Journal of
open research software

Slugocki, M, et al. 2019 BayesFit: A Tool for Modeling Psychophysical Data Using Bayesian Inference. *Journal of Open Research Software*, 7: 2. DOI: https://doi.org/10.5334/jors.202

SOFTWARE METAPAPER

BayesFit: A Tool for Modeling Psychophysical Data Using Bayesian Inference

Michael Slugocki, Allison B. Sekuler and Patrick J. Bennett
McMaster University, CA
Corresponding author: Michael Slugocki (slugocm@mcmaster.ca)

BayesFit is a module for Python that allows users to fit models to psychophysical data using Bayesian inference. The module aims to make it easier to develop probabilistic models for psychophysical data in Python by providing users with a simple API that streamlines the process of defining psychophysical models, obtaining fits, extracting outputs, and visualizing fitted models. Our software implementation uses numerical integration as the primary tool to fit models, which avoids the complications that arise in using Markov Chain Monte Carlo (MCMC) methods [1]. The source code for BayesFit is available at https://github.com/slugocm/bayesfit and API documentation at http://www.slugocm.ca/bayesfit/. This module is extensible, and many of the functions primarily rely on Numpy [2] and therefore can be reused as newer versions of Python are developed to ensure researchers always have a tool available to ease the process of fitting models to psychophysical data.

Keywords: Psychophysics; Psychometrics; Psychometric function; Bayesian inference; Numerical integration; Curve fitting; Python

(1) Overview

Introduction

Fitting statistical models to behavioural data is an important part of analyzing task performance within the field of Psychology [3]. Within the domain of psychophysics, a fundamental statistical model is the psychometric function, which relates performance – usually response accuracy or latency – to the stimulus, and is used to quantify stimulus detection or discrimination [3]. In experiments that measure response accuracy in a stimulus detection task, the psychometric function indicates how the probability of a correct detection response is related to stimulus intensity. Typically, correct responses are made more frequently as the stimulus intensity increases [3, 4].

Although fitting a psychometric function to data is a common component of the analysis of behavioural data, very few tools have been designed to help researchers streamline this fitting process [1, 3, 4, 5], and only one of these has been adapted for use in Python [1]. Furthermore, the only tool available for Python does not support versions 3.x or greater. We therefore created a module supporting newer versions of Python (3.x) that implements a Bayesian algorithm to fit psychometric functions to behavioural data with as few steps as possible. The resulting software, BayesFit, accomplishes this task by taking advantage of a simple API that provides a convenient method for users to build probabilistic models using numerical integration in Python. Our module also makes it easy for users to fit models to large numbers of datasets while constraining the parameters used across model fits.

Usage Example 1: Fitting Models to Psychophysical Data using MLE

The full API documentation for BayesFit can be found at: http://www.slugocm.ca/bayesfit/.

To demonstrate the functionality of this software, below we provide the results from a simulated two-interval, forced-choice (2-IFC) detection experiment in which each experimental trial consists of two successive, temporal intervals, and the observer must determine whether the target stimulus was presented in the first or second interval. Stimulus intensity is varied across trials, and response accuracy varies from near chance levels at low intensity to nearly perfect accuracy at high stimulus intensity (see **Figure 1**). Detection threshold is estimated by fitting a curve to the response accuracy data and then determining the stimulus intensity that produces an intermediate level of response accuracy (e.g., 75% correct).

BayesFit expects that data provided by the user are organized into a *m*-row by 3-column Numpy array, where the 1st column corresponds to the stimulus intensities used, the 2nd column the number of correct responses made by an observer at each stimulus intensity, and the 3rd column the number of trials run at each intensity

level. Data in columns 2 and 3 should be organized such that values along each row correspond to the level of intensity specified along the same row in column 1 (see **Figure 2**).

Once data are appropriately organized, we are ready to use BayesFit to perform the model fitting procedure. We first import the BayesFit module into our workspace via:

```
import bayesfit as bf
```

Next, we specify the output variables and options that we want to use for our fitting procedure. BayesFit uses only one main function called *fitmodel* to streamline the process of fitting models to psychophysical data. If no priors are provided to the function, the fitting procedure is equivalent to performing maximum likelihood estimation (MLE). A basic example in using the *fitmodel* function to fit a model using MLE would be as follows:

```
metrics,options = bf.fitmodel(data = data,
                              batch = False,
                              logspace = None,
                              nafc = 2,
                              sigmoid_type = 'norm',
                              param_ests = None,
                              priors = None,
                              param_free = [True,
                                     True, False,
                                     False],
                              threshold = 0.75,
                              density = 100
                              )
```

Although we have listed all options that are available to the user in fitting models using BayesFit, the only mandatory argument to provide the *fitmodel* function with is data. However, if additional options are not specified, default values for these options will be assigned. Complete API documentation describing each of these options, as well as their default assignments, are located at: http://www.slugocm.ca/bayesfit/. Following the fitting procedure, the *fitmodel* function outputs two variables that include:

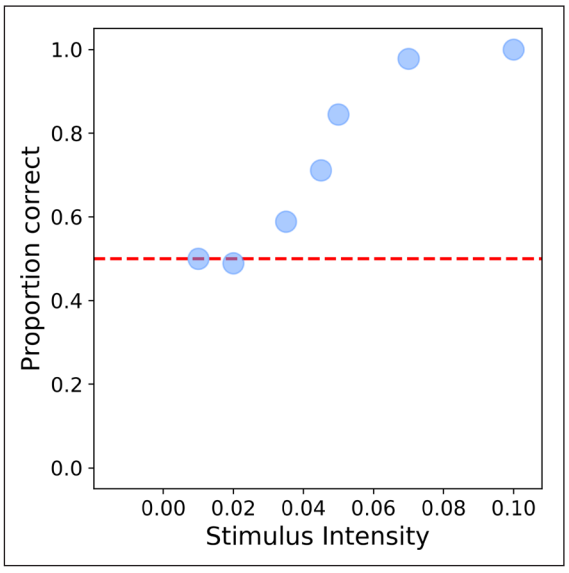

Figure 1: Plot of synthetic data generated from an observer detecting the flash of a light onscreen. The abscissa represents stimulus intensity, while the ordinate axis represents proportion correct.

1. **Metrics** – a dictionary containing parameter estimates from the fitted model (s), metrics about the goodness-of-fit of each parameter, arrays corresponding to the marginal distributions, the likelihood surface (posterior if priors provided), as well as a numerical approximation of the threshold from the fitted function (at level specified in options).
2. **Options** – the options used to fit model (s) to data.

Using these outputted variables, we can generate a plot of the fitted model using the *plot_psyfcn* function of the BayesFit module. For example, to visualize a psychometric function (Cumulative Normal sigmoid) fit using the BayesFit module to the data shown in **Figure 1**, we would specify the following snippet of code:

```
bf.plot_psyfcn (data, options, metrics)
```

As can be seen in the example above, only a few lines of code are needed to generate parameter estimates for our model and quickly visualize the results (see **Figure 3**). Although our example above was performed using only a single dataset, the BayesFit module also includes a method for fitting models to batch datasets so that the same model definitions can be used across all models fit. Once again, we encourage the reader to visit the API

m x 3 Numpy array

x	y	N
0.1	59	100
0.2	57	100
0.3	65	100
⋮	⋮	⋮

Figure 2: Example of how data from a single psychophysical experiment should be formatted when passing these data to BayesFit for model fitting.

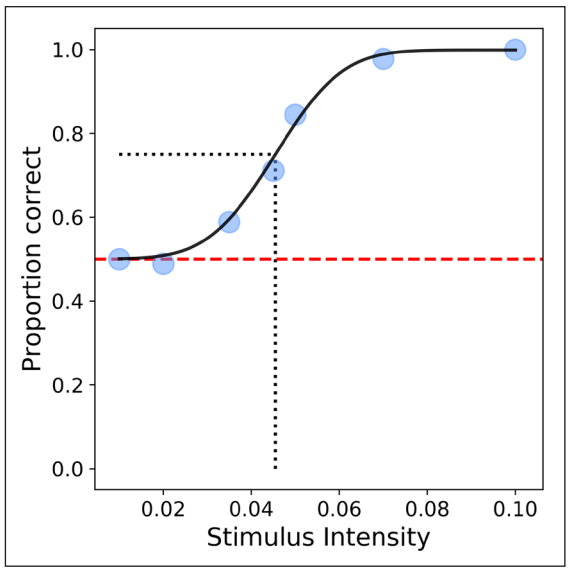

Figure 3: Plot of psychometric function using a Cumulative Normal sigmoid fit to data using BayesFit.

documentation for detailed examples of how to make use of such operations.

Usage Example 2: Fitting Models to Psychophysical Data using Bayesian Inference

Our example above used an MLE fitting procedure, as no prior distributions were provided to BayesFit for parameters to-be-estimated. However, if we have prior knowledge about what the values of our parameters should be, along with what the shape of the distribution in likelihood of different values are (e.g., Uniform, Normal, etc.), we can pass these arguments along to the *fitmodel* function of BayesFit during our fitting procedure. The fundamental difference in passing priors to BayesFit is that we are now using Bayesian inference methods rather than MLE in estimating parameters of the psychometric function, and thus are computing the posterior distribution.

BayesFit provides users with 5 different distributions that can be used to specify priors for each parameter. These include:

1. **'Unif (a,b)'** – Uniform
2. **'Norm (a,b)'** – Normal
3. **'Log-Norm (a,b)'** – Log-normal
4. **'Beta (a,b)'** – Beta
5. **'Gamma (a,b)'** – Gamma

Therefore, if the user knew that certain values were more likely compared to others for the parameters controlling the scale and slope of the psychometric function, then the user could pass those arguments along to the *fitmodel* function of BayesFit as follows:

```
# Define priors for scale and slope parameters of
# Cumulative Normal function
priors = ['Norm (0.05,0.01)', 'Norm
(0.012,0.001)', None, None]

# Fit model passing along priors
metrics,options = bf.fitmodel(data = data,
                              batch = False,
                              logspace = None,
                              nafc = 2,
                              sigmoid_type = 'norm',
                              param_ests = None,
                              param_free = [True,
                                     True, False,
                                     False],
                              priors = priors,
                              threshold = 0.75,
                              density = 100
                              )
```

From this example, it can also be seen that choosing prior distributions, and the values that define them, can be difficult. Therefore, it is best to use priors only when you are confident that the probability of certain values for parameters to-be-estimated are more likely than others based on theoretical or empirical grounds. Following the fitting procedure, the user can now visualize the priors used during the fitting procedure using the *plot_priors* function from BayesFit (see **Figure 4**):

```
bf.plot_priors (options, metrics)
```

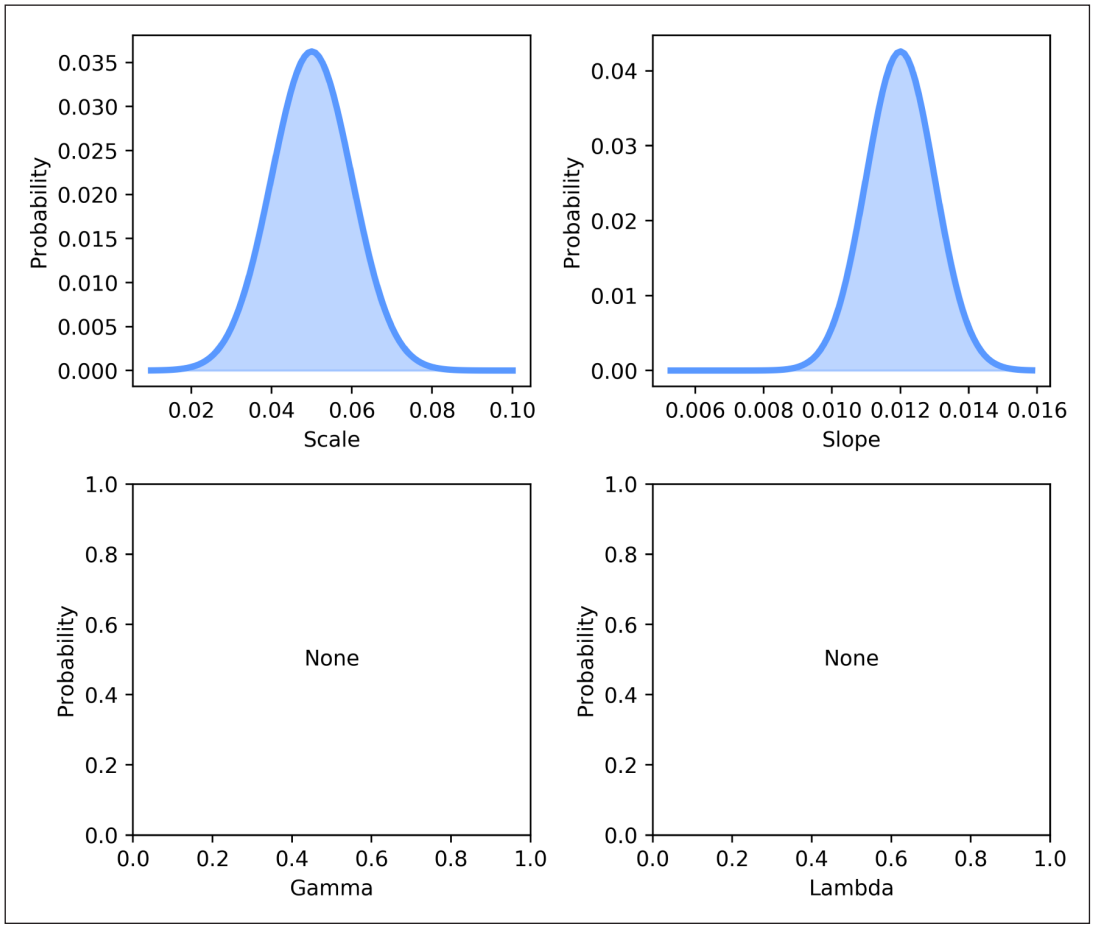

Figure 4: Plot of prior distributions used during Bayesian inference of parameters of psychometric function.

The marginal distributions for each parameter can be plotted via the *plot_marginals* functions (see **Figure 5**):

```
# Plot marginal distributions
bf.plot_marginals (metrics)
```

The posterior surface can be visualized via the *plot_posterior* function (see **Figure 6**):

```
# Plot posterior collapsed across gamma and lambda
bf.plot_posterior (metrics)
```

Usage Example 3: Batch Fitting Models to Datasets

Occasions might arise where the user would like to automatically, rather than manually, fit multiple psychometric models across several datasets. The BayesFit module makes this task easy by requiring only two small changes to the procedure used to fit a single model. The first change is that the batch input argument for the *fitmodel* function must be set to *True*. The second change is that each dataset must be stored in a single dictionary object, with each dataset occupying a separate key (see **Figure 7**).

Here is a simple example of using the *fitmodel* function to perform batch fitting on multiple datasets:

```
# Store each dataset in a separate key under a
# dictionary object
data = dict ()
data['dataset_01'] = data01
data['dataset_02'] = data02
data['dataset_03'] = data03

# Use fitmodel to fit functions across multiple
# datasets
metrics, options = bf.fitmodel (data = data,
                                batch = True
                                )
```

Note that only user-specified values for fixed parameters are used when performing batch fitting. Furthermore, to

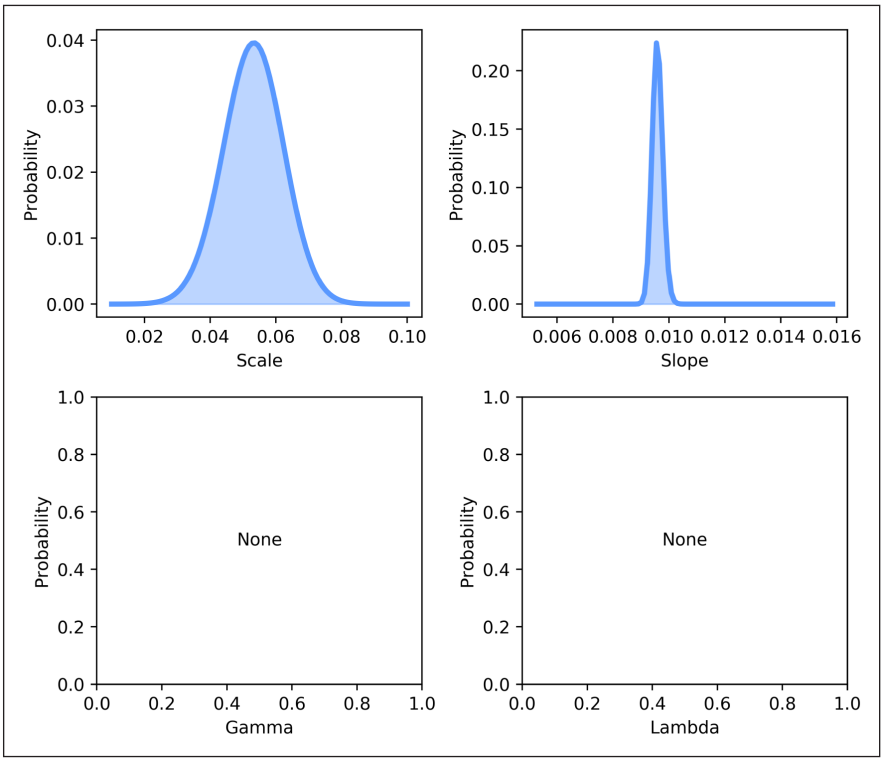

Figure 5: Plot of marginal distributions of parameters extracted from posterior.

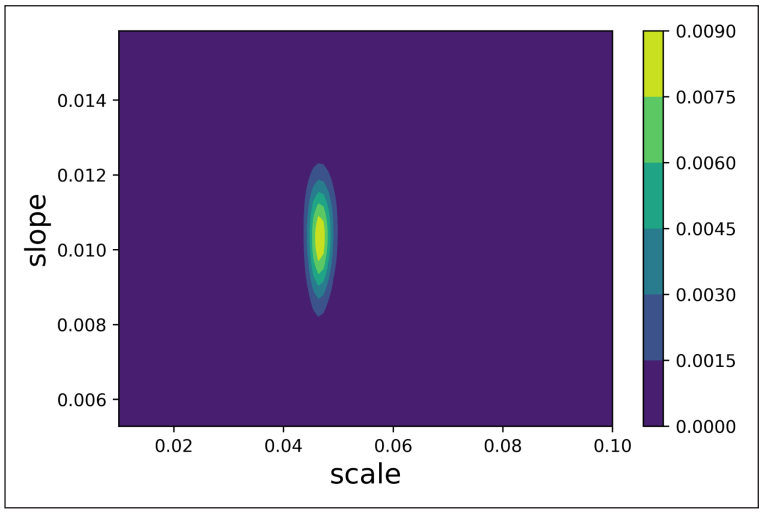

Figure 6: Plot of posterior surface for scale and slope parameters, collapsed across gamma and lambda.

prevent memory issues, only select metrics will be saved for each psychometric model fit.

Performance evaluation using simulated data

In order to assess the accuracy of parameter estimates using BayesFit, we generated 100 synthetic datasets from simulated observers in a psychophysical experiment. The datasets were generated to mimic the response of observers performing a 2-interval forced choice task with with eight equally-spaced stimulus intensities (range: 0.1–0.9). The true probability for an observer of correctly responding to a given level of intensity, x, was modelled as:

$$p(x,\alpha,\beta,\gamma,\lambda) = \gamma + (1-\gamma-\lambda)\times Weibull_{CDF}(x,\alpha,\beta) \quad (1)$$

where x is the intensity of the stimulus, α and β determine the scale and shape of the Weibull sigmoid function, γ is the guess rate, and λ the lapse rate. With the exception of the parameter γ which was fixed at 0.5 (i.e., 1/nafc), the parameters that define an observers› sensitivity at different stimulus intensities were drawn from the distributions:

$$\begin{aligned} \alpha &\sim Unif(0.45,\ 0.65) \\ \beta &\sim Unif(2,\ 8) \\ \lambda &\sim Unif(0,\ 0.10) \end{aligned} \quad (2)$$

These parameters were used to generate 100 binomial random responses at each stimulus intensity, for a total of 800 responses. In other words, 100 responses were simulated for each observer at each stimulus intensity. Data from each simulated observer were fit using the **default** settings for BayesFit, with the exception that a Weibull function was specified for the sigmoid type.

Results regarding the accuracy of parameter estimates for α, β, and λ are shown in **Figures 8** and **9**. Overall,

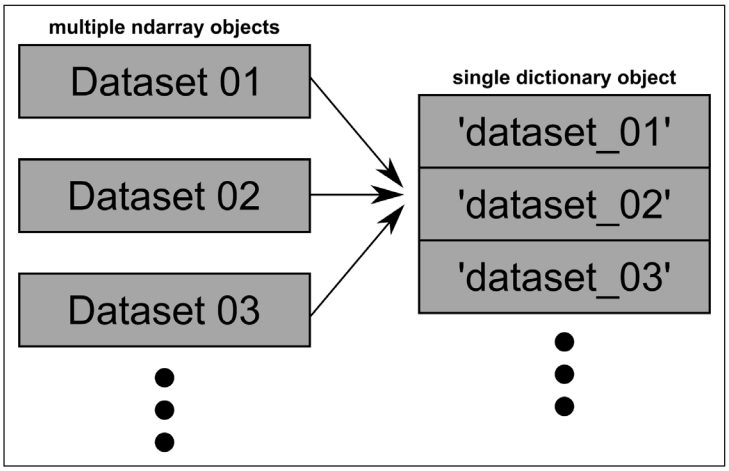

Figure 7: Example of how multiple datasets should be combined into a single dictionary object before being passed as an argument to BayesFit for batch fitting.

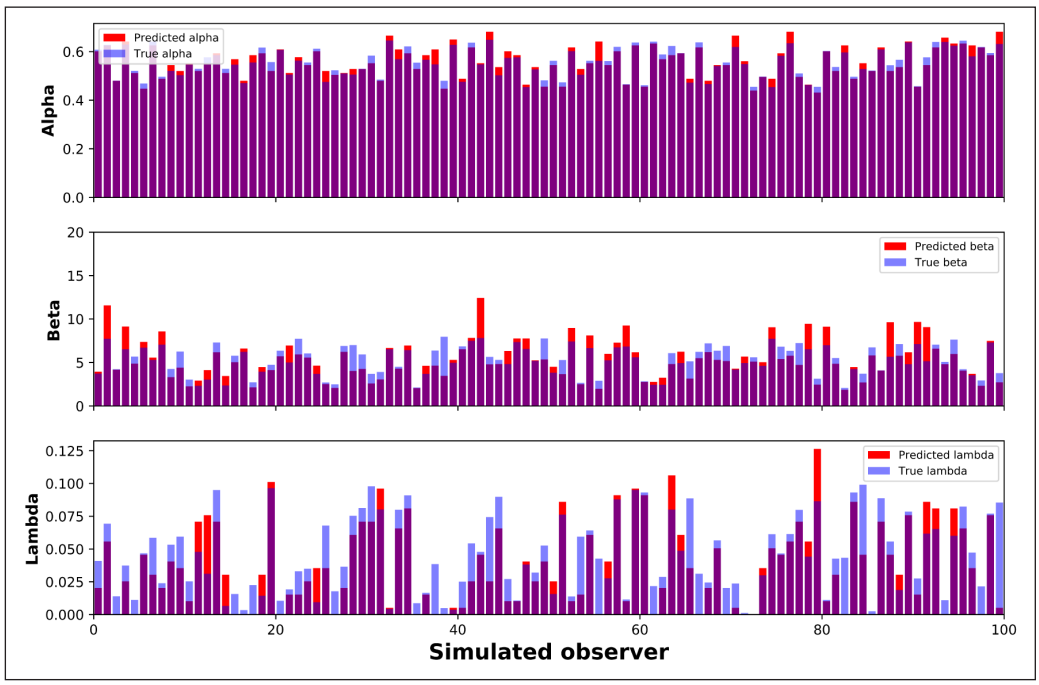

Figure 8: Three plots comparing the accuracy of predicted parameter values versus the true values used to generate data for each simulated observer. The ordinate axis provides the value of the parameter, either estimated or true, and the abscissa is an index for each simulated observer.

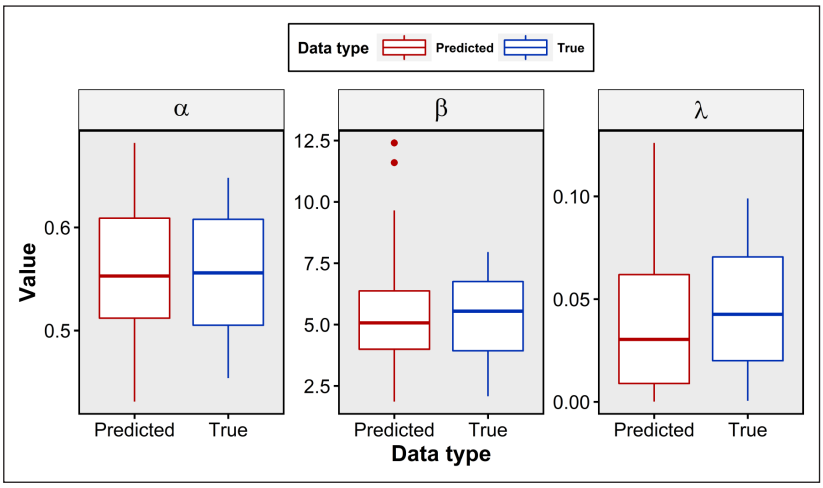

Figure 9: Boxplots of parameter estimates generated using BayesFit compared to target distributions.

parameter estimates using BayesFit with default settings are extremely good. Estimates for α, a parameter that is typically used to represent the threshold of an observer, is very accurate. Estimates for parameters β and λ are also quite close to the true parameters used to generate data for a given observer, but are more variable in their accuracy to true values compared to parameter α.

These simulations demonstrate that BayesFit can be trusted for use in estimating the parameters of models used to fit psychophsyical data. It should be noted that, when possible, greater care should be taken in choosing the options used to fit models to psychophysical data. However, these simulations serve as a worst case scenario where the user fails to specify prior knowledge that may help the fitting procedure, and yet, BayesFit still performs extremely well.

Comparison in accuracy of estimating thresholds to Psignifit 4.0

To evaluate the performance of BayesFit in comparison to Psignifit 4.0, the only other module the authors are aware of for fitting models to psychophysical data in Python (albeit 2.x), we make use of the simulated data used above for judging the accuracy of BayesFit in parameter estimation. Because BayesFit uses definitions for psychometric functions according to those specified in [6], and Psignifit 4.0 uses parameterizations of the psychometric functions according to [7], a direct comparison between parameter estimates obtained using Psignifit 4.0 and BayesFit cannot be readily made.

However, we can assess the accuracy of threshold estimates obtained from the model fit to a simulated observer's dataset using each module. Therefore, we used Psignifit 4.0 to fit models to simulated data using default settings except for specifying the sigmoid type as a Weibull function, and also defining threshold at 75% proportion correct response.

Figure 10 show boxplots of the distributions of 75% threshold estimates obtained using BayesFit and Psignifit 4.0 compared to the distribution of true threshold values. Results show that threshold estimates obtained using BayesFit are remarkably similar to those obtained using

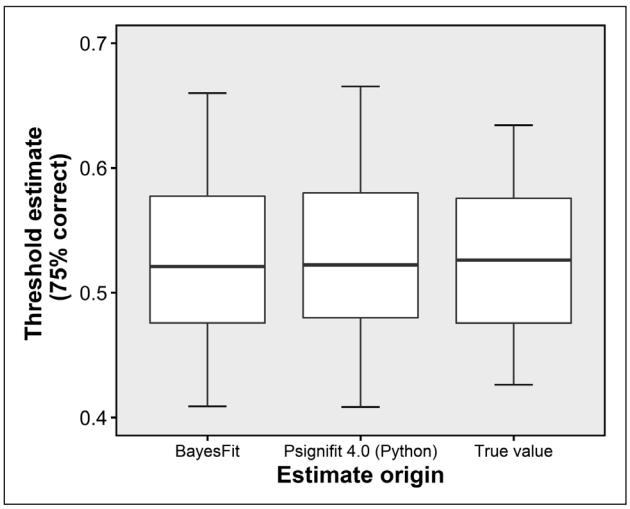

Figure 10: Boxplots of distributions for values of threshold estimated at 75% correct performance using BayesFit and Psignifit 4.0 compared to target distribution.

Psignifit 4.0, and both distributions accurately resemble the target distribution for threshold defined at 75% correct response. These results suggest that both BayesFit and Psignifit 4.0 perform similarly in estimating thresholds for data obtained from a simulated psychophysical experiment.

Implementation and architecture

BayesFit was written in Python [8], and makes extensive use of the Numpy module [2] in constructing functions for BayesFit. The core function of BayesFit called *fitmodel* computes posterior estimates using numerical integration, and serves as the main function used in fitting models. Besides this core function, BayesFit uses several utility functions to compute parameter estimates. Although most of these functions require only Numpy, probability distributions currently are defined using a subset of functions from the module SciPy [9]. BayesFit also contains several plotting functions dedicated to visualizing results and analyses which make extensive use of the module Matplotlib [10]. **Figure 11** shows a schematic of the architecture for BayesFit, along with displaying the module dependencies required by a given function.

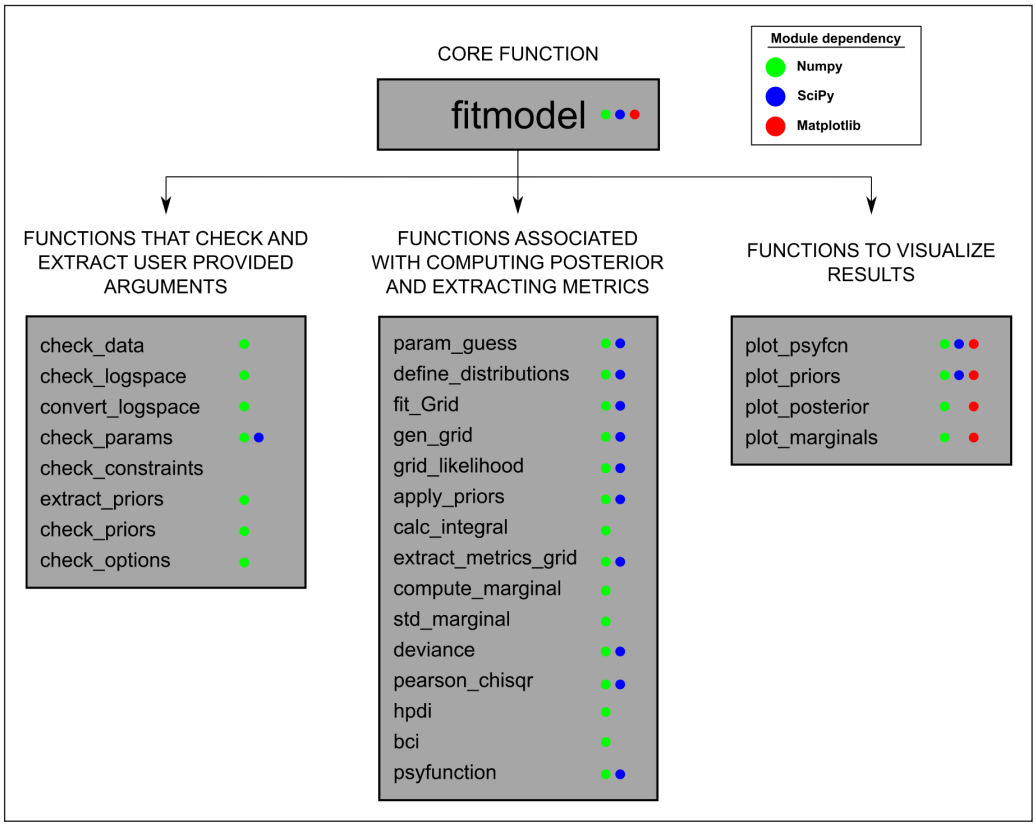

Figure 11: Schematic of the architecture for BayesFit, also displaying module dependencies for each function, whether directly or using a function that also depended upon use of a specific module.

Quality control

Both unit and functional testing has been performed on BayesFit. Travis CI (https://travis-ci.org/) is used to perform regular unit tests on BayesFit to meet strict quality control standards. A coverage report can also be found on the GitHub repository for BayesFit via the coverage badge.

Issue tracker

Although BayesFit undergoes extensive testing to ensure the software is functioning properly, if any issues arise during use, please let us know under the issue tracker for BayesFit on Github: https://github.com/SlugocM/bayesfit/issues.

(2) Availability

Operating system

BayesFit should function on any software capable of running Python 3.5 or greater. Current operating systems that BayesFit has been tested on include macOS Sierra, Windows 10, and Ubuntu 16.04.

Programming language

Python (version 3.5 or greater.)

Additional system requirements

The minimum system requirements needed to run Python 3.5 or greater.

Dependencies

numpy >= 0.19.0
matplotlib >= 2.0.0
scipy >= 1.1.0

List of contributors

Michael Slugocki is the sole developer, and current maintainer of the BayesFit module. The code for this software has been developed in the lab of Dr. Allison B. Sekuler and Dr. Patrick J. Bennett, who are co-authors of this article.

Software location

Code repository

Name: GitHub
Persistent identifier: https://github.com/slugocm/bayesfit
Licence: Apache 2.0
Date published: 02/10/17

Language

English

(3) Reuse potential

The need to estimate parameters of models fit to psychophysical data exists in many different domains within Psychology. Therefore, the potential for reuse of this software to be high. The core code of BayesFit has been written using only the Numpy module, and therefore many functions in BayesFit can be used in other projects that want to use numerical integration and Bayesian inference in modelling psychophysical data. Some highlighted functionality includes:

1. Checking user arguments for fitting psychometric functions.

2. Extracting user provided prior distributions.
3. Defining psychometric function definitions.
4. Computing likelihood surface via numerical integration.
5. Computing posterior distribution via numerical integration.
6. Compute various parameters from fitted psychometric function (e.g., MAP estimate, Bayesian Credible Intervals, etc).
7. Plotting model fits from parameter estimates.
8. Plotting prior, marginal, and posterior distributions.

Acknowledgements

We thank the editorial team and reviewers for their insightful comments and efforts spent towards improving this manuscript, and the BayesFit module.

Competing Interests

The authors have no competing interests to declare.

References

1. **Schütt, H, Harmeling, S, Macke, J** and **Wichmann, F** 2015 Psignifit 4: Pain-free Bayesian inference for psychometric functions. In: *15th Annual Meeting of the Vision Sciences Society (VSS 2015)*. DOI: https://doi.org/10.1167/15.12.474
2. **Oliphant, T E** 2006 *Aguide to NumPy*. USA: Trelgol Publishing.
3. **Wichmann, F A** and **Hill, N J** 2001 The psychometric function: I. Fitting, sampling, and goodness of fit. *Attention, Perception, & Psychophysics*, 63(8): 1293–1313. DOI: https://doi.org/10.3758/BF03194544
4. **Prins, N** and **Kingdon, F A A** 2009 Palamedes: Matlab routines for analyzing psychophysical data. http://www.palamedestoolbox.org.
5. **Linares, D** and **López-Moliner, J** 2016 Quickpsy: An R package to fit psychometric functions for multiple groups. *The R Journal*, 8(1): 122–131.
6. **Prins, N** 2016 *Psychophysics: A practical introduction*. Academic Press.
7. **Alcalá-Quintana, R** and **García-Pérez, M A** 2004 The role of parametric assumptions in adaptive Bayesian estimation. *Psychological Methods*, 9(2): 250. DOI: https://doi.org/10.1037/1082-989X.9.2.250
8. **Van Rossum, G** 2007 June Python Programming Language. In: *USENIX Annual Technical Conference*, 41: 36.
9. **Jones, E, Oliphant, E, Peterson, P,** et al. 2001 SciPy: Open Source Scientific Tools for Python. http://www.scipy.org/ [Online; accessed 2018-09-17].
10. **Hunter, J D** 2007 Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9: 90–95. DOI: https://doi.org/10.1109/MCSE.2007.55

How to cite this article: Slugocki, M, Sekuler, A B and Bennett, P J 2019 BayesFit: A Tool for Modeling Psychophysical Data Using Bayesian Inference. *Journal of Open Research Software*, 7: 2. DOI: https://doi.org/10.5334/jors.202

Submitted: 02 November 2017 **Accepted:** 27 November 2018 **Published:** 17 January 2019