USING OPEN SOURCE CODE LIBRARIES FOR ROBUST DESIGN ANALYSIS

Otto, Kevin; Wang, Jiahui; Uyan, Tekin

Aalto University

ABSTRACT

The design of systems today often involves computer simulation to assess performance and design margins. Understanding how variability erases design margin is important to assure adequacy of margins, especially in optimization efforts. In this paper, we develop a toolchain using open source code libraries in Python, and encapsulate it in Jupyter notebooks, to provide an open source, interactive uncertainty quantification and sensitivity analysis toolchain. This works generally with simulation tools, where a reference folder is created containing a script that reads an input file of parameter values and runs the simulation. With that easily created, the toolchain executes the necessary uncertainty quantification steps with replicates of that reference folder. This approach fits within a broader workflow outlined that defines the variation modes to study, maps to simulation inputs, and screens the variables for sensitivity before conducting an uncertainty quantification. An example is shown in the simulation analysis of a Stirling engine.

Keywords: Robust design, Simulation, Open source design

Contact:
Otto, Kevin
Aalto University
Department of Mechanical Engineering
Finland
kevin.otto@aalto.fi

Cite this article: Otto, K., Wang, J., Uyan, T. (2019) 'Using Open Source Code Libraries for Robust Design Analysis', in *Proceedings of the 22nd International Conference on Engineering Design (ICED19),* Delft, The Netherlands, 5-8 August 2019. DOI:10.1017/dsi.2019.179

INTRODUCTION

The design and development of systems today is becoming more complex, involving the consideration of many dimensions and features that all interact and contribute to providing the overall system performance. In concert with this increased complexity, the ability to diagnose when problems occur has become more difficult. For example, Tan *et al.* (2017) report on recent examples of large programs that had experienced unexpected cost overruns and delays, and found unexpected redesign effort due to design margin loss on critical requirements as contributors. Detailed inputs when varying interact in unforseen ways reducing constraint margins, causing extensive debugging and lost productivity. The aim of this paper is to provide a accessible means for engineers to study variability and design margins, to make use of and encode available algorithms and code libraries on uncertainty analysis into readily adopted scripts for a design engineer. In this paper, we present a workflow procedure implemented in the open source scripting language Python to quantify the design margin capability against uncertainties using computer simulation tools and Quasi-Monte-Carlo simulation. We adopt previous research methods and combine them into a scripted process to define, initialize, compute and report the uncertainty in a system and its causal sources.

Complex system design often involves assuring a hierarchy of system, subsystem and component operational constraints such as temperature, pressure and loading limit constraints while supplying necessary system performance. Typically design engineers meet constraints using *margins*, an allowance between the failure limit to be avoided and the limit of operation over the population of units built. Margins are necessary to cover the range of uncertainty, including the variability in as-used conditions and variability in as-manufactured units.

The degree of design margin needed is typically based on past experiences, where for example insufficient margins on past designs have caused failures and so increased. With the expanding adoption of design optimization, the ability to get design margins right is becoming more critical, as the optimization analysis can often drive the design solution right up to the design margin constraint limit. Therefore, estimation of the design margin limits becomes more critical.

The design margin limit often is made difficult given uncertainty tolerances on inputs. While design optimization studies generate high performance designs, it also selects the nominal configuration with active constraints. This often leaves open that as-built units, which can be anywhere in the tolerance domain, may also exceed the design margin boundary slightly. Some of the design margin is consumed by as-built manufacturing variations. Engineers need the means to quantify and estimate the level of design margin used up by manufacturing variability.

To improve the estimate of necessary design margin, the uncertainties can be quantified and studied for impact on design margins. Given a simulation or parametric model of the design constraints, it can be used to compute the uncertainty in the design margin and associate risk of failure. Input variations are represented as probability distributions, and uncertainty quantification methods applied to compute the design margin distribution.

While apparently straightforward for implementation, in practice barriers remain to adoption of the previous research which has developed these techniques. Uncertainty quantification methods remain difficult for many firms to adopt and limited (Wallace, 2011, Arvidsson *et al*., 2010). The aim of this paper to provide workflows of manual and computational tasks for engineers to execute the analysis as readily adopted open source scripted procedures. This includes systematically studying a design for necessary input variations to modeling, mapping these variations to simulations and variables, screening the many variables, and quantifying the contribution of the several sensitive inputs. In this paper, we present a workflow procedure implemented in the open source scripting language Python to quantify the design margin capability against uncertainties using computer simulation tools and Quasi-Monte-Carlo simulation, and to clarify the largest input contributors.

Many researchers have explored robust design and uncertainty quantification in design research. Chen *et al*. (2006) pioneered analysis methods to compute robust designs. Fang *et al*. (2005) have described various sampling methods and results with examples from the automotive industry. Jin *et al*. (2001) discuss alternative surrogate models for simulation in design. These previous research efforts demonstrated the necessary algorithms and methods to implement robust design, we adopt the methods here and make it available using open source code libraries within a scripted workflow.

Rikard *et al*. (2006) have developed tools and methods to consider uncertainties in compliant assemblies, and also has studied geometric variation management and impact on performance

https://doi.org/10.1017/dsi.2019.179 Published online by Cambridge University Press

(Forslund *et al*., 2018). Howard *et al*. (2017) have studied variation management, mapping variability from processes through to system variability. Freund *et al*. (2017) offer guidelines for improving robustness. These work are related to that here as studies in robust design of related mechanical systems.

1 MODEL BASED DESIGN CAPABILITY ANALYSIS METHODS

To analyse a design for margin loss through uncertainty, uncertainty quantification is needed (Eifler *et al*., 2013, Arvent *et al*., 2013). These are computational methods to sample an input space of variables and use computer simulation tools to compute response values and fit a distribution to the results. In this way, the risk of a constraint failure limit margin can be computed as the probability beyond the failure limit.

The uncertainty is typically viewed as a histogram of simulation output data points computed at input variable values sampled over an input space. Among others, common methods for sampling include traditional factorial sampling, random sampling, Latin hypercube sampling, and low discrepancy sequence sampling such as Hammersly, Sobol or Halton sequences (Garud *et al*., 2017).

To analyze for contributions, global sensitivity analysis methods computing Sobol indices are needed. Methods for computing Sobol indices include FAST or Saltelli's method which builds off a base sampling approach such as Sobol sampling (Saltelli *et al*., 2008). Unfortunately for problems of typical size of over 10 variables, thousands of samples are needed for reasonably small confidence intervals on the results. Therefore the computational effort needed for these becomes prohibitive without some form of screening of variables.

One approach for screening variables for contribution is Morris' method (Saltelli 2008). Here a set of one-factor-at-a-time variable changes are made as path sequences. The results are integrated to provide a rank ordering of variables on both their main effect and on their nonlinear effects (higher order or interaction effects). While not quantifying the uncertainty contribution, the Morris method does indicate which variables are more important than others. This can be used to screen a large set of variables down to many which ought be investigated with Sobol indices calculations.

Another approach for reducing the computational burden is using surrogate models. Methods include ordinary least squares regression, radial basis function interpolation, Kriging or Gaussian process modeling in general, and support vector regression, among many others. Qian *et al*. (2005) discuss such surrogate model use in mechanical design.

These works describe the methods and tools needed for efficient uncertainty quantification and sensitivity analysis in robust design. We next demonstrate a toolchain of these algorithms using open source libraries and tools, made explicit in an easily used robust design workflow implemented in Python and Jupyter notebooks. The approach makes these methods accessible and easily implemented.

2 CAPABILITY ANALYSIS WORKFLOW AND TOOLCHAIN

A *workflow* is an orchestrated repeatable sequence of tasks or procedures to be execute to solve a problem. We are here concerned with a workflow for robust design, to define, initialize and solve for the uncertainty in a system output and explain it in terms of system input variations. This can be repeated for different design concepts. Robust design has both manual and computational tasks. Manual tasks include defining what inputs ought be considered for study. Computational tasks include quantifying the uncertainty or decomposing it with sensitivity analysis. When executing a workflow, standard input and output artifacts are generated at each step. When the output of a computational task is the input to the next computational tasks, the chain of computational tasks can be scripted into a *toolchain*.

Typically toolchains are defined as a sequence of commands in scripted languages such as Matlab, Python, or shell scripts. Here the toolchain is developed in Python, an open source complete programming language. With limitations, the toolchain could have been developed in other languages. Python is useful for several reasons. First, it is inherently a complete programming language, whereas others are perhaps not. It is extensive with many developed code libraries. It is also modern, most packages are complete with features such as automated testing and peer-reviewed documentation. Finally, as open source it is constantly improved and debugged by the community.

https://doi.org/10.1017/dsi.2019.179 Published online by Cambridge University Press

2.1 Python toolchain

The tool chain is a scripted set of procedures that implement the calculations within robust design workflow. The Python toolchain is built within the open source Jupyter Notebook environment (Kluyver *et al*., 2016). This environment is a system of interactive webpages, a webserver, and a computational core that executes commands entered in the webpages. The computation core can be one of several languages such as Python, R, C#, or otherwise; here we work with Python.

Jupyter notebook pages are edited by the user and consist of blocks, where each block is either a descriptive Markdown (html) documentation block or an operative Python code block. Workflows are easily implemented in Jupyter notebooks as a sequence of code blocks that can be interactively modified and executed, block by block. Operative code blocks are programmed in Python. Typically, each Python code block is separated by a Markdown description block, which offers highly descriptive html documentation comments and instructions for the next code block.

The open source Python coding language offers many relevant packages for uncertainty quantification, including statistical analysis and graphics, design of experiments, sampling procedures, surrogate modeling, and global sensitivity analysis. Many Python packages are used in this toolchain.

For uncertainty quantification sampling methods, several packages are used in the toolchain depending on user selections. The Python package pyDOE (Baudin 2015) provides factorial and Latin hypercube sampling. The package sobol_seq (Lawless 2015) offers low discrepancy Sobol sequence samples, and the package ghalton (De Rainville, 2017) offers low discrepancy Halton sequence samples.

For surrogate modeling, the Python package scikit-learn (Pedregosa *et al*., 2011) is used. This package offers many surrogate modeling alternatives including ordinary least squares regression, radial basis functions, support vector regression, Kriging, Gaussian process models, PCE models, and many others.

For sensitivity analysis, the Python package SALib (Herman *et al*., 2017) is used. This package offers multiple sensitivity analysis methods, including FAST, Saltelli sampling, and fractional sampling methods.

2.2 Robust design workflow

While these codes and methods are available as code libraries, they are not convenient for a system design engineer to apply. An experienced design engineer often would be a novice at these packages and would not know where to start. We here implement these codes and necessary activities into a robust design workflow diagrammed in Figure 1. This is a combination of designer assessment tasks and computational tasks. The workflow starts by considering a system with a variation problem, and first clarifies the issues to be modelled. This is mapped to available simulations, and then variations defined for uncertainty propagation with the simulation model.

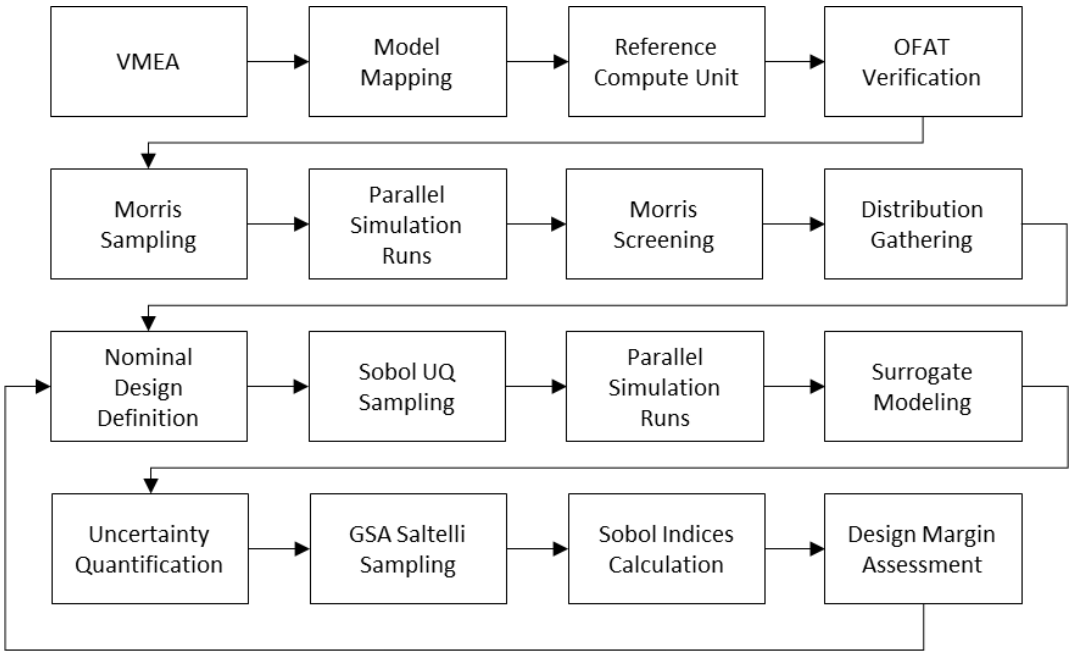

Figure 1: Model based robust design workflow.

https://doi.org/10.1017/dsi.2019.179 Published online by Cambridge University Press

There are several steps, however, that become necessary in this workflow to make the calculations practical. These include analyses to detect convergence issues in the black-box simulation, analyses to screen the variables to a smaller set for uncertainty quantification, and surrogate modeling to make the sensitivity analyses computationally practical. We detail each step of the workflow next.

The first step of the workflow is for the important variation modes to be identified. Rather than simply consider a simulation model and varying its inputs, a design engineer ought examine the system and make clear the necessary inputs to consider. Using the *Variation Modes and Effects Analysis* (VMEA) methodology (Johansson *et al.*, 2006), a system is analyzed for causal sources of variations in each component of the system, the *variation modes*. This is a manual task, often executed as a workshop of engineers and results in a list of hundreds of possible input variation modes for a typical system.

Next in step 2 of Figure 1, given the list of possible input variation mode causes and available models, the variation modes identified are mapped to model variables. Typically any available model does not necessarily represent all variation modes, some sources of variation are of too fine a geometric or dynamic resolution compared with a simulation model that may happen to be available. As such, the modeling choices are interrogated here over scale, amongst more rapid lumped parameter models versus finer resolution using slower computing models. The result, however, is a choice of multiscale models used and their inter-dependencies as a computational flow. For example, a dynamic simulation model of a device can generate force loadings passed on to a finite element stress model of one component at several conditions which can generate stress inputs which are in turn passed on to a cycle stress-strain material model to predict component life. Linking the original causal inputs to the ultimate design margin constraint in this way using perhaps several inter-linked models is what we will call a *simulation compute unit*.

To easily construct this, we create a *reference folder* on the computer within which the simulation will be executed, Step 3 in Figure 1. Within this folder, we create a file input.csv which is a simple two column table of input variable names and values. We then define a batch file (shell script) in the folder that is executed. The batch file (shell script) calls an executable file that reads the input.csv file, populates the simulation code with the values for the variables, and generates the outputs. These are written to a file output.csv which again is a simple two column table of output variable names and values. This reference folder is created by the user, and contains all codes and files needed for a batch file (shell script) to generate the particular output file from the input file. The simulation compute unit is thereby encapsulated within a reference folder on the computer, as shown in Figure 2. This folder will be copied and the input file modified for each new input sample used in the uncertainty quantification. With the reference folder and compute unit so constructed, initially it is known to generate validated predictions at the nominal design.

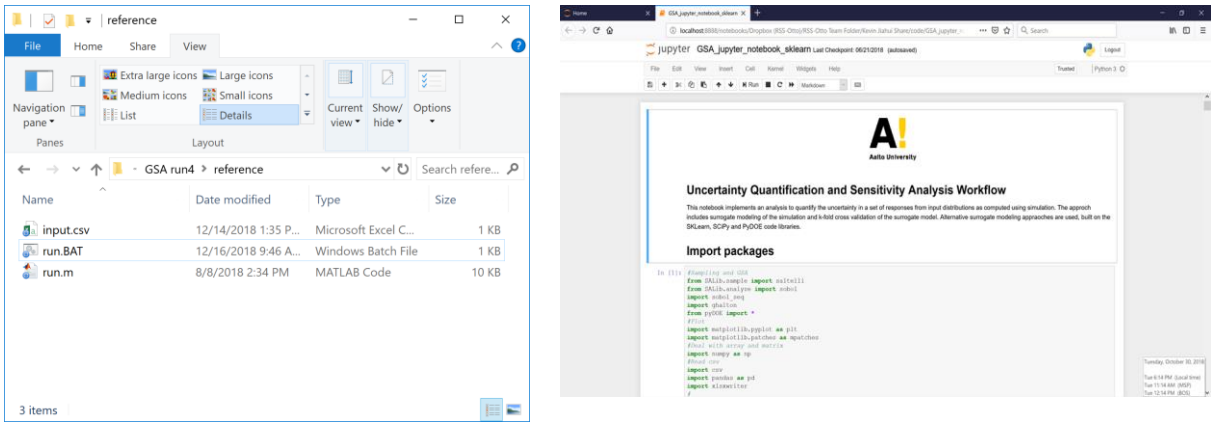

Figure 2: Reference folder for a simulation and the Jupyter Notebook.

Given the simulation models are constructed as a compute unit and it generated validated predictions at the nominal design, there remains no assurance the compute unit will converge and generate a value at input values other than the nominal configuration. It can often be the case that changing an input by any amount will cause the simulation to fail to converge. This is a problem with most probabilistic sampling strategies

https://doi.org/10.1017/dsi.2019.179 Published online by Cambridge University Press

which vary all inputs simultaneously, and would result in all samples failing. To check for this, we first conduct a very simple one-factor-at-a-time change in input values, to verify the simulation, Step 4 in Figure 1. In this step, the reference folder is copied N times for each of the N input variables, and the input.csv file contains the nominal inputs except for the i^{th} variable which is increased by a small amount such as 1%. Executing these N simulations and ensuring all produce viable results indicates the simulation can be used for simultaneously varying all N inputs.

The next step is to screen the many variables that might possible affect the response down to those which have non-negligible sensitivity. The contribution of any variable is conceptually it's input distribution multiplied by it sensitivity. If the variable's sensitivity is negligible, then it is not worth the effort to gather data on its input variability. To compute a rank order of the variables on sensitivity, the Morris method can be applied (Saltelli *et al*., 2008). Here each input is varied over a range of 1%, and a set of Morris samples are generated, Step 5 in Figure 1. Here, these samples are generated by the Python toolchain, and a set of replicated reference folders generated.

These folders are each executed by the toolchain, as a set of parallel runs, Step 6 in Figure 1. The results in the output.csv files of these runs are then collected and analyzed into a Morris contribution analysis, Step 7 in Figure 1. The result is a computed filtered rank order of the inputs into those which significantly contribute sensitivity and those which do not.

The next step is to gather input distribution data on the screened input variables, Step 8 in Figure 1. This is a manual laborious task of gathering and estimating C_{pk} data to inform the input distributions of the uncertainty quantification. Typically, this means measuring the standard deviation of characteristics of a sample from production parts similar to the new design.

With the variability quantified, these can be used as variations around the nominal design, Step 9 in Figure 1. Then a sample of points in this input uncertainty space are created, Step 10. Sampling methods are typically Sobol or Latin hypercube samples. These samples are generated by the Python toolchain using sampling code libraries as discussed in the previous section, and a set of replicated reference folders generated. The samples are distributed according to the input distributions. These folders are each executed by the toolchain, as a set of parallel runs, Step 11.

These output sample points are then collected from the output.csv files in the folders and shown as a histogram, and a distribution fit to the data through maximum likelihood in Step 13. This completes one view of the uncertainty quantification, directly using the simulation results. Confidence intervals on the statistics provide an indication of adequate sample size in Step 10.

With the input and output samples, next in step 12 of the workflow a surrogate model is computed for use with the sensitivity analysis, which typically needs many more sample points than are realistic with direct simulation. The surrogate model is created by the Python toolchain using the Scikit-learn code library. A correlation analysis and confidence interval on the fit is generated using bootstrapping, and so the goodness of fit and adequacy of the sample size in Step 10 can be assessed.

With a surrogate model, the intensive sampling needed for global sensitivity analysis can be executed to compute Sobol indices, to determine how much each input variable conributes to the output uncertainty distribution. These are computed in Steps 14-15 of the workflow using the SALib code library which creates a large sample set where at each sample the surrogate model is computed. The result is the first, second and total order contribution of each variable, and for each a confidence interval computed using bootstrapping. If the confidence interval is too large, a larger sample size can be used.

These steps complete one cycle through the toolchain. As a last step in the workflow, the computed uncertainty distribution can be compared against the design margin considering the large input contributors. If the output distribution uncertainty is too large compared to the design margin, alternative design configurations can be considered and run through the toolchain to compute if the new design has higher capability.

2.3 Example: Stirling engine robust design

While we have used the workflow in industrial use cases including industrial electric motors, aircraft brake systems, air management systems and commercial air conditioning equipment, an issue is that industrial use cases are proprietary. We here consider a systems design problem with complete open data available for all to study, a Stirling engine project created within the university. The high thermal efficiency and the ability to operate on non-traditional heat sources has kept Stirling engines of research interest and under constant optimization (Çınar *et al*., 2018). It will be used to demonstrate

https://doi.org/10.1017/dsi.2019.179 Published online by Cambridge University Press

the analysis of performance uncertainty due to variability on input manufacturing sources. This provides an example of the intended workflow, to define and study the performance uncertainty of a system design.

A miniature Stirling engine was used in a university machine design project course. The tight dimensional and geometrical tolerance requirements of a Stirling engine make it an ideal system to develop machining skills. Students were asked to machine selected parts of a Stirling engine kit, and the remainder of the parts provided. A total of 85 students working in teams of 5 produced 17 engines. Each engine was assembled and tested, measuring the operational no-load speed. The no-load speed varied between 200 rpm and 600 rpm, depending on the variability of the manufactured parts and assembly process. This activity indicated the Stirling engine is sensitive to manufacturing and assembly tolerances. However it was not clear which component features or combination of tolerances were the most critical in the Stirling engine.

Separately, Ureili (2010) created a simulation model of the Stirling engine and made it available as a Matlab simulation code. Given geometry and temperatures, a Schmidt analysis of the Stirling engine cycle is calculated which computes the thermodynamic energy output per cycle and thereby the thermodynamic power produced. Given the variability of no load speed observed in the engine sample, an uncertainty quantification and sensitivity analysis of the thermodynamic power predicted from the Schmidt analysis model could indicate which inputs are most sensitive and need most care when fabricating.

To study this problem, the toolchain was applied to study the performance uncertainty and to determine contributing sources, as a demonstrator. The goal was to find the critical parameters affecting to the power output performance of the engine. Graduate student teaching research assistants applied and executed this workflow, in support of their course teaching, to gain understanding of which inputs were critical.

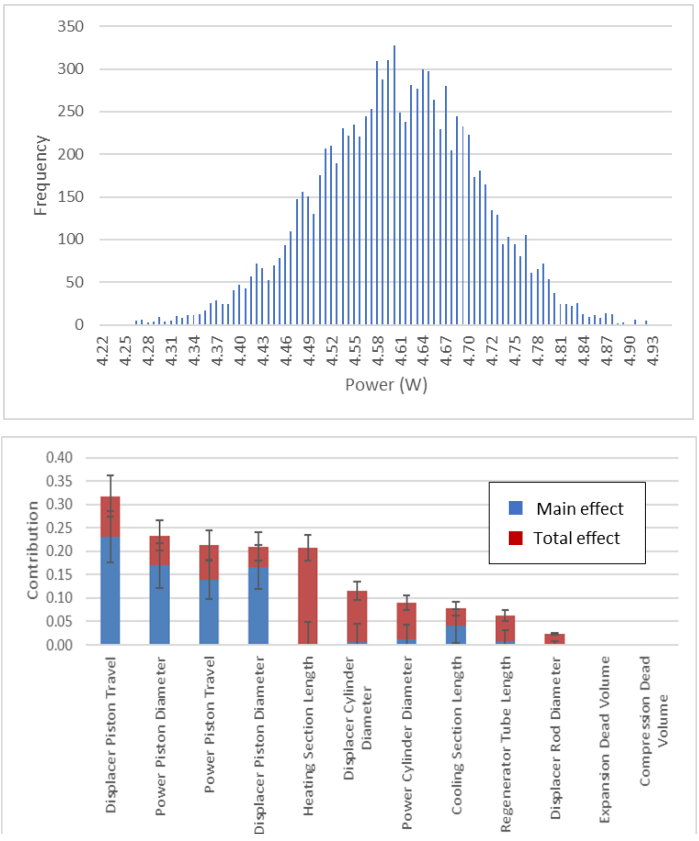

Figure 3: Stirling engine uncertainty quantification and sensitivity analysis results.

The Stirling engine simulation was written by Ureili (2010) in Matlab for execution. A reference folder was created with a file input.csv which had 28 input variables including part diameters and lengths in one column and values to use for them in the second column. Within this folder a Matlab command script was created which read the input file, ran the Stirling engine simulation code with

https://doi.org/10.1017/dsi.2019.179 Published online by Cambridge University Press

these input values, and then wrote the output power value to a newly created file output.csv, which had one row of data with the output power name and value. Then a BAT batch file was created that launched Matlab to run the command script, with all errors directed to a newly created error log file in the folder. This reference folder of the input.csv file, Matlab script and batch file will then be copied for every execution of the simulation.
For the Stirling engine, the VMEA analysis identified 42 sources of variation that might affect the no-load speed. Of these, 12 input variables were prioritized and considered for their impact on computed engine power generated. The results of the uncertainty assessment and contributors are shown in Figure 3. The computed engine power uncertainty distribution range was ±6%, a large range of performance variability. The sensitivity analysis was then executed, and the results show the uncertainty was primarily due to the contribution of the piston displacements and diameters (Figure 3). This was consistent with the experimental results in the course, where engines with tight power piston fits were slower, and engines with larger displacer volumes were faster.

3 CONCLUSIONS

A workflow and toolchain for computing uncertainty and sensitivity using simulation is necessary for enabling engineers to compute robust design calculations. This proves useful for complex system design problems where a performance response has excess variability due to poorly understood input manufacturing variations. Yet, running Monte-Carlo simulation studies of an available simulation model inputs alone offers no assurance of solving the variability problem. An entire workflow from identifying variation modes to model setup to actual sensitivity analysis is all necessary.
A Python toolchain making use of open source code libraries can offer rapid setup and execution, as embodied in a self-documenting Jupyter notebook environment. This is freely available as an open source tool. The structure of the toolchain makes it easier for engineers to make use, given the construction of a simulation with input and output files in a reference folders.
Wallace (2011) noted that transfer of design research to industrial practice is difficult and often due to lack of a champion. Design researchers pursue a research objective and demonstrate a novel result whereas engineers in industry pursue the launch of a new product. These are disconnected. While causes for the disconnection are many, high complexity and perceived irrelevance are contributors (Wallace 2011). The work here is to create a self-documented toolchain using easily available open source code libraries, to thereby make robust design accessible. We have used this toolchain in several industrial contexts started by simply providing the toolchain, from large electric motors to large commercial air conditioning systems. We find that to insert a new methodology such as robust design into a company, a workflow such as this is likely a necessary starting point.
To demonstrate the robust design workflow, we provide an open data example from the university context, the design of a Stirling engine. We use the workflow and toolchain to demonstrate the intent of the workflow, to analyze a complex system for the input causal sources of performance variability. The graduate student teaching assistants readily executed an analysis for uncertainty using the workflow and came to quick conclusions of what to inspect and assure for a high performing engine.

REFERENCES

Arendt, P.D., Apley, D.W. and Chen, W. (2013), "Objective-oriented sequential sampling for simulation based robust design considering multiple sources of uncertainty", *Journal of Mechanical Design*, Vol. 135 No. 5, p. 051005.

Arvidsson, M., Gremyr, I. and Johansson, P. (2003), "Use and knowledge of robust design methodology: a survey of Swedish industry", *Journal of engineering design*, Vol. 14 No. 2, pp. 129–143.

Baudin, M. 2015. pyDOE. https://github.com/tisimst/pyDOE

Chen, W., Jin, R. and Sudjianto, A. (2006), "Analytical global sensitivity analysis and uncertainty propagation for robust design", *Journal of quality technology*, Vol. 38 No. 4, pp. 333–348.

Çınar, C., Aksoy, F., Solmaz, H., Yılmaz, E. and Uyumaz, A. (2018), "Manufacturing and testing of an α-type Stirling engine", *Applied Thermal Engineering*, Vol. 130, pp. 1373–1379.

Eifler, T., Ebro, M. and Howard, T.J. (2013). "A classification of the industrial relevance of robust design methods", In *Proceedings of the 19th International Conference on Engineering Design*, Vol. 9: Design Methods and Tools, Seoul, pp. 427–436.

Fang, K.T., Li, R. and Sudjianto, A. (2005), *Design and modeling for computer experiments*, Chapman and Hall/CRC.

https://doi.org/10.1017/dsi.2019.179 Published online by Cambridge University Press

De Rainville, F., (2017), ghalton 0.6.1, A Generalized Halton Number Generator. pypi.org/project/ghalton/
Forslund, A., Madrid, J., Söderberg, R., Isaksson, O., Lööf, O. and Frey, D. (2018), "Evaluating How Functional Performance in Aerospace Components Is Affected by Geometric Variation", *SAE International Journal of Aerospace*, 01-11-01-0001.
Freund, T., Würtenberger, J., Lotz, J., Rommel, C. and Kirchner, E. (2017), "Design for robustness-Systematic application of design guidelines to control uncertainty", In *Proceedings of the 21st International Conference on Engineering Design*, Vancouver, Canada, pp. 277–286.
Garud, S.S., Karimi, I.A. and Kraft, M. (2017), "Design of computer experiments: A review", *Computers & Chemical Engineering*, Vol. 106, pp. 71–95.
Herman, J. and Usher, W. (2017), "SALib: an open-source Python library for sensitivity analysis", *The Journal of Open Source Software*, Vol. 2 No. 9.
Howard, T., Eifler, T., Pedersen, S., Göhler, S., Boorla, S. and Christensen, M. (2017), "The variation management framework (VMF): A unifying graphical representation of robust design", *Quality Engineering*, Vol. 29 No. 4, pp. 563–572.
Jin, R., Chen, W. and Simpson, T. (2001), "Comparative studies of metamodelling techniques under multiple modelling criteria", *Structural and multidisciplinary optimization*, Vol. 23 No. 1, pp. 1–13.
Johansson, P., Chakhunashvili, A., Barone, S. and Bergman, B. (2006), "Variation mode and effect analysis: a practical tool for quality improvement", *Quality and reliability engineering international*, Vol. 22 No. 8, pp. 865–876.
Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B.E., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J.B., Grout, J., Corlay, S. and Ivanov, P. (2016), May "Jupyter Notebooks-a publishing format for reproducible computational workflows", *ELPUB*, pp. 87–90.
Lawless, C. (2015), sobol_seq 0.1.2, pypi.org/project/sobol_seq/.
Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J. (2011), "Scikit-learn: Machine learning in Python", *Journal of machine learning research*, Vol. 12 No. Oct, pp. 2825–2830.
Qian, Z., Seepersad, C.C., Joseph, V.R., Allen, J.K. and Wu, C.J. (2006), "Building surrogate models based on detailed and approximate simulations", *Journal of Mechanical Design*, Vol. 128 No. 4, pp. 668–677.
Söderberg, R., Lindkvist, L. and Dahlström, S. (2006), "Computer-aided robustness analysis for compliant assemblies", *Journal of Engineering Design*, Vol. 17 No. 5, pp. 411–428.
Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M. and Tarantola, S. (2008), *Global sensitivity analysis: the primer*, John Wiley & Sons.
Tan, J., Otto, K. and Wood, K. (2017), "Relative Impact of Early versus Late Design Decisions in Systems Development", *Design Science Journal*.
Ureili, I. (2010), *Stirling Cycle Machine Analysis*, www.ohio.edu/mechanical/stirling/
Wallace, K. (2011), "Transferring design methods into practice", In *The future of design methodology*, Springer, London, pp. 239–248.

ACKNOWLEDGMENTS

This work was made possible with support from an Academy of Finland, Project Number 310252.

https://doi.org/10.1017/dsi.2019.179 Published online by Cambridge University Press