Output values at stop_time per run
output_names	List of observed variables
result_morris	Elementary effects statistics (μ, μ*, σ)
result_sobol	First-order, total-order (and optional second-order) indices
accept_results	Boolean and rationale for pass/fail

Table 4: Results Schema of the Sensitivity Analysis FMU Tests Library

The results add to this structure (depicted in Table 4) with raw run data, values at the stop time, the list of observed outputs, method-specific results (result_morris and result_sobol), and the final acceptance decisions (accept_results). The configuration and results dictionary carries a unique experiment identifier (guid).

A single-directory layout prioritizes accessibility: users can drop the module into Jupyter Notebooks or Python scripts for rapid experimentation, or package it as a wheel for integration with larger toolchains. Performance-wise, wall-clock time is dominated by the FMU's own runtime; the library adds only negligible overhead from function calls, dictionary bookkeeping, and NumPy-based data handling. The library targets Python 3.1x and relies on NumPy (Harris et al., 2020) and tqdm (Da Costa-Luis, 2019) in the core, with FMPy and SALib as default adapters. It introduces no native extensions and therefore runs wherever Python, the simulator, and the analyzer run, namely Linux, Windows, and macOS. Installation can be as simple as adding the module directory to a project or distributing a wheel; the absence of heavy dependencies eases adoption in controlled environments. Because the orchestrator's overhead is small relative to FMU execution, the achievable throughput is typically bounded by the simulator's implementation and the complexity of the FMU itself.

Integration into MBSE toolchains follows a familiar pattern. FMUs are exported from modeling tools (>250 commercial and open source tools at the time of writing), while architecture models supply the scenario variables, parameter bounds, and verification intent. These artifacts are transformed into an experiment configuration with a unique Globally Unique Identifier (GUID). The library executes the experiment, computes sensitivity indices, and emits a pass/fail decision alongside raw and derived data. The GUID, together with the preserved configuration, enables straightforward back-annotation into engineering repositories and supports optional audit trails for requirement verification. Users can replace the simulator adapter to target alternative FMU runtimes or master algorithms or swap the analysis adapter to integrate custom methods while preserving the library's public API and data contracts. Domain-specific acceptance policies can be implemented within accept_result() without touching sampling or evaluation code, allowing teams to encode verification criteria such as minimum total-order influence of a controller gain on a safety-critical output or stability of Morris rankings across trajectories.

The reference repository includes scripts to re-run experiments end-to-end with deterministic seeds, pinned dependency versions, and schema validation for both configuration and results. Continuous integration workflows can execute reference experiments on a small suite of FMUs to guard against regressions and produce platform fingerprints for transparency. Together, these choices produce a concise, extensible, and traceable framework for sensitivity-analysis-based unit testing of FMUs that integrates cleanly with FMI based co-simulation and the broader engineering artifact verification process.

Conclusion and Outlook

This paper has presented a cohesive workflow and a lightweight Python library that integrate Morris screening and Sobol analysis directly into FMU-based co-simulations, addressing longstanding challenges such as ad-hoc scripting, high-dimensional parameter spaces, subsystem coupling, and poor traceability. By adopting a two-adapter architecture—one for FMU simulation and one for sensitivity analysis—and driving experiments through JSON configurations, the implementation ensures interoperability with diverse tools, reproducibility of results, and seamless integration into digital engineering toolchains. Performance evaluations confirm that the library adds only minimal orchestration overhead, with throughput primarily bounded by the FMU's own runtime, thereby enabling efficient and scalable sensitivity studies.

The unified API for sampling, evaluation, index computation, and pass/fail decision-making substantially reduces manual effort and error-proneness, while the embedded metadata (experiment ID, FMU hash, adapter versions, git commit) supports full audit trails and requirement-based verification. In doing so, the proposed framework elevates sensitivity analysis from an ad-hoc, expert-driven activity to a systematic, traceable component of model-based validation. Ultimately, this work advances model validity for complex cyber-physical systems by making sensitivity analysis more accessible, reliable, and fit for decision support in industrial settings. Beyond these technical gains, our approach directly strengthens risk management and uncertainty quantification in modeling and simulation of complex systems. By systematically uncovering which parameters and interactions drive output variability, stakeholders can allocate validation and testing resources more effectively, reducing the likelihood of hidden vulnerabilities. The ability to trace each perturbation through to its impact on model outputs builds confidence in the fidelity of predictions and supports robust decision-making under uncertainty. In safety-critical domains—such as aerospace, automotive, and energy—this translates into more resilient designs, clearer risk profiles, and defensible verification evidence for regulatory compliance.

Despite our solution enabling system architects and integrators rather than model Domain experts to perform reliable sensitivity analyses, defining upper and lower bounds for each parameter still relies on domain expertise and must be specified manually for every model. To streamline this process and to enhance traceability and reproducibility we intend to centralize parameter - bound metadata in a model registry or embed it within the FMU artifact itself, making ranges available

automatically at runtime. Although users can already control when a co-simulation stops, a comprehensive temporal sensitivity study often demands multiple runs with progressively extended horizons to capture long-term effects and delayed interactions. Introducing adaptive sampling strategies that concentrate computational effort where output uncertainty remains highest, thereby tightening risk bounds and allow for fewer model Evaluations, further improving efficiency in resource-constrained environments even with in-Depth temporal analyses. Closer integration with digital engineering toolchains (for example, via connectors to requirements-management platforms and real-time risk-monitoring dashboards) will embed sensitivity analysis more deeply into industrial workflows, strengthening decision support and enhancing confidence in complex system simulations.

References

Blockwitz, T., Otter, M., Akesson, J., Arnold, M., Clauss, C., Elmqvist, H., . . . Viel, A. (2012). Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models. In Linköping Electronic Conference Proceedings, Proceedings of the 9th International MODELICA Conference, September 3-5, 2012, Munich, Germany (pp. 173–184). Linköping University Electronic Press. https://doi.org/10.3384/ecp12076173

Cederbladh, J., Cicchetti, A., & Jongeling, R. (2025). A Road-Map to Readily Available Early Validation and Verification of System Behaviour in Model-Based Systems Engineering using Software Engineering Best Practices. ACM Transactions on Software Engineering and Methodology, 34(5), 1–30. https://doi.org/10.1145/3708520

Da Costa-Luis, C. O. (2019). tqdm: A Fast, Extensible Progress Meter for Python and CLI. The Journal of Open Source Software, 4(37), 1277. https://doi.org/10.21105/joss.01277

Giselle Fernández-Godino, M. (2023). Review of multi-fidelity models. Advances in Computational Science and Engineering, 1(4), 351–400. https://doi.org/10.3934/acse.2023015

Hällqvist, R., Munjulury, R. C., Braun, R., Eek, M., & Krus, P. (2022). Realizing Interoperability between MBSE Domains in Aircraft System Development. Electronics, 11(18), 2901. https://doi.org/10.3390/electronics11182901

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., . . . Oliphant, T. E. (2020). Array programming with NumPy. Nature, 585(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Herman, J. [Jon], & Usher, W. [Will] (2017). SALib: An open-source Python library for Sensitivity Analysis. The Journal of Open Source Software, 2(9), 97. https://doi.org/10.21105/joss.00097

Horber, D., Pickel, J., Goetz, S., & Wartzack, S. (2024). Utilizing System Models for Multicriteria Decision-Making—A Systematic Literature Review on the Current State of the Art. IEEE Open Journal of Systems Engineering, 2, 135–147. https://doi.org/10.1109/OJSE.2024.3434310

Iooss, B., & Lemaître, P. (2015). A Review on Global Sensitivity Analysis Methods. In G. Dellino & C. Meloni (Eds.), Operations Research/Computer Science Interfaces Series. Uncertainty Management in Simulation-Optimization of Complex Systems (Vol. 59, pp. 101–122). Boston, MA: Springer US. https://doi.org/10.1007/978-1-4899-7547-8_5

Iwanaga, T., Usher, W. [William], & Herman, J. [Jonathan] (2022). Toward SALib 2.0: Advancing the accessibility and interpretability of global sensitivity analyses. Socio-Environmental Systems Modelling, 4, 18155. https://doi.org/10.18174/sesmo.18155

Khaitan, S. K., & McCalley, J. D. (2015). Design Techniques and Applications of Cyberphysical Systems: A Survey. IEEE Systems Journal, 9(2), 350–365. https://doi.org/10.1109/JSYST.2014.2322503

Kleine, O. (2020). FMPy: A Python library to generate and simulate Functional Mock\-up Units (FMUs). Retrieved from https://github.com/CATIA-Systems/FMPy

Li, D., Jiang, P., Hu, C., & Yan, T. (2023). Comparison of local and global sensitivity analysis methods and application to thermal hydraulic phenomena. Progress in Nuclear Energy, 158, 104612. https://doi.org/10.1016/j.pnucene.2023.104612

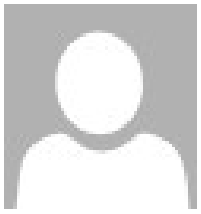
Liu, D., Macchiarella, N., & Vincenzi, D. (2008). Simulation Fidelity. In P. Hancock, D. Vincenzi, J. Wise, & M. Mouloua (Eds.), Human Factors in Simulation and Training (pp. 61–73). CRC Press. https://doi.org/10.1201/9781420072846.ch4

Murray-Smith, D. J. (2015). Testing and Validation of Computer Simulation Models. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-15099-4

Reed, P. M., Hadjimichael, A., Malek, K., Karimi, T., Vernon, C. R., Srikrishnan, V., . . . Rice, J. S. (2025). Addressing Uncertainty in Multisector Dynamics Research [Computer software]. Zenodo: Zenodo.

Zimmermann, T. C., Manoury, M., & Lindow, K. (2024). SysML v2 for Automated Co-simulation from Systems Architecture Models. In A. Salado, R. Valerdi, R. Steiner, & L. Head (Eds.), Conference on Systems Engineering Research Series. The Proceedings of the 2024 Conference on Systems Engineering Research (pp. 35–46). Cham: Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-62554-1_4

Biography



Thomas Zimmermann

Thomas C. Zimmermann holds a Master of Science in Aerospace, Aeronautical and Astronautical Engineering from Technische Universität Berlin and a Master of Engineering in Systems Engineering from Stevens Institute of Technology. He is currently a Researcher at the Fraunhofer Institute for Production Systems and Design Technology IPK. His research interests include Systems Engineering, Model-Based Systems Engineering (MBSE) and Modeling & Simulation.

Benjamin Futász

Benjamin Futász holds a Bachelor's degree in Mathematics from the University of Halle, Germany. He is currently a Master's student at the Technical University of Berlin. As a research assistant at Fraunhofer IPK, he leverages applied statistics in various research projects. His research interests include software development, co-simulation, artificial intelligence, and applied statistics.

Johan Cederbladh

Johan has a master's degree in dependable aeronautical systems engineering from Mälardalen University, Sweden. He is currently a Ph.D. student at Mälardalen University, where his current research is focused on early stages of MBSE development. His research is applied at Volvo Construction Equipment where he is involved in several pilot projects for MBSE. His research interests include System Architecture, Co-simulation, DevOps, Early validation, etc.