Guide to variational Bayesian parameter estimation in Python

Version v1.0

Manuel Eichenlaub

13/11/2020

Table of Contents

1 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 log 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.

2 Model specification

2.1 Model formulation

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

$$\begin{aligned} \frac{dx_1(t)}{dt} &= f_1(\boldsymbol{x}(\boldsymbol{t}), \boldsymbol{u}, \boldsymbol{\theta}) \qquad x_1(0) = x_{01}, \\ &\vdots \\ \frac{dx_n(t)}{dt} &= f_n(\boldsymbol{x}(\boldsymbol{t}), \boldsymbol{u}, \boldsymbol{\theta}) \qquad x_n(0) = x_{0n}, \end{aligned} \tag{2.1}$$

where $\boldsymbol{x}$ is the vector of n model states (n is the model order), f_1 to f_n the possible nonlinear functions describing the model evolution, $\boldsymbol{\theta}$ a vector of n_θ unknown evolution parameters and $\boldsymbol{u}$ the known inputs. The time t is thereby considered to be an known input. The unknown initial conditions are described by vector $\boldsymbol{x_0}$.

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

$$\begin{aligned} y_1 &= g_1(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\varphi}) + \varepsilon_1, \\ &\vdots \\ y_{n_y} &= g_{n_y}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\varphi}) + \varepsilon_{n_y}, \end{aligned} \tag{2.2}$$

where $y_1$to y_{n_y} are the n_y observations described by possibly nonlinear functions g_1 to g_{n_y}, partially dependant on the n_φ observation parameters in vector $\boldsymbol{\varphi}$**.** The observations are affected by the additive noise vector $\boldsymbol{\varepsilon}$ following a normal distribution with zero mean

$$\boldsymbol{\varepsilon} \sim \mathcal{N}(0;\ \sigma^{-1}\boldsymbol{Q}(\boldsymbol{t})^{-\mathbf{1}}) \tag{2.3}$$

The unknown parameter σ describes the measurement noise precision and the matrix $\boldsymbol{Q}$ of size $[n_y \times n_y]$ 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 f_1 to f_n as well as functions g_1 to g_{n_y}. Additionally, some derivates of these functions are required. Firstly, for the ODEs, the derivatives of the functions f_1 to f_n with respect to states $\boldsymbol{x}$ and evolution parameters $\boldsymbol{\theta}$ must be specified in the form of

$$\boldsymbol{dFdX} = \begin{bmatrix} \frac{df_1}{dx_1} & \cdots & \frac{df_1}{dx_n} \\ \vdots & \ddots & \vdots \\ \frac{df_n}{dx_1} & \cdots & \frac{df_n}{dx_{1n}} \end{bmatrix} \quad and \quad \boldsymbol{dFd\theta} = \begin{bmatrix} \frac{df_1}{d\theta_1} & \cdots & \frac{df_1}{d\theta_{n_\theta}} \\ \vdots & \ddots & \vdots \\ \frac{df_n}{d\theta_1} & \cdots & \frac{df_n}{d\theta_{n_\theta}} \end{bmatrix} \tag{2.4}$$

where $\boldsymbol{dFdX}$ is a $[n \times n]$ and $\boldsymbol{dFd\theta}$ is a $[n \times n_\theta]$ matrix.

Secondly, for the observations, the derivatives of the functions g_1 to g_{n_y} with respect to states $\boldsymbol{x}$ and observation parameters $\boldsymbol{\varphi}$ must be specified, i.e.

$$\boldsymbol{dGdX} = \begin{bmatrix} \frac{dg_1}{dx_1} & \cdots & \frac{dg_1}{dx_n} \\ \vdots & \ddots & \vdots \\ \frac{dg_{n_y}}{dx_1} & \cdots & \frac{dg_{n_y}}{dx_n} \end{bmatrix} \quad and \quad \boldsymbol{dGd\varphi} = \begin{bmatrix} \frac{dg_1}{d\varphi_1} & \cdots & \frac{dg_1}{d\varphi_{n_\varphi}} \\ \vdots & \ddots & \vdots \\ \frac{dg_{n_y}}{d\varphi_1} & \cdots & \frac{dg_{n_y}}{d\varphi_{n_\varphi}} \end{bmatrix} \tag{2.5}$$

where $\boldsymbol{dGdX}$ is a $[n_y \times n]$ and $\boldsymbol{dGd\theta}$ is a $[n_y \times n_\varphi]$ matrix.

2.2 Parameter distributions

The unknown parameters that a can be estimated with the VB method are the evolution parameters $\boldsymbol{\theta}$, the initial conditions $\boldsymbol{x_0}$, the observation parameters $\boldsymbol{\varphi}$ and the measurement noise precision σ. Parameter groups $\boldsymbol{\theta}$, $\boldsymbol{x_0}$ and $\boldsymbol{\varphi}$ thereby follow a Gaussian PDF (prior and posterior) specified with vectors of means and covariance matrices. The measurement noise precision σ follows a Gamma PDF (prior and posterior) specified by shape and rate parameters a and b.

3 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]. The following python packages are required:

- numpy
- scipy
- matplotlib

3.1 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: $[1 \times n_t]$ 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"*: $[n_y \times n_D]$ matrix containing the datapoints
  - *"t"*: $[1 \times n_D]$ 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):* $[n_u \times n_t]$ array containing the n_u 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 σ
  - *"b":* scale parameter of prior Gamma PDF of the measurement noise precision σ
  - *"muTheta"*: $[n_\theta \times 1]$ array containing the prior means of the normally distributed evolution parameters $\boldsymbol{\theta}$**.**
  - *"SigmaTheta"*: $[n_\theta \times n_\theta]$ matrix containing the prior covariance matrix of the normally distributed evolution parameters $\boldsymbol{\theta}$**.** If any diagonal elements are specified as zero the respective parameter is not updated during inversion.
  - *"muX0"*: $[n \times 1]$ array containing the prior means of the normally distributed initial conditions $\boldsymbol{x_0}$**.**

- *"SigmaX0"*: $[n \times n]$ matrix containing the prior covariance matrix of the normally distributed initial conditions $\boldsymbol{x_0}$**.** If any diagonal elements are specified as zero the respective parameter is not updated during inversion.
- *"muPhi"*: $[n_\varphi \times 1]$ array containing the prior means of the normally distributed observation parameters $\boldsymbol{\varphi}$**.**
- *"SigmaPhi"*: $[n_\varphi \times n_\varphi]$ matrix containing the prior covariance matrix of the normally distributed observation parameters $\boldsymbol{\varphi}$**.** If any diagonal elements are specified as zero the respective parameter is not updated during inversion.
- *"iQy"* (optional): A Python list of n_D $[n_y \times n_y]$ matrices specifying the time dependant measurement noise covariance matrix $\boldsymbol{Q(t)}$**.** If this key is not specified, a list of n_D $[n_y \times n_y]$ identity matrices is used. This assumes that the measurement errors at every measurement time point follow the distribution $\boldsymbol{\varepsilon} \sim \mathcal{N}(0;\ \sigma^{-1}\boldsymbol{I})$.

- *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
    - *"n_theta":* Number of evolution parameters n_θ
    - *"n_phi":* Number of observation parameters n_φ
  - *"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 σ 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, R^2, BIC, AIC, etc., as well as posterior correlation matrices of the unknown parameters

3.2 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*: $[n \times 1]$ array giving the current state of the model $\boldsymbol{x}$ at any given time on the integration time grid

- *th:* $[n_\theta \times 1]$ array containing the current means of the normally distributed evolution parameters $\boldsymbol{\theta}$
- *u:* $[n_u \times 1]$ 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*: $[n \times 1]$ array implementing functions f_1 to f_n
- *dFdX:* $[n \times n]$ matrix containing the derivatives of the functions f_1 to f_n with respect to the model states $\boldsymbol{x}$ as defined in expression (2.4)
- *dFdTh:* $[n \times n_\theta]$ matrix containing the derivatives of the functions f_1 to f_n with respect to the evolution parameters $\boldsymbol{\theta}$ as defined in expression (2.4)

The function *f_obs* specifies the observation equations and takes the following inputs:

- *X*: $[n \times 1]$ array giving the current state of the model $\boldsymbol{x}$ at any given time on the integration time grid
- *phi:* $[n_\varphi \times 1]$ array containing the current means of the normally distributed evolution parameters $\boldsymbol{\varphi}$
- *u:* $[n_u \times 1]$ 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*: $[n \times 1]$ array implementing functions g_1 to g_{n_y}
- *dGdX:* $[n_y \times n]$ matrix containing the derivatives of the functions g_1 to g_{n_y} with respect to the model states $\boldsymbol{x}$ as defined in expression (2.5)
- *dGdPhi:* $[n_y \times n_\varphi]$ matrix containing the derivatives of the functions g_1 to g_{n_y} with respect to the observation parameters $\boldsymbol{\varphi}$ as defined in expression (2.5)

3.3 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*: $[1 \times n_D]$ 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:* $[n_u \times n_t]$ array containing the n_u 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.

Output is the following variable:

- *yd*: $[n_y \times n_D]$ matrix containing the simulated datapoints

4 Examples

4.1 Van der Pol Oscillator

Based on an example in the MATLAB VB toolbox, we also use the van der Pol oscillator described by the following ODEs with a single unknown evolution parameter:

$$\begin{aligned} \frac{dx_1(t)}{dt} &= x_2(t) & x_1(0) &= x_{01} \\ \frac{dx_2(t)}{dt} &= \theta_1(1 - x_1(t)^2)x_2(t) - x_1(t) & x_2(0) &= x_{02} \end{aligned} \tag{4.1}$$

The derivatives of the ODEs with respect to the states and evolution parameters are as follows

$$\boldsymbol{dFdX} = \begin{bmatrix} 0 & 1 \\ -2\theta_1 x_1(t)x_2(t) - 1 & \theta_1(1 - x_1(t)^2) \end{bmatrix}$$
$$\boldsymbol{dFd\theta} = \begin{bmatrix} 0 \\ (1 - x_1(t)^2)x_2(t) \end{bmatrix} \tag{4.2}$$

The system is partially observed trough a logistic function with unknown slope

$$g(x) = \frac{1}{1 + \exp(-\varphi_1 x_1(t))} \tag{4.3}$$

The derivatives of the observation equations with respect to the states and evolution parameters are as follows.

$$\boldsymbol{dGdX} = \begin{bmatrix} \dfrac{\varphi_1 \exp(-\varphi_1 x_1(t))}{\left(1 + \exp\left(-\varphi_1 x_1(t)\right)\right)^2} & 0 \end{bmatrix}$$
$$\boldsymbol{dGd\varphi} = \frac{x_1 \exp(-\varphi_1 x_1(t))}{\left(1 + \exp\left(-\varphi_1 x_1(t)\right)\right)^2} \tag{4.4}$$

We recommend using a symbolic computation tool such as Mathematica or Wolfram Alpha (https://www.wolframalpha.com/) to compute the derivatives, as they are crucial to the success of the model inversion. Future versions of the software package might include the option to have the derivatives calculated numerically.

This means we have the following model dimensions

$$n = 2 \qquad n_\theta = 1 \qquad n_\varphi = 1 \tag{4.5}$$

We choose the following priors

$$\varphi_1 \sim \mathcal{N}(0.1; 1)$$ $$\theta_1 \sim \mathcal{N}(0.1; 1)$$ $$\boldsymbol{x_0} \sim \mathcal{N}(\begin{bmatrix}0.1\\0.1\end{bmatrix}; \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix})$$ $$\sigma \sim Gamma(0.1; 0.1)$$	(4.6)

It is often useful to set the prior means so that a certain amount of dynamical behaviour is induced in the model output, i.e. it is not recommended to set the priors means of either parameter to zero.

Simulating the model with the values

$$\varphi_1 = 2$$ $$\theta_1 = 1$$ $$\boldsymbol{x_0} = \begin{bmatrix}1\\1\end{bmatrix}$$ $$\sigma = 500$$	(4.7)

the following inversion results are obtained

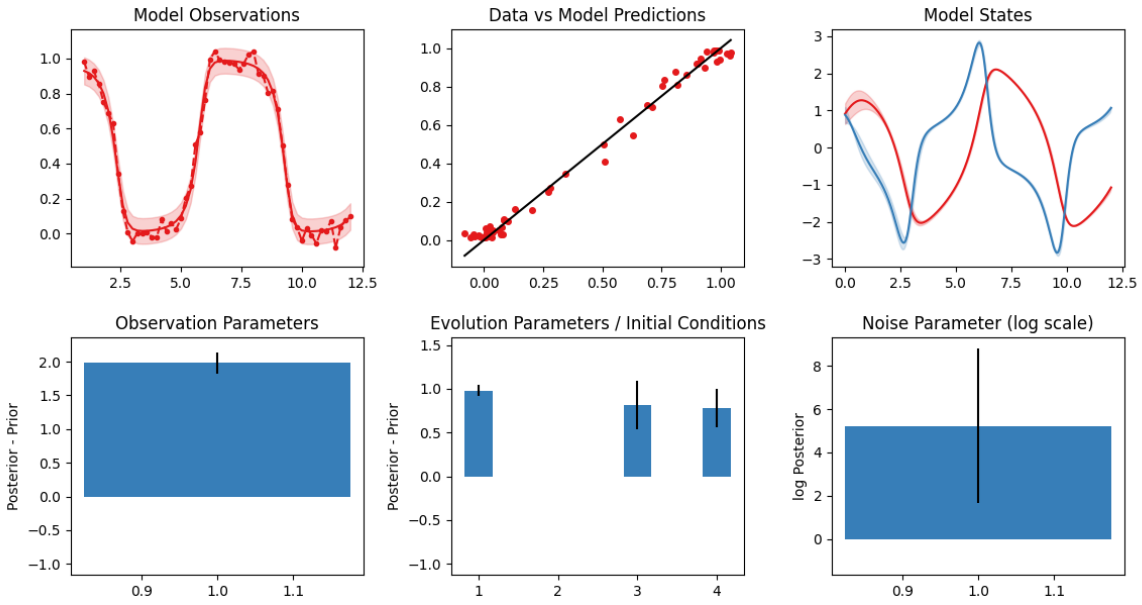

Figure 1: Final state of the inversion figure window for the van der Pol oscillator model

Comparing the inversion results to the true values used for simulation leads to the following results

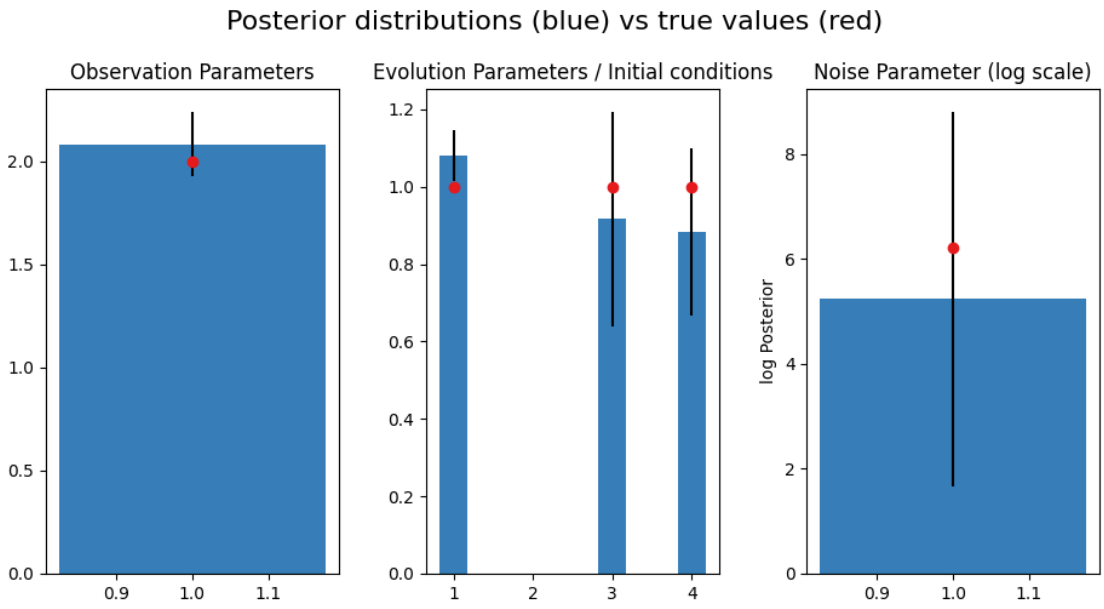

Figure 2: Comparison between the posterior distributions over unknown parameters and the true values used for simulation.

5 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/.