BI documentation

Sebastian Sosa

2024-09-09

Table of contents

1

2 Introduction

This document is a guide to Bayesian analysis and the implementation of Bayesian inference (BI). 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. Subsequent chapters present increasingly complex models. Each following 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 formulas 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.

3 Bayesian analysis

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

4 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 use them to describe the different components of a Bayesian model, how parameter optimization 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.

4.0.1 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 BI through the class `bi.dist.XXX` where `XXX` is the name of the distribution you want to use. Within any `bi.dist.XXX` class, you can define which distribution needs to be used as a likelihood by adding an argument `obs`, e.g., `m.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 , given the independent variable , by maximizing the likelihood.

4.1 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 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.

4.2 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 relationship between the data. By using a different distribution, you will need to use a different *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); 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 BI can be accessed through the class `BI.link.XXX` where `XXX` is the name of the link function.

Link Function	**Mathematical Form**	**Typical Use / Model**	**Interpretation & Range**
Identity	$g(\mu) = \mu$	Linear regression (Normal)	Directly models μ; μ can be any real number.
Logit	$g(\mu) = \log\left(\frac{\mu}{1-\mu}\right)$	Logistic regression (Binomial)	Models probabilities (0, 1); coefficients reflect log-odds.
Probit	$g(\mu) = \Phi^{-1}(\mu)$	Probit regression (Binomial)	Similar to logit; uses the inverse standard normal CDF.
Log	$g(\mu) = \log(\mu)$	Poisson, Gamma regression (Count data)	Ensures $\mu > 0$; coefficients represent multiplicative effects.
Inverse	$g(\mu) = \frac{1}{\mu}$	Gamma regression	Models positive μ; relates changes inversely to μ.

4.3 The Prior Distributions

For each parameter of our equation that describes μ, 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 standard deviation σ.

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

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

$$\sigma \sim \text{Uniform}(0, 1)$$

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

$$\begin{aligned} y &\sim \text{Normal}(\mu, \sigma) \\ \mu &= \alpha + \beta x \\ \alpha &\sim \text{Normal}(0, 1) \\ \beta &\sim \text{Normal}(0, 1) \\ \sigma &\sim \text{Uniform}(0, 1) \end{aligned}$$

In BI, you will need to define this model within a function in which you will be able to use any probability distributions, link functions, and mathematical operations required for your model. BI 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 BI as:

```
from BI import bi
import jax.numpy as jnp

m = bi(platform='cpu')
dat = dict(
    x = m.dist.normal(178, 20, sample = True),
    y = m.dist.lognormal( 0, 1, sample = True)
)
m.data_on_model = dat

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.uniform( 0, 50, name = 'sigma', shape = (1,))
    m.normal(alpha + beta * x, sigma, obs=y)
```

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

1. First, you need to initialize a bi 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 BI 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 must be set to `False`.
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.

4.3.1 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 is *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 and interpretability. 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 this document, we will assume that the data are scaled.

4.4 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.

We can use *Bayesian updating* using the *Bayesian theorem* to 'reshape' the prior distributions by considering every possible combination of values for our parameters, and scoring each combination by its relative plausibility in light of the data. These relative plausibilities are the posterior probabilities of each combination of our parameters: the *posterior distributions*. 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 the Bayesian Inference. In BI, we use MCMC and it can be called as `m.run(model)` where `model` is the function that describes the model.

4.5 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 BI within the class `BI.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	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)
potential scale reduction factor (Rhat)	Check convergence across multiple chains	Rhat 1 (typically <1.1)	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

Diagnostic Tool	Purpose	Key Indicator	Interpretation
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

4.6 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 BI, 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
Bayes Factor (BF)	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

5 Linear Regression for a Continuous Variable

5.1 General Principles

To study relationships between a continuous independent variable and a continuous dependent variable (e.g., height and weight), we can use linear regression. Essentially, we draw a line that passes through the point cloud of the two variables being tested. For this, we need to have:

1) An intercept α, which represents the origin of the line—the expected value of the dependent variable (height) when the independent variable (weight) is equal to zero.

2) A coefficient β, which informs us about the slope of the line. In other words, it tells us how much Y (height) increases for each increment of the independent variable (weight).

3) A variance term σ, which informs us about the spread of points around the line, i.e., the variance around the prediction.

5.2 Considerations

Caution

- Bayesian models consider model parameter uncertainty , allowing for the quantification of confidence or uncertainty through the parameters' posterior distribution . Therefore, we need to declare prior distributions for each model parameter, in this case for: α, β, and σ^2.
- Prior distributions are built following these considerations:
  - As the data is scaled (see introduction), we can use a Normal distribution for α and β, with a mean of 0 and a standard deviation of 1.
  - Since σ is strictly positive, we can use any distribution that is positively defined, such as the Exponential or Uniform distribution.
- Gaussian regression deals directly with continuous outcomes, estimating a linear relationship between predictors and the outcome variable without needing a link function (see introduction). This simplifies interpretation, as coefficients represent direct changes in the outcome variable.

5.3 Example

Below is an example code snippet demonstrating Bayesian linear regression using the Bayesian Inference (BI) package. Data consist of two continuous variables (height and weight), and the goal is to estimate the effect of weight on height.

5.4 Python

```python
from BI import bi

# Setup device------------------------------------------------
m = bi(platform='cpu')

# Import Data & Data Manipulation ------------------------------------------------
# Import
from importlib.resources import files
data_path = files('BI.resources.data') / 'Howell1.csv'
m.data(data_path, sep=';')
m.df = m.df[m.df.age > 18] # Manipulate
m.scale(['weight']) # Scale
```

```
# Define model ------------------------------------------------
def model(weight, height):
    a = m.dist.normal(178, 20, name = 'a')
    b = m.dist.lognormal(0, 1, name = 'b')
    s = m.dist.uniform(0, 50, name = 's')
    m.normal(a + b * weight , s, obs = height)

# Run mcmc ------------------------------------------------
m.run(model)  # Optimize model parameters through MCMC sampling

# Summary ------------------------------------------------
m.summary() # Get posterior distributions
```

5.5 R

```
library(BI)
m=importbi(platform='cpu')

# Load csv file
m$data(paste(system.file(package = "BI"),"/data/Howell1.csv", sep = ''), sep=';')

# Filter data frame
m$df = m$df[m$df$age > 18,]

# Scale
m$scale(list('weight'))

# Convert data to JAX arrays
m$data_to_model(list('weight', 'height'))

# Define model ------------------------------------------------
model <- function(height, weight){
  # Parameter prior distributions
  s = bi.dist.uniform(0, 50, name = 's')
  a = bi.dist.normal(178, 20,  name = 'a')
  b = bi.dist.normal(0, 1, name = 'b')

  # Likelihood
  m$normal(a + b * weight, s, obs = height)
}
```

```
# Run mcmc ------------------------------------------------
m$run(model) # Optimize model parameters through MCMC sampling

# Summary -------------------------------------------------
m$summary()
```

5.6 Mathematical Details

5.6.1 *Frequentist Formulation*

The following equation allows us to draw a line:

$$Y_i = \alpha + \beta X_i + \sigma_i$$

Where:

- Y_i is the dependent variable for observation i.
- α is the intercept term.
- β is the regression coefficient.
- X_i is the input variable for observation i.
- σ_i is the error term for observation i.

5.6.2 *Bayesian Formulation*

In the Bayesian formulation, we define each parameter with priors . We can express a Bayesian version of this regression model using the following model:

$$Y_i \sim Normal(\alpha + \beta X_i, \sigma)$$

$$\alpha \sim Normal(0, 1)$$

$$\beta \sim Normal(0, 1)$$

$$\sigma \sim Uniform(0, 50)$$

Where:

- Y_i is the dependent variable for observation i.
- α and β are the intercept and regression coefficient, respectively.
- X_i is the input variable for observation i.
- σ is the standard deviation of the normal distribution, which describes the variance in the relationship between the dependent variable Y and the independent variable X.

5.7 Notes

Note

We can observe a difference between the *Frequentist* and the *Bayesian* formulation regarding the error term. Indeed, in the *Frequentist* formulation, the error term σ_i represents random fluctuations around the predicted values. This assumption leads to point estimates for α and β, without accounting for uncertainty in these estimates. In contrast, the *Bayesian* formulation treats σ as a parameter with its own prior distribution. This allows us to incorporate our uncertainty about the error term into the model.

5.8 Reference(s)

McElreath (2018)

6 Multiple continuous variables model

6.1 General Principles

To study relationships between multiple continuous independent variables (e.g., the effect of weight and age on height), we can use a multiple regression approach. Essentially, we extend Linear Regression for continuous variable by adding a regression coefficient β_x for each continuous variable (e.g., β_{weight} and β_{age}).

6.2 Considerations

Caution

- We have the same considerations as for the Regression for continuous variable.
- The model interpretation of the regression coefficients β_x is considered for fixed values of the other independent variable(s)' regression coefficients—i.e., for a given age, β_{weight} represents the expected change in the dependent variable (height) for each one-unit increase in weight, holding all other variable(s) constant (age).

6.3 Example

Below is example code demonstrating Bayesian multiple linear regression using the Bayesian Inference (BI) package. Data consist of three continuous variables (*height*, *weight*, *age*), and the goal is to estimate the effect of *weight* and *age* on *height*.

6.3.1 Python

```
from BI import bi

# Setup device------------------------------------------------
m = bi(platform='cpu')
```

```
# Import Data & Data Manipulation ------------------------------------------------
from importlib.resources import files
# Import
data_path = files('BI.resources.data') / 'Howell1.csv'
m.data(data_path, sep=';')
m.df = m.df[m.df.age > 18] # Manipulate
m.scale(['weight', 'age']) # Scale

# Define model ------------------------------------------------
def model(height, weight, age):
    # Parameter prior distributions
    alpha = m.dist.normal(0, 0.5, name = 'alpha')
    beta1 = m.dist.normal(0, 0.5, name = 'beta1')
    beta2 = m.dist.normal(0, 0.5, name = 'beta2')
    sigma = m.dist.uniform(0,50, name = 'sigma')
    # Likelihood
    m.normal(alpha + beta1 * weight + beta2 * age, sigma, obs = height)

# Run MCMC ------------------------------------------------
m.run(model) # Optimize model parameters through MCMC sampling

# Summary ------------------------------------------------
m.summary()
```

6.3.2 R

```
library(BI)
m=importbi(platform='cpu')

# Import Data & Data Manipulation ------------------------------------------------
m$data(paste(system.file(package = "BI"),"/data/Howell1.csv", sep = ''), sep=';')# Import
m$df = m$df[m$df$age > 18,] # Manipulate
m$scale(list('weight', 'age')) # Scale
m$data_to_model(list('weight', 'height', 'age')) # Send to model (convert to jax array)

# Define model ------------------------------------------------
model <- function(height, weight, age){
  # Parameter prior distributions
  alpha = bi.dist.normal( 0, 0.5, name = 'a')
  beta1 = bi.dist.normal( 0, 0.5, name = 'b1')
  beta2 = bi.dist.normal(  0, 0.5, name = 'b2')
  sigma = bi.dist.uniform(0, 50, name = 's')
  # Likelihood
  m$normal(alpha + beta1 * weight + beta2 * age, sigma, obs=height)
}

# Run MCMC ------------------------------------------------
m$run(model) # Optimize model parameters through MCMC sampling

# Summary ------------------------------------------------
m$summary() # Get posterior distributions
```

6.4 Mathematical Details

6.4.1 *Frequentist formulation*

We model the relationship between the independent variables $(X_{1i}, X_{2i}, ..., X_{ni})$ and the dependent variable Y using the following equation:

$$Y_i = \alpha + \beta_1 X_{1i} + \beta_2 X_{2i} + ... + \beta_n X_{ni} + \sigma_i$$

Where:

- Y_i is the dependent variable for observation i.
- α is the intercept term.
- X_{1i}, X_{2i}, ..., X_{ni} are the values of the independent variables for observation i.
- β_1, β_2, ..., β_n are the regression coefficients.
- σ_i is the error term for observation i.

6.4.2 *Bayesian formulation*

In the Bayesian formulation, we define each parameter with priors . We can express the Bayesian model as follows:

$$Y \sim Normal(\alpha + \sum_{k}^{n} \beta_k X, \sigma^2)$$

$$\alpha \sim Normal(0, 1)$$

$$\beta_k \sim Normal(0, 1)$$

$$\sigma \sim Uniform(0, 50)$$

Where:

- Y_i is the dependent variable for observation i.
- α is the prior distribution for the intercept.
- β_k are the prior distributions for the k distinct regression coefficients.
- X_{1i}, X_{2i}, ..., X_{ni} are the values of the independent variables for observation i.
- σ is the prior distribution for the standard deviation, ensuring that it is positive.

6.5 Reference(s)

McElreath (2018)

7 Interaction terms

7.1 General Principles

To study the relationships between two independent continuous variables and their interaction effect on a dependent variable (e.g., temperature and humidity affecting energy consumption), we can use Regression Analysis with Interaction Terms. In this approach, we extend the simple linear regression model to include an interaction term (a multiplication) between the two continuous variables.

7.2 Considerations

Caution

- We have the same assumptions as for Regression for continuous variable.
- We wish to model the relationship between dependent variable Y and independent variable X_1 to vary as a function of independent variable X_2. To do this, we explicitly model the hypothesis that the slope between Y and X_1 depends—is conditional—upon X_2.
- For continuous interactions with scaled data, the intercept becomes the grand mean of the outcome variable.
- The interpretation of estimates is more complex. The estimate for a non-interaction term reflects the expected change in Y when X_1 increases by one unit, holding X_2 constant at its average value. The estimate for the interaction term represents how the effect of X_1 on Y changes depending on the value of X_2, and vice versa, showing how the relationship between the two variables influences the outcome Y.
- Triptych plots are very handy for understanding the impact of interactions, especially when more than two interactions are present.

7.3 Example

Below is example code demonstrating Bayesian regression with an interaction term between two continuous variables using the Bayesian Inference (BI) package. The data consist of three continuous variables (temperature, humidity, energy consumption), and the goal is to estimate the effect of the interaction between temperature and humidity on energy consumption.

7.3.1 Python

```python
from BI import bi

# Setup device------------------------------------------------
m = bi(platform='cpu')

# Import Data & Data Manipulation ------------------------------------------------
# Import
from importlib.resources import files
data_path = files('BI.resources.data') / 'tulips.csv'
m.data(data_path, sep=';')
m.scale(['blooms', 'water', 'shade']) # Scale


# Define model ------------------------------------------------
def model(blooms,shade, water):
    sigma = m.dist.exponential(1, name = 'sigma', shape = (1,))
    bws = m.dist.normal(0, 0.25, name = 'bws', shape = (1,))
    bs = m.dist.normal(0, 0.25, name = 'bs', shape = (1,))
    bw = m.dist.normal(0, 0.25, name = 'bw', shape = (1,))
    a = m.dist.normal(0.5, 0.25, name = 'a', shape = (1,))
    mu = a + bw*water + bs*shade + bws*water*shade
    m.normal(mu, sigma, obs=blooms)

# Run mcmc ------------------------------------------------
m.run(model) # Optimize model parameters through MCMC sampling

# Summary ------------------------------------------------
m.summary()
```

7.3.2 R

```r
library(BI)
m=importbi(platform='cpu')

# Load csv file
m$data(paste(system.file(package = "BI"),"/data/tulips.csv", sep = ''), sep=';')
m$scale(list('blooms', 'water', 'shade')) # Scale
m$data_to_model(list('blooms', 'water', 'shade')) # Send to model (convert to jax array)

# Define model ------------------------------------------------
model <- function(blooms, water,shade){
  # Parameter prior distributions
  alpha = bi.dist.normal( 0.5, 0.25, name = 'a')
  beta1 = bi.dist.normal( 0,  0.25, name = 'b1')
  beta2 = bi.dist.normal(  0,  0.25, name = 'b2')
  beta_interaction_ = bi.dist.normal(  0, 0.25, name = 'bint')
  sigma = bi.dist.normal(0, 50, name = 's')
  # Likelihood
  m$normal(alpha + beta1*water + beta2*shade + beta_interaction_*water*shade, sigma, obs=bloo
}

# Run mcmc ------------------------------------------------
m$run(model) # Optimize model parameters through MCMC sampling

# Summary ------------------------------------------------
m$summary() # Get posterior distributions
```

7.4 Mathematical Details

7.5 *Frequentist formulation*

We model the relationship between the input features (X_1 and X_2) and the target variable (Y) using the following equation:

$$Y_i = \alpha + \beta_1 X_{1i} + \beta_2 X_{2i} + \beta_{interaction} X_{1i} X_{2i} + \sigma$$

Where:

- Y_i is the dependent variable for observation i.

- α is the intercept term.
- X_{1i} and X_{2i} are the two values of the independent continuous variables for observation i.
- β_1 and β_2 are the regression coefficients for X_1 and X_2, respectively.
- $\beta_{interaction}$ is the regression coefficient for the interaction term $(X_1 X_2)$.
- σ is the error term, assumed to be normally distributed.

In this context, the interaction term $X_{1i} * X_{2i}$ captures the joint effect of X_{1i} and X_{2i} on the target variable Y_i.

7.5.1 *Bayesian formulation*

In the Bayesian formulation, we define each parameter with priors . We can express the Bayesian regression model accounting for prior distributions as follows:

$$Y \sim Normal(\alpha + \beta_1 X_{1i} + \beta_2 X_{2i} + \beta_{interaction} X_{1i} X_{2i}, \sigma)$$

$$\alpha \sim Normal(0, 1)$$

$$\beta_1 \sim Normal(0, 1)$$

$$\beta_2 \sim Normal(0, 1)$$

$$\beta_{interaction} \sim Normal(0, 1)$$

$$\sigma \sim Exponential(1)$$

Where:

- Y_i is the dependent variable for observation i.
- α is the prior distribution for the intercept.
- β_1, β_2, and $\beta_{interaction}$ are the prior distributions for the regression coefficients.
- X_{1i} and X_{2i} are the two values of the independent continuous variables for observation i.
- σ is the prior distribution for the standard deviation, ensuring it is positive.

7.6 Reference(s)

McElreath (2018)

8 Regression for Categorical Variables

8.1 General Principles

To study the relationship between a categorical independent variable and a continuous dependent variable, we use a *Categorical model* which applies *stratification*.

Stratification involves modeling how the k different categories of the independent variable affect the target continuous variable by performing a regression for each k category and assigning a regression coefficient for each category. To implement stratification, categorical variables are often encoded using one-hot encoding or by converting categories to indices .

8.2 Considerations

> Caution
>
> - We have the same considerations as for Regression for a Continuous Variable.
> - As we generate regression coefficients for each k category, we need to specify a prior with a shape equal to the number of categories k in the code (see comments in the code).
> - To compare differences between categories, we need to compute the distribution of the differences between categories, known as the contrast distribution. **Never compare confidence intervals or p-values directly**.

8.3 Example

Below is an example of code that demonstrates Bayesian regression with an independent categorical variable using the Bayesian Inference (BI) package. The data consist of one continuous dependent variable (*kcal_per_g*), representing the caloric value of milk per gram, and a categorical independent variable, representing species clade membership. The goal is to estimate the differences in milk calories between clades.

8.3.1 Python

```python
from main import*

# Setup device------------------------------------------------
m = bi(platform='cpu')

# Import Data & Data Manipulation ------------------------------------------------
# Import
from importlib.resources import files
data_path = files('BI.resources.data') / 'milk.csv'
m.data(data_path, sep=';')
m.index(["clade"]) # Manipulate
m.scale(['kcal_per_g']) # Scale
m.data_to_model(['kcal_per_g', "index_clade"]) # Send to model (convert to jax array)

# Define model ------------------------------------------------
def model(kcal_per_g, index_clade):
    a = m.bi.dist.normal(0, 0.5, shape=(4,), name = 'a') # shape based on the number of clade
    s = m.bi.dist.exponential( 1, name = 's')
    mu = a[index_clade]
    m.normal(mu, s, obs=kcal_per_g)


# Run mcmc ------------------------------------------------
m.run(model) # Optimize model parameters through MCMC sampling

# Summary ------------------------------------------------
m.summary()
```

8.3.2 R

```r
library(BI)
m=importbi(platform='cpu')

# Load csv file
m$data(paste(system.file(package = "BI"),"/data/milk.csv", sep = ''), sep=';')
m$scale(list('kcal.per.g')) # Manipulate
m$index(list('clade')) # Scale
m$data_to_model(list('kcal_per_g', 'index_clade')) # Send to model (convert to jax array)
```

```
# Define model ------------------------------------------------
model <- function(kcal_per_g, index_clade){
  # Parameter prior distributions
  beta = bi.dist.normal( 0, 0.5, name = 'beta', shape = c(4))  # shape based on the number o
  sigma =bi.dist.exponential(1, name = 's')
  # Likelihood
  m$normal(beta[index_clade], sigma, obs=kcal_per_g)
}

# Run mcmc ------------------------------------------------
m$run(model) # Optimize model parameters through MCMC sampling

# Summary ------------------------------------------------
m$summary() # Get posterior distributions
```

8.4 Mathematical Details

8.4.1 *Frequentist formulation*

We model the relationship between the categorical input feature (X) and the target variable (Y) using the following equation:

$$Y_i = \alpha + \beta_k X_i + \sigma$$

Where:

- Y_i is the dependent variable for observation i.
- α is the intercept term.
- β_k are the regression coefficients for each k category.
- X_i is the encoded categorical input variable for observation i.
- σ is the error term.

We can interpret β_i as the effect of each category on Y relative to the baseline (usually one of the categories or the intercept).

8.4.2 *Bayesian formulation*

In the Bayesian formulation, we define each parameter with priors . We can express the Bayesian regression model accounting for prior distributions as follows:

$$Y \sim Normal(\alpha + \beta_k X, \sigma)$$

$$\alpha \sim Normal(0, 1)$$

$$\beta_k \sim Normal(0, 1)$$

$$\sigma \sim Exponential(1)$$

Where:

- Y_i is the dependent variable for observation i.
- α is the prior distribution for the intercept.
- β_k are k prior distributions for k regression coefficients.
- X_i is the encoded categorical input variable for observation i.
- σ is the prior distribution for the standard deviation, ensuring it is positive.

8.5 Notes

ℹ Note

- We can apply multiple variables similarly to Chapter 2: Multiple Continuous Variables.
- We can apply interaction terms similarly to Chapter 3: Interaction between Continuous Variables.

8.6 Reference(s)

McElreath (2018)

9 Binomial Model

9.1 General Principles

To model the relationship between a binary dependent variable—e.g., success/failure, yes/no, or 1/0—and one or more independent variables, we can use a *Binomial model*.

9.2 Considerations

Caution

- We have the same considerations as for Regression for continuous variable.
- The first link function is the *logit*. The *logit* link function in the Bayesian binomial model converts the linear combination of predictor variables into probabilities, making it suitable for modeling binary outcomes. It helps to estimate the relationship between predictors and the probability of success, ensuring results fall within the bounds of the binomial distribution $\in [0, 1]$.

9.3 Example

Below is an example code snippet that demonstrates Bayesian binomial regression using the Bayesian Inference (BI) package. The data consist of one binary dependent variable (*pulled_left*), which represents which side individuals pulled. The goal is to evaluate the probability of pulling the left side.

9.3.1 Python

```
from BI import bi

# setup platform------------------------------------------------
m = bi(platform='cpu')
```

```python
# import data ------------------------------------------------
# Import
from importlib.resources import files
data_path = files('BI.resources.data') / 'chimpanzees.csv'
m.data(data_path, sep=';')
m.data_to_model(['pulled_left'])

# Define model ------------------------------------------------
def model(pulled_left):
    a = m.dist.normal( 0, 10, shape=(1,), name = 'a')
    m.binomial(logits=a[0], obs=pulled_left)

# Run sampler ------------------------------------------------
m.run(model, num_samples=500)

# Diagnostic ------------------------------------------------
m.summary()
```

9.3.2 R

```r
library(BI)

# setup platform------------------------------------------------
m=importbi(platform='cpu')

# import data ------------------------------------------------
m$data(paste(system.file(package = "BI"),"/data/chimpanzees.csv", sep = ''), sep=';')
m$data_to_model(list('pulled_left')) # Send to model (convert to jax array)

# Define model ------------------------------------------------
model <- function(pulled_left){
  # Parameters priors distributions
  alpha = bi.dist.normal( 0, 10, name = 'alpha')
  # Likelihood
  m$binomial(logits = alpha, obs=pulled_left)
}


# Run MCMC ------------------------------------------------
m$run(model) # Optimize model parameters through MCMC sampling
```

```
# Summary ------------------------------------------------
m$summary() # Get posterior distribution
```

9.4 Mathematical Details

9.4.1 *Frequentist formulation*

We model the relationship between the independent variable (X_i) and the binary dependent variable (Y_i) using the following equation:

$$logit(Y_i) = \alpha + \beta X_i$$

Where:

- Y_i is the probability of success (i.e., the probability of the binary outcome being 1) for observation i.
- α is the intercept term.
- β is the regression coefficient.
- X_i is the value of the independent variable for observation i.
- $logit(Y_i)$ is the log-odds of success, calculated as the log of the odds ratio of success. Through this link function, the relationship between the independent variables and the log-odds of success is modeled linearly, allowing us to interpret the effect of each independent variable on the log-odds of success for observation i.

9.4.2 *Bayesian formulation*

priors . We can express the Bayesian regression model accounting for prior distributions as follows:

$$Y_i \sim Binomial(n = 1, p)$$

$$logit(p) \sim \alpha + \beta X_i$$

$$\alpha \sim Normal(0, 1)$$

$$\beta \sim Normal(0, 1)$$

Where:

- Y_i is the probability of success (i.e., the probability of the binary outcome being 1) for observation i.
- $n = 1$ represents the number of trials in the binomial distribution (binary outcome).
- β and α are the prior distributions for the regression coefficient and intercept, respectively.
- *logit* is the log-odds of success, calculated as the log of the odds ratio of success. Through this link function, the relationship between the independent variables and the log-odds of success is modeled linearly, allowing us to interpret the effect of each independent variable on the log-odds of success for observation i.

9.5 Notes

ℹ Note

- We can apply multiple variables similarly to chapter 2.
- We can apply interaction terms similarly to chapter 3.
- We can apply categorical variables similarly to chapter 4.
- Below is an example code snippet demonstrating a Bayesian binomial model for multiple categorical variables using the Bayesian Inference (BI) package. The data consist of one binary dependent variable (*pulled_left*), which represents which side individuals pulled, and three independent variables (*actor*, *side*, *cond*). The goal is to evaluate, for each individual, the probability of pulling the left side, accounting for whether the individual is left-handed or right-handed, as well as the different conditions.

```python
from BI import bi
m = bi(platform='cpu')
m.data('../resources/data/chimpanzees.csv', sep=';')
m.df['treatment'] =  m.df.prosoc_left + 2 * m.df.condition
m.df['actor'] = m.df['actor'] - 1

m.data_to_model(['actor', 'treatment', 'pulled_left'])

def model(actor, treatment, pulled_left):
    a = m.dist.normal(0, 1.5, shape = (7,), name='a')
    b = m.dist.normal(0, 0.5, shape = (4,), name='b')
    p = a[actor] + b[treatment]
    m.lk("y", m.dist.binomial(1, logits=p), obs=pulled_left)

# Run sampler ------------------------------------------------
m.run(model)
# Diagnostic ------------------------------------------------
m.summary()
```

9.6 Reference(s)

McElreath (2018)

10 Beta-Binomial Model

10.1 General Principles

To model the relationship between a binary outcome variable representing success counts and one or more independent variables with overdispersion , we can use the *Beta-Binomial model*.

10.2 Considerations

Caution

- We have the same considerations as for Binomial regression.
- A Beta-Binomial model assumes that each binomial count observation has its own probability of success. The model estimates the distribution of probabilities of success across cases, instead of a single probability of success.
- A Beta distribution has two parameters: the rates for each probability and a shape parameter .   influences how probabilities are distributed between 0 and 1. Specifically, it consists of two parameters, α and β, which determine the concentration of probability around 0 and 1.
  - If both are equal to or greater than 1, the distribution is bell-shaped and centered around 0.5.
  - If $\alpha > \beta$, the distribution is skewed toward 1, and if $\beta > \alpha$, it is skewed toward 0. Thus, the shape parameters γ and η provide flexibility in modeling various types of prior beliefs about probabilities.

10.3 Example

Below is an example code snippet demonstrating Bayesian Beta-Binomial regression using the Bayesian Inference (BI) package. The data consist of:

1) One binary dependent variable (admit), which represents candidates' admission status.
2) One independent categorical variable representing individuals' gender (gid).
3) Additionally, we have the number of applications (applications) per gender, which will be used to account for independent rates.

The goal is to evaluate whether the probability of admission is different between genders, while accounting for differences in the number of applications between genders.

10.3.1 Python

```python
from BI import bi

# setup platform------------------------------------------------
m = bi(platform='cpu')

# Import Data & Data Manipulation ------------------------------------------------
# Import
from importlib.resources import files
data_path = files('BI.resources.data') / 'UCBadmit.csv'
m.data(data_path, sep=';')
m.df["gid"] = (m.df["applicant.gender"] != "male").astype(int)

# Define model ------------------------------------------------
def model(gid, applications, admit):
    phi = m.dist.exponential(1,  name = 'phi')
    alpha = m.dist.normal( 0., 1.5, shape=(2,), name = 'alpha')
    theta =  phi + 2
    pbar = jax.nn.sigmoid(alpha[gid])
    concentration1 = pbar*theta
    concentration0 = (1 - pbar) * theta

    m.dist.betabinomial(total_count = applications, concentration1 = concentration1, concent

# Run MCMC ------------------------------------------------
m.run(model) # Optimize model parameters through MCMC sampling

# Summary ------------------------------------------------
m.summary()
```

10.3.2 R

```
library(BI)

# setup platform------------------------------------------------
m=importbi(platform='cpu')

# import data ------------------------------------------------
m$data(paste(system.file(package = "BI"),"/data/UCBadmit.csv", sep = ''), sep=';')
m$df["gid"] = as.integer(ifelse(m$df["applicant.gender"] == "male", 0, 1)) # Manipulate
m$data_to_model(list('gid', 'applications', 'admit' )) # Send to model (convert to jax array)

# Define model ------------------------------------------------
model <- function(gid, applications, admit){
  # Parameter prior distributions
  phi = bi.dist.exponential(1, name = 'phi',shape=c(1))
  alpha = bi.dist.normal(0., 1.5, shape= c(2), name='alpha')
  t = phi + 2
  pbar = jax$nn$sigmoid(alpha[gid])
  gamma = pbar * t
  eta = (1 - pbar) * t
  # Likelihood
  m$betabinomial(total_count=applications, concentration1=gamma, concentration0=eta, obs=admi
}

# Run MCMC ------------------------------------------------
m$run(model) # Optimize model parameters through MCMC sampling

# Summary ------------------------------------------------
m$summary() # Get posterior distribution
```

10.4 Mathematical Details

10.4.1 *Bayesian Model*

In the Bayesian formulation, we define each parameter with priors . We can express the Bayesian regression model accounting for prior distributions as follows:

$$Y_i \sim BetaBinomial(n_i, \gamma_i, \eta_i)$$

$$\gamma_i = \overline{\rho}\tau$$

$$\eta_i = (1 - \overline{\rho})\tau$$

$$\overline{\rho} = logit(\alpha_i)$$

$$\tau = \phi + 2$$

$$\alpha \sim Normal(0, 1)$$

$$\phi \sim Exponential(1)$$

Where:

- Y_i is the count of successes for the i-th observation, which follows a beta-binomial distribution with n_i trials.
- γ_i represents the concentration parameter for the number of successes, derived from the probability of success and scaled by τ.
- η_i represents the concentration parameter for failures, derived from the probability of failure $(1 - \overline{\rho})$ and also scaled by τ.
- $\overline{\rho}$ is the probability of success for the i-th observation. The logit function transforms the linear predictor (which can take any real value) into a probability value between 0 and 1.
- τ is derived from and is used as a scaling factor for the shape parameters and .
- is a vector of parameters, each representing the effect of group i on the success probability.
- is a random variable following an exponential distribution with a rate of 1.

10.5 Reference(s)

McElreath (2018)

11 Poisson model

11.1 General Principles

To model the relationship between a count outcome variable—e.g., counts of events occurring in a fixed interval of time or space—and one or more independent variables, we can use the *Poisson model*.

This is a special shape of the binomial distribution; it is useful because it models binomial events for which the number of trials n is unknown or uncountably large.

11.2 Considerations

Caution

- We have the same considerations as for Regression for a continuous variable.
- We have the second link function : *log*. The *log* link ensures that is always positive.
- To invert the log link function and linearly model the relationship between the predictor variables and the log of the mean rate parameter, we can apply the exponential function (see comment in code).

11.3 Example

Below is an example code snippet demonstrating a Bayesian Poisson model using the Bayesian Inference (BI) package. Data consist of:

1) A continuous dependent variable *total_tools*, which represents the number of tools produced by a civilization.
2) A continuous independent variable *population* representing population size.
3) A categorical independent variable *cid* representing different civilizations.

The goal is to estimate the production of tools based on population size, accounting for each civilization.

11.3.1 Python

```python
from BI import bi
import jax.numpy as jnp
# Setup device------------------------------------------------
m = bi(platform='cpu')

# import data ------------------------------------------------
# Import
from importlib.resources import files
data_path = files('BI.resources.data') / 'Kline.csv'
m.data(data_path, sep=';')
m.scale(['population'])
m.df["cid"] = (m.df.contact == "high").astype(int)
#m.data_to_model(['total_tools', 'population', 'cid'])
def model(cid, population, total_tools):
    a = m.dist.normal(3, 0.5, shape= (2,), name='a')
    b = m.dist.normal(0, 0.2, shape=(2,), name='b')
    l = jnp.exp(a[cid] + b[cid]*population)
    m.poisson(l, obs=total_tools)

# Run sampler ------------------------------------------------
m.run(model)

# Diagnostic ------------------------------------------------
m.summary()
```

11.3.2 R

```r
library(BI)

# Setup platform------------------------------------------------
m=importbi(platform='cpu')

# import data ------------------------------------------------
m$data(paste(system.file(package = "BI"),"/data/Kline.csv", sep = ''), sep=';')
m$scale(list('population'))# Scale
```

```
m$df["cid"] =  as.integer(ifelse(m$df$contact == "high", 1, 0)) # Manipulate
m$data_to_model(list('total_tools', 'population', 'cid' )) # Send to model (convert to jax a

# Define model ------------------------------------------------
model <- function(total_tools, population, cid){
  # Parameter prior distributions
  alpha = bi.dist.normal(3, 0.5, name='alpha', shape = c(2))
  beta = bi.dist.normal(0, 0.2, name='beta', shape = c(2))
  l = jnp$exp(alpha[cid] + beta[cid]*population)
  # Likelihood
  m$poisson(l, obs=total_tools)
}

# Run MCMC ------------------------------------------------
m$run(model) # Optimize model parameters through MCMC sampling

# Summary ------------------------------------------------
m$summary() # Get posterior distribution
```

11.4 Mathematical Details

11.4.1 *Frequentist formulation*

We model the relationship between the predictor variable (X_i) and the count outcome variable (Y_i) using the following equation:

$$\log(\lambda_i) = \alpha + \beta X_i$$

Where:

- λ_i is the mean rate parameter of the Poisson distribution (expected count) for observation i, modeled as the exponential function of the linear combination of predictors.
- $\log(\lambda_i)$ is the log of the mean rate parameter for observation i, ensuring it is positive.
- β is the regression coefficient.
- α is the intercept term.
- X_i is the value of the independent variable for observation i.

11.4.2 *Bayesian formulation*

In the Bayesian formulation, we define each parameter with priors . We can express the Bayesian regression model accounting for prior distributions as follows:

$$Y \sim Poisson(\lambda_i)$$

$$\log(\lambda_i) = \alpha + \beta X_i$$

$$\alpha \sim Normal(0, 1)$$

$$\beta \sim Normal(0, 1)$$

Where:

- Y_i is the dependent variable for observation i.
- λ_i is the mean rate parameter of the Poisson distribution for observation i, modeled as the exponential function of the linear combination of predictors.
- $\log(\lambda_i)$ is the log of the mean rate parameter for observation i.
- α and β are the prior distributions for the intercept and the regression coefficients, respectively.
- λ_i is the mean rate parameter of the Poisson distribution, modeled as the exponential function of the linear combination of predictors.
- X_i is the value of the independent variable for observation i.

11.5 Notes

ℹ Note

- We can apply multiple variables similarly to chapter 2.
- We can apply interaction terms similarly to chapter 3.
- We can apply categorical variables similarly to chapter 4.

11.6 Reference(s)

McElreath (2018)

12 Gamma-Poisson model

12.1 General Principles

To model the relationship between a count outcome variable and one or more independent variables with overdispersion , we can use the *Negative Binomial model*.

12.2 Considerations

> Caution
>
> - We have the same considerations as for the Poisson model.
> - Overdispersion is handled because the Negative Binomial model assumes that each Poisson count observation has its own rate. This is an additional parameter specified in the model (in the code, it is `log_days`).

12.3 Example

Below is an example code snippet demonstrating a Bayesian Gamma-Poisson model using the Bayesian Inference (BI) package:

12.3.1 Python

```python
from BI import bi
# Setup device ------------------------------------------------
m = bi(platform='cpu') # Import

# Import Data & Data Manipulation ------------------------------------------------
# Import
from importlib.resources import files
data_path = files('BI.resources.data') / 'Sim dat Gamma poisson.csv'
```

```
m.data(data_path, sep=',')
m.data_to_model(['log_days', 'monastery', 'y']) # Send to model (convert to jax array)

# Define model -----------------------------------------------
def model(log_days, monastery, y):
    a = m.dist.normal(0, 1, name = 'a', shape=(1,))
    b = m.dist.normal(0, 1, name = 'b', shape=(1,))
    l = m.jnp.exp(log_days + a + b * monastery)
    m.poisson(rate = l, obs=y)
# Run MCMC -----------------------------------------------
m.run(model) # Optimize model parameters through MCMC sampling

# Summary -----------------------------------------------
m.summary() # Get posterior distributions
```

12.3.2 R

```
library(BI)

# Setup platform-----------------------------------------------
m=importbi(platform='cpu')

# Import data -----------------------------------------------
m$data(paste(system.file(package = "BI"),"/data/Sim dat Gamma poisson.csv", sep = ''), sep='
m$data_to_model(list('log_days', 'monastery', 'y' )) # Send to model (convert to jax array)

# Define model -----------------------------------------------
model <- function(log_days, monastery, y){
  # Parameter prior distributions
  alpha = bi.dist.normal(0, 1, name='alpha', shape=c(1))
  beta = bi.dist.normal(0, 1, name='beta', shape=c(1))
  l = jnp$exp(log_days + alpha + beta * monastery)
  # Likelihood
  m$poisson(rate=l, obs=y)
}

# Run MCMC -----------------------------------------------
m$run(model) # Optimize model parameters through MCMC sampling

# Summary -----------------------------------------------
m$summary() # Get posterior distributions
```

12.4 Mathematical Details

12.4.1 *Frequentist formulation*

We model the relationship between the independent variable X and the count outcome variable Y using the following equation:

$$\log(\lambda_i) = \exp(\text{rates}_i + \alpha + \beta X_i)$$

Where:

- λ_i is the mean rate parameter of the negative binomial distribution (expected count) for observation i.
- $\log(\lambda_i)$ is the log of the mean rate parameter, ensuring it is positive for observation i.
- α is the intercept term.
- β is the regression coefficient.
- X_i is the value of the predictor variable for observation i.

12.4.2 *Bayesian model*

In the Bayesian formulation, we define each parameter with priors . We can express the Bayesian regression model accounting for prior distributions as follows:

$$Y_i \sim \text{Poisson}(\lambda_i)$$

$$\log(\lambda_i) = \text{rates}_i + \alpha + \beta X_i$$

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

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

Where:

- Y_i is the dependent variable for observation i.
- λ_i is the mean rate parameter of the Poisson distribution for observation i, assuming that each Poisson count observation has its own $rate_i$.
- $\log(\lambda_i)$ is the log of the mean rate parameter for observation i, ensuring it is positive.
- α is the intercept term.
- β is the regression coefficient.
- X_i is the value of the predictor variable for observation i.

12.5 Notes

ℹ Note

- We can apply multiple variables similarly as in chapter 2.
- We can apply interaction terms similarly as in chapter 3.
- We can apply categorical variables similarly as in chapter 4.

12.6 Reference(s)

McElreath (2018)

13 Multinomial model

13.1 General Principles

To model the relationship between a categorical outcome variable with more than two categories and one or more independent variables, we can use a *Multinomial* model.

13.2 Considerations

Caution

- We have the same considerations as for Regression for continuous variable.
- One way to interpret a multinomial model is to consider that we need to build $K-1$ linear models, where K is the number of categories. Once we get the linear prediction for each category, we can convert these predictions to probabilities by building a simplex . To do this, we convert the regression outputs using the softmax function (see the "jax.nn.softmax" line in the code).
- The intercept α captures the difference in the log-odds of the outcome categories; thus, different categories need different intercepts.
- On the other hand, as we assume that the effect of each predictor on the outcome is consistent across all categories, the regression coefficients β are shared across categories.
- The relationship between the predictor variables and the log-odds of each category is modeled linearly, allowing us to interpret the effect of each predictor on the log-odds of each category.

13.3 Example

Below is an example code snippet demonstrating a Bayesian multinomial model using the Bayesian Inference (BI) package:

13.3.1 Python

```python
from BI import bi
import jax.numpy as jnp
import pandas as pd
import jax
# Setup device ------------------------------------------------
m = bi('cpu')

# Import Data & Data Manipulation ------------------------------------------------
# Import
from importlib.resources import files
data_path = files('BI.resources.data') / 'Sim data multinomial.csv'
m.data(data_path, sep=',')

# Define model ------------------------------------------------
def model(career, income):
    a = m.dist.normal(0, 1, shape=(2,), name = 'a')
    b = m.dist.halfnormal(0.5, shape=(1,), name = 'b')
    s_1 = a[0] + b * income[0]
    s_2 = a[1] + b * income[1]
    s_3 = [0] #pivot
    p = jax.nn.softmax(jnp.stack([s_1[0], s_2[0], s_3[0]]))
    m.dist.categorical(probs=p, obs=career)

# Run sampler ------------------------------------------------
m.run(model)  # Optimize model parameters through MCMC sampling

# Summary ------------------------------------------------
m.summary() # Get posterior distributions
```

13.3.2 R

```r
library(BI)

# setup platform------------------------------------------------
m=importbi(platform='cpu')

# import data ------------------------------------------------
m$data(paste(system.file(package = "BI"),"/data/Sim data multinomial.csv", sep = ''), sep=',
keys <- c("income", "career")
```

```
income = unique(m$df$income)
income = income[order(income)]
values <- list(jnp$array(as.integer(income)),jnp$array( as.integer(m$df$career)))
m$data_on_model = py_dict(keys, values, convert = TRUE)

# Define model ------------------------------------------------
model <- function(income, career){
  # Parameter prior distributions
  alpha = bi.dist.normal(0, 1, name='alpha', shape = c(2))
  beta = bi.dist.halfnormal(0.5, name='beta')

  s_1 = alpha[0] + beta * income[0]
  s_2 = alpha[1] + beta * income[1]
  s_3 = 0 # reference category

  p = jax$nn$softmax(jnp$stack(list(s_1, s_2, s_3)))

  # Likelihood
  m$categorical(probs=p[career], obs=career)
}

# Run MCMC ------------------------------------------------
m$run(model) # Optimize model parameters through MCMC sampling

# Summary ------------------------------------------------
m$summary() # Get posterior distribution
```

13.4 Mathematical Details

13.4.1 *Frequentist formulation*

We model the relationship between the predictor variables (X1, X2, ..., Xn) and the categorical outcome variable (Y_i) using the following equation:

$$logit(p_{ik}) = \alpha_k + \beta X_i$$

Where:

- p_{ik} is the probability of the -th observation being in category .
- α_k is the intercept for category .

- β is the regression coefficients common to all categories.
- X_i is the vector of independent variables for the -th observation.
- A reference category is often chosen to simplify the model.

13.4.2 *Bayesian model*

In Bayesian multinomial modeling, the likelihood function of the data is specified using a multinomial distribution. The multinomial distribution models the counts of outcomes falling into different categories. For an outcome variable with categories, the multinomial likelihood function is:

$$Multinomial(y|\theta) = \frac{N!}{\prod_{k=1}^{K} y_k!} \prod_{k=1}^{K} \theta_k^{y_k}$$

Where:

- $y = (y_1, y_2, ..., y_K)$ represents the counts of observations in each of the categories.
- N is the total number of observations or trials.
- $\theta = (\theta_1, \theta_2, ..., \theta_K)$ is a simplex of category probabilities, with θ_k representing the probability of category .
- $\frac{N!}{\prod_{k=1}^{K} y_k!}$ is the multinomial coefficient that accounts for the number of ways to arrange the observations into the categories. This coefficient ensures that the likelihood function properly accounts for the permutations of the counts across different categories.

13.5 Reference(s)

McElreath (2018)

14 Dirichlet Model

14.1 General Principles

To model the relationship between a categorical outcome variable with more than two categories and one or more independent variables with overdispersion , we can use a *Dirichlet* model.

14.2 Considerations

> Caution
>
> - We have the same considerations as for the Multinomial model.
> - One major difference from the multinomial model is that the Dirichlet model doesn't require a simplex but rather strictly positive values. We can thus exponentiate the outputs from the categorical regressions instead of using the softmax function.
> -

14.3 Example

```
from BI import bi
import jax.numpy as jnp
import pandas as pd
import jax
# Setup device ------------------------------------------------
m = bi('cpu')

# Import Data & Data Manipulation ------------------------------------------------
# Import
from importlib.resources import files
data_path = files('BI.resources.data') / 'Sim data multinomial.csv'
```

```
m.data(data_path, sep=',')

# Define model ------------------------------------------------
def model(income, career):
    # Parameter prior distributions
    alpha = m.dist.normal(0, 1, shape=(2,), name='a')
    beta = m.dist.halfnormal(0.5, shape=(1,), name='b')
    s_1 = alpha[0] + beta * income[0]
    s_2 = alpha[1] + beta * income[1]
    s_3 = alpha[0] + beta * income[0]
    p = jax.nn.exp(jnp.stack([s_1[0], s_2[0], s_3[0]]))
    # Likelihood
    m.dirichletmultinomial(p[career], lambda_, obs=career)

# Run sampler ------------------------------------------------
m.run(model)

# Summary ------------------------------------------------
m.summary()
```

14.4 Mathematical Details

14.4.1 *Formula*

14.4.2 *Bayesian Model*

In the Bayesian formulation, we define each parameter with priors . We can express the Bayesian regression model accounting for prior distributions as follows:

14.5 Reference(s)

McElreath (2018)

15 Zero-Inflated

15.1 General Principles

Zero-Inflated Regression models are used when the outcome variable is a count variable with an excess of zero counts. These models combine a count model (e.g., Poisson or Negative Binomial) with a separate model for predicting the probability of excess zeros.

15.2 Considerations

Caution

- In Bayesian Zero-Inflated regression, we consider uncertainty in the model parameters and provide a full posterior distribution over them. We need to declare prior distributions for $W_{1\pi}, W_{2\pi}, ..., W_{n\pi}$, $W_{1\lambda}, W_{2\lambda}, ..., W_{n\lambda}$, b_{π}, and b_{λ}.

15.3 Example

Below is an example code snippet demonstrating Bayesian Zero-Inflated Poisson regression using the Bayesian Inference (BI) package. The data represent the production of books in a monastery (y), which is affected by the number of days that individuals work, as well as the number of days individuals drink.

15.3.1 Python

```
from BI import bi
from jax.scipy.special import expit
# Setup device ------------------------------------------------
m = bi('cpu')

# Simulated data------------------------------------------------
prob_drink = 0.2  # 20% of days
```

```
rate_work = 1     # average 1 manuscript per day

# Sample one year of production
N = 365
drink = m.dist.binomial(1, prob_drink, shape = (N,), sample = True)
y = (1 - drink) * m.dist.poisson(rate_work, shape = (N,), sample = True)

# Setup device------------------------------------------------
m = bi(platform='cpu')
m.data_on_model = dict(
    y=jnp.array(y)
)

# Define model ------------------------------------------------
def model(y):
    al = dist.normal(1, 0.5, name='al')
    ap = dist.normal(-1.5, 1, name='ap')
    p = expit(ap)
    lambda_ = jnp.exp(al)
    m.zeroinflatedpoisson(p, lambda_, obs=y)

# Run MCMC ------------------------------------------------
m.run(model)

# Summary ------------------------------------------------
m.summary()
```

15.3.2 R

```
library(BI)

# setup platform------------------------------------------------
m=importbi(platform='cpu')

# Simulate data ------------------------------------------------
prob_drink = 0.2  # 20% of days
rate_work = 1     # average 1 manuscript per day
# sample one year of production
N = as.integer(365)
drink = bi.dist.binomial(total_count = as.integer(1), probs = prob_drink, shape = c(N), sampl
y = (1 - drink) *  bi.dist.poisson(rate_work, shape = c(N), sample = T)
```

```
data = list()
data$y = y
m$data_on_model = data

# Define model ------------------------------------------------
model <- function(y){
  al = bi.dist.normal(1, 0.5, name='al', shape=c(1))
  ap = bi.dist.normal(-1, 1, name='ap', shape=c(1))
  p = jax$scipy$special$expit(ap)
  lambda_ = jnp$exp(al)
  m$zeroinflatedpoisson(p, lambda_, obs=y)
}

# Run MCMC ------------------------------------------------
m$run(model) # Optimize model parameters through MCMC sampling

# Summary ------------------------------------------------
m$summary() # Get posterior distribution
```

16 Mathematical Details

16.0.1 *Frequentist formulation*

We model the relationship between the independent variable X and the count outcome variable Y using two components:

1) A logistic regression model to predict the probability of an excess zero.
2) A count model (e.g., Poisson or Negative Binomial) to predict the count outcome.

The overall model can be represented as follows:

$$\begin{aligned}
&\text{logit}(\pi) = \alpha_\pi + \beta_\pi X_i \\
&\log(\lambda) = \alpha_\lambda + \beta_\lambda X_i \\
&Y_i \sim \begin{cases} 0 & \text{with probability } \pi \\ \text{CountModel}(\lambda) & \text{with probability } (1 - \pi) \end{cases}
\end{aligned}$$

Where:

- π is the probability of an excess zero.
- λ is the mean rate parameter of the count model.
- α_π and β_π are respectively the intercept and the regression coefficient for the logistic model.
- α_λ and β_λ are respectively the intercept and the regression coefficient for the count model.
- X_i are the independent variables' values for observation i.

16.0.2 *Bayesian formulation*

In the Bayesian formulation, we define each parameter with priors . We can express the Bayesian regression model accounting for prior distributions as follows:

$$Y \sim ZIPoisson(\pi, \lambda)$$

$$\text{logit}(\pi) = \alpha_\pi + \beta_\pi X$$

$$\log(\lambda) = \alpha_\lambda + \beta_\lambda X$$

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

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

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

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

Where:

- π is the probability of an excess zero.
- λ is the mean rate parameter of the count model.
- α_π and β_π are respectively the intercept and the regression coefficient for the logistic model.
- α_λ and β_λ are respectively the intercept and the regression coefficient for the count model.
- X_i are the independent variables' values for observation i.

16.1 Reference(s)

McElreath (2018)

17 Survival Analysis

17.1 General Principles

Survival analysis studies the time until an event of interest (e.g., death, recovery, information acquisition) occurs. When analyzing binary survival outcomes (e.g., alive or dead), we can use models such as Cox proportional hazards to evaluate the effect of predictors on survival probabilities.

Key concepts include:

1. **Hazard Function**: The instantaneous risk of the event occurring at a given time.
2. **Survival Function**: The probability of surviving beyond a given time.
3. **Covariates**: Variables (e.g., age, treatment) that may affect survival probabilities.
4. **Baseline Hazard**: The hazard when all covariates are zero, which forms the reference for comparing different conditions.

17.2 Considerations

🔥 Caution

- Bayesian models provide a framework to account for uncertainty in parameter estimates through posterior distributions. You will need to define prior distributions for all model parameters, such as baseline hazard, covariate effects, and variance terms.
- In survival analysis:
  - The **baseline hazard** can follow distributions like Exponential, Weibull, or Gompertz, depending on the data.
  - Censoring (when the event is not observed for some subjects) must be accounted for in the likelihood function. Proper handling is essential for unbiased results.
- Bayesian survival models allow flexible handling of time-dependent covariates, random effects, and incorporate uncertainty more naturally than Frequentist methods.

17.3 Example

Here's an example of a Bayesian survival analysis using the **Bayesian Inference (BI)** package. The data come from a clinical trial of mastectomy for breast cancer. The goal is to estimate the effect of the `metastasized` covariate, coded as 0 (no metastasis) and 1 (metastasis), on the survival outcome `event` for each patient. Time is continuous and censoring is indicated by the event variable.

17.3.1 Python

```python
from BI import bi

# Setup device------------------------------------------------
m = bi(platform='cpu')

# Import Data & Data Manipulation ------------------------------------------------
# Import
from importlib.resources import files
data_path = files('BI.resources.data') / 'mastectomy.csv'
m.data(data_path, sep=',')

m.df.metastasized = (m.df.metastasized == "yes").astype(np.int64)
m.df.event = jnp.array(m.df.event.values, dtype=jnp.int32)

## Create survival object
m.surv_object(time='time', event='event', cov='metastasized', interval_length=3)

# Plot censoring ------------------------------------------------
m.plot_censoring(cov='metastasized')

# Model ------------------------------------------------
def model(intervals, death, metastasized, exposure):
    # Parameter prior distributions-------------------------
    ## Base hazard distribution
    lambda0 = m.dist.gamma(0.01, 0.01, shape= intervals.shape, name = 'lambda0')
    ## Covariate effect distribution
    beta = m.dist.normal(0, 1000, shape = (1,),  name='beta')
    ### Likelihood
    #### Compute hazard rate based on covariate effect
    lambda_ = m.hazard_rate(cov = metastasized, beta = beta, lambda0 = lambda0)
    #### Compute exposure rates
    mu = exposure * lambda_

    # Likelihood calculation
    y = m.poisson(mu + jnp.finfo(mu.dtype).tiny, obs = death)

# Run mcmc ------------------------------------------------
m.run(model, num_samples=500)

# Summary ------------------------------------------------
print(m.summary())

# Plot hazards and survival function ------------------------------------------------
m.plot_surv()
```

17.3.2 R

17.4 Mathematical Details

17.4.1 *Frequentist formulation*

The Cox proportional hazards model can be expressed as:

$$h(t|X) = h_0(t)\exp(\beta^T X)$$

- Where:
  - $h(t|X)$ is the hazard at time t for covariates X.
  - $h_0(t)$ is the baseline hazard function (e.g., exponential, Weibull).
  - X represents the covariates (such as age, treatment).
  - β are the regression coefficients to be estimated.
- Censoring is accounted for by multiplying the hazard function by a factor that depends on the censoring distribution, usually modeled as independent censoring with a rate (t):
  - $Y_i(t) = Poisson(h(t|X) * \delta(t))$

17.4.2 *Bayesian formulation*

In Bayesian survival analysis, we define priors for each parameter:

1. **Hazard Function**: The hazard rate at time t for an individual is given by:

$$Y_i(t) = Poisson(\lambda(t) * censoring(t))$$

$$\lambda(t) = \lambda_0(t) \exp(x\beta)$$

$$\beta \sim Normal(\mu_\beta, \sigma_\beta^2)$$

$$\mu_\beta \sim Normal(0, 10^2)$$

$$\sigma_\beta^2 \sim Uniform(0, 10)$$

Where:

- $Y_i(t)$ is the status of the i-th subject at time t coded as a binary variable:

$$Y_i(t) = \begin{cases} 1 & \text{if subject } i \text{ died at time } t \text{ ,} \\ 0 & \text{otherwise.} \end{cases}$$

- $\lambda(t)$: Hazard function at time t.
- $\lambda_0(t)$: Baseline hazard function (e.g., exponential or Weibull).
- x: Covariates (e.g., age, treatment).
- β: Regression coefficients capturing the effect of x on the hazard are assigned a normal prior.
- μ_β: Mean of the normal distribution.
- σ_β^2: Variance of the normal distribution.

17.5 Reference(s)

18 Varying Intercepts

18.1 General Principles

To model the relationship between a dependent variable and an independent variable while allowing for different intercepts across groups or clusters, we can use a *Varying Intercepts* model. This approach is particularly useful when data is grouped (e.g., by subject, location, or time period) and we expect the baseline level of the outcome to vary across these groups.

18.2 Considerations

Caution

- We have the same considerations as for Regression for a continuous variable.
- The main idea of varying intercepts is to generate an intercept for each group, allowing each group to start at different levels. Thus, the intercept α_k is defined based on the k declared groups.
- Each intercept has its own prior - i.e., a hyper-prior .
- In the code below, the *hyper-prior* is `a_bar`.

18.3 Example

Below is an example code snippet demonstrating Bayesian regression with varying intercepts using the Bayesian Inference (BI) package. The data consists of a dependent variable representing individuals' survival (*surv*) and an independent categorical variable (*tank*), which indicates the tank where the individual was born, with a total of 48 tanks.

18.4 Python

```python
from BI import bi
import numpy as np

# Setup device------------------------------------------------
m = bi(platform='cpu')

# Import Data & Data Manipulation ------------------------------------------------
# Import
from importlib.resources import files
data_path = files('BI.resources.data') / 'reedfrogs.csv'
m.data(data_path, sep=';')
# Manipulate
m.df["tank"] = np.arange(m.df.shape[0])

# Define model ------------------------------------------------
def model(tank, surv, density):
    sigma = m.dist.exponential( 1,  name = 'sigma')
    a_bar = m.dist.normal( 0., 1.5,  name = 'a_bar')
    alpha = m.dist.normal( a_bar, sigma, shape= tank.shape, name = 'alpha')
    p = alpha[tank]
    m.dist.binomial(total_count = density, logits = p, obs=surv)

# Run sampler ------------------------------------------------
m.run(model)

# Diagnostic ------------------------------------------------
m.summary()
```

18.5 R

```r
library(BI)

# setup platform------------------------------------------------
m=importbi(platform='cpu')

# Import data ------------------------------------------------
m$data(paste(system.file(package = "BI"),"/data/reedfrogs.csv", sep = ''), sep=';')
m$df$tank = c(0:(nrow(m$df)-1)) # Manipulate
```

```r
m$data_to_model(list('tank', 'surv', 'density')) # Manipulate
m$data_on_model$tank = m$data_on_model$tank$astype(jnp$int32) # Manipulate
m$data_on_model$surv = m$data_on_model$surv$astype(jnp$int32) # Manipulate


# Define model -------------------------------------------------
model <- function(tank, surv, density){
  # Parameter prior distributions
  sigma = bi.dist.exponential( 1,  name = 'sigma',shape=c(1))
  a_bar =  bi.dist.normal(0, 1.5, name='a_bar',shape=c(1))
  alpha = bi.dist.normal(a_bar, sigma, name='alpha', shape =c(48))
  p = alpha[tank]
  # Likelihood
  m$binomial(total_count = density, logits = p, obs=surv)
}

# Run MCMC -------------------------------------------------
m$run(model) # Optimize model parameters through MCMC sampling

# Summary -------------------------------------------------
m$summary() # Get posterior distribution
```

18.6 Mathematical Details

18.6.1 *Frequentist formulation*

We model the relationship between the independent variable X and the outcome variable Y with varying intercepts α for each group k using the following equation:

$$Y_{ik} = \alpha_k + \beta X_{ik} + \sigma$$

Where:

- Y_{ik} is the outcome variable for observation i in group k.
- α_k is the varying intercept for group k.
- X_{ik} is the independent variable for observation i in group k.
- β is the regression coefficient.
- σ is the error term, typically assumed to be normally distributed and positive.

18.6.2 Bayesian Model

We can express the Bayesian regression model accounting for priors distributions as follows:

$$
\begin{aligned}
Y_{ik} &\sim \text{Normal}(\mu_{ik}, \sigma) \\
\mu_{ik} &= \alpha_k + \beta X_{ik} \\
\alpha_k &\sim \text{Normal}(\mu_{\alpha_k}, \sigma_{\alpha_k}) \\
\beta &\sim \text{Normal}(0, 1) \\
\sigma &\sim \text{Exponential}(1) \\
\mu_{\alpha_k} &\sim \text{Normal}(0, 1) \\
\sigma_{\alpha_k} &\sim \text{Exponential}(1)
\end{aligned}
$$

Where:

- Y_{ik} is the outcome variable for observation i in group k.
- α_k is the varying intercept for group k.
- μ_{α_k} is the overall mean intercept.
- σ_{α_k} is the variance of the intercepts across groups.
- β is the regression coefficient.
- σ is the standard deviation of the error term.

18.7 Notes

Note

- We can apply multiple variables similarly to Chapter 2.
- We can apply interaction terms similarly to Chapter 3.
- We can apply categorical variables similarly to Chapter 4.
- We can apply varying intercepts with any distribution developed in previous chapters.

18.8 Reference(s)

McElreath (2018)

19 Varying slopes

19.1 General Principles

To model the relationship between predictor variables and a dependent variable while allowing for varying effects across groups or clusters, we use a *varying slopes* model.

This approach is useful when we expect the relationship between predictors and the dependent variable to differ across groups (e.g., different slopes for different subjects, locations, or time periods). This allows every unit in the data to have its own unique response to any treatment, exposure, or event, while also improving estimates via pooling.

19.2 Considerations

Caution

- We have the same considerations as for 12. Varying intercepts.
- The idea is pretty similar to categorical models, where a slope is specified for each category. However, here, we also estimate relationships between different groups. This leads to a different mathematical approach, as to model these relationships between groups, we model a matrix of covariance .
- The covariance matrix requires a correlation matrix distribution which is modeled using an $LKJcorr$ distribution that holds a parameter η. η is usually set to 2 to define a weakly informative prior that is skeptical of extreme correlations near -1 or 1. When we use $LKJcorr(1)$, the prior is flat over all valid correlation matrices. When the value is greater than 1, then extreme correlations are less likely.
- The Half-Cauchy distribution is used when modeling the covariance matrix to specify strictly positive values for the diagonal of the covariance matrix, ensuring positive variances.

19.3 Example

Below is an example code snippet demonstrating Bayesian regression with varying effects:

19.3.1 Simulated data

19.4 Python

```python
from BI import bi
# Setup device------------------------------------------------
m = bi(platform='cpu')

# Import Data & Data Manipulation ------------------------------------------------
# Import
from importlib.resources import files
data_path = files('BI.resources.data') / 'Sim data multivariatenormal.csv'
m.data(data_path, sep=',')

# Define model ------------------------------------------------
def model(cafe, wait, N_cafes, afternoon):
    a = m.dist.normal(5, 2,  name = 'a')
    b = m.dist.normal(-1, 0.5, name = 'b')
    sigma_cafe = m.dist.exponential(1, shape=(2,),  name = 'sigma_cafe')
    sigma = m.dist.exponential( 1,  name = 'sigma')
    Rho = m.dist.lkj(2, 2, name = 'Rho')
    cov = jnp.outer(sigma_cafe, sigma_cafe) * Rho
    a_cafe_b_cafe = m.dist.multivariatenormal(jnp.stack([a, b]), cov, shape = [N_cafes], name

    a_cafe, b_cafe = a_cafe_b_cafe[:, 0], a_cafe_b_cafe[:, 1]
    mu = a_cafe[cafe] + b_cafe[cafe] * afternoon
    m.dist.normal(mu, sigma, obs=wait)

# Run sampler ------------------------------------------------
m.run(model)
```

19.5 R

```r
library(BI)

# Setup platform------------------------------------------------
m=importbi(platform='cpu')

# Import data ------------------------------------------------
m$data(paste(system.file(package = "BI"),"/data/Sim data multivariatenormal.csv", sep = ''),
m$data_to_model(list('cafe', 'wait', 'afternoon'))

# Define model ------------------------------------------------
model <- function(cafe, afternoon, wait, N_cafes = as.integer(20) ){
  a = bi.dist.normal(5, 2, name = 'a')
  b = bi.dist.normal(-1, 0.5, name = 'b')
  sigma_cafe = bi.dist.exponential(1, shape= c(2), name = 'sigma_cafe')
  sigma = bi.dist.exponential( 1, name = 'sigma')
  Rho = bi.dist.lkj(as.integer(2), as.integer(2), name = 'Rho')
  cov = jnp$outer(sigma_cafe, sigma_cafe) * Rho

  a_cafe_b_cafe = bi.dist.multivariatenormal(
    jnp$squeeze(jnp$stack(list(a, b))),
    cov, shape = c(N_cafes), name = 'a_cafe')

  a_cafe = a_cafe_b_cafe[, 0]
  b_cafe = a_cafe_b_cafe[, 1]

  mu = a_cafe[cafe] + b_cafe[cafe] * afternoon

  m$normal(mu, sigma, obs=wait)
}

# Run MCMC ------------------------------------------------
m$run(model) # Optimize model parameters through MCMC sampling

# Summary ------------------------------------------------
m$summary() # Get posterior distribution
```

19.6 Mathematical Details

19.6.1 *Formula*

We model the relationship between the independent variable X and the outcome variable Y with varying intercepts (α) and varying slopes (β) for each group (k) using the following equation:

$$Y_{ik} = \alpha_k + \beta_k X_{ik} + \sigma$$

Where:

- Y_{ik} is the outcome variable for observation i in group k.
- X_{ik} is the independent variable for observation i in group k.
- α_k is the varying intercept for group k.
- β_k is the varying regression coefficient for group k.
- σ is the error term, assumed to be strictly positive.

19.6.2 *Bayesian Model*

We can express the Bayesian regression model accounting for prior distributions as follows:

$$\begin{gathered}
Y_{ik} \sim \text{Normal}(\mu_{ik}, \sigma) \\
\mu_{ik} = \alpha_k + \beta_k X_{ik} \\
\alpha_k \sim Normal(0, 1) \\
\beta_k \sim Normal(0, 1) \\
\sigma \sim Exponential(1)
\end{gathered}$$

The varying intercepts (α_k) and slopes (β_k) are modeled using a *Multivariate Normal distribution*:

$$\begin{pmatrix} \alpha_k \\ \beta_k \end{pmatrix} \sim \text{MultivariateNormal}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \sigma_\alpha^2 & \rho_{\alpha\beta}\sigma_\alpha\sigma_\beta \\ \rho_{\alpha\beta}\sigma_\alpha\sigma_\beta & \sigma_\beta^2 \end{pmatrix} \right)$$

Where:

- $\left(\begin{array}{c} 0 \\ 0 \end{array} \right)$ is the prior for the average intercept.

- $\begin{pmatrix} \sigma_\alpha^2 & \rho_{\alpha\beta}\sigma_\alpha\sigma_\beta \\ \rho_{\alpha\beta}\sigma_\alpha\sigma_\beta & \sigma_\beta^2 \end{pmatrix}$ is the covariance matrix which specifies the variance and covariance of α_k and β_k,
- where:
  - σ_α^2 is the variance of α_k.
  - σ_β^2 is the variance of β_k.
  - $\rho_{\alpha\beta}\sigma_\alpha\sigma_\beta$ is the covariance between α_k and β_k.

For computational reasons, it is often better to implement a non-centered parameterization that is equivalent to the *Multivariate Normal distribution* approach:

$$\begin{pmatrix} \alpha_k \\ \beta_k \end{pmatrix} \sim \begin{pmatrix} \sigma_\alpha \\ \sigma_\beta \end{pmatrix} \circ L \cdot \begin{pmatrix} \hat{\alpha}_k \\ \hat{\beta}_k \end{pmatrix}$$

- Where:
  - $\sigma_\alpha \sim$ Exponential(1) is the prior standard deviation among intercepts.
  - $\sigma_\beta \sim$ Exponential(1) is the prior standard deviation among slopes.
  - $L \sim$ LKJcorr(η) is the prior for the correlation matrix using the Cholesky Factor

The full non-centered version of the model is thus:

$$Y_{ik} \sim \text{Normal}(\mu_{ik}, \sigma)$$

$$\mu_{ik} = \alpha_k + \beta_k X_{ik}$$

$$\begin{pmatrix} \alpha_k \\ \beta_k \end{pmatrix} \sim \begin{pmatrix} \sigma_\alpha \\ \sigma_\beta \end{pmatrix} \circ L \cdot \begin{pmatrix} \hat{\alpha}_k \\ \hat{\beta}_k \end{pmatrix}$$

$$\alpha \sim Normal(0, 1)$$

$$\beta \sim Normal(0, 1)$$

$$\sigma_\alpha \sim Exponential(1)$$

$$\sigma_\beta \sim Exponential(1)$$

$$L \sim LKJcorr(2)$$

19.7 *Multivariate Model with One Random Slope for Each Variable*

We can apply a multivariate model similarly to Chapter 2. In this case, we apply the same principle, but with a covariance matrix with a dimension equal to the number of varying slopes we define. For example, if we want to generate random slopes for i observations in a model with two independent variables X_1 and X_2, we can define the formula as follows:

$$p(Y_i|\mu_i, \sigma) \sim \text{Normal}(\mu_i, \sigma)$$

$$\mu_i = \alpha_i + \beta_{1i} X_{1i} + \beta_{2i} X_{2i}$$

$$\begin{pmatrix} \alpha_i \\ \beta_{1i} \\ \beta_{2i} \end{pmatrix} \sim \begin{pmatrix} \sigma_\alpha \\ \sigma_{\beta_1} \\ \sigma_{\beta_2} \end{pmatrix} \circ L \cdot \begin{pmatrix} \hat{\alpha}_i \\ \hat{\beta}_{1i} \\ \hat{\beta}_{2i} \end{pmatrix}$$

$$\sigma_\alpha \sim Exponential(1)$$
$$\sigma_{\beta_1} \sim Exponential(1)$$
$$\sigma_{\beta_2} \sim Exponential(1)$$

$$L \sim LKJcorr(2)$$

19.8 *Multivariate Random Slopes on a Single Variable*

For more than two varying effects, we apply the same principle but with a covariance matrix for each varying effect that is summed to generate the varying intercept and slope. For example, if we want to generate random slopes for i actors and k groups, we can define the formula as follows:

$$p(Y_i|\mu_i, \sigma) \sim \text{Normal}(\mu_i, \sigma)$$

$$\mu_i = \alpha_i + \beta_i X_i$$
$$\alpha_i = \alpha + \alpha_{actor[i]} + \alpha_{group[i]}$$
$$\beta_i = \beta + \beta_{actor[i]} + \beta_{group[i]}$$

$$\alpha \sim Normal(0,1)$$

$$\beta \sim Normal(0,1)$$

$$\begin{pmatrix}\alpha_{\text{actor}}\\ \beta_{\text{actor}}\end{pmatrix} \sim \begin{pmatrix}\sigma_{\alpha a}\\ \sigma_{\beta a}\end{pmatrix} \circ L_a \cdot \begin{pmatrix}\hat{\alpha}_{ka}\\ \hat{\beta}_{ka}\end{pmatrix}$$

$$\sigma_{\alpha a} \sim Exponential(1)$$
$$\sigma_{\beta a} \sim Exponential(1)$$
$$L_a \sim LKJcorr(2)$$

$$\begin{pmatrix}\alpha_{\text{group}}\\ \beta_{\text{group}}\end{pmatrix} \sim \begin{pmatrix}\sigma_{\alpha g}\\ \sigma_{\beta g}\end{pmatrix} \circ L_g \cdot \begin{pmatrix}\hat{\alpha}_{kg}\\ \hat{\beta}_{kg}\end{pmatrix}$$

$$\sigma_{\alpha g} \sim Exponential(1)$$
$$\sigma_{\beta g} \sim Exponential(1)$$
$$L_g \sim LKJcorr(2)$$

19.9 Notes

Note

- We can apply interaction terms similarly to Chapter 3.
- We can apply categorical variables similarly to Chapter 4.
- We can apply varying slopes with any distribution presented in previous chapters. Below is the formula and the code snippet for a Binomial multivariate model with an interaction between two independent variables X_1 and X_2 and multiple varying effects for each actor and each group.

$$p(Y_i|n, p_i) \sim \text{Binomial}(n=1, p_i)$$

$$\text{logit}(p_i) = \alpha_i + \beta_{1i}X_{1i} + \beta_{2i}X_{1i}X_{2i}$$

$$\alpha_i = \alpha + \alpha_{actor[i]} + \alpha_{group[i]}$$
$$\beta_{1i} = \beta_1 + \beta_{1,actor[i]} + \beta_{1,group[i]}$$
$$\beta_{2i} = \beta_2 + \beta_{2,actor[i]} + \beta_{2,group[i]}$$

$$\alpha \sim Normal(0,1)$$
$$\beta_1 \sim Normal(0,1)$$
$$\beta_2 \sim Normal(0,1)$$

$$\begin{pmatrix} \alpha_{\text{actor}} \\ \beta_{1,\text{actor}} \\ \beta_{2,\text{actor}} \end{pmatrix} \sim \begin{pmatrix} \sigma_{\alpha a} \\ \sigma_{\beta_1 a} \\ \sigma_{\beta_2 a} \end{pmatrix} \circ L_a \cdot \begin{pmatrix} \hat{\alpha}_{ka} \\ \hat{\beta}_{1,ka} \\ \hat{\beta}_{2,ka} \end{pmatrix}$$

$$\sigma_{\alpha a} \sim Exponential(1)$$

$$\sigma_{\beta_1 a} \sim Exponential(1)$$
$$\sigma_{\beta_2 a} \sim Exponential(1)$$
$$L_a \sim LKJcorr(2)$$

$$\begin{pmatrix} \alpha_{\text{group}} \\ \beta_{1,\text{group}} \\ \beta_{2,\text{group}} \end{pmatrix} \sim \begin{pmatrix} \sigma_{\alpha g} \\ \sigma_{\beta_1 g} \\ \sigma_{\beta_2 g} \end{pmatrix} \circ L_g \cdot \begin{pmatrix} \hat{\alpha}_{kg} \\ \hat{\beta}_{1,kg} \\ \hat{\beta}_{2,kg} \end{pmatrix}$$

$$\sigma_{\alpha g} \sim Exponential(1)$$
$$\sigma_{\beta_1 g} \sim Exponential(1)$$
$$\sigma_{\beta_2 g} \sim Exponential(1)$$
$$L_g \sim LKJcorr(2)$$

```
from main import *
# Setup device-----------------------------------------------
m = bi(platform='cpu')
# Import data
m.read_csv("../data/chimpanzees.csv", sep=";")
m.df["block_id"] = m.df.block
m.df["treatment"] = 1 + m.df.prosoc_left + 2 * m.df.condition
m.data_to_model(['pulled_left', 'treatment', 'actor', 'block_id'])


def model(tid, actor, block_id, L=None, link=False):
    # fixed priors
    g = dist.normal(0, 1, name = 'g', shape = (4,))
    sigma_actor = dist.exponential(1, name = 'sigma_actor', shape = (4,))
    L_Rho_actor = dist.lkjcholesky(4, 2, name = "L_Rho_actor")
    sigma_block = dist.exponential(1, name = "sigma_block", shape = (4,))
    L_Rho_block = dist.lkjcholesky(4, 2, name = "L_Rho_block")

    # adaptive priors - non-centered
    z_actor = dist.normal(0, 1, name = "z_actor", shape = (4,7))
    z_block = dist.normal(0, 1, name = "z_block", shape = (4,3))
    alpha = deterministic(
        "alpha", ((sigma_actor[..., None] * L_Rho_actor) @ z_actor).T
    )
    beta = deterministic(
        "beta", ((sigma_block[..., None] * L_Rho_block) @ z_block).T
    )

    logit_p = g[tid] + alpha[actor, tid] + beta[block_id, tid]
    dist("L", dist.Binomial(logits=logit_p), obs=L)

    # compute ordinary correlation matrices from Cholesky factors
    if link:
        deterministic("Rho_actor", L_Rho_actor @ L_Rho_actor.T)
        deterministic("Rho_block", L_Rho_block @ L_Rho_block.T)
        deterministic("p", expit(logit_p))

# Run mcmc ------------------------------------------------
m.run(model)

# Summary ------------------------------------------------
m.sampler.print_summary(0.89)
```

19.10 Reference(s)

20 Gaussian Processes

20.1 General Principles

20.2 Considerations

Caution

- To capture complex, non-linear relationships in data where the underlying function is smooth but has an unknown functional form, GPs use a kernel .
- GPs assume normally distributed errors and may not be appropriate for all types of noise.
- The choice of kernel hyperparameters can significantly impact results; thus, GPs require choosing an appropriate kernel function that captures the expected behavior of your data.
- Through kernel definition, we can incorporate domain knowledge.
- They scale poorly with dataset size ($\mathrm{O}(\mathrm{n}^3)$ complexity) due to matrix operations; thus, memory requirements can be substantial for large datasets, which has led to neural networks being used instead to resolve large non-linear problems.

20.3 Example

20.4 Python

```python
from BI import bi
import jax.numpy as jnp
import numpyro
# Setup device------------------------------------------------
m = bi(platform='cpu')

# Import Data & Data Manipulation ------------------------------------------------
# Import
from importlib.resources import files
data_path = files('BI.resources.data') / 'Kline2.csv'
m.data(data_path, sep=';')

data_path2 = files('BI.resources.data') / 'Kline2.csv'
islandsDistMatrix = pd.read_csv(data_path2, index_col=0)

m.data_to_model(['total_tools', 'population'])
m.data_on_model["society"] = jnp.arange(0,10)# index observations
m.data_on_model["Dmat"] = islandsDistMatrix.values # Distance matrix


def model(Dmat, population, society, total_tools):
    a = m.dist.exponential(1, name = 'a')
    b = m.dist.exponential(1, name = 'b')
    g = m.dist.exponential(1, name = 'g')

    # non-centered Gaussian Process prior
    etasq = m.dist.exponential(2, name = 'etasq')
    rhosq = m.dist.exponential(0.5, name = 'rhosq')
    SIGMA = cov_GPL2(Dmat, etasq, rhosq, 0.01)
    k = m.dist.multivariatenormal(0, SIGMA, name = 'k')
    #k = m.gaussian.gaussian_process(Dmat, etasq, rhosq, 0.01, shape = (10,))
    k = m.gaussian.kernel_L2(Dmat, etasq, rhosq, 0.01)
    lambda_ = a * population**b / g * jnp.exp(k[society])
    m.dist.poisson(lambda_, obs=total_tools)

# Run sampler ------------------------------------------------
m.run(model)
m.summary()
```

20.5 R

```
library(BI)
pd=import('pandas')
# setup platform------------------------------------------------
m=importbi(platform='cpu')

# Import data ------------------------------------------------
m$data(paste(system.file(package = "BI"),"/data/Kline2.csv", sep = ''), sep=';')
islandsDistMatrix = pd$read_csv(paste(system.file(package = "BI"),"/data/islandsDistMatrix.cs
m$data_to_model(list('total_tools', 'population'))
m$data_on_model$society = jnp$arange(0,10, dtype='int64')
m$data_on_model$Dmat = jnp$array(islandsDistMatrix)


# Define model ------------------------------------------------
model <- function(Dmat, population, society, total_tools){
  a = bi.dist.exponential(1, name = 'a')
  b = bi.dist.exponential(1, name = 'b')
  g = bi.dist.exponential(1, name = 'g')

  # non-centered Gaussian Process prior
  etasq = bi.dist.exponential(2, name = 'etasq')
  rhosq = bi.dist.exponential(0.5, name = 'rhosq')
  z = bi.dist.normal(0,1, name = 'z', shape = c(10))
  r = m$kernel_sq_exp(Dmat, z, etasq, rhosq, 0.01)
  SIGMA = r[[1]]
  L_SIGMA = r[[2]]
  k = r[[3]]
  lambda_ = a * population**b / g * jnp$exp(k[society])
  m$poisson(lambda_, obs=total_tools)
}

# Run MCMC ------------------------------------------------
```

```
m$run(model) # Optimize model parameters through MCMC sampling

# Summary -------------------------------------------------
m$summary() # Get posterior distribution
```

20.6 Mathematical Details

20.6.1 *Formula*

-
-
-
-

-
-
-

-
-
-
-
-
-

20.6.2 *Bayesian model*

-
-
-
-
-
-
-

20.7 Notes

Note

Common kernel functions include:

- *Radial Basis Function* (RBF) or Squared Exponential Kernel:

$$k(x, x') = \sigma^2 \exp\left(-\frac{||x - x'||^2}{2l^2}\right)$$

- *Rational Quadratic Kernel*, this kernel is equivalent to adding together many RBF kernels with different length scales:

$$k(x, x') = \sigma^2 \left(1 + \frac{||x - x'||^2}{2l^2}\right)^{-\alpha}$$

- *Periodic kernel* allows for modeling functions that repeat themselves exactly:

$$k(x, x') = \sigma^2 \exp\left(-\frac{2\sin^2(\pi||x - x'||/p)}{l^2}\right)$$

- *Locally Periodic Kernel*:

$$k(x, x') = \sigma^2 \exp\left(-\frac{2\sin^2(\pi||x - x'||/p)}{l^2}\right) \exp\left(-\frac{||x - x'||^2}{2l^2}\right)$$

- GPs can be extended to classification problems using link functions.
- Multi-output problems can be addressed using matrix-valued kernels.
- Deep learning can be combined with GPs through Deep Kernel Learning.
- Computational tricks for large datasets include:
  - Sparse approximations (e.g., FITC, VFE)
  - Inducing points methods
  - Random Fourier features

20.8 Reference(s)

21 Measuring error

21.1 General Principles

21.2 Example

21.3 Python

```
from BI import bi

# Setup device------------------------------------------------
m = bi(platform='cpu')

# Import Data & Data Manipulation ------------------------------------------------
# Import
from importlib.resources import files
```

```
data_path = files('BI.resources.data') / 'WaffleDivorce.csv'
m.data(data_path, sep=';')
m.scale(['Divorce', 'Divorce SE', 'MedianAgeMarriage']) # Scale
dat = dict(
    D_obs = jnp.array(m.df['Divorce'].values),
    D_sd = jnp.array(m.df['Divorce'].values),
    A = jnp.array(m.df['MedianAgeMarriage'].values),
    N = m.df.shape[0]
)
m.data_on_model = dat # Send to model (convert to jax array)


# Define model ------------------------------------------------
def model(D_obs, D_sd, A, N):
    a = m.dist.normal(0, 0.2, name = 'a')
    bA = m.dist.normal(0, 0.5, name = 'bA')
    s = m.dist.exponential(1, name = 's')
    mu = a + bA * A + bM * M
    D_true = m.dist.normal(mu, s, name = 'D_true')
    m.normal(D_true , D_sd, obs = D_obs)

# Run MCMC ------------------------------------------------
m.run(model)  # Optimize model parameters through MCMC sampling

# Summary ------------------------------------------------
m.summary() # Get posterior distributions
```

21.4 Mathematical Details

21.4.1 *Bayesian formulation*

-
-
-
-
-
-
-

21.5 Notes

Note

This is an approach that can be extended to any kind of model previously described. For example, one could generate a Bernoulli measurement error model by generating a process for the probabilities of success and failure. We can even go further by potentially having an error rate that is present only in one of the two outcomes.

21.6 Reference(s)

22 Missing data

22.1 General Principles

22.2 Considerations

Caution

22.3 Example

22.4 Python

22.5 Mathematical Details

22.5.1 *Frequentist formulation*

22.5.2 *Bayesian formulation*

22.6 Notes

> ℹ Note

22.7 Reference(s)

23 Latent Variables

23.1 General Principles

23.2 Considerations

23.3 Example

```python
from BI import bi
import numpy as np
import jax.numpy as jnp

# Setup device------------------------------------------------
m = bi(platform='cpu')

# Data Simulation ------------------------------------------------
NY = 4  # Number of dependent variables or outcomes (e.g., dimensions for latent variables)
NV = 8  # Number of observations or individual-level data points (e.g., subjects)

# Initialize the matrix Y2 with shape (NV, NY) filled with NaN values, to be filled later
Y2 = np.full((NV, NY), np.nan)

# Generate the means and offsets for the data
# means: Generate random normal means for each of the NY outcomes
# offsets: Generate random normal offsets for each of the NV observations
means = m.dist.normal(0, 1, shape=(NY,), sample=True, seed=10)
offsets = m.dist.normal(0, 1, shape=(NV, 1), sample=True, seed=20)

# Fill the matrix Y2 with simulated data based on the generated means and offsets
# Each observation (i) is the sum of an individual-specific offset and an outcome-specific me
for i in range(NV):
    for k in range(NY):
        Y2[i, k] = means[k] + offsets[i]

# Simulate individual-level random effects (e.g., random slopes or intercepts)
# b_individual: A matrix of size (N, K) where N is the number of individuals and K is the nu
b_individual = BI.distribution.normal(0, 1, shape=(N, K), sample=True, seed=0)

# mu: Add an additional effect 'a' to the individual-level random effects 'b_individual'
# 'a' could represent a population-level effect or a baseline
mu = b_individual + a

# Convert Y2 to a JAX array for further computation in a JAX-based framework
Y2 = jnp.array(Y2)
```

```
# Set data ------------------------------------------------
dat = dict(
    NY = NY,
    NV = NV,
    Y2 = Y2
)
m.data_on_model = dat

# Define model ------------------------------------------------
def model(NY, NV, Y2):
    means = m.dist.normal(0, 1, shape=(NY,), name='means')
    offset = m.dist.normal(0, 1, shape=(NV, 1), name='offset')
    sigma = m.dist.exponential(1, shape=(NY,), name='sigma')
    tmp = jnp.tile(means, (NV, 1)).reshape(NV, NY)
    mu_l = tmp + offset
    m.normal(mu_l, jnp.tile(sigma, [NV, 1]), obs=Y2)

# Run sampler ------------------------------------------------
m.run(model)

# Summary ------------------------------------------------
m.summary()
```

23.4 Mathematical Details

23.5 Interpretation of Latent Variables

-

-

23.6 Use Cases

-

-

-

24 Principal Component Analysis

24.1 General Principles

24.1.1 Goal:

-
-

24.1.2 Use Cases

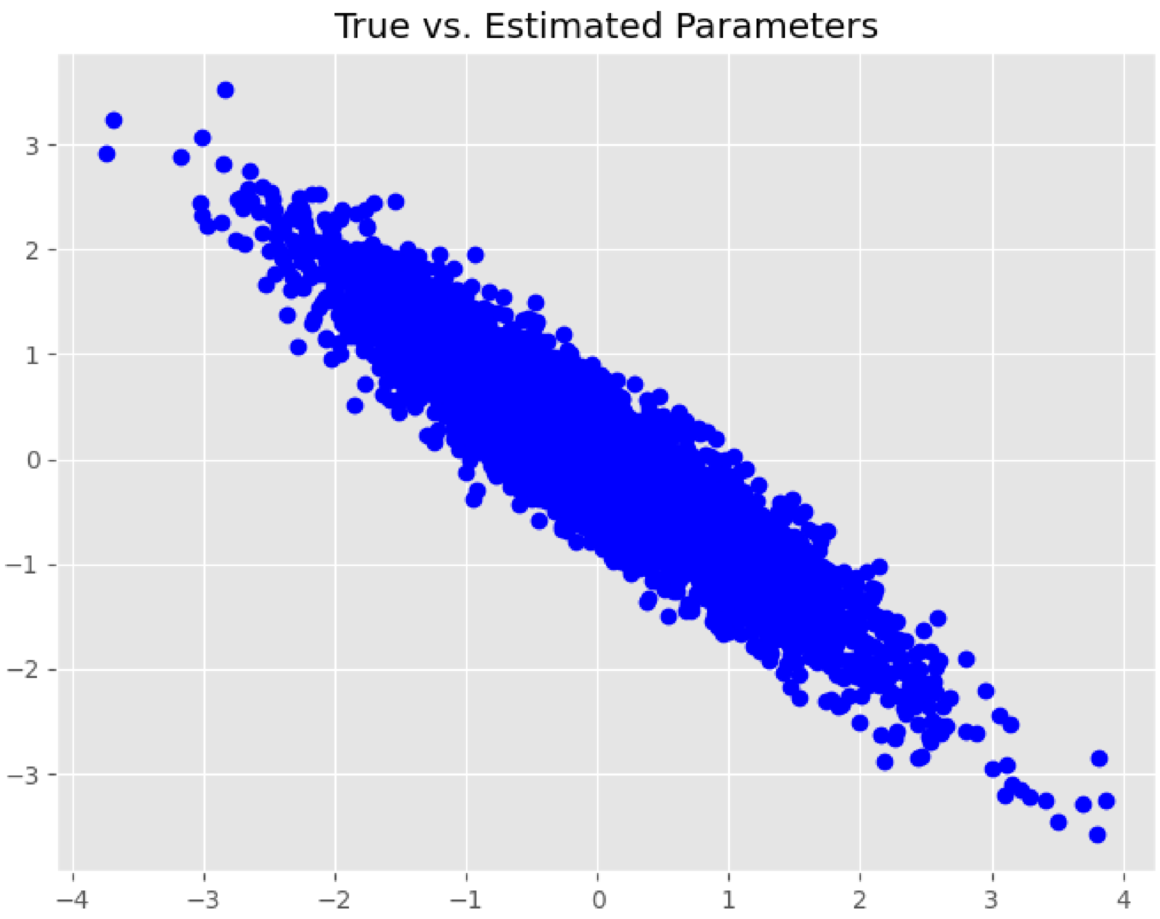
-
-
-
-

-

24.2 Considerations

24.3 Example

```
from main import *
import seaborn as sns

m = bi(platform='cpu')

# Data simulation ------------------------------------------

plt.style.use("ggplot")
warnings.filterwarnings('ignore')

num_datapoints = 5000
data_dim = 2
latent_dim = 1
stddv_datapoints = 0.5

# Simulate data
def sim_data(data_dim, latent_dim, num_datapoints, stddv_datapoints, seed = 0):
    w = bi.dist.normal(0, 1, shape=(data_dim, latent_dim), name='w', sample=True, seed=seed)
    z = bi.dist.normal(0, 1, shape=(latent_dim, num_datapoints), name='z', sample=True, seed
    x = bi.dist.normal(w @ z, stddv_datapoints, name='x', sample=True, seed=seed)
    return w, z, x

actual_w, actual_z, x_train =sim_data(data_dim, latent_dim, num_datapoints, stddv_datapoints
plt.scatter(x_train[0, :], x_train[1, :], color='blue', alpha=0.1)
plt.axis([-20, 20, -20, 20])
plt.title("Dataset")
plt.show()


# Model using simulated data
def model(x_train, data_dim, latent_dim, num_datapoints, stddv_datapoints, seed = 0):
    w = bi.dist.normal(0, 1, shape=(data_dim, latent_dim), name='w')
    z = bi.dist.normal(0, 1, shape=(latent_dim, num_datapoints), name='z')
    lk('Y', Normal(w @ z, stddv_datapoints), obs = x_train)

m.data_on_model = dict(
    x_train = x_train,
    data_dim = data_dim,
    latent_dim = latent_dim,
    num_datapoints = num_datapoints,
    stddv_datapoints = stddv_datapoints
)

m.run(model)
summary = m.summary()
real_data = jnp.concatenate([actual_w.flatten(), actual_z.flatten()]) # concatenate the actua
posteriors = summary.iloc[:,0]


plt.figure(figsize=(8, 6))
plt.plot(real_data, posteriors, marker='o', linestyle='None', color='b', label='Posteriors')
```



24.4 Mathematical Details

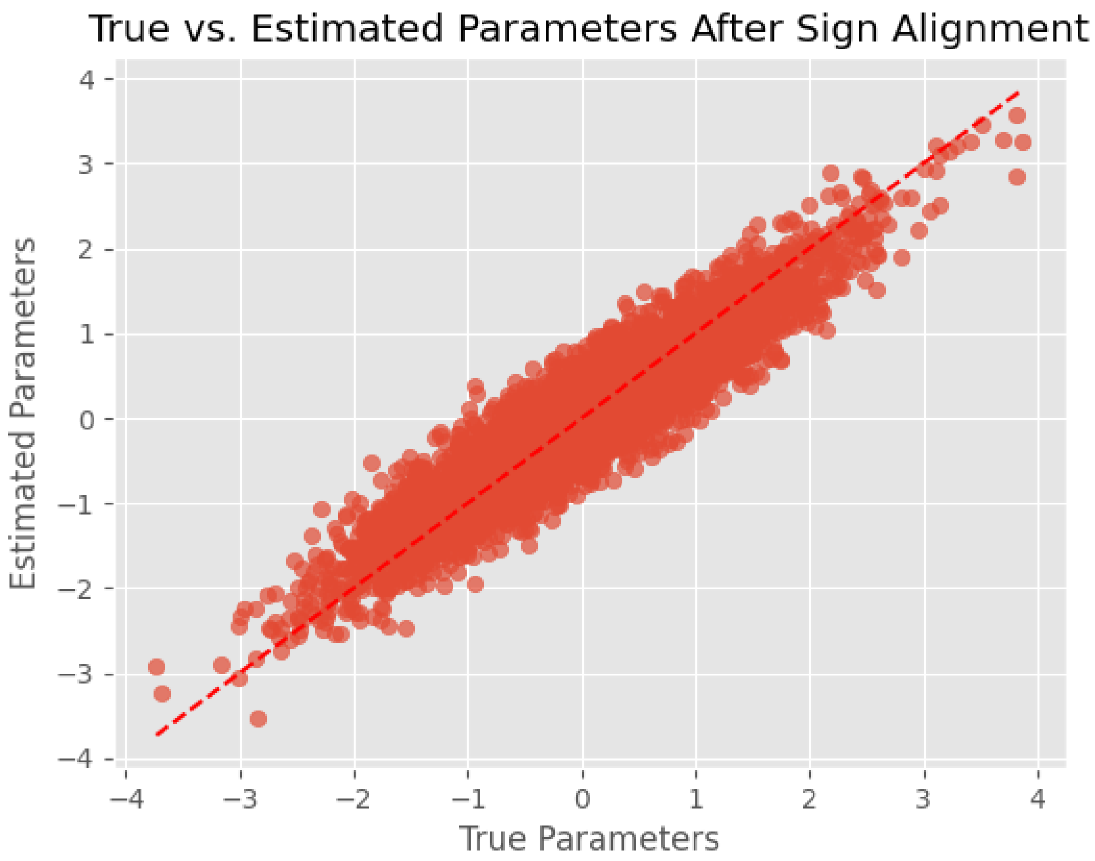
24.4.1 Formulation

-
-
-
-

24.5 Note

-

```
true_params = jnp.array(real_data)
estimated_params = jnp.array(posteriors)

# Compute dot product
dot_product = jnp.dot(true_params, estimated_params)

# Align signs if necessary
if dot_product < 0:
    estimated_params = -estimated_params

# Plot the aligned parameters
plt.scatter(true_params, estimated_params, alpha=0.7)
plt.plot([min(true_params), max(true_params)], [min(true_params), max(true_params)], 'r--')
plt.xlabel('True Parameters')
plt.ylabel('Estimated Parameters')
plt.title('True vs. Estimated Parameters After Sign Alignment')
plt.show()
```



24.6 Reference(s)

25

26 Dirichlet Process Gaussian Mixture Model

26.1 General Principles

1.
2.
3.

26.2 Considerations

Caution

- A DPMM is a Bayesian model that considers uncertainty in all its parameters. The core idea is to use the Dirichlet Process prior that allows for a potentially infinite number of clusters. In practice, we use a finite approximation called the Stick-Breaking Process .
- The key parameters and their priors are:
  - **Concentration** α: This single parameter controls the tendency to create new clusters. A low favors fewer, larger clusters, while a high allows for many smaller clusters. We typically place a `Gamma` prior on α to learn its value from the data.
  - **Cluster Weights** `w`: Generated via the Stick-Breaking process from α. These are the probabilities of drawing a data point from any given cluster.
  - **Cluster Parameters** (μ, σ): Each potential cluster has a mean μ and a covariance matrix σ. If the data have multiple dimensions, we use a multivariate normal distribution (see chapter, 14). Howver, if the data is one-dimensional,

we use a univariate normal distribution.

- The model is often implemented in its marginalized form . Instead of explicitly assigning each data point to a cluster, we integrate out this choice. This creates a smoother probability surface for the inference algorithm to explore, leading to much more efficient computation.

26.3 Example

26.4 Python

```python
from BI import bi

def dpmm_marginal(data, T=10):
    # 1) stick-breaking weights
    alpha = m.dist.gamma(1.0, 10.0,name='alpha')
    beta = m.dist.beta(1, alpha,name='beta',shape=(T-1,))
    w = numpyro.deterministic("w",dist.transforms.StickBreakingTransform()(beta))


    # 2) component parameters
    data_mean = jnp.mean(data, axis=0)
    with numpyro.plate("components", T):
        mu = m.dist.multivariatenormal(loc=data_mean, covariance_matrix=100.0*jnp.eye(D),name
        sigma = m.dist.halfcauchy(1,shape=(D,),event=1,name='sigma')# shape (T, D)
        Lcorr = m.dist.lkjcholesky(dimension=D, concentration=1.0,name='Lcorr')# shape (T, D

        scale_tril = sigma[..., None] * Lcorr  # shape (T, D, D)

    # 3) marginal mixture over obs
    m.dist.mixturesamefamily(
        mixing_distribution=m.dist.categoricalprobs(w,name='cat', create_obj=True),
        component_distribution=m.dist.multivariatenormal(loc=mu, scale_tril=scale_tril,name=
        name="obs",
        obs=data
```