bayesim: a tool for fast model fitting with Bayesian inference

Rachel Kurchin
Department of Materials Science & Engineering, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139, USA

Giuseppe Romano and Tonio Buonassisi*
Department of Mechanical Engineering, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139, USA

Target journal: Computer Physics Communications

INTRODUCTION

There are a plethora of examples across diverse scientific and engineering fields of mathematical models used to simulate the results of experimental observations. In many cases, there are input parameters to these models which are difficult to determine via direct measurement, and it is desirable to invert the numerical model – that is, use the experimental observations to determine values of the input parameters. Bayesian inference is a fruitful framework within which to do such fitting, since the resulting posterior probability distribution over the parameters of interest can give rich insights into not just the most likely values of the parameters, but also uncertainty about these values and the potentially complicated ways in which they can covary to give equally good fits to observations.

We have previously demonstrated the value of a Bayesian approach in using automated high-throughput temperature- and illumination-dependent current-voltage measurements (JVTi) to fit material/interface properties and defect recombination parameters in photovoltaic (PV) absorbers [1, 2]. In cases such as these, when the data model is not a simple analytical equation but rather a computationally intensive numerical model, efficient, nonredundant sampling of the parameter space when computing likelihoods becomes critical to making the fit feasible.

In this work, we introduce `bayesim`, a Python-based code that utilizes adaptive grid sampling to perform Bayesian parameter estimation. We discuss the structure of the code, its implementation, and provide several examples of its usage. While the authors' expertise is in the realm of semiconductor physics and thus the examples herein are drawn from that space, we also discuss the general characteristics of a problem amenable to this approach so that researchers from other fields might adopt it as well.

TECHNICAL BACKGROUND

Bayes' Theorem is a relationship between conditional probabilities. It states

$$P(H|E) = \frac{P(H)P(E|H)}{P(E)} \tag{1}$$

where the notation $P(A|B)$ indicates the probability of A being true given that B is true. H is a *hypothesis* and E the observed *evidence*. $P(H)$ is termed the *prior*, $P(E|H)$ the *likelihood*, $P(H|E)$ the *posterior*, and $P(E)$ is a normalizing constant. If there are n pieces of evidence, this can generalize to an iterative process where

$$P(H|\{E_1, E_2, ...E_n\}) = \frac{P(H|\{E_1, E_2...E_{n-1}\})P(E_n|H)}{P(E_n)} \tag{2}$$

In a multidimensional parameter estimation problem, each hypothesis H is a tuple of possible values for the fitting parameters, i.e. a point in the parameter space, while the evidence E is an observed output of a measurement as a function of various experimental conditions.

In order to compute a likelihood, a model capable of simulating that observed output as a function of both the fitting parameters and the experimental conditions is required. In `bayesim`, likelihoods are calculated for each point in parameter space using a Gaussian where the argument is the difference between observed and simulated output at that point, and the standard deviation is the sum of experimental uncertainty and model uncertainty. The experimental uncertainty is a number provided by the user that quantifies any noise/irreproducibility inherent to the measurement,

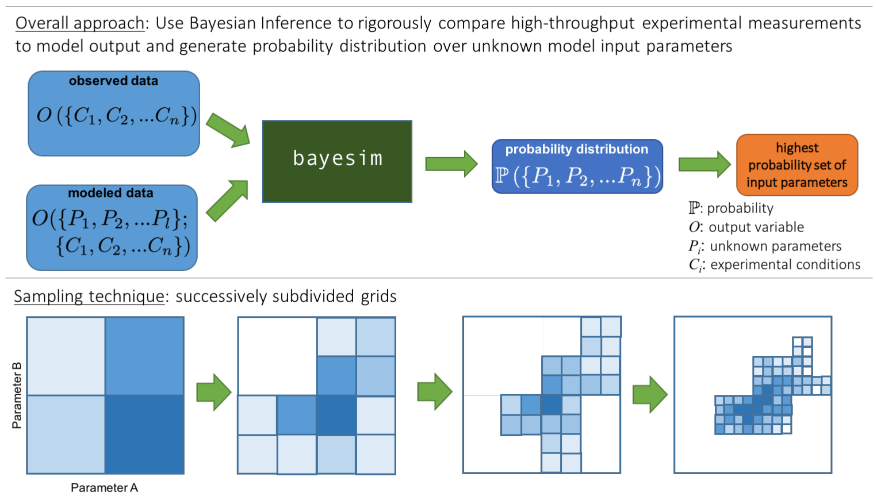

FIG. 1. (Figure copied from docs website for now, will make a publication-appropriate version)

while the model uncertainty is calculated by `bayesim` and reflects the sparseness of the parameter space grid, i.e. how much simulated output changes from one grid point to another.

A high-level summary of what `bayesim` does is shown in Figure 1. Part (a) is a flowchart showing that observed (as a function of experimental conditions $\{C\}$ and simulated (as a function of fitting parameters $\{P\}$ and experimental conditions $\{C\}$) outputs are compared to produce a probability distribution over $\{P\}$. Part (b) schematically indicates the adaptive grid sampling approach for a hypothetical two-dimensional parameter space, wherein grid boxes exceeding some threshold probability are subdivided and lower-probability regions discarded, allowing attainment of a high fitting precision without needing to sample the entire parameter space at the same high density.

SOFTWARE ARCHITECTURE AND INTERFACE

The structure of `bayeim` is shown in Figure 2. Detailed and up-to-date documentation of classes, functions, and example applications is maintained online. [3]

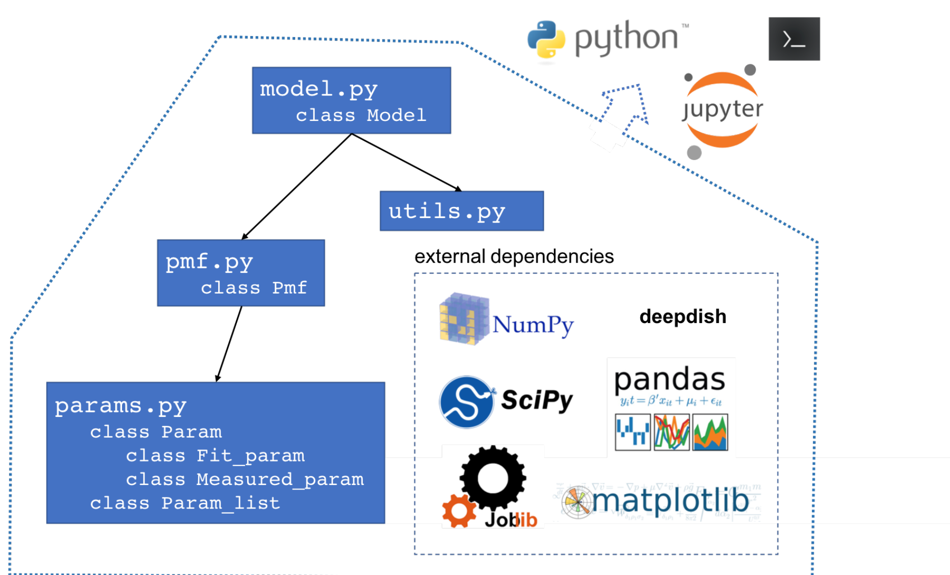

FIG. 2. `bayesim` software structure, indicating class definitions, internal and external dependencies, and interfaces.

Structure

The top-level object with which users interact is implemented in the `Model` class. The `params` module defines classes to store information about the various types of parameters (fitting parameters, experimental conditions, and measured output) while the `Pmf` class stores the probability distribution and implements the manipulations required for Bayesian updates.

Interfaces

The most flexible way to interact with `bayesim` is via Python scripting or through literate programming in a Jupyter notebook. There is also a command line interface (CLI) for users less familiar with coding in Python.

Dependencies

`bayesim` relies on a variety of external open-source packages. These include `numpy` [4] and `scipy` [5] for a variety of mathematical functions and vectorized implementations, `joblib` [6] for simple parallelism, `deepdish` [7] for saving and loading HDF5 files, `pandas` [8] for data manipulation, and `matplotlib` [9] for visualization.

APPLICATION EXAMPLES

Ideal Diode Model

As a first example, we fit a simple two-parameter model of solar cell current density J (as a function of voltage V and temperature T) known as the ideal diode model:

$$J(V,T) = J_L + J_0 \left(\exp \frac{qV}{nkT} - 1 \right) \tag{3}$$

where $k = 8.61733 \times 10^{-5}$ eV/K is Boltzmann's constant, by convention J_L (the light current) is negative and J_0 (the saturation current) is positive but strongly dependent on temperature, a dependence we can approximate as:

$$J_0 \approx B' T^{3/n} \exp \left(\frac{-E_{g0}}{nkT} \right) \tag{4}$$

where E_{g0} is the zero-temperature bandgap of the material. For this example, we will set this to 1.2 eV and J_L to -0.03 mA/cm^2. There are thus two parameters to be fit, the coefficient B' (which can range over a wide variety of values) and the ideality factor n (which is by definition between 1 and 2).

As validation, we generate the "observations" using the same model we'll use for fitting, with parameter values of $n = 1.36$ and $B' = 258$, and four temperature values of 150, 200, 250, and 300 K.

SnS Solar Cell

- more practical example - replicating fit from Joule paper
- figure 4 showing PMF and comparison of JV curves

CONCLUSIONS

talk about broader applicability of approach

ACKNOWLEDGEMENTS

APPENDIX

- include minimal code to run ideal diode example
- link to Github repo (which has installation instructions and documentation as well as list of planned future features)

---

* buonassi@mit.edu

[1] R. E. Brandt, R. C. Kurchin, V. Steinmann, D. Kitchaev, C. Roat, S. Levcenco, G. Ceder, T. Unold, T. Buonassisi, and H. Z. Berlin, "Rapid semiconductor device characterization through Bayesian parameter estimation," *Joule*, vol. 1, no. 4, pp. 843–856, 2017.

[2] R. C. Kurchin, J. R. Poindexter, D. Kitchaev, V. Vähänissi, C. del Cañizo, L. Zhe, H. S. Laine, C. Roat, S. Levcenco, G. Ceder, and T. Buonassisi, "Semiconductor parameter extraction via current-voltage characterization and bayesian inference methods," 6 2018.

[3] R. C. Kurchin and G. Romano, "bayesim." `https://pv-lab.github.io/bayesim/_build/html/index.html`, 2018.

[4] T. E. Oliphant, *A guide to NumPy*. Trelgol Publishing, 2006.

[5] E. Jones, T. Oliphant, P. Peterson, *et al.*, "SciPy: Open source scientific tools for Python," 2001.

[6] J. developers, "Joblib." `https://joblib.readthedocs.io/en/latest/`.

[7] G. Larsson and M. Stoehr, "deepdish." `https://github.com/uchicago-cs/deepdish`.

[8] W. McKinney, "Data structures for statistical computing in python," 2010.

[9] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.