Varying Slopes Models

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.

Considerations

> **Note**
>
> - 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 .
> - To construct the covariance matrix, we use an SRS decomposition where S is a diagonal matrix of standard deviations and R is a correlation matrix. To model the correlation matrix, we use an $LKJcorr$ distribution parametrized with a single control parameter η that controls the amount of regularization. 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 standard deviations in S are model with a prior that constrains them to strictly positive values.

Example

Below is an example code snippet demonstrating Bayesian regression with varying effects. This example is based on McElreath (2018).

Simulated data

Python (Raw)

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

# Import Data & Data Manipulation ------------------------------------------------
# Import
from importlib.resources import files
data_path = m.load.sim_multivariate_normal(only_path=True)
m.data(data_path, sep=',')
m.data_on_model = dict(
    cafe = jnp.array(m.df.cafe.values, dtype=jnp.int32),
    wait = jnp.array(m.df.wait.values, dtype=jnp.float32),
    N_cafes = len(m.df.cafe.unique()),
    afternoon = jnp.array(m.df.afternoon.values, dtype=jnp.float32)
)

# 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.multivariate_normal(jnp.stack([a, b]), cov, shape = [N_cafes], nam

    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.fit(model)
```

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

```
jax.local_device_count 32
```

```
  0%|          | 0/1000 [00:00<?, ?it/s]warmup:   0%|          | 1/1000 [00:00<15:36,  1.07it
arviz - WARNING - Shape validation failed: input_shape: (1, 500), minimum_shape: (chains=2, d
/home/sosa/work/3.12venv/lib/python3.12/site-packages/arviz/stats/diagnostics.py:991: Runtime
```

```
invalid value encountered in scalar divide
```

	mean	sd	hdi_5.5%	hdi_94.5%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_ha
Rho[0, 0]	1.00	0.00	1.00	1.00	0.00	NaN	500.00	500.00	NaN
Rho[0, 1]	-0.48	0.19	-0.78	-0.19	0.01	0.01	422.99	367.44	NaN
Rho[1, 0]	-0.48	0.19	-0.78	-0.19	0.01	0.01	422.99	367.44	NaN
Rho[1, 1]	1.00	0.00	1.00	1.00	0.00	0.00	454.62	436.09	NaN
a	3.52	0.20	3.24	3.87	0.01	0.01	746.37	288.40	NaN
a_b_cafe[0, 0]	3.55	0.23	3.21	3.92	0.01	0.01	452.61	381.98	NaN
a_b_cafe[0, 1]	-1.53	0.30	-1.98	-1.04	0.01	0.01	509.28	348.57	NaN
a_b_cafe[1, 0]	5.32	0.23	4.97	5.69	0.01	0.01	678.84	388.59	NaN
a_b_cafe[1, 1]	-1.37	0.32	-1.91	-0.91	0.01	0.02	717.67	407.82	NaN
a_b_cafe[2, 0]	3.50	0.23	3.19	3.91	0.01	0.01	584.24	402.89	NaN
a_b_cafe[2, 1]	-1.24	0.29	-1.66	-0.78	0.01	0.01	565.87	254.37	NaN
a_b_cafe[3, 0]	4.40	0.21	4.09	4.71	0.01	0.01	595.46	337.76	NaN
a_b_cafe[3, 1]	-1.28	0.28	-1.73	-0.85	0.01	0.01	511.32	319.38	NaN
a_b_cafe[4, 0]	3.56	0.22	3.22	3.92	0.01	0.01	651.57	395.53	NaN
a_b_cafe[4, 1]	-1.58	0.29	-2.04	-1.12	0.01	0.01	580.06	351.02	NaN
a_b_cafe[5, 0]	4.23	0.21	3.93	4.60	0.01	0.01	708.60	355.22	NaN
a_b_cafe[5, 1]	-1.73	0.29	-2.13	-1.21	0.01	0.01	657.42	399.55	NaN
a_b_cafe[6, 0]	4.08	0.23	3.70	4.42	0.01	0.01	536.86	282.44	NaN
a_b_cafe[6, 1]	-0.19	0.29	-0.64	0.25	0.01	0.01	501.31	399.55	NaN
a_b_cafe[7, 0]	3.78	0.22	3.45	4.12	0.01	0.01	842.35	436.09	NaN
a_b_cafe[7, 1]	-1.08	0.30	-1.52	-0.56	0.01	0.01	741.73	425.51	NaN
a_b_cafe[8, 0]	3.49	0.22	3.15	3.82	0.01	0.01	928.83	448.99	NaN
a_b_cafe[8, 1]	-1.51	0.29	-1.95	-1.01	0.01	0.01	900.17	323.82	NaN
a_b_cafe[9, 0]	3.25	0.21	2.93	3.57	0.01	0.01	633.95	365.86	NaN
a_b_cafe[9, 1]	-0.34	0.28	-0.82	0.04	0.01	0.01	652.68	393.66	NaN
a_b_cafe[10, 0]	3.29	0.22	2.92	3.58	0.01	0.01	810.20	365.91	NaN
a_b_cafe[10, 1]	-0.60	0.28	-1.06	-0.18	0.01	0.01	659.36	385.50	NaN

	mean	sd	hdi_5.5%	hdi_94.5%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_ha
a_b_cafe[11, 0]	3.83	0.20	3.50	4.13	0.01	0.01	577.10	304.11	NaN
a_b_cafe[11, 1]	-1.44	0.27	-1.82	-0.96	0.01	0.01	593.52	297.44	NaN
a_b_cafe[12, 0]	1.92	0.22	1.57	2.23	0.01	0.01	600.90	314.72	NaN
a_b_cafe[12, 1]	-0.77	0.30	-1.18	-0.24	0.01	0.01	724.12	342.76	NaN
a_b_cafe[13, 0]	4.37	0.22	4.04	4.71	0.01	0.01	549.10	369.49	NaN
a_b_cafe[13, 1]	-2.18	0.30	-2.64	-1.67	0.01	0.01	565.11	332.48	NaN
a_b_cafe[14, 0]	2.88	0.21	2.55	3.22	0.01	0.01	708.31	473.41	NaN
a_b_cafe[14, 1]	-0.73	0.29	-1.15	-0.25	0.01	0.01	644.40	297.55	NaN
a_b_cafe[15, 0]	2.26	0.22	1.90	2.59	0.01	0.01	598.20	408.34	NaN
a_b_cafe[15, 1]	-0.45	0.29	-0.90	-0.01	0.01	0.01	500.02	219.35	NaN
a_b_cafe[16, 0]	4.68	0.22	4.28	5.00	0.01	0.01	601.60	399.55	NaN
a_b_cafe[16, 1]	-2.08	0.30	-2.53	-1.54	0.01	0.01	584.76	437.40	NaN
a_b_cafe[17, 0]	2.46	0.21	2.17	2.84	0.01	0.01	647.23	279.36	NaN
a_b_cafe[17, 1]	-1.22	0.28	-1.67	-0.80	0.01	0.02	634.28	211.75	NaN
a_b_cafe[18, 0]	2.51	0.21	2.20	2.84	0.01	0.01	518.14	369.15	NaN
a_b_cafe[18, 1]	-0.16	0.28	-0.59	0.31	0.01	0.01	514.85	364.46	NaN
a_b_cafe[19, 0]	2.82	0.20	2.48	3.15	0.01	0.01	570.10	327.00	NaN
a_b_cafe[19, 1]	-0.57	0.27	-1.02	-0.16	0.01	0.01	766.08	470.00	NaN
b	-1.09	0.15	-1.34	-0.86	0.01	0.01	634.77	375.64	NaN
sigma	0.51	0.03	0.46	0.54	0.00	0.00	800.09	512.95	NaN
sigma_cafe[0]	0.92	0.15	0.67	1.14	0.01	0.01	541.45	397.11	NaN
sigma_cafe[1]	0.68	0.13	0.47	0.87	0.01	0.01	401.57	358.18	NaN

Python (Build in function)

```
from BI import bi

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

# Import Data & Data Manipulation ----------------------------------------------
# Import
from importlib.resources import files
data_path = m.load.sim_multivariate_normal(only_path=True)
m.data(data_path, sep=',')
m.data_on_model = dict(
    cafe = jnp.array(m.df.cafe.values, dtype=jnp.int32),
    wait = jnp.array(m.df.wait.values, dtype=jnp.float32),
    N_cafes = len(m.df.cafe.unique()),
```

```
    afternoon = jnp.array(m.df.afternoon.values, dtype=jnp.float32)
)


# Define model ------------------------------------------------
def model(cafe, wait, N_cafes, afternoon):

    sigma = m.dist.exponential( 1,  name = 'sigma')

    varying_intercept, varying_slope = m.effects.varying_effects(
        N_vars = 1,
        N_group = N_cafes,
        group_id = cafe,
        group_name = 'cafe',
        centered = False)


    mu = varying_intercept + varying_slope* afternoon
    m.dist.normal(mu, sigma, obs=wait)

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

R

```
library(BayesianInference)
jnp = reticulate::import('jax.numpy')

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

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

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

# 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.multivariate_normal(
    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

  bi.dist.normal(mu, sigma, obs=wait)
}

# 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.sim_multivariate_normal(only_path = true)

m.data(data_path, sep=',')
m.data_on_model = pybuiltins.dict(
    cafe = jnp.array(m.df.cafe.values, dtype=jnp.int32),
```

```
    wait = jnp.array(m.df.wait.values, dtype=jnp.float32),
    N_cafes = length(m.df.cafe.unique()),
    afternoon = jnp.array(m.df.afternoon.values, dtype=jnp.float32)
)


# Define model ------------------------------------------------
@BI function 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.multivariate_normal(jnp.stack([a, b]), cov, shape = [N_cafes], nam

    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)

end

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

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

Mathematical Details

Centered parameterization

The Gaussian Mixture Model is a hierarchical model where each data point is generated from one of K distinct multivariate Gaussian distributions.

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

$$\begin{pmatrix} \alpha_k \\ \beta_k \end{pmatrix} \sim \text{MultivariateNormal}\left(\begin{pmatrix} \bar{\alpha} \\ \bar{\beta} \end{pmatrix}, \text{diag}(\varsigma)\ \Omega\ \text{diag}(\varsigma) \right)$$

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

Where:

- $\begin{pmatrix} \bar{\alpha} \\ \bar{\beta} \end{pmatrix}$ is a vector composed from concatenating a parameter for the global intercept and a parameter vector of the global slopes.
- ς is a vector giving the standard deviation of the random effects for the intercept and slopes across groups.
- Ω is the correlation matrix.

Non-centered parameterization

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

$$\begin{pmatrix} \alpha_k \\ \beta_k \end{pmatrix} = \begin{pmatrix} \bar{\alpha} \\ \bar{\beta} \end{pmatrix} + \varsigma \circ \left(L \cdot \begin{pmatrix} \hat{\alpha}_k \\ \hat{\beta}_k \end{pmatrix} \right)$$

$$\bar{\alpha} \sim \text{Normal}(0, 1)$$
$$\bar{\beta} \sim \text{Normal}(0, 1)$$
$$\varsigma \sim \text{Exponential}(1)$$
$$L \sim \text{LKJ Cholesky}(\eta)$$

$$\hat{\alpha}_k \sim \text{Exponential}(1)$$

$$\hat{\beta}_k \sim \text{Exponential}(1)$$

- Where:
  - $\sigma_\alpha \sim \text{Exponential}(1)$ is the prior standard deviation among intercepts.
  - $\sigma_\beta \sim \text{Exponential}(1)$ is the prior standard deviation among slopes.
  - $L \sim \text{LKJcorr}(\eta)$ is the a cholesky factor of the correlation matrix matrix using the Cholesky Factor

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:

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

$$\mu_i = \alpha_i + \beta_{k(i)} X_{1i} + \gamma_{k(i)} X_{2i}$$

$$\begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} \sim \begin{pmatrix} \bar{\alpha} \\ \bar{\beta} \\ \bar{\gamma} \end{pmatrix} + \varsigma \circ \left(L \cdot \begin{pmatrix} \hat{\alpha}_k \\ \hat{\beta}_k \\ \hat{\gamma}_k \end{pmatrix} \right)$$

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

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

$$\varsigma \sim \text{Exponential}(1)$$
$$L \sim \text{LKJ Cholesky}(2)$$

$$\hat{\alpha}_k \sim \text{Exponential}(1)$$

$$\hat{\beta}_k \sim \text{Exponential}(1)$$

$$\hat{\gamma}_k \sim \text{Exponential}(1)$$

Notes

Note

- We can apply interaction terms similarly to Chapter 3.
- We can apply categorical variables similarly to Chapter 4.
- We can apply multiple random effects in Bayesian inference as follows:

```
def model(cafe, wait, N_cafes, afternoon, state):
    sigma = m.dist.exponential( 1,  name = 'sigma')

    #  Note: `alpha_bar` and `beta_bar` are set to zero to ensure the same intercepts for
    varying_intercept_1, varying_slope_1 = m.effects.varying_effects(
        N_vars = 1,
        N_group = N_cafes,
        group_id = cafe,
        group_name='cafe',
        alpha_bar = jnp.array([0]),
        beta_bar = jnp.array([0]),
        centered=True,
    )

    varying_intercept_2, varying_slope_2 = m.effects.varying_effects(
        N_vars = 1,
        N_group = 4,
        group_id = states,
        group_name='states',
        alpha_bar = jnp.array([0]),
        beta_bar = jnp.array([0]),
        centered=True,
    )

    varying_intercept = varying_intercept_1 + varying_intercept_2
    varying_slope = varying_slope_1 + varying_slope_2

    mu = varying_intercept + varying_slope[:,0] * afternoon
    m.dist.normal(mu, sigma, obs=wait
```

Where `state` is a second varying effect, representing cafes clustered by state.
Note: `alpha_bar` and `beta_bar` are set to zero to ensure the same intercepts for all varying effects.

Reference(s)

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