**Guide to variational Bayesian parameter estimation in Python**

Version v1.0

Manuel Eichenlaub

13/11/2020

Table of Contents

[1 Introduction 2](#_Toc56451758)

[2 Model specification 3](#_Toc56451759)

[2.1 Model formulation 3](#_Toc56451760)

[2.2 Parameter distributions 4](#_Toc56451761)

[3 Python implementation 5](#_Toc56451762)

[3.1 Main function 5](#_Toc56451763)

[3.2 Model functions 7](#_Toc56451764)

[3.3 Model simulation 8](#_Toc56451765)

[4 References 9](#_Toc56451766)

# Introduction

Variational Bayesian (VB) analysis provides a technique for fully Bayesian parameter estimation in nonlinear models formulated with ordinary differential equations (ODEs). This means that all unknown parameters are considered to be continuous random variables following a parametric probability density function (PDF). In addition to the unknown parameters, the VB method provides a lower bound to the marginal likelihood, also known as model evidence, which can be used for model selection. This lower bound is known as Free Energy and is iteratively maximised by the VB method. The VB method is fully deterministic thereby using semi-analytical approximations to the true posterior distributions and provides an efficient alternative to Markov Chain Monte Carlo approaches. The VB approach for the inversion of stochastic ODE models has been published in [1] and implemented in an open-source MATLAB toolbox [2].

The purpose of the here described software package is to provide an implementation of VB method in Python. The main difference to the MATLAB toolbox is that we only allow the inversion of deterministic ODE models. Additionally, the focus lies on the inversion of models used in physiology.

This guide describes the form of the models that can be inverted, the details of the Python software, i.e. its in- and outputs and illustrates its use in a few examples.

# Model specification

## Model formulation

The models that can be inverted with the present toolbox take the following form

|  |  |
| --- | --- |
|  | (2.1) |

where is the vector of model states ( is the model order), to the possible nonlinear functions describing the model evolution, a vector of unknown evolution parameters and the known inputs. The time is thereby considered to be an known input. The unknown initial conditions are described by vector

The process of observing the model, i.e. collecting data, is described as follows

|  |  |
| --- | --- |
|  | (2.2) |

where to are the observations described by possibly nonlinear functions to , partially dependant on the observation parameters in vector **.** The observations are affected by the additive noise vector following a normal distribution with zero mean

|  |  |
| --- | --- |
|  | (2.3) |

The unknown parameter describes the measurement noise precision and the matrix of size describes any known and possibly time dependant covariances between the errors of different observations.

To implement the model in Python it is necessary to specify functions to as well as functions to . Additionally, some derivates of these functions are required. Firstly, for the ODEs, the derivatives of the functions to with respect to states and evolution parameters must be specified in the form of

|  |  |
| --- | --- |
|  | (2.4) |

where is a and is a matrix.

Secondly, for the observations, the derivatives of the functions to with respect to states and observation parameters must be specified, i.e.

|  |  |
| --- | --- |
|  | (2.5) |

where is a and is a matrix.

## Parameter distributions

The unknown parameters that a can be estimated with the VB method are the evolution parameters , the initial conditions , the observation parameters and the measurement noise precision . Parameter groups **,** and thereby follow a Gaussian PDF (prior and posterior) specified with vectors of means and covariance matrices. The measurement noise precision follows a Gamma PDF (prior and posterior) specified by shape and rate parameters and .

# Python implementation

The software package was written in Python 3.8 and has only been tested with this Python version. It is heavily based on the MATLAB implementation [2].

## Main function

All vectors are defined as *ndarrays* from the package *numpy*. The main script is called *VBA.py* and needs to be imported from the folder *Functions*. The main function to be called from this script is called *main* and takes the following inputs:

* *t*: array specifying the ODE integration time grid used for the numerical solution of the ODE model. The first element must be zero and the time steps must be equally spaced.
* *data*: A dictionary containing the following keys:
  + *“y”*: matrix containing the datapoints
  + *“t”*: array containing the time points at with the data points in “y” have been recorded. The first and last time points must lie within the integration time grid previously specified
  + *“u” (optional):*  array containing the known inputs specified on the integration time grid *t*. If the time *t* is explicitly used in the model formulation, it can be implemented as a known input.
* *priors:* A dictionary specifying the prior distributions over the unknown parameters containing the following keys:
  + *“a”:* shape parameter of prior Gamma PDF of the measurement noise precision
  + *“b”:* scale parameter of prior Gamma PDF of the measurement noise precision
  + *“muTheta”*: array containing the prior means of the normally distributed evolution parameters **.**
  + *“SigmaTheta”*: matrix containing the prior covariance matrix of the normally distributed evolution parameters **.** If any diagonal elements are specified as zero the respective parameter is not updated during inversion.
  + *“muX0”*: array containing the prior means of the normally distributed initial conditions **.**
  + *“SigmaX0”*: matrix containing the prior covariance matrix of the normally distributed initial conditions **.** If any diagonal elements are specified as zero the respective parameter is not updated during inversion.
  + *“muPhi”*: array containing the prior means of the normally distributed observation parameters **.**
  + *“SigmaPhi”*: matrix containing the prior covariance matrix of the normally distributed observation parameters **.** If any diagonal elements are specified as zero the respective parameter is not updated during inversion.
  + *“iQy”* (optional): A Python list of matrices specifying the time dependant measurement noise covariance matrix **.** If this key is not specified, a list of identity matrices is used. This assumes that the measurement errors at every measurement time point follow the distribution .
* *options*: A dictionary specifying the model inversion options with the following keys. Those marked as optional can be left out and will automatically be set to the default value:
  + *“dim”*: A dictionary specifying the model dimensions with the following keys:
    - *“n”*: Model order
    - *“n\_theta”:* Number of evolution parameters
    - *“n\_phi”:* Number of observation parameters
  + *“f\_model”:* User defined function specifying the ODEs from equations (2.1) (more details later)
  + *“f\_obs”:* User defined function specifying the observation functions from equations (2.2) (more details later)
  + *“inF”* (optional)*:* A dictionary that is passed to the evolution function *f\_model* and can contain user specified constants or other information
  + *“inG”* (optional)*:* A dictionary that is passed to the observation function *f\_obs* and can contain user specified constants or other information
  + *“ODESolver”* (optional)*:* A string that specifies either “Euler” (default) if the Euler method shall be used to numerically solve the ODE or “RK” if the fourth-order Runge-Kutta method shall be used to solve the ODE. NB: The Runge-Kutta method is more accurate for the same integration step size but is computationally more expensive.
  + *"MaxIter"* (optional): Maximum number of iterations of the main free energy maximisation scheme. Default: 32
  + *"GnTolFun"* (optional): Minimal change in the free energy that terminates the main free energy maximisation scheme. Default: 10-5
  + *"GnMaxIter"* (optional): Maximum number of iterations of the Gauss-Newton optimisation scheme with the main free energy maximisation. Default: 32
  + *"GnTolFun"* (optional): Minimal change in the variational energy that terminates the Gauss-Newton optimisation. Default: 10-5
  + *“updateHP”* (optional): Boolean variable (*True* or *False*) that specifies whether the PDF of the measurement noise precision shall be updated during model inversion or kept at its prior PDF. Default: *True*
  + *“Display”* (optional): Boolean variable (*True* or *False*) that specifies whether a figure showing the inversion process in shown. NB: Displaying the figure slows the inversion process. Default: *True*
  + *“verbose”* (optional): Boolean variable (*True* or *False*) that specifies whether intermediate steps of the model inversion are printed to the terminal. Default: *True*

The outputs of the *main* function are as follows:

* *posterior*: Dictionary of containing the posterior distributions of the unknown parameters analogous to the *priors* dictionary
* *out*: Dictionary containing additional inversion results in the following keys:
  + *“it”*: Number of iterations of the main free energy maximisation routine
  + *“F”*: Free energy, i.e. lower bound on the log model evidence of the inverted model
  + *“ModelOut”*: Dictionary containing various information of the model evolution
  + *“suffStat”:* Dictionary containing various intermediate variables calculated during model inversion

Future versions of the software package will include additional model performance measures such as RMSE, R2, BIC, AIC, etc., as well as posterior correlation matrices of the unknown parameters

## Model functions

The Python functions specifying the ODEs and observations must be specified in separate function and passed the VB routine via the options dictionary, as previously described. Both functions should be specified in a single Python script appropriately named.

The function *f\_model* specifies the ODE evolution and takes the following inputs:

* *X*: array giving the current state of the model at any given time on the integration time grid
* *th:* array containing the current means of the normally distributed evolution parameters
* *u:* array giving the known inputs at the current integration time step
* *inF:* Dictionary containing additional information specified in the *options* dictionary.

The outputs are as follows:

* *dx*: array implementing functions to
* *dFdX:* matrix containing the derivatives of the functions to with respect to the model states as defined in expression (2.4)
* *dFdTh:* matrix containing the derivatives of the functions to with respect to the evolution parameters as defined in expression (2.4)

The function *f\_obs* specifies the observation equations and takes the following inputs:

* *X*: array giving the current state of the model at any given time on the integration time grid
* *phi:* array containing the current means of the normally distributed evolution parameters
* *u:* array giving the known inputs at the current integration time step
* *inG:* Dictionary containing additional information specified in the *options* dictionary.

The outputs are as follows:

* *dx*: array implementing functions to
* *dGdX:* matrix containing the derivatives of the functions to with respect to the model states as defined in expression (2.5)
* *dGdPhi:* matrix containing the derivatives of the functions to with respect to the observation parameters as defined in expression (2.5)

## Model simulation

In order to simulate a model for any given parameter values and settings the function *simulate* in the main *VBA.py* script can be used. The function takes the following inputs:

* *td*: array specifying the time points at which observations are to be calculated
* *t:* ODE integration time grid, identical to the input of the *main* function
* *u:*  array containing the known inputs specified on the integration time grid *t*, identical to the input of the *main* function. If no input is specified, please provide an empty list as *[]*
* *priors:* Dictionary containing the prior distributions of the unknown model parameters, identical to the input of the *main* function. The *simulate* function only uses the prior means for the simulations.
* *options:* see input of *main* function
* *plotting:* Boolean variable (True or False) specify whether the simulation results are displayed.

# References

|  |  |
| --- | --- |
| [1] | J. Daunizeau, K. Friston and S. Kiebel, “Variational Bayesian identification and prediction of stochastic nonlinear dynamic causal models,” *Physica D,* pp. 2089-2118, 238 2009. |
| [2] | J. Daunizeau und L. Rigoux, „VBA (Variational Bayesian Analysis),“ [Online]. Available: https://mbb-team.github.io/VBA-toolbox/. |