Bayesian analysis with BF

This document is a guide to Bayesian analysis and the implementation of *BayesForge* (BF) package. It is intended for users ranging from those with little or no experience to advanced practitioners. In this introduction, we outline the main steps of Bayesian analysis. Each of the subsequent chapters present increasingly complex models. Each chapter will have the same structure in order to allow users to easily find the information they are looking for. The structure is as follows:

1. *Introduction to the model with non-mathematical explanations.*
2. *Specific considerations for the model.*
3. *Code to run the model.*
4. *Mathematical representation of the model.*
5. *Notes related to the model.*

We recommend reading the introduction first since some key concepts here will not be revisited in later chapters.

Bayesian analysis

Bayesian analysis is a statistical approach that uses probability theory to update beliefs about the parameters of a model as new data become available. Bayesian methods have several key advantages, as they allow direct uncertainty quantification , incorporation of prior knowledge and flexibility for complex models .

Model set-up, Bayesian linear regression example

To illustrate the basics of setting up a Bayesian model, let's consider a simple linear regression example. In this example, we build a simple model where we predict a dependent variable Y from a single independent variable X. Here, we will not focus on the equations and their details, but rather describe the different components of a Bayesian model, show how parameter estimations algorithms work in Bayesian analysis, and provide basic technical vocabulary related to Bayesian methods. For further details on the mathematical formulas of the model, please refer to Chapter 1: Linear Regression for continuous variable.

The Likelihood

The *likelihood* can be considered as your main equation that combines the different terms of the model. Depending on the type of problem you are trying to solve (classification, regression, etc.) and the type of data you are working with (continuous, discrete, binomial, etc.), the likelihood will be different. Additionally, a model can have multiple likelihoods (e.g., network models).

As both the *dependent variable* Y and the *independent variable* X are continuous variables, we can use a *Gaussian model* to describe the relationship between these two variables. As we are using Bayesian methods, we are describing this relationship using a *probability distribution* :

$$Y \sim \text{Normal}(\mu, \sigma)$$

Basically, our likelihood is saying that the *dependent variable* Y is normally distributed with a mean μ and a standard deviation σ. *Probability distributions* are available in BF through the class `bf.dist.XXX` where `XXX` is the name of the distribution you want to use. Within any `bf.dist.XXX` class, you can define which distribution needs to be used as a likelihood by adding an argument `obs`, e.g., `b.dist.normal(alpha + beta * x, sigma, obs=y)`. You can see this as instructing the model to adjust its parameters to best explain the observed dependent variable Y, given the independent variable X, by maximizing the likelihood.

Modeling Likelihood

Once the *likelihood* is defined, we can now define the mathematical equations that describe our parameters (μ and σ) and their relationship with the *dependent variable* Y. We can express this relationship in the form of a linear function:

$$\mu = \alpha + \beta X$$

Where α is the *intercept* and β is the *slope* of the regression line. These parameters are the *unknowns* that we want to estimate to evaluate the strength and direction of the relationship between the *independent variable* X and the *dependent variable* Y.

Link functions

Depending on the type of problem you are trying to solve (classification, regression, etc.) and the type of data you are working with (continuous, discrete, binomial, etc.), you will need to choose the appropriate distribution to describe the relationships in the data. For each different outcome distribution, you will need to use an appropriate *link function* .

For the moment, we just need to know that these different distributions require a *link function* (for each specific family we will discuss the corresponding link function in their respective chapters); however, below is a table summarizing some of the most common link functions, the mathematical form of each, their typical applications, and how to interpret them. *Link functions* in BF can be accessed through the class `BF.link.XXX` where `XXX` is the name of the link function. Let μ be a real number output from a linear model, and $g(\mu)$ be the corresponding link function. Then the table shows the most common link functions and their interpretations.

Link Function	**Mathematical Form of Inverse Link**	**Typical Use / Model**	**Interpretation & Range**
Identity	$g(\mu) = \mu$	Linear regression (Normal)	Directly models μ; the outputs space is also the set of real numbers.
Logit	$g(\mu) = \frac{1}{1+\exp(-\mu)}$	Logistic regression (Binomial)	The output space is the unit interval $[0, 1]$; coefficients reflect log-odds.
Probit	$g(\mu) = \Phi(\mu)$	Probit regression (Binomial)	Similar to logit; uses the standard normal CDF, Φ.
Log	$g(\mu) = \exp(\mu)$	Poisson, Gamma regression (Count data)	The output space is the set of positive reals.

The Prior Distributions

For each parameter in our model, we need to define a *prior distribution* that encodes our initial beliefs about the parameter. In the case of the linear regression model, we need to specify *prior distributions* for the intercept, α, the slope, β, and the standard deviation, σ.

$$\alpha \sim \text{Normal}(0, 1)$$

$$\beta \sim \text{Normal}(0, 1)$$

$$\sigma \sim \text{Exponential}(1)$$

And with this, we can write our entire model as:

$$Y \sim \text{Normal}(\mu, \sigma)$$
$$\mu = \alpha + \beta X$$
$$\alpha \sim \text{Normal}(0, 1)$$
$$\beta \sim \text{Normal}(0, 1)$$
$$\sigma \sim \text{Exponential}(1)$$

In BF, you code all this statements within a single function. In that function you can use any probability distribution, link function, and mathematical operations required for your model (if they are supported by *JAX*). BF has been designed to allow you to declare your model as close as possible to the mathematical notation. For example, the model above can be written in BF as:

Python

```
from BayesForge import bf
import jax.numpy as jnp

m = bf(platform='cpu')
beta = 2.5
alpha = 0.5
sigma = 1.0
X = m.dist.normal(0, 1, sample = True)
Y = m.dist.normal(alpha + beta * X, sigma, sample = True)

m.data_on_model = dict(X=X, Y=Y)

def model(X, Y):
    alpha = m.dist.normal( 0, 1, name = 'alpha', shape= (1,))
```

```
    beta = m.dist.normal( 0, 1, name = 'beta', shape= (1,))
    sigma = m.dist.exponential(1, name = 'sigma', shape = (1,))
    m.dist.normal(alpha + beta * X, sigma, obs=Y)
```

```
bf v 0.0.48 package loaded
```

```
E0527 08:58:19.539349 1346861 cuda_dnn.cc:523] Loaded runtime CuDNN library: 9.1.0 but source
E0527 08:58:19.541416 1346861 cuda_dnn.cc:523] Loaded runtime CuDNN library: 9.1.0 but source
```

```
jax.local_device_count 32
```

R

```
library(BayesForge)

m = importBF(platform="cpu")
beta = 2.5
alpha = 0.5
sigma = 1.0
X = bf.dist.normal(0, 1, sample = TRUE)
Y = bf.dist.normal(alpha + beta * X, sigma, sample = TRUE)

m.data_on_model = list(X=X, Y=Y)

mode <- function(X, Y){
    alpha = bf.dist.normal( 0, 1, name = 'alpha', shape= c(1))
    beta = bf.dist.normal( 0, 1, name = 'beta', shape= c(1))
    sigma = bf.dist.exponential(1, name = 'sigma', shape = c(1))
    bf.dist.normal(alpha + beta * X, sigma, obs=Y)
}

m$fit(mode)
```

Julia

```
using BayesForge

m = importBF(platform="cpu")
```

```
beta = 2.5
alpha = 0.5
sigma = 1.0
X = m.dist.normal(0, 1, sample = true)
Y = m.dist.normal(alpha + beta * X, sigma, sample = true)

m.data_on_model = Dict("X" => X, "Y" => Y)

@BF function model(X, Y)
    alpha = m.dist.normal(0, 1, name="alpha", shape=(1,))
    beta = m.dist.normal(0, 1, name="beta", shape=(1,))
    sigma = m.dist.exponential(1, name="sigma", shape=(1,))
    m.dist.normal(alpha + beta * X, sigma, obs=Y)
end

m.fit(model)
```

The code snippet provides several key features of the *BF* package:

1. First, you need to initialize a BF object.
2. Then, you can store data as a JAX array dictionary using the `m.data_on_model` function. If all the data can be stored in a data frame (e.g., Linear Regression for continuous variable), you do not need to use `m.data_on_model`, as the BF object automatically detects the data provided in the model arguments. However, sometimes you may need different data structures such as vectors and 2D arrays (e.g., Network model).
3. Regarding distribution parameters, note the difference depending on whether you are generating data outside a function (e.g., for simulation purposes) or specifying priors inside a model function. In the former case, the argument `sample` should be set to `True`. However, if you are specifying priors within a model function, this argument is `False` by default.
4. Finally, note that each parameter declared in the model must have a unique `name` as well as a `shape`. The shape refers to the number of parameters you want to estimate. For example, if you want to estimate a different β for each independent variable, you would declare β with a shape equal to the number of independent variables. By default, the shape is one, so technically you don't need to specify it. In this example, we highlight this feature explicitly.

Which prior distribution range to use?

The choice of prior ranges can significantly affect Bayesian analysis results. There are several approaches to selecting them:

- **Expert Knowledge**: The prior distributions can be based on expert knowledge or historical data. This approach is useful when there is a lot of information available about the parameters.
- **Noninformative Priors**: When there is little or no information about the parameters, noninformative priors can be used. These priors are designed to have minimal influence on the posterior distribution, allowing the data to dominate the inference process.
- **Scaled data**: If the data are *scaled* , the prior distributions can be chosen to reflect this. For example, if the data are scaled, the prior distributions for the intercept and slope can be centered around 0 and 1, respectively. By scaling the independent variable, we obtain a unit of change based on variance; that is, the effect represents a one–standard–deviation change in X on Y. Scaling the data improves both numerical stability. When all data are scaled to the same range, it leads to more stable numerical behavior during estimation. Additionally, it facilitates setting priors that are both meaningful and relatively uninformative. By aligning the scale of the data with the scale assumed in the priors, we ensure that the posterior distributions exhibit reasonable spread and that our uncertainty quantification is consistent with the data's scale. For the remainder of the document, we will assume that the data are scaled. However, users should be aware that it is often necessary to rescale parameters estimates by the standard deviation of the data in order to get parameters that are interpretable on the natural scale.

Model fit and posterior distribution

Once data are observed, *Bayes' Theorem* is used to evaluate how well a given set of parameter values fits the data:

$$P(\theta \mid \text{data}) = \frac{P(\text{data} \mid \theta) \cdot P(\theta)}{P(\text{data})}$$

Where:

- θ represents the unknown parameters we are interested in.
- $P(\theta)$ is the **prior distribution**, representing our beliefs about θ before seeing the data.
- $P(\text{data} \mid \theta)$ is the **likelihood**, representing the model of how the data are generated given θ.

- $P(\theta \mid \text{data})$ is the **posterior distribution**, representing our updated beliefs after observing the data. It tells us not only the most likely value of θ (e.g., α, β, and σ in our case) but also quantifies the uncertainty in these estimates.

Various techniques can be used to approximate the mathematical definition of Bayes' theorem: *grid approximation*, *quadratic approximation*, and Markov chain Monte Carlo (*MCMC*). Descriptions of these algorithms are out of the scope of this document. For more information, please refer to Wikipedia. In BF, we use MCMC and it can be called as `m.fit(model)` where `model` is the function that describes the model.

Model diagnostic

Once a Bayesian model has been fit, it is crucial to evaluate how well it captures the observed data and to assess whether the Markov chain Monte Carlo (MCMC) sampling has converged. Bayesian model diagnostics help us answer questions like: "Are our uncertainty estimates reliable?", "Does the model generate data similar to what we observed?", and "Have the chains mixed well?" Multiple diagnostics approaches can be used to assess the model's performance. Below are some key diagnostic tools and techniques available in BF within the class `BF.diag.XXX` where `XXX` is the name of the diagnostic tool.

Diagnostic Tool	**Purpose**	**Key Indicator**	**Interpretation**
posterior predictive checks (PPCs)	Assess if the model can reproduce observed data	Graphs, p-values, summary stats	Good fit if simulated data resemble observed data
Credible Interval (CI)	Quantify uncertainty in parameter estimates	95% CI or other percentage	95% probability the parameter lies within the interval
highest posterior density intervals (HPDI)	Identify the narrowest interval containing a given probability mass density	95% HPDI	Smallest interval capturing 95% of the posterior density
effective sample size (ESS)	Measure independent information in the chain	ESS value (ideally high)	Low ESS indicates high autocorrelation (poor mixing) ot to few samples correlations

Diagnostic Tool	Purpose	Key Indicator	Interpretation
potential scale reduction factor (Rhat)	Check convergence across multiple chains	Rhat 1 (typically <1.01)	Values near 1 indicate convergence; >1 suggests non-convergence
Trace plots	Visualize the sampling path to check convergence and mixing	Plot showing parameter values over iterations	Stationary, 'hairy caterpillar' pattern suggests convergence
autocorrelation plots	Assess dependency between samples over lags	Autocorrelation values across lags	Rapid decay to zero suggests good mixing; slow decay indicates poor mixing
density plots	Visualize the posterior distribution of a parameter	Smoothness and shape of the curve	Unimodal and smooth suggests convergence; multimodal or irregular may suggest poor mixing

Model comparison

Model comparison is performed by evaluating how well different models explain the observed data while accounting for model complexity. Multiple criteria can be used to compare models, and are summarized in the table below. In BF, we can compare models using *Watanabe-Akaike Information Criterion (WAIC)* with the function `m.diag.waic(model1, model2)`.

Criterion	Purpose	Interpretation	Strengths	Weaknesses
DIC (Deviance Information Criterion)	Measures model fit while penalizing complexity	Lower values indicate better model fit	Simple to compute, useful for hierarchical models	Sensitive to the number of parameters, not always reliable in complex models
WAIC (Watanabe-Akaike Information Criterion)	Estimates out-of-sample predictive accuracy while penalizing complexity	Lower values indicate better models	More robust than DIC, accounts for overfitting	Computationally intensive for large models

Criterion	Purpose	Interpretation	Strengths	Weaknesses
BF (Bayes Factor)	Quantifies relative support for two models based on marginal likelihoods	`BF > 1` favors the numerator model, `BF < 1` favors the denominator	Provides direct evidence comparison, works with different model types	Sensitive to prior choices, requires good model specification