Grama: A Grammar of Model Analysis

Zachary del Rosario

1 Visiting Professor, Olin College of Engineering

Summary

Grama is a Python package implementing a functional grammar of model analysis emphasizing the quantification of uncertainties. In Grama a model contains both a function mapping inputs to outputs as well as a distribution characterizing uncertainties on those inputs. This conceptual object unifies the engineer/scientist’s definition of a model with that of a statistician. Grama provides an implementation of this model concept, as well as verbs to carry out model-building and model-analysis.

Statement of Need

Uncertainty Quantification (UQ) is the science of analyzing uncertainty in scientific problems and using those results to inform decisions. UQ has important applications to building safety-critical engineering systems, and to making high-consequence choices based on scientific models. However, UQ is generally not taught at the undergraduate level: Many engineers leave their undergraduate training with a purely deterministic view of their discipline, which can lead to probabilistic design errors that negatively impact safety (del Rosario, Fenrich, & Iaccarino, 2020). To that end, I have developed a grammar of model analysis—Grama—to facilitate rapid model analysis, communication of results, and the teaching of concepts, all with quantified uncertainties. Intended users of Grama are scientists and engineers at the undergraduate level and upward, seeking to analyze computationally-lightweight models.

Differentiating Attributes

Packages similar to Grama exist, most notably Sandia National Lab’s Dakota (Adams, 2017) and UQLab (Marelli & Sudret, 2014) out of ETH Zurich. While both of these packages are mature and highly featured, Grama has several differentiating attributes. First, Grama emphasizes an explicit but flexible model object: this object enables sharp decomposition of a UQ problem into a model-building stage and a model-analysis stage. This logical decomposition enables simplified syntax and a significant reduction in boilerplate code. Second, Grama implements a functional programming syntax to emphasize operations against the model object, improving readability of code. Finally, Grama is designed from the ground-up as a pedagogical and communication tool. For learnability: Its verb-prefix syntax is meant to remind the user how functions are used based solely on their name, and the package is shipped with fill-in-the-blank Jupyter notebooks (Kluyver et al., 2016) to take advantage of the pedagogical benefits of active learning (Freeman et al., 2014). For communication: The model object and functional syntax abstract away numerical details for presentation in a notebook, while preserving tracability and reproducibility of results through the inspection of source code.
Inspiration and Dependencies

Grama relies heavily on the SciKit package ecosystem for its numerical backbone (Hunter, 2007; McKinney, 2010; Pedregosa et al., 2011; van der Walt, Colbert, & Varoquaux, 2011; Virtanen et al., 2020). The functional design is heavily inspired by the Tidyverse (Wickham et al., 2019), while its implementation is built upon dfply (Katovich, 2019). Additional functionality for materials data via an optional dependency on Matminer (Ward et al., 2018).

Acknowledgements

I acknowledge contributions from Richard W. Fenrich on the laminate plate model.

References


Welcome to the tidyverse. *Journal of Open Source Software*, 4(43), 1686. doi:10.21105/joss.01686