Varying Intercepts Models

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 are grouped (e.g., by subject, location, or time period) and we expect the baseline level of the outcome to vary across these groups.

Considerations

> **Note**
>
> - 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 uniquely for each of the K declared groups.
> - In the code below, the intercept `alpha` for each of the k declared groups shares two priors, `a_bar` and `sigma`, which are respectively modeled by a Normal and an Exponential distribution.

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. This example is based on McElreath (2018).

Python (Raw)

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

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

# Import Data & Data Manipulation ------------------------------------------------
# Import
from importlib.resources import files
data_path = m.load.reedfrogs(only_path=True)
m.data(data_path, sep=';')
# Manipulate
m.df["tank"] = jnp.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.fit(model, progress_bar=False)

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

```
BI v 0.0.26 package loaded
jax.local_device_count 32

E0525 10:54:14.430444  506814 cuda_dnn.cc:523] Loaded runtime CuDNN library: 9.1.0 but source
E0525 10:54:14.432162  506814 cuda_dnn.cc:523] Loaded runtime CuDNN library: 9.1.0 but source
```

	mean	sd	hdi_5.5%	hdi_94.5%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
a_bar	1.35	0.26	0.94	1.75	0.00	0.00	3834.21	2968.85	1.0
alpha[0]	2.14	0.86	0.79	3.52	0.01	0.01	4979.85	2954.34	1.0
alpha[1]	3.08	1.11	1.38	4.79	0.02	0.01	4333.78	2411.09	1.0

	mean	sd	hdi_5.5%	hdi_94.5%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
alpha[2]	1.02	0.65	-0.02	2.05	0.01	0.01	5097.49	3017.37	1.0
alpha[3]	3.08	1.10	1.34	4.80	0.02	0.01	4551.94	2776.54	1.0
alpha[4]	2.14	0.90	0.66	3.47	0.01	0.01	4310.13	2200.45	1.0
alpha[5]	2.13	0.87	0.76	3.46	0.01	0.01	5083.25	2802.82	1.0
alpha[6]	3.07	1.11	1.34	4.83	0.02	0.01	4594.50	2655.09	1.0
alpha[7]	2.15	0.89	0.70	3.50	0.01	0.01	4199.79	2854.03	1.0
alpha[8]	-0.17	0.62	-1.09	0.90	0.01	0.01	4582.23	2424.98	1.0
alpha[9]	2.14	0.88	0.83	3.58	0.01	0.01	4571.70	2439.08	1.0
alpha[10]	0.99	0.68	-0.18	1.98	0.01	0.01	4971.55	2497.84	1.0
alpha[11]	0.59	0.64	-0.40	1.63	0.01	0.01	5735.23	3076.15	1.0
alpha[12]	1.00	0.68	-0.04	2.08	0.01	0.01	5589.19	3084.19	1.0
alpha[13]	0.20	0.61	-0.76	1.16	0.01	0.01	4779.38	2802.07	1.0
alpha[14]	2.14	0.85	0.78	3.47	0.01	0.01	4784.65	2608.70	1.0
alpha[15]	2.15	0.87	0.74	3.48	0.01	0.01	5729.82	2630.65	1.0
alpha[16]	2.91	0.79	1.62	4.06	0.01	0.01	3841.21	2417.77	1.0
alpha[17]	2.38	0.65	1.36	3.39	0.01	0.01	4773.26	2672.22	1.0
alpha[18]	2.02	0.60	1.04	2.94	0.01	0.01	4716.07	2648.00	1.0
alpha[19]	3.65	1.01	2.01	5.10	0.02	0.01	4360.67	2731.38	1.0
alpha[20]	2.38	0.66	1.33	3.34	0.01	0.01	5329.63	2530.29	1.0
alpha[21]	2.40	0.66	1.31	3.35	0.01	0.01	4993.93	2981.52	1.0
alpha[22]	2.40	0.65	1.35	3.39	0.01	0.01	4256.52	2654.77	1.0
alpha[23]	1.72	0.54	0.84	2.54	0.01	0.01	5807.75	2665.81	1.0
alpha[24]	-1.00	0.44	-1.67	-0.26	0.01	0.00	5890.22	3129.53	1.0
alpha[25]	0.17	0.41	-0.45	0.85	0.01	0.00	6071.92	2947.29	1.0
alpha[26]	-1.43	0.50	-2.22	-0.65	0.01	0.00	5964.15	3038.19	1.0
alpha[27]	-0.48	0.40	-1.10	0.18	0.01	0.00	5068.89	3092.02	1.0
alpha[28]	0.17	0.39	-0.51	0.73	0.01	0.00	5310.95	3095.96	1.0
alpha[29]	1.45	0.50	0.64	2.23	0.01	0.01	4834.71	2728.30	1.0
alpha[30]	-0.64	0.42	-1.28	0.02	0.01	0.00	5476.04	2773.30	1.0
alpha[31]	-0.31	0.40	-0.95	0.32	0.01	0.00	5023.47	3169.33	1.0
alpha[32]	3.21	0.79	2.02	4.43	0.01	0.01	4116.48	2283.51	1.0
alpha[33]	2.72	0.65	1.67	3.68	0.01	0.01	4588.37	2610.15	1.0
alpha[34]	2.71	0.65	1.70	3.67	0.01	0.01	4683.01	2567.85	1.0
alpha[35]	2.06	0.52	1.20	2.85	0.01	0.01	4844.07	2358.63	1.0
alpha[36]	2.07	0.51	1.31	2.91	0.01	0.01	4573.20	2752.89	1.0
alpha[37]	3.89	0.98	2.32	5.32	0.02	0.01	4231.43	2521.72	1.0
alpha[38]	2.70	0.64	1.64	3.62	0.01	0.01	4322.33	2569.17	1.0
alpha[39]	2.34	0.55	1.42	3.15	0.01	0.01	4705.56	2566.87	1.0
alpha[40]	-1.80	0.47	-2.52	-1.07	0.01	0.00	5663.69	3198.80	1.0
alpha[41]	-0.58	0.35	-1.13	-0.01	0.00	0.00	5231.16	2817.50	1.0
alpha[42]	-0.46	0.35	-1.02	0.10	0.00	0.00	5481.04	2856.80	1.0

	mean	sd	hdi_5.5%	hdi_94.5%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
alpha[43]	-0.34	0.34	-0.86	0.21	0.00	0.00	5622.36	3168.15	1.0
alpha[44]	0.58	0.34	-0.03	1.08	0.00	0.00	5259.45	3008.82	1.0
alpha[45]	-0.57	0.35	-1.13	-0.02	0.00	0.00	5481.65	3084.27	1.0
alpha[46]	2.07	0.53	1.26	2.91	0.01	0.01	5138.86	2678.20	1.0
alpha[47]	-0.00	0.34	-0.55	0.51	0.00	0.00	4967.65	3062.32	1.0
sigma	1.62	0.21	1.28	1.94	0.00	0.00	2349.15	2457.32	1.0

Python (Build in function)

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


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

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

# Define model -------------------------------------------------
def model(tank, surv, density):
    alpha = m.effects.varying_intercept(N_groups=48,group_id=tank, group_name = 'tank')
    m.dist.binomial(total_count = density, logits = alpha, obs=surv)

# Run sampler --------------------------------------------------
m.fit(model, progress_bar=False)

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

```
jax.local_device_count 32
```

	mean	sd	hdi_5.5%	hdi_94.5%	mcse_mean	mcse_sd	ess_bulk	ess_tail
global_intercept_tank	1.35	0.26	0.96	1.78	0.00	0.00	3128.12	2795.06
intercept_tank[0]	2.13	0.85	0.74	3.41	0.01	0.01	4896.42	2977.89
intercept_tank[1]	3.10	1.12	1.34	4.77	0.02	0.01	3333.45	2244.17
intercept_tank[2]	1.01	0.68	-0.11	2.04	0.01	0.01	4006.84	2518.35
intercept_tank[3]	3.09	1.10	1.43	4.88	0.02	0.01	4039.46	2370.83
intercept_tank[4]	2.16	0.89	0.66	3.50	0.01	0.01	4107.92	2448.18
intercept_tank[5]	2.15	0.88	0.69	3.39	0.01	0.01	4944.03	2837.00
intercept_tank[6]	3.07	1.11	1.42	4.85	0.02	0.01	4331.09	2669.68
intercept_tank[7]	2.14	0.89	0.72	3.50	0.01	0.01	4676.57	2710.76
intercept_tank[8]	-0.17	0.62	-1.17	0.80	0.01	0.01	4758.66	2968.60
intercept_tank[9]	2.14	0.89	0.72	3.49	0.01	0.01	4105.76	2176.15
intercept_tank[10]	1.00	0.69	-0.08	2.09	0.01	0.01	4926.33	2771.45
intercept_tank[11]	0.59	0.64	-0.39	1.62	0.01	0.01	5618.92	2532.68
intercept_tank[12]	0.99	0.68	-0.08	2.08	0.01	0.01	5699.74	3091.73
intercept_tank[13]	0.20	0.60	-0.77	1.16	0.01	0.01	4517.82	3282.02
intercept_tank[14]	2.13	0.83	0.86	3.48	0.01	0.01	4330.61	2838.22
intercept_tank[15]	2.13	0.88	0.69	3.43	0.01	0.01	5491.65	2816.60
intercept_tank[16]	2.92	0.80	1.67	4.17	0.01	0.01	3714.04	2200.71
intercept_tank[17]	2.40	0.67	1.28	3.36	0.01	0.01	4742.72	2658.82
intercept_tank[18]	2.02	0.60	1.00	2.94	0.01	0.01	4770.20	2576.51
intercept_tank[19]	3.64	0.98	2.14	5.19	0.01	0.01	4511.41	2613.67
intercept_tank[20]	2.37	0.63	1.40	3.38	0.01	0.01	4807.48	2924.05
intercept_tank[21]	2.40	0.66	1.30	3.33	0.01	0.01	5326.42	3409.34
intercept_tank[22]	2.40	0.66	1.41	3.49	0.01	0.01	4326.56	2649.60
intercept_tank[23]	1.71	0.53	0.88	2.53	0.01	0.00	6182.40	2765.81
intercept_tank[24]	-1.01	0.44	-1.69	-0.27	0.01	0.00	4693.38	3158.50
intercept_tank[25]	0.17	0.40	-0.45	0.84	0.01	0.00	5440.26	3232.16
intercept_tank[26]	-1.45	0.50	-2.24	-0.66	0.01	0.01	4787.53	3209.81
intercept_tank[27]	-0.48	0.41	-1.14	0.16	0.01	0.00	5371.22	3073.55
intercept_tank[28]	0.17	0.39	-0.48	0.75	0.01	0.00	5524.77	3064.63
intercept_tank[29]	1.45	0.49	0.65	2.20	0.01	0.00	4955.46	2997.19
intercept_tank[30]	-0.64	0.43	-1.29	0.04	0.01	0.00	5545.06	2746.35
intercept_tank[31]	-0.31	0.40	-0.95	0.32	0.01	0.00	5515.18	3231.43
intercept_tank[32]	3.21	0.79	1.96	4.41	0.01	0.01	4177.53	2401.12
intercept_tank[33]	2.73	0.66	1.62	3.70	0.01	0.01	4409.42	2737.95
intercept_tank[34]	2.71	0.64	1.71	3.69	0.01	0.01	4860.33	2668.98
intercept_tank[35]	2.06	0.51	1.26	2.89	0.01	0.01	4312.01	2396.63
intercept_tank[36]	2.07	0.51	1.27	2.89	0.01	0.01	4848.59	2593.87
intercept_tank[37]	3.89	0.98	2.32	5.34	0.01	0.01	4605.46	2403.61
intercept_tank[38]	2.71	0.64	1.65	3.63	0.01	0.01	4782.85	2836.53
intercept_tank[39]	2.33	0.55	1.42	3.18	0.01	0.01	3994.05	2812.19

	mean	sd	hdi_5.5%	hdi_94.5%	mcse_mean	mcse_sd	ess_bulk	ess_tail
intercept_tank[40]	-1.81	0.47	-2.51	-1.02	0.01	0.00	5382.02	3046.55
intercept_tank[41]	-0.57	0.35	-1.16	-0.05	0.00	0.00	5302.09	3115.69
intercept_tank[42]	-0.46	0.35	-1.00	0.11	0.01	0.00	4749.29	2987.73
intercept_tank[43]	-0.34	0.34	-0.89	0.19	0.00	0.00	5657.56	3111.34
intercept_tank[44]	0.58	0.35	0.01	1.13	0.01	0.00	4815.44	3005.92
intercept_tank[45]	-0.57	0.34	-1.10	-0.02	0.00	0.00	5504.29	3210.16
intercept_tank[46]	2.05	0.52	1.24	2.87	0.01	0.01	4768.77	2698.03
intercept_tank[47]	0.01	0.33	-0.54	0.51	0.00	0.00	4584.66	3164.46
sd_tank	1.62	0.21	1.29	1.96	0.00	0.00	2070.88	2143.18

R

```r
library(BayesianInference)

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

# Import data -------------------------------------------------
m$data(m$load$reedfrogs(only_path=T), sep=';')
m$df$tank = c(0:(nrow(m$df)-1)) # Manipulate
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$dist$binomial(total_count = density, logits = p, obs=surv)
}

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

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

Julia

```
using BayesianInference

# Setup device-------------------------------------------------
m = importBI(platform="cpu")

# Import Data & Data Manipulation ------------------------------------------------
# Import
data_path = m.load.reedfrogs(only_path = true)
m.data(data_path, sep=';')
m.df["tank"] = jnp.arange(m.df.shape[0])

# Define model -------------------------------------------------
@BI function model(tank, surv, density)
    alpha = m.effects.varying_intercept(N_groups=48,group_id=tank, group_name = "tank")
    m.dist.binomial(total_count = density, logits = alpha, obs=surv)
end

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

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

Mathematical Details

We model the relationship between the independent variable X and the outcome variable Y while accounting for varying intercepts α for each group where $k(i)$ give us group belonging for observation i, using the following equation:

$$Y_i \sim \text{Normal}(\mu_i, \sigma)$$
$$\mu_i = \alpha_{[k(i)]} + \beta X_i$$

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

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

$$\alpha_{[k]} \sim \text{Normal}(\bar{\alpha}, \varsigma)$$

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

Where:

- Y_i is the outcome variable for observation i.
- $\alpha_{[k(i)]}$ is the varying intercept corresponding to the group k of observation i.
- β is the regression coefficient.
- σ is a standard deviation parameter, which here has an Exponential prior that constrains it to be positive.
- $\bar{\alpha}$ is the overall mean intercept.
- ς is the variance of the intercepts across groups.

Notes

> **Note**
>
> - We can apply multiple variables similarly to Multiple continuous variables.
> - 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.

Reference(s)

McElreath, Richard. 2018. *Statistical Rethinking: A Bayesian course with examples in R and Stan.* Chapman; Hall/CRC.