Enabling Active Learning Pedagogy and Insight Mining with a Grammar of Model Analysis

Zachary del Rosario[‡][*]

Abstract—Modern engineering models are complex, with dozens of inputs, uncertainties arising from simplifying assumptions, and dense output data. While major strides have been made in the computational scalability of complex models, relatively less attention has been paid to user-friendly, reusable tools to explore and make sense of these models. Grama is a python package aimed at supporting these activities. Grama is a grammar of model analysis: an ontology that specifies data (in tidy form), models (with quantified uncertainties), and the verbs that connect these objects. This definition enables a reusable set of evaluation "verbs" that provide a consistent analysis toolkit across different grama models. This paper presents three case studies that illustrate pedagogy and engineering work with grama: 1. Providing teachable moments through errors for learners, 2. Providing reusable tools to help users self-initiate productive modeling behaviors, and 3. Enabling *exploratory model analysis* (EMA) – exploratory data analysis augmented with data generation.

Index Terms—engineering, engineering education, exploratory model analysis, software design, uncertainty quantification

Introduction

Modern engineering relies on scientific computing. Computational advances enable faster analysis and design cycles by reducing the need for physical experiments. For instance, finite-element analysis enables computational study of aerodynamic flutter, and Reynolds-averaged Navier-Stokes simulation supports the simulation of jet engines. Both of these are enabling technologies that support the design of modern aircraft [KN05]. Modern areas of computational research include heterogeneous computing environments [MV15], task-based parallelism [BTSA12], and big data [SS13]. Another line of work considers the development of *integrated tools* to unite diverse disciplinary perspectives in a single, unified environment (e.g., the integration of multiple physical phenomena in a single code [EVB+20] or the integration of a computational solver and data analysis tools [MTW+22]). Such integrated computational frameworks are highlighted as *essential* for applications such as computational analysis and design of aircraft [SKA+14]. While engineering computation has advanced along the aforementioned axes, the conceptual understanding of practicing engineers has lagged in key areas.

Every aircraft you have ever flown on has been designed using probabilistically-flawed, potentially dangerous criteria [dRFI21]. The fundamental issue underlying these criteria is a flawed heuristic for uncertainty propagation; initial human subjects work suggests that engineers' tendency to misdiagnose sources of variability as inconsequential noise may contribute to the persistent application of flawed design criteria [AFD+21]. These flawed treatments of uncertainty are not limited to engineering design; recent work by Kahneman et al. [KSS21] highlights widespread failures to recognize or address variability in human judgment, leading to bias in hiring, economic loss, and an unacceptably capricious application of justice.

Grama was originally developed to support model analysis under uncertainty; in particular, to enable *active learning* [FEM+14] – a form of teaching characterized by active student engagement shown to be superior to lecture alone. This toolkit aims to *integrate* the disciplinary perspectives of computational engineering and statistical analysis within a unified environment to support a *coding to learn* pedagogy [Bar16] – a teaching philosophy that uses code to teach a discipline, rather than as a means to teach computer science or coding itself. The design of grama is heavily inspired by the Tidyverse [WAB+19], an integrated set of R packages organized around the 'tidy data' concept [Wic14]. Grama uses the tidy data concept and introduces an analogous concepts for *models*.

Grama: A Grammar of Model Analysis

Grama [dR20] is an integrated set of tools for working with *data* and *models*. Pandas [pdt20], [WM10] is used as the underlying data class, while grama implements a `Model` class. A grama model includes a number of functions – mathematical expressions or simulations – and domain/distribution information for the deterministic/random inputs. The following code illustrates a simple grama model with both deterministic and random inputs[1].

```
# Each cp_* function adds information to the model
md_example = (
    gr.Model("An example model")
    # Overloaded `>>` provides pipe syntax
    >> gr.cp_vec_function(
        fun=lambda df: gr.df_make(f=df.x+df.y+df.z),
        var=["x", "y", "z"],
        out=["f"],
    )
    >> gr.cp_bounds(x=(-1, +1))
    >> gr.cp_marginals(
        y=gr.marg_mom("norm", mean=0, sd=1),
        z=gr.marg_mom("uniform", mean=0, sd=1),
    )
```

* Corresponding author: zdelrosario@olin.edu
‡ Assistant Professor of Engineering and Applied Statistics, Olin College of Engineering

1. Throughout, `import grama as gr` is assumed.

```
    >> gr.cp_copula_gaussian(
        df_corr=gr.df_make(
            var1="y",
            var2="z",
            corr=0.5,
        )
    )
)
```

While an engineer's interpretation of the term "model" focuses on the input-to-output mapping (the simulation), and a statistician's interpretation of the term "model" focuses on a distribution, the grama model integrates both perspectives in a single model.

Grama models are intended to be *evaluated* to generate data. The data can then be analyzed using visual and statistical means. Models can be *composed* to add more information, or *fit* to a dataset. Figure 1 illustrates this interplay between data and models in terms of the four categories of function "verbs" provided in grama.

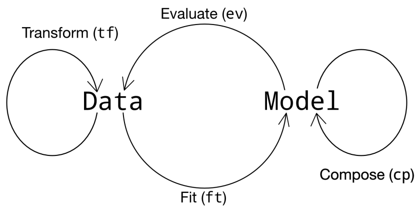

Fig. 1: Verb categories in grama. These grama functions start with an identifying prefix, e.g. `ev_*` *for evaluation verbs.*

Defaults for Concise Code

Grama verbs are designed with sensible default arguments to enable concise code. For instance, the following code visualizes input sweeps across its three inputs, similar to a *ceteris paribus* profile [KBB19], [Bie20].

```
(
    ## Concise default analysis
    md_example
    >> gr.ev_sinews(df_det="swp")
    >> gr.pt_auto()
)
```

This code uses the default number of sweeps and sweep density, and constructs a visualization of the results. The resulting plot is shown in Figure 2.

Grama imports the plotnine package for data visualization [HK21], both to provide an expressive grammar of graphics, but also to implement a variety of "autoplot" routines. These are called via a dispatcher `gr.pt_auto()` which uses metadata from evaluation verbs to construct a default visual. Combined with sensible defaults for keyword arguments, these tools provide a concise syntax even for sophisticated analyses. The same code can be slightly modified to change a default argument value, or to use plotnine to create a more tailored visual.

```
(
    md_example
    ## Override default parameters
    >> gr.ev_sinews(df_det="swp", n_sweeps=10)
    >> gr.pt_auto()
)

(
    md_example
    >> gr.ev_sinews(df_det="swp")
    ## Construct a targeted plot
```

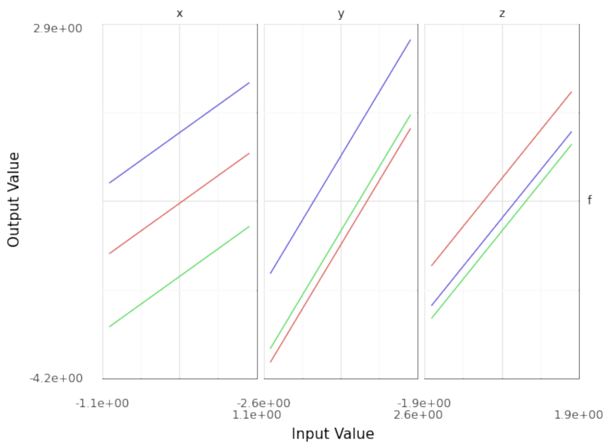

Fig. 2: Input sweep generated from the code above. Each panel visualizes the effect of changing a single input, with all other inputs held constant.

```
    >> gr.tf_filter(DF.sweep_var == "x")
    >> gr.ggplot(gr.aes("x", "f", group="sweep_ind"))
    + gr.geom_line()
)
```

This system of defaults is important for pedagogical design: Introductory grama code can be made extremely simple when first introducing a concept. However, the defaults can be overridden to carry out sophisticated and targeted analyses. We will see in the Case Studies below how this concise syntax encourages sound analysis among students.

Pedagogy Case Studies

The following two case studies illustrate how grama is designed to support *pedagogy*: the formal method and practice of teaching. In particular, grama is designed for an active learning pedagogy [FEM+14], a style of teaching characterized by active student engagement.

Teachable Moments through Errors for Learners

An advantage of a unified modeling environment like grama is the opportunity to introduce design *errors for learners* in order to provide teachable moments.

It is common in probabilistic modeling to make problematic assumptions. For instance, Cullen and Frey [CF99] note that modelers frequently and erroneously treat the normal distribution as a default choice for all unknown quantities. Another common issue is to assume, by default, the independence of all random inputs to a model. This is often done *tacitly* – with the independence assumption unstated. These assumptions are problematic, as they can adversely impact the validity of a probabilistic analysis [dRFI21].

To highlight the dependency issue for novice modelers, grama uses error messages to provide just-in-time feedback to a user who does not articulate their modeling choices. For example, the following code builds a model with no dependency structure specified. The result is an error message that summarizes the conceptual issue and points the user to a primer on random variable modeling.

```
md_flawed = (
    gr.Model("An example model")
    >> gr.cp_vec_function(
        fun=lambda df: gr.df_make(f=df.x+df.y+df.z),
        var=["x", "y", "z"],
        out=["f"],
    )
    >> gr.cp_bounds(x=(-1, +1))
    >> gr.cp_marginals(
        y=gr.marg_mom("norm", mean=0, sd=1),
        z=gr.marg_mom("uniform", mean=0, sd=1),
    )
    ## NOTE: No dependency specified
)
(
    md_flawed
    ## This code will throw an Error
    >> gr.ev_sample(n=1000, df_det="nom")
)
```

Error `ValueError`: Present model copula must be defined for sampling. Use `CopulaIndependence` only when inputs can be guaranteed independent. See the Documentation chapter on Random Variable Modeling for more information. https://py-grama.readthedocs.io/en/latest/source/rv_modeling.html

Grama is designed both as a teaching tool and a scientific modeling toolkit. For the student, grama offers teachable moments to help the novice grow as a modeler. For the scientist, grama enforces practices that promote scientific reproducibility.

Encouraging Sound Analysis

As mentioned above, concise grama syntax is desirable to *encourage sound analysis practices*. Grama is designed to support higher-level learning outcomes [Blo56]. For instance, rather than focusing on *applying* programming constructs to generate model results, grama is intended to help users *study* model results ("evaluate," according to Bloom's Taxonomy). Sound computational analysis demands study of simulation results (e.g., to check for numerical instabilities). This case study makes this learning outcome distinction concrete by considering *parameter sweeps*.

Generating a parameter sweep similar to Figure 2 with standard Python libraries requires a considerable amount of boilerplate code, manual coordination of model information, and explicit loop construction. The following code generates parameter sweep data using standard libraries. Note that this code sweeps through values of `x` holding values of `y` fixed; additional code would be necessary to construct a sweep through `y`[2].

```
## Parameter sweep: Manual approach
# Gather model info
x_lo = -1; x_up = +1;
y_lo = -1; y_up = +1;
f_model = lambda x, y: x**2 * y
# Analysis parameters
nx = 10               # Grid resolution for x
y_const = [-1, 0, +1] # Constant values for y
# Generate data
data = np.zeros((nx * len(y_const), 3))
for i, x in enumerate(
        np.linspace(x_lo, x_up, num=nx)
    ):
    for j, y in enumerate(y_const):
        data[i + j*nx, 0] = f_model(x, y)
        data[i + j*nx, 1] = x
        data[i + j*nx, 2] = y
# Package data for visual
df_manual = pd.DataFrame(
    data=data,
    columns=["f", "x", "y"],
)
```

The ability to write low-level programming constructs – such as the loops above – is an obviously worthy learning outcome in a course on scientific computing. However, not all courses should focus on low-level programming constructs. Grama is not designed to support low-level learning outcomes; instead, the package is designed to support a "coding to learn" philosophy [Bar16] focused on higher-order learning outcomes to support sound modeling practices.

Parameter sweep functionality can be achieved in grama without explicit loop management and with sensible defaults for the analysis parameters. This provides a "quick and dirty" tool to inspect a model's behavior. A grama approach to parameter sweeps is shown below.

```
## Parameter sweep: Grama approach
# Gather model info
md_gr = (
    gr.Model()
    >> gr.cp_vec_function(
        fun=lambda df: gr.df_make(f=df.x**2 * df.y),
        var=["x", "y"],
        out=["f"],
    )
    >> gr.cp_bounds(
        x=(-1, +1),
        y=(-1, +1),
    )
)
# Generate data
df_gr = gr.eval_sinews(
    md_gr,
    df_det="swp",
    n_sweeps=3,
)
```

Once a model is implemented in grama, generating and visualizing a parameter sweep is trivial, requiring just two lines of code and zero initial choices for analysis parameters. The practical outcome of this software design is that users will tend to *self-initiate* parameter sweeps: While students will rarely choose to write the extensive boilerplate code necessary for a parameter sweep (unless required to do so), students writing code in grama will tend to self-initiate sound analysis practices.

For example, the following code is unmodified from a student report[3]. The original author implemented an ordinary differential equation model to simulate the track time `"finish_time"` of an electric formula car, and sought to study the impact of variables such as the gear ratio `"GR"` on `"finish_time"`. While the assignment did not require a parameter sweep, the student chose to carry out their own study. The code below is a self-initiated parameter sweep of the track time model.

```
## Unedited student code
md_car = (
    gr.Model("Accel Model")
    >> gr.cp_function(
        fun = calculate_finish_time,
        var = ["GR", "dt_mass", "I_net" ],
        out = ["finish_time"],
    )

    >> gr.cp_bounds(
        GR=(+1,+4),
        dt_mass=(+5,+15),
        I_net=(+.2,+.3),
```

2. Code assumes `import numpy as np; import pandas as pd`.

3. Included with permission of the author, on condition of anonymity.

```
    )
)

gr.plot_auto(
    gr.eval_sinews(
        md_car,
        df_det="swp",
        #skip=True,
        n_density=20,
        n_sweeps=5,
        seed=101,
    )
)
```

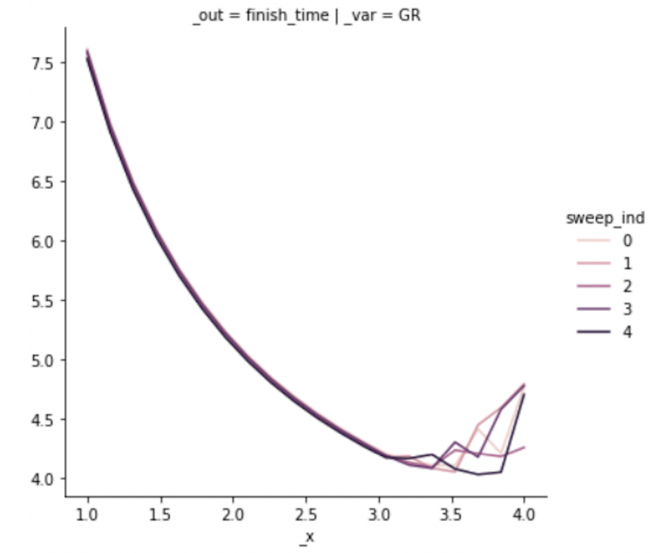

Fig. 3: *Input sweep generated from the student code above. The image has been cropped for space, and the results are generated with an older version of grama. The jagged response at higher values of the input are evidence of solver instabilities.*

The parameter sweep shown in Figure 2 gives an overall impression of the effect of input "GR" on the output "finish_time". This particular input tends to dominate the results. However, variable results at higher values of "GR" provide evidence of numerical instability in the ODE solver underlying the model. Without this sort of model evaluation, the student author would not have discovered the limitations of the model.

Exploratory Model Analysis Case Study

This final case study illustrates how grama supports exploratory model analysis. This iterative process is a computational approach to mining insights into physical systems. The following use case illustrates the approach by considering the design of boat hull cross-sections.

Static Stability of Boat Hulls

Stability is a key consideration in boat hull design. One of the most fundamental aspects of stability is *static stability*; the behavior of a boat when perturbed away from static equilibrium [LE00]. Figure 4 illustrates the physical mechanism governing stability at small perturbations from an upright orientation.

As a boat is rotated away from its upright orientation, its center of buoyancy (COB) will tend to migrate. If the boat is in vertical equilibrium, its buoyant force will be equal in magnitude to its weight. A stable boat is a hull whose COB migrates in such a way

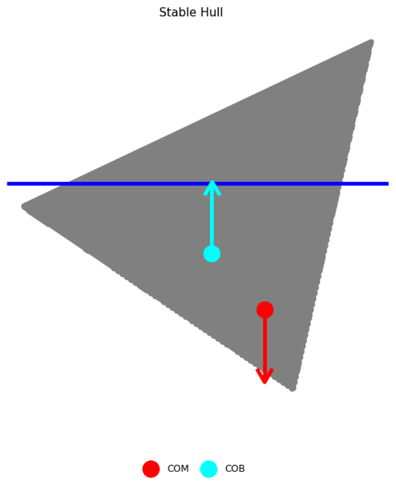

Fig. 4: *Schematic boat hull rotated to* $22.5°$. *The forces due to gravity and buoyancy act at the center of mass (COM) and center of buoyancy (COB), respectively. Note that this hull is upright stable, as the couple will rotate the boat to upright.*

that a restoring torque is generated (Fig. 4). However, this upright stability is not guaranteed; Figure 5 illustrates a boat design that does not provide a restoring torque near its upright angle. An upright-unstable boat will tend to capsize spontaneously.

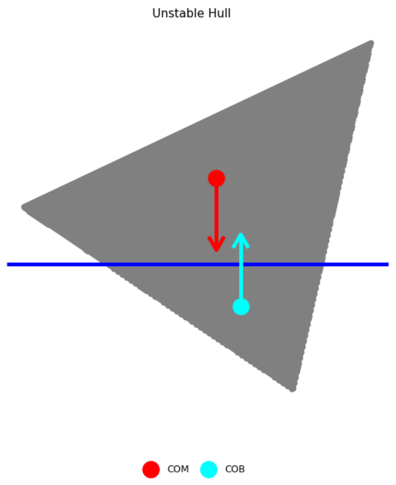

Fig. 5: *Schematic boat hull rotated to* $22.5°$. *Gravity and buoyancy are annotated as in Figure 4. Note that this hull is upright unstable, as the couple will rotate the boat away from upright.*

Naval engineers analyze the stability of a boat design by constructing a *moment curve*, such as the one pictured in Figure 6. This curve depicts the net moment due to buoyancy at various angles, assuming the vessel is in vertical equilibrium. From this figure we can see that the design is upright-stable, as it possesses a negative slope at upright $\theta = 0°$. Note that a boat may not have an unlimited range of stability as Figure 6 exhibits an *angle of vanishing stability* (AVS) beyond which the boat does not recover to upright.

The classical way to build intuition about boat stability is via mathematical derivations [LE00]. In the following section we present an alternative way to build intuition through exploratory

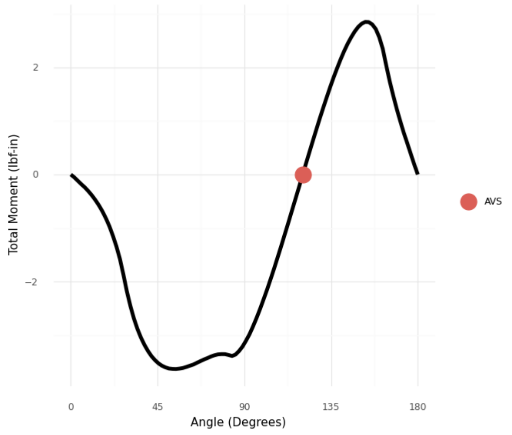

Fig. 6: *Total moment on a boat hull as it is rotated through* 180°. *A negative slope at upright* $\theta = 0°$ *is required for upright stability. Stability is lost at the* angle of vanishing stability *(AVS).*

model analysis.

EMA for Insight Mining

Generation and post-processing of the moment curve are implemented in the grama model `md_performance`[4]. This model parameterizes a 2d boat hull via its height `H`, width `W`, shape of corner `n`, the vertical height of the center of mass `f_com` (as a fraction of the height), and the *displacement ratio* `d` (the ratio of the boat's mass to maximum water mass displaced). Note that a boat with `d > 1` is incapable of flotation. A smaller value of `d` corresponds to a boat that floats higher in the water. The model `md_performance` returns `stability = -dMdtheta_0` (the negative of the moment curve slope at upright) as well as the `mass` and AVS `angle`. A positive value of `stability` indicates upright stability, while a larger value of `angle` indicates a wider range of stability.

The EMA process begins by generating data from the model. However, the generation of a moment curve is a nontrivial calculation. One should exercise care in choosing an initial sample of designs to analyze. The statistical problem of selecting efficient input values for a computer model is called the *design of computer experiments* [SSW89]. The grama verb *gr.tf_sp()* implements the support points algorithm [MJ18] to reduce a large dataset of target points to a smaller (but representative) sample. The following code generates a sample of input design values via `gr.ev_sample()` with the `skip=True` argument, uses `gr.tf_sp()` to "compact" this large sample, then evaluates the performance model at the smaller sample.

```
df_boats = (
    md_performance
    >> gr.ev_sample(
        n=5e3,
        df_det="nom",
        seed=101,
        skip=True,
    )
    >> gr.tf_sp(n=1000, seed=101)
    >> gr.tf_md(md=md_performance)
)
```

With an initial sample generated, we can perform an exploratory analysis relating the inputs and outputs. The verb `gr.tf_iocorr()` computes correlations between every pair of input variables `var` and outputs `out`. The routine also attaches metadata, enabling an autoplot as a tileplot of the correlation values.

```
(
    df_boats
    >> gr.tf_iocorr(
        var=["H", "W", "n", "d", "f_com"],
        out=["mass", "angle", "stability"],
    )
    >> gr.pt_auto()
)
```

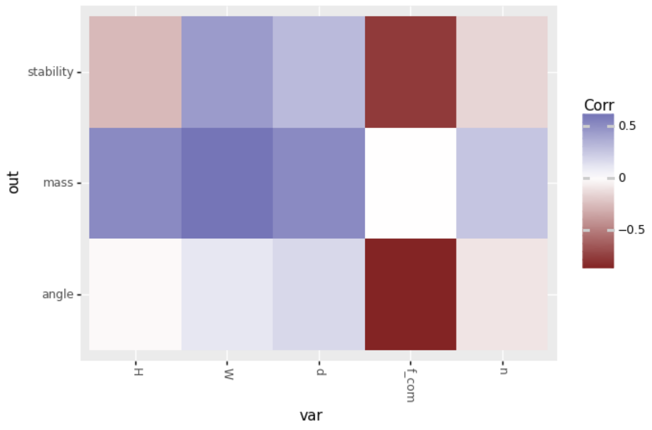

Fig. 7: *Tile plot of input/output correlations; autoplot gr.pt_auto() visualization of gr.tf_iocorr() output.*

The correlations in Figure 7 suggest that `stability` is positively impacted by increasing the width `W` and displacement ratio `d` of a boat, and by decreasing the height `H`, shape factor `n`, and vertical location of the center of mass `f_com`. The correlations also suggest a similar impact of each variable on the AVS `angle`, but with a weaker dependence on `H`. These results also suggest that `f_com` has the strongest effect on both `stability` and `angle`.

Correlations are a reasonable first-check of input/output behavior, but linear correlation quantifies only an average, linear association. A second-pass at the data would be to fit an accurate surrogate model and inspect parameter sweeps. The following code defines a gaussian process fit [RW05] for both `stability` and `angle`, and estimates model error using k-folds cross validation [JWHT13]. Note that a non-default kernel is necessary for a reasonable fit of the latter output[5].

```
## Define fitting procedure
ft_common = gr.ft_gp(
    var=["H", "W", "n", "d", "f_com"],
    out=["angle", "stability"],
    kernels=dict(
        stability=None,  # Use default
        angle=RBF(length_scale=0.1),
    )
)
## Estimate model accuracy via k-folds CV
(
    df_boats
    >> gr.tf_kfolds(
        ft=ft_common,
        out=["angle", "stability"],
    )
)
```

4. The analysis reported here is available as a jupyter notebook.

5. `RBF` is imported as `from sklearn.gaussian_process.kernels import RBF`.

angle	stability	k
0.771	0.979	0
0.815	0.976	1
0.835	0.95	2
0.795	0.962	3
0.735	0.968	4

TABLE 1: *Accuracy (R^2) estimated via k-fold cross validation of gaussian process model.*

The k-folds CV results (Tab. 1) suggest a highly accurate model for `stability`, and a moderately accurate model for `angle`. The following code defines the surrogate model over a domain that includes the original dataset, and performs parameter sweeps across all inputs.

```
md_fit = (
    df_boats
    >> ft_common()
    >> gr.cp_marginals(
        H=gr.marg_mom("uniform", mean=2.0, cov=0.30),
        W=gr.marg_mom("uniform", mean=2.5, cov=0.35),
        n=gr.marg_mom("uniform", mean=1.0, cov=0.30),
        d=gr.marg_mom("uniform", mean=0.5, cov=0.30),
        f_com=gr.marg_mom(
            "uniform",
            mean=0.55,
            cov=0.47,
        ),
    )
    >> gr.cp_copula_independence()
)

(
    md_fit
    >> gr.ev_sinews(df_det="swp", n_sweeps=5)
    >> gr.pt_auto()
)
```

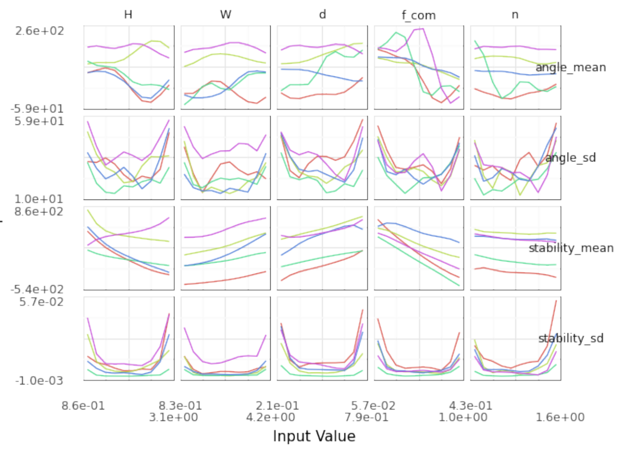

Fig. 8: *Parameter sweeps for fitted GP model. Model `*_mean` and predictive uncertainty `*_sd` values are reported for each output `angle`, `stability`.*

Figure 8 displays parameter sweeps for the surrogate model of `stability` and `angle`. Note that the surrogate model reports both a mean trend `*_mean` and a predictive uncertainty `*_sd`. The former is the model's prediction for future values, while the latter quantifies the model's confidence in each prediction.

The parameter sweeps of Figure 8 show a consistent and strong effect of `f_com` on the `stability_mean` of the boat; note that all the sweeps across `f_com` for `stability_mean` tend to be monotone with a fairly steep slope. This is in agreement with the correlation results of Figure 7; the `f_com` sweeps tend to have the steepest slopes. Given the high accuracy of the model for `stability` (as measured by k-folds CV), this trend is reasonably trustworthy.

However, the same figure shows an inconsistent (non-monotone) effect of most inputs on the AVS `angle_mean`. These results are in agreement with the k-fold CV results shown above. Clearly, the surrogate model is untrustworthy, and we should resist trusting conclusions from the parameter sweeps for `angle_mean`. This undermines the conclusion we drew from the input/output correlations pictured in Figure 7. Clearly, `angle` exhibits more complex behavior than a simple linear correlation with each of the boat design variables.

A different analysis of the boat hull `angle` data helps develop useful insights. We pursue an active subspace analysis of the data to reduce the dimensionality of the input space by identifying directions that best explain variation in the output [dCI17], [Con15]. The verb `gr.tf_polyridge()` implements the variable projection algorithm of Hokanson and Constantine [HC18]. The following code pursues a two-dimensional reduction of the input space. Note that the hyperparameter `n_degree=6` is set via a cross-validation study.

```
## Find two important directions
df_weights = (
    df_boats
    >> gr.tf_polyridge(
        var=["H", "W", "n", "d", "f_com"],
        out="angle",
        n_degree=6, # Set via CV study
        n_dim=2,    # Seek 2d subspace
    )
)
```

Direction	H	W	n	d	f_com
1	-0.0277	0.0394	-0.1187	0.4009	-0.9071
2	-0.6535	0.3798	-0.0157	-0.6120	-0.2320

TABLE 2: *Subspace weights in `df_weights`.*

The subspace weights are reported in Table 2. Note that the leading direction 1 is dominated by the displacement ratio `d` and COM location `f_com`. Essentially, this describes the "loading" of the vessel. The second direction corresponds to "widening and shortening" of the hull cross-section (in addition to lowering `d` and `f_com`).

Using the subspace weights in Table 2 to produce a 2d projection of the feature space enables visualizing all boat geometries in a single plot. Figure 9 reveals that this 2d projection is very successful at separating universally-stable (`angle==180`), upright-unstable (`angle==0`), and intermediate cases (`0 < angle < 180`). Intermediate cases are concentrated at higher values of the second active variable. There is a phase transition between universally-stable and upright-unstable vessels at lower values of the second active variable.

Interpreting Figure 9 in light of Table 2 provides us with deep insight about boat stability: Since active variable 1 corresponds to loading (high displacement ratio `d` with a low COM `f_com`), we can see that the boat's loading conditions are key to determining its stability. Since active variable 2 depends on the aspect ratio

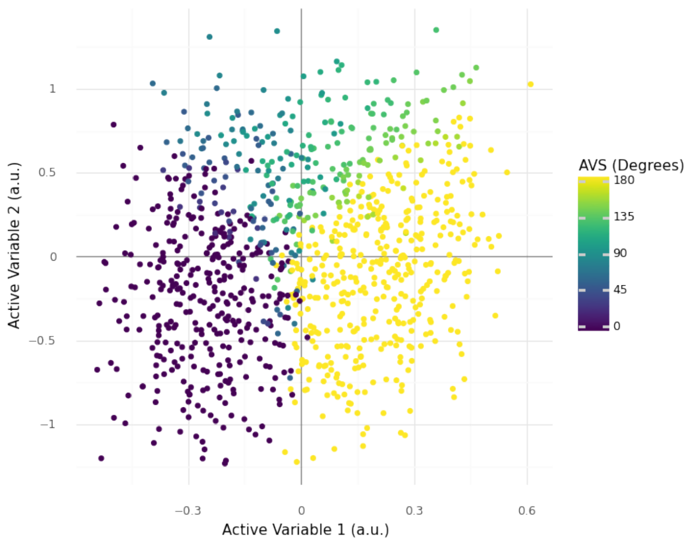

Fig. 9: *Boat design feature vectors projected to 2d active subspace. The origin corresponds to the mean feature vector.*

(higher width, shorter height), Figure 9 suggests that only wider boats will tend to exhibit intermediate stability.

Conclusions

Grama is a Python implementation of a grammar of model analysis. The grammar's design supports an active learning approach to teaching sound scientific modeling practices. Two case studies demonstrated the teaching benefits of grama: *errors for learners* help guide novices toward a more sound analysis, while concise syntax encourages novices to carry out sound analysis practices. Grama can also be used for exploratory model analysis (EMA) – an exploratory procedure to mine a scientific model for useful insights. A case study of boat hull design demonstrated EMA. In particular, the example explored and explained the relationship between boat design parameters and two metrics of boat stability.

Several ideas from the grama project are of interest to other practitioners and developers in scientific computing. Grama was designed to support model analysis *under uncertainty*. However, the **data/model and four-verb ontology** (Fig. 1) underpinning grama is a much more general idea. This design enables very concise model analysis syntax, which provides much of the benefit behind grama.

The design idiom of **errors for learners** is not simply focused on writing "useful" error messages, but is rather a design orientation to use errors to introduce teachable moments. In addition to writing error messages "for humans" [Bry20], an *errors for learners* philosophy designs errors not simply to avoid fatal program behavior, but rather introduces exceptions to prevent conceptually invalid analyses. For instance, in the case study presented above, designing *gr.tf_sample()* to assume independent random inputs when a copula is unspecified would lead to code that throws errors less frequently. However, this would silently endorse the conceptually problematic mentality of "independence is the default." While throwing an error message for an unspecified dependence structure leads to more frequent errors, it serves as a frequent reminder that dependency is an important part of a model involving random inputs.

Finally, exploratory model analysis holds benefits for both learners and practitioners of scientific modeling. EMA is an alternative to derivation for the activities in an active learning approach. Rather than structuring course around deriving and implementing scientific models, course exercises could have students explore the behavior of a pre-implemented model to better understand physical phenomena. Lorena Barba [Bar16] describes some of the benefits in this style of lesson design. EMA is also an important part of the modeling practitioner's toolkit as a means to verify a model's implementation and to develop new insights. Grama supports both novices and practitioners in performing EMA through a concise syntax.

References

[AFD+21] Riya Aggarwal, Mira Flynn, Sam Daitzman, Diane Lam, and Zachary Riggins del Rosario. A qualitative study of engineering students' reasoning about statistical variability. In *2021 Fall ASEE Middle Atlantic Section Meeting*, 2021. URL: https://peer.asee.org/38421.

[Bar16] Lorena Barba. Computational thinking: I do not think it means what you think it means. Technical report, 2016. URL: https://lorenabarba.com/blog/computational-thinking-i-do-not-think-it-means-what-you-think-it-means/.

[Bie20] Przemyslaw Biecek. *ceterisParibus: Ceteris Paribus Profiles*, 2020. R package version 0.4.2. URL: https://cran.r-project.org/package=ceterisParibus.

[Blo56] Benjamin Samuel Bloom. *Taxonomy of educational objectives: The classification of educational goals*. Addison-Wesley Longman Ltd., 1956.

[Bry20] Jennifer Bryan. object of type closure is not subsettable. 2020. rstudio::conf 2020. URL: https://rstd.io/debugging.

[BTSA12] Michael Bauer, Sean Treichler, Elliott Slaughter, and Alex Aiken. Legion: Expressing locality and independence with logical regions. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE, 2012. URL: https://ieeexplore.ieee.org/document/6468504, doi:10.1109/SC.2012.71.

[CF99] Alison C Cullen and H Christopher Frey. *Probabilistic Techniques In Exposure Assessment: A Handbook For Dealing With Variability And Uncertainty In Models And Inputs*. Springer Science & Business Media, 1999.

[Con15] Paul G. Constantine. *Active Subspaces: Emerging Ideas for Dimension Reduction in Parameter Studies*. SIAM Philadelphia, 2015. doi:10.1137/1.9781611973860.

[dCI17] Zachary del Rosario, Paul G. Constantine, and Gianluca Iaccarino. Developing design insight through active subspaces. In *19th AIAA Non-Deterministic Approaches Conference*, page 1090, 2017. URL: https://arc.aiaa.org/doi/10.2514/6.2017-1090, doi:10.2514/6.2017-1090.

[dR20] Zachary del Rosario. Grama: A grammar of model analysis. *Journal of Open Source Software*, 5(51):2462, 2020. URL: https://doi.org/10.21105/joss.02462, doi:10.21105/joss.02462.

[dRFI21] Zachary del Rosario, Richard W Fenrich, and Gianluca Iaccarino. When are allowables conservative? *AIAA Journal*, 59(5):1760–1772, 2021. URL: https://doi.org/10.2514/1.J059578, doi:10.2514/1.J059578.

[EVB+20] M Esmaily, L Villafane, AJ Banko, G Iaccarino, JK Eaton, and A Mani. A benchmark for particle-laden turbulent duct flow: A joint computational and experimental study. *International Journal of Multiphase Flow*, 132:103410, 2020. URL: https://www.sciencedirect.com/science/article/abs/pii/S030193222030519X, doi:10.1016/j.ijmultiphaseflow.2020.103410.

[FEM+14] Scott Freeman, Sarah L Eddy, Miles McDonough, Michelle K Smith, Nnadozie Okoroafor, Hannah Jordt, and Mary Pat Wenderoth. Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the National Academy of Sciences*, 111(23):8410–8415, 2014. doi:10.1073/pnas.1319030111.

[HC18] Jeffrey M Hokanson and Paul G Constantine. Data-driven polynomial ridge approximation using variable projection. *SIAM Journal on Scientific Computing*, 40(3):A1566–A1589, 2018. doi:10.1137/17M1117690.

[HK21] Jan Katins gdowding austin matthias-k Tyler Funnell Florian Finkernagel Jonas Arnfred Dan Blanchard et al. Hassan Kibirige, Greg Lamp. has2k1/plotnine: v0.8.0. Mar 2021. doi:10.5281/zenodo.4636791.

[JWHT13] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R*, volume 112. Springer, 2013. URL: https://www.statlearning.com/.

[KBB19] Michał Kuźba, Ewa Baranowska, and Przemysław Biecek. pyceterisparibus: explaining machine learning models with ceteris paribus profiles in python. *Journal of Open Source Software*, 4(37):1389, 2019. URL: https://joss.theoj.org/papers/10.21105/joss.01389, doi:10.21105/joss.01389.

[KN05] Andy Keane and Prasanth Nair. *Computational Approaches For Aerospace Design: The Pursuit Of Excellence*. John Wiley & Sons, 2005.

[KSS21] Daniel Kahneman, Olivier Sibony, and Cass R Sunstein. *Noise: A flaw in human judgment*. Little, Brown, 2021.

[LE00] Lars Larsson and Rolf Eliasson. *Principles of Yacht Design*. McGraw Hill Companies, 2000.

[MJ18] Simon Mak and V Roshan Joseph. Support points. *The Annals of Statistics*, 46(6A):2562–2592, 2018. doi:10.1214/17-AOS1629.

[MTW+22] Kazuki Maeda, Thiago Teixeira, Jonathan M Wang, Jeffrey M Hokanson, Caetano Melone, Mario Di Renzo, Steve Jones, Javier Urzay, and Gianluca Iaccarino. An integrated heterogeneous computing framework for ensemble simulations of laser-induced ignition. *arXiv preprint arXiv:2202.02319*, 2022. URL: https://arxiv.org/abs/2202.02319, doi:10.48550/arXiv.2202.02319.

[MV15] Sparsh Mittal and Jeffrey S Vetter. A survey of cpu-gpu heterogeneous computing techniques. *ACM Computing Surveys (CSUR)*, 47(4):1–35, 2015. URL: https://dl.acm.org/doi/10.1145/2788396, doi:10.1145/2788396.

[pdt20] The pandas development team. pandas-dev/pandas: Pandas, February 2020. URL: https://doi.org/10.5281/zenodo.3509134, doi:10.5281/zenodo.3509134.

[RW05] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 11 2005. URL: https://doi.org/10.7551/mitpress/3206.001.0001, doi:10.7551/mitpress/3206.001.0001.

[SKA+14] Jeffrey P Slotnick, Abdollah Khodadoust, Juan Alonso, David Darmofal, William Gropp, Elizabeth Lurie, and Dimitri J Mavriplis. Cfd vision 2030 study: A path to revolutionary computational aerosciences. Technical report, 2014. URL: https://ntrs.nasa.gov/citations/20140003093.

[SS13] Seref Sagiroglu and Duygu Sinanc. Big data: A review. In *2013 International Conference on Collaboration Technologies and Systems (CTS)*, pages 42–47. IEEE, 2013. URL: https://ieeexplore.ieee.org/document/6567202, doi:10.1109/CTS.2013.6567202.

[SSW89] Jerome Sacks, Susannah B. Schiller, and William J. Welch. Designs for computer experiments. *Technometrics*, 31(1):41–47, 1989. URL: http://www.jstor.org/stable/1270363, doi:10.2307/1270363.

[WAB+19] Hadley Wickham, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, Alex Hayes, Lionel Henry, Jim Hester, et al. Welcome to the tidyverse. *Journal of Open Source Software*, 4(43):1686, 2019. doi:10.21105/joss.01686.

[Wic14] Hadley Wickham. Tidy data. *Journal of Statistical Software*, 59(10):1–23, 2014. doi:10.18637/jss.v059.i10.

[WM10] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010. doi:10.25080/Majora-92bf1922-00a.