Liesel: News and Use

Johannes Brachem, Jens Lichter, Felipe Queiroz, Jinbo Hao, Gianmarco Callegher

University of Göttingen, Germany

September 2025

Contents

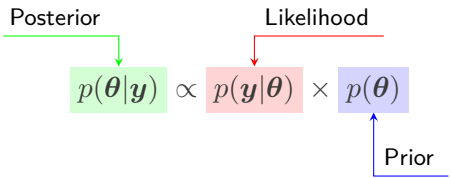
https://github.com/liesel-devs/statistics-retreat-2025

Liesel — Recap and New Developments

Johannes Brachem

python™

Liesel is for Bayesian modeling.



Two fundamental steps of the Bayesian workflow:

1. Set up the model by defining a likelihood and prior.
2. Approximate the posterior distribution.

Liesel: What?

Liesel is a Python library that supports Bayesian modeling.

Liesel is a Python library that supports Bayesian modeling.

1) Set up the model by defining a likelihood and prior.

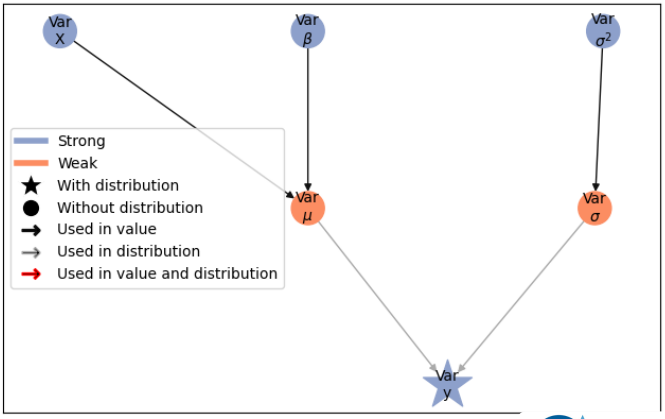
- Build models by connecting variables.
- Represent models as graphs.
- Be very explicit about intermediate calculations and variable transformations.
- Manipulate existing models.

2) Approximate the posterior distribution.

- Build custom inference algorithms.
- Automatic differentiation via JAX saves setup time.
- Just-in-time compilation via JAX saves computation time.

Models as graphs in Liesel

$$y \sim N(\mu, \sigma^2), \quad \mu = \mathbf{X}\boldsymbol{\beta}$$



Blocked MCMC Sampling in Liesel

1. We want to draw samples from the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{y})$
2. To make the problem easier, we partition the parameter vector into P blocks: $\boldsymbol{\theta} = [\boldsymbol{\theta}'_1, \ldots, \boldsymbol{\theta}'_P]'$
3. Then, for each iteration $t = 1, \ldots, T$, we draw samples from the full conditional distributions:
   - $\boldsymbol{\theta}_1^{[t]}$ drawn from $p(\theta_1|\boldsymbol{y}, \boldsymbol{\theta}_{-1}^{[t-1]})$ using a kernel
   - $\vdots$
   - $\boldsymbol{\theta}_P^{[t]}$ drawn from $p(\theta_P|\boldsymbol{y}, \boldsymbol{\theta}_{-P}^{[t]})$ using a kernel

A kernel is any technique to obtain a sample from a full conditional distribution. Liesel currently offers Gibbs updates, Metropolis-Hastings with different proposals (random walk, iteratively weighted least squares), Hamiltonian Monte Carlo, and the No-U-turn Sampler.

Liesel development team

Hannes Riebl
City of Lübeck, Germany

Paul Wiemann
Ohio State University

Johannes Brachem
University of Göttingen, Germany

Gianmarco Callegher
University of Göttingen, Germany

What's the difference to other software?

There are great packages for applying common models.

Liesel is intended for development & experimentation

Images: ChatGPT

Liesel: News

- Prior predictive sampling
  (*big update in v0.4.0*)
- MCMC-Sampling setup
  (*updated in v0.4.0*)
- Predictions
  (*new in v0.4.0*)
- Posterior predictive sampling
  (*new in v0.4.0*)

Liesel: News

- Prior predictive sampling
  (*big update in v0.4.0*)
- MCMC-Sampling setup
  (*updated in v0.4.0*)
- Predictions
  (*new in v0.4.0*)
- Posterior predictive sampling
  (*new in v0.4.0*)

Structured Additive Models in Liesel

Experimental add-on library `liesel_gam` (v.0.0.5, https://github.com/liesel-devs/liesel_gam) will receive a big update later this year.

Mixture models in Liesel: the zero-and-one inflated beta regression models

Felipe Queiroz

We will focus on bounded continuous data in the unit interval that may include the values 0 and/or 1:

- proportion of family income spent on children's education;
- mortality or infection rate for a given disease;
- proportion of natural vegetation area preserved in a region;
- fraction of electricity generated from renewable sources;

and many others...

Natural approach: regression model where the dependent variable has a probability distribution on $(0, 1)$ with a point-mass at 0 and/or 1 (mixture discrete-continuous distribution).

- **Inflated beta regression models** (Ospina & Ferrari, 2010, 2012)

The zero-and-one inflated beta distribution

The zero-and-one inflated beta distribution: mixture between a beta distribution and a Bernoulli distribution.

$$\text{BEINF}(y; p_0, p_1, \mu, \phi) = p_2 \text{Ber}(y; p_1/p2) + (1 - p_2)F(y; \mu, \phi), \quad y \in [0, 1],$$

where $p_2 = p_0 + p_1$ is the mixture parameter, $0 < p_0, p_1, \mu < 1$, $p_0 + p_1 < 1$, $\phi > 0$, and $F(\cdot; \mu, \phi)$ is the CDF of a beta-distributed random variable (Ferrari & Cribari-Neto, 2004).

Remark: the zero-or-one inflated beta distribution is defined by replacing $\text{Ber}(y; p_1/p2)$ with $\mathbb{I}_{[c,\infty)}(y)$, $c = 0$ or $c = 1$, depending on the inflation type. One sets $p_1 = 0$ for zero inflation and setting $p_0 = 0$ for one inflation.

The zero-and-one inflated beta distribution

The PDF of a zero-and-one inflated beta distributions is given by

$$f(y; p_0, p_1, \mu, \phi) = \left\{ p_0^{\mathbb{I}_{\{0\}}(y)} p_1^{\mathbb{I}_{\{1\}}(y)} (1-p_2)^{\mathbb{I}_{(0,1)}(y)} \right\} g(y; \mu, \phi)^{\mathbb{I}_{(0,1)}(y)}, \quad y \in [0,1],$$

where $p_0 = P(y = 0)$, $p_1 = \mathbb{P}(y = 1)$, and $g(y; \mu, \phi)$ is the PDF of the beta distribution with parameters μ and ϕ. We write $y \sim \text{BEINF}(p_0, p_1, \mu, \phi)$.

The PDF f factorizes in two terms: one depending only on (p_0, p_1) (discrete part) and the other only on (μ, ϕ) (continuous part). Hence, (p_0, p_1) and (μ, ϕ) are separable.

The zero-and-one inflated beta regression models

We define the zero-and-one inflated beta regression models as follows.

$$y_i \overset{\text{ind}}{\sim} \text{BEINF}(p_{0,i}, p_{1,i}, \mu_i, \phi), \quad i = 1, \ldots, n;$$

$$\log\left(\frac{p_{0,i}}{p_{2,i}}\right) = \eta_{p_0 p_2,i} = \boldsymbol{z_i}^\top \boldsymbol{\gamma},$$
$$\log\left(\frac{p_{1,i}}{p_{2,i}}\right) = \eta_{p_1 p_2,i} = \boldsymbol{s_i}^\top \boldsymbol{\tau},$$
$$\text{logit}(\mu_i) = \eta_{\mu,i} = \boldsymbol{x_i}^\top \boldsymbol{\beta},$$

$\boldsymbol{\theta} = (\boldsymbol{\gamma}^\top, \boldsymbol{\tau}^\top, \boldsymbol{\beta}^\top, \phi)^\top$ is the parameter vector;

$\boldsymbol{z}_i = (z_{i1}, \ldots, z_{im})^\top$, $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{ip})^\top$, and $\boldsymbol{s}_i = (s_{i1}, \ldots, s_{iq})^\top$ are the observations of the covariates.

The likelihood function of $\boldsymbol{\theta}$ is $L(\boldsymbol{\theta}) = L_1(\boldsymbol{\gamma}, \boldsymbol{\tau}) L_2(\boldsymbol{\beta}, \phi)$;

- $L_1(\boldsymbol{\gamma}, \boldsymbol{\tau}) = \prod_{i=1}^{n} \left\{ p_{0,i}^{\mathbb{I}_{\{0\}}(y_i)} p_{1,i}^{\mathbb{I}_{\{1\}}(y_i)} (1 - p_{2,i})^{\mathbb{I}_{(0,1)}(y_i)} \right\}$: likelihood function of a regression model for a three level multinomial response (discrete part), i.e.,

$$(\mathbb{I}_{\{0\}}(y_i), \mathbb{I}_{\{1\}}(y_i), \mathbb{I}_{(0,1)}(y_i)) \sim \text{Multinomial}(1, p_{0,i}, p_{1,i}, p_{2,i}).$$

- $L_2(\boldsymbol{\beta}, \boldsymbol{\tau}, \lambda) = \prod_{i:y_i \in (0,1)} g(y_i; \mu_i, \phi)$: likelihood function of a beta regression model based on the observations within $(0, 1)$ (continuous part).

We implement the zero-and-one inflated beta regression model under the Bayesian approach in Liesel.

Normal priors with mean 0 and variance 100^2 are assigned to the γ's, τ's, and β's, while a Gamma$(0.001, 0.001)$ prior is used for ϕ.

The main idea is to split the data into two parts and fit a multinomial regression for the discrete part and a beta regression for the continuous part.

We use the `lsl.Calc` function to split the data as follows.

Continuous part

```
y_continuous_calc = lsl.Calc(
    lambda y: jnp.asarray(y[(y > 0.0) & (y < 1.0)], dtype=jnp.float32),
    y=yvar
)
y_cont = lsl.Var(
    y_continuous_calc,
    distribution=lsl.Dist(tfd.Beta,
                          concentration1=shape1,
                          concentration0=shape2),
    name="$y \\in (0, 1)$",
)
```

Discrete part

```
dichotomized_calc = lsl.Calc(
  lambda y: jax.nn.one_hot(
      jnp.where(y == 0.0, 0,
                jnp.where(y == 1.0, 1, 2)),
      num_classes=3
  ).astype(jnp.float32),
  y=yvar
)
y_dichotomized = lsl.Var(
    dichotomized_calc,
    distribution=lsl.Dist(
        tfd.Multinomial,
        total_count=jnp.array(1, dtype=jnp.float32),
        logits=logits
    ),
    name="(I_{\\{0\\}}(y), I_{\\{1\\}}(y), I_{(0,1)}(y))",
)
```

The model is then specified using both `y_dichotomized` and `y_cont`:

Model

```
model = lsl.Model([y_dichotomized, y_cont])
```

Liesel implementation

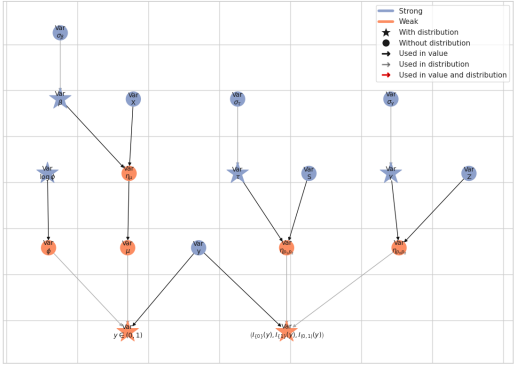


Figure 1: Plot of model variables.

One can implement the zero-inflated beta regression model by modifying both the calculation of the discrete variable and its associated distribution, as follows:

Discrete part

```
dichotomized_calc = lsl.Calc(
    lambda y: jnp.where(y == 0.0, 1.0, 0.0).astype(jnp.float32),
    y=yvar
)
y_dichotomized = lsl.Var(
    dichotomized_calc,
    distribution=lsl.Dist(tfd.Bernoulli, logits=logit_p0),
    name="I_{\\{0\\}}(y)",
)
```

- Structured additive predictors (Fahrmeir et al., 2021, Chapter 9) can be implemented using the `liesel_gam` package.
- Different distributions may be assigned to the continuous part.
- Inflated regression models for positive response variables can be implemented in the same way, provided that the parameters of the discrete and continuous components are separable.

Distributional Regression with Bivariate Response for Plot based Design Studies

Jens Lichter

- Bivariate Gaussian distribution:

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix} \right)$$

- Bivariate Gaussian distribution:

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix} \right)$$

- Modeling mean, scale and correlation for bivariate Gaussian response:

$$\begin{aligned} \mu_{ti} &= \beta_0^{\mu_t} + \beta_1^{\mu_t} x_i \quad \forall\, t = 1, 2, \\ \log(\sigma_{ti}) &= \beta_0^{\sigma_t} + \beta_1^{\sigma_t} x_i \quad \forall\, t = 1, 2, \end{aligned}$$

$$\varsigma(\rho) = \beta_0^{\rho} + \beta_1^{\rho} x_i.$$

- Simulate bivariate response conditional on binary variable x:

$$\mu_{ti} = 2 + 3x_i \qquad \forall\, t = 1, 2,$$

$$\log(\sigma_{ti}) = \frac{1}{2} x_i \qquad \forall\, t = 1, 2,$$

$$\varsigma(\rho) = 1 - 2x_i.$$

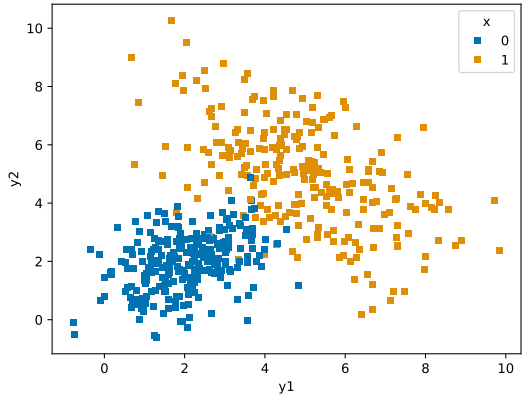


Figure 2: Simulated bivariate Gaussian data with binary variable x influencing location, scale and correlation.

Fit in Python

- Use `pyvireg` package to set up model:

```
li = linear('x', dummy=True, coef_prior=(0, 100))
fo = {'mean1': li, 'mean2': li,
      'sigma1': li, 'sigma2': li,
      'rho12': li}
comp = build_predictors(
   y=['y1', 'y2'], formula=fo, df=dat
)

```

- Fit with `Liesel` within `pyvireg`:

```
fit = dgam(
    components=comp, family='BiGaussian', warmup=500,
  nsamples=500, thin=2, nchains=3, kernel_sampler='IWLS'
)

```

- Extracting results:

```
fit_reform = DGAMBayes(fit)
fit_reform.linear_summary()

```

mean1 equation:

	0.05% CI	mean	0.95% CI
intercept	1.9500	2.0465	2.1489
x::1	2.7260	2.9283	3.1385

⋮

sigma2 equation:

	0.05% CI	mean	0.95% CI
intercept	-0.0877	-0.0214	0.0506
x::1	0.5068	0.6092	0.7072

rho12 equation:

	0.05% CI	mean	0.95% CI
intercept	0.5951	0.7828	0.9799
x::1	-2.2673	-1.9696	-1.6740

- Simulate bivariate response conditional on continuous variables x_1 and x_2, treatment variable z (treatments A, B and C) and spatial variable $\boldsymbol{s}$:

$$\mu_{ti} = \beta_0^{\mu_t} + \beta_1^{\mu_t} x_{ti} + \beta_2^{\mu_t} z_{i\mathsf{B}} + \beta_3^{\mu_t} z_{i\mathsf{C}} + \sum_{q=1}^{\tilde{q}} f_{geo,q}^{\mu_t}(\boldsymbol{s}_{ik}) \quad \forall\, t = 1, 2,$$

$$\log(\sigma_{ti}) = \beta_0^{\sigma_t} + \beta_1^{\sigma_t} x_{ti} + \beta_2^{\sigma_t} z_{i\mathsf{B}} + \beta_3^{\sigma_t} z_{i\mathsf{C}} \quad \forall\, t = 1, 2,$$

$$\varsigma(\rho) = \beta_0^{\rho} + \beta_1^{\rho} x_{1i} + \beta_2^{\rho} z_{i\mathsf{B}} + \beta_3^{\rho} z_{i\mathsf{C}}.$$

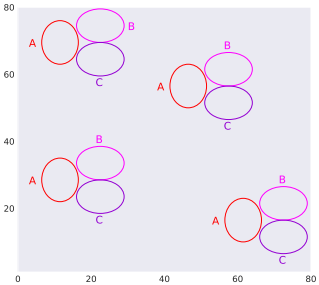


Figure 3: Simulated plots in space with always 3 plots in close proximity. Illustrated are simulated plot based spatial effect (left) and within plot spatial effects (right) based on a stationary Gaussian random field.

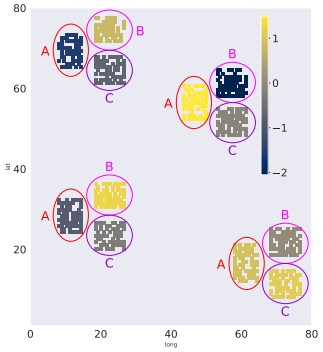


Figure 3: Simulated plots in space with always 3 plots in close proximity. Illustrated are simulated plot based spatial effect (left) and within plot spatial effects (right) based on a stationary Gaussian random field.

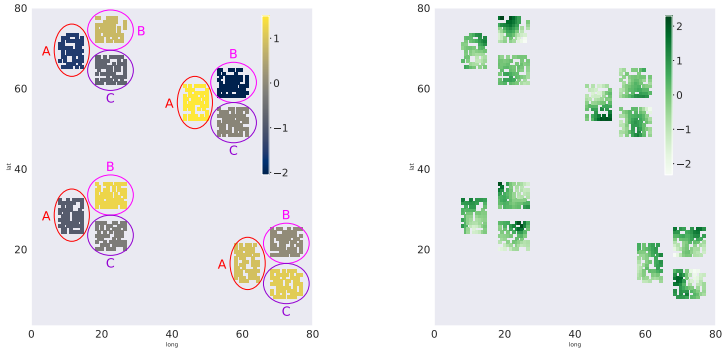


Figure 3: Simulated plots in space with always 3 plots in close proximity. Illustrated are simulated plot based spatial effect (left) and within plot spatial effects (right) based on a stationary Gaussian random field.

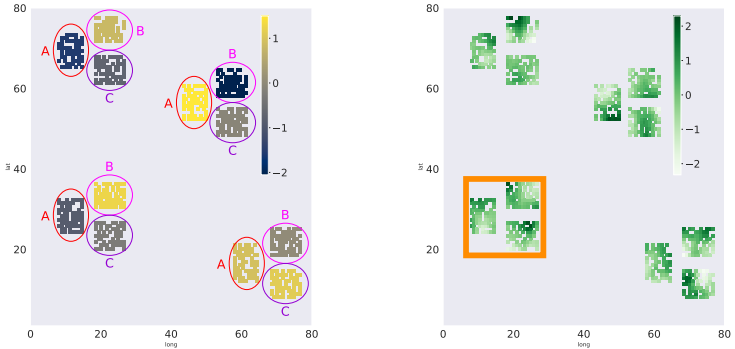


Figure 3: Simulated plots in space with always 3 plots in close proximity. Illustrated are simulated plot based spatial effect (left) and within plot spatial effects (right) based on a stationary Gaussian random field.

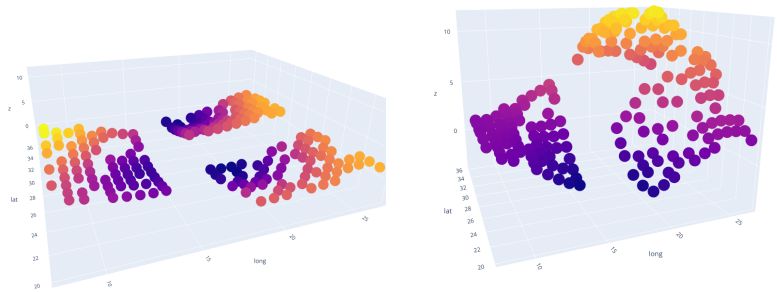


Figure 4: Sum over basis functions for whithin plot spatial effect centered (left) and uncentered (right).

Fit in Python

- Use `pyvireg` package to set up model:

```
li = linear(['x', 'treat'],
            dummy=[False, 'A'], coef_prior=(0, 100))
pl = cle('pl',    tau2_hyper    =(0.1, 0.1))
wpl = geo(['long', 'lat', 'pl'],
          kn=60, range_param=3, corrfu='matern1',
          blocked_pen=True,    tau2_hyper    =(.1, .1))
fo = {'mean1': li + pl + wpl, 'mean2': li + pl + wpl,
      'sigma1': li, 'sigma2': li,
      'rho12': li}
comp = build_predictors(y=['y1', 'y2'], formula=fo, df=dat)

```

- Fit with `Liesel` within `pyvireg`:

```
fit = dgam(components=comp, family='BiGaussian')

```

Fit in Python

- Use `pyvireg` package to set up model:

```
wpl = geo(['long', 'lat', 'pl', 'pl'],
          kn=60, range_param=3, corrfu='matern1',
          blocked_pen=True,    tau2_hyper    =(.1, .1))
fo = {'mean1': li + pl + wpl, 'mean2': li + pl + wpl,
      'sigma1': li, 'sigma2': li,
      'rho12': li}
comp = build_predictors(y=['y1', 'y2'], formula=fo, df=dat)

```

- Fit with `Liesel` within `pyvireg`:

```
fit = dgam(components=comp, family='BiGaussian')

```

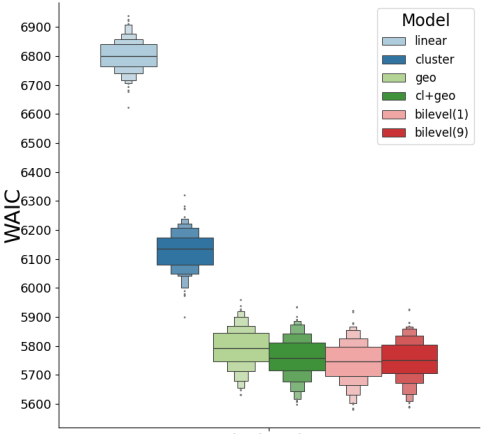


Figure 5: The WAIC across different models.

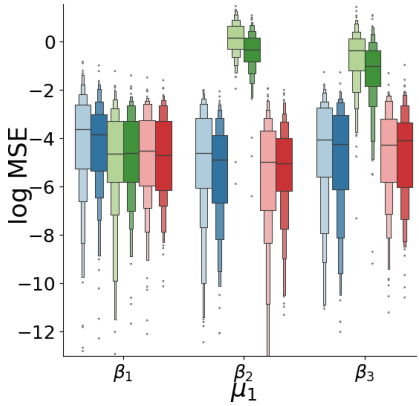


Figure 6: How good is the model in capturing the linear effects in μ_1 (left) and the intercept of ρ (right, true is red line $= 1$).

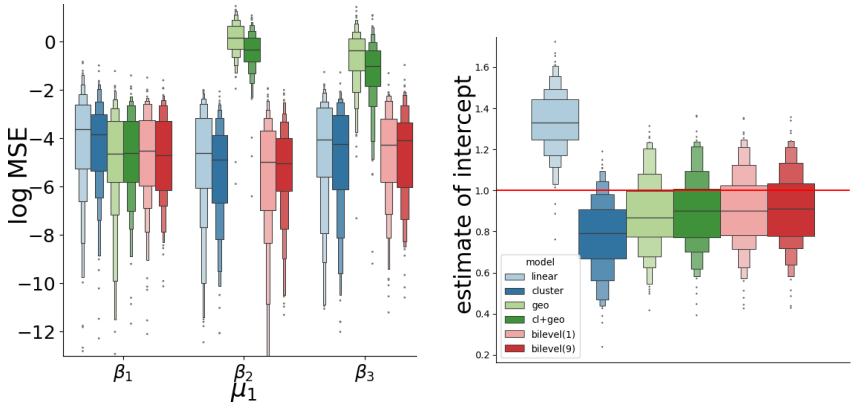


Figure 6: How good is the model in capturing the linear effects in μ_1 (left) and the intercept of ρ (right, true is red line $= 1$).

- You can use the `pyvireg`-package to easily set up distributional regression models.

- You can use the `pyvireg`-package to easily set up distributional regression models.
- For plot-based design studies you can set up the specific spatial structures in the `pyvireg`-package that might help better separating treatment, plot and within plot spatial effects.

Penalized Transformation Model with Censored Survival Time

Jinbo Hao

Motivation: Traditional AFT models rely on a fixed error distribution.

Problem: Misspecification leads to biased results.

Research Question: How can we relax strict assumptions on the error distribution?

Proposed Solution: Penalized Transformation Models (PTMs) with modified likelihood contributions for censored survival times.

Base Model: Location–scale regression

$$T = \mu(\boldsymbol{x}) + \sigma(\boldsymbol{x})\varepsilon \tag{1}$$

Extension in PTM:

- Residual distribution defined via a monotone transformation: $F_\varepsilon(\varepsilon) = F_Z(h(\varepsilon))$
- $h(\cdot)$ estimated nonparametrically (e.g., B-splines)
- Allows flexible, data-driven shape of the conditional distribution

Definition:

- **Uncensored:** The exact event time is observed $\rightarrow$ no bounds are needed $(T_i = t_i)$.
- **Right censoring:** The event has not yet occurred by the observation time $\rightarrow$ only a lower bound L_i is known $(T_i > L_i)$.
- **Left censoring:** The event has already occurred before observation starts $\rightarrow$ only an upper bound U_i is known $(T_i \leq U_i)$.
- **Interval Censoring:** The event occurs within a known time window $\rightarrow$ both lower and upper bounds are observed $(L_i < T_i \leq U_i)$.

Setup: Survival time T with density $f(t)$, residual $\varepsilon = \frac{t-\mu(x)}{\sigma(x)}$, and transformation $h(\cdot)$

- Uncensored Observation:

$$L_u = f_T(t_i|x_i) = f_Z(h(\varepsilon_i)) \cdot h'(\varepsilon_i) \cdot \frac{1}{\sigma(x_i)} \tag{2}$$

- Right Censored Observation:

$$L_r = P(T > t_i \mid x_i) = 1 - F_Z\Big(h\Big(\tfrac{t_i-\mu(x_i)}{\sigma(x_i)}\Big)\Big) \tag{3}$$

Likelihood Contribution for Censored Time (Continued)

- Left Censored Observation:

$$L_l = F_Z\left(h\left(\frac{\bar{t} - \mu(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right)\right) \quad \text{for } t \in (\infty, \bar{t}). \tag{4}$$

- Interval Censored Observation:

$$L_{in} = F_Z\left(h\left(\frac{\bar{t} - \mu(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right)\right) - F_Z\left(h\left(\frac{\underline{t} - \mu(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right)\right) \quad \text{for } t \in (\underline{t}, \bar{t}) \tag{5}$$

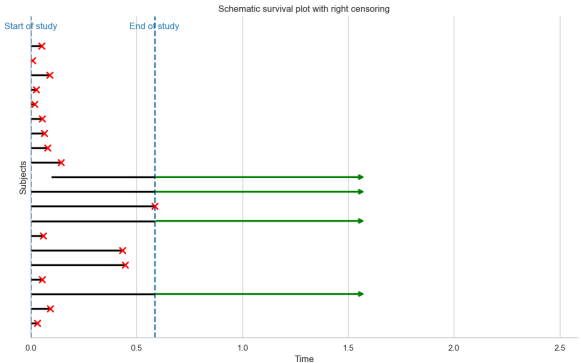


Figure 7: Plot of observations.

Illustration Example (Continued)

Build Model

```
class RightCensoredNormal(tfd.Distribution):
    ...
    def _log_prob(self, value):
        x = jnp.asarray(value)

        lp_unc = self._normal.log_prob(x) # Uncensored: density
        lp_cen = jnp.log(self._normal.survival_function(x)) # Censored: log survival

        return jnp.where(self._is_censored == 1.0, lp_cen, lp_unc)
```

Illustration Example (Continued)

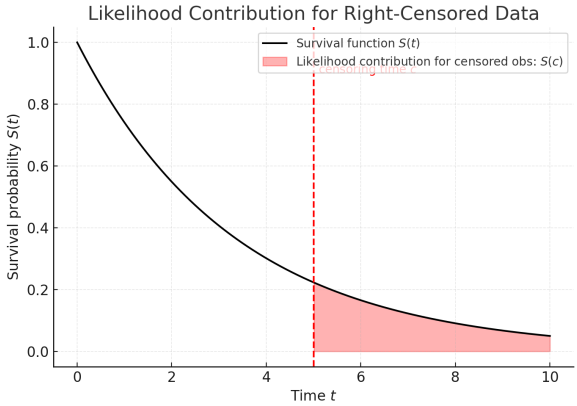


Figure 8: Plot of likelihood contribution for right censoring.

Build Model

```
1 import liesel.model as lsl
2 import liesel.goose as gs
3 dist = lsl.Dist(RightCensoredNormal, loc=mu, scale=sigma, is_censored=log_y_obs[:, 4])
4 y = lsl.Var.new_obs(log_y_obs[:, 0], distribution=dist, name="y")
5 model = lsl.Model([y], to_float32=False)
6 model.plot_vars()
```

Illustration Example (Continued)

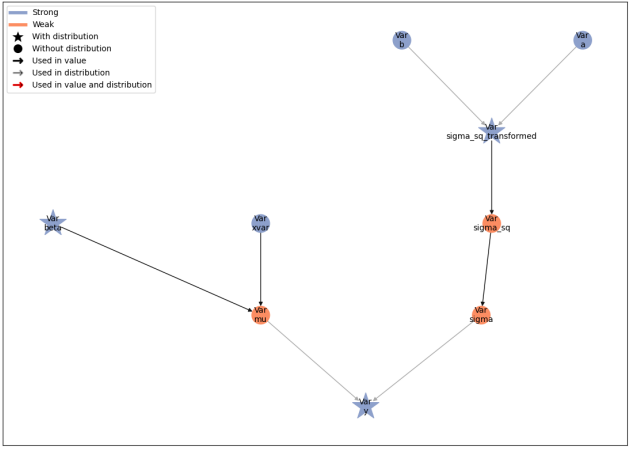


Figure 9: Plot of the Model.

Illustration Example (Continued)

Build Model with censored likelihood

```
eb = gs.LieselMCMC(model).get_engine_builder(seed=1, num_chains=4)
eb.set_duration(warmup_duration=300, posterior_duration=500)
engine = eb.build()
engine.sample_all_epochs()
result = engine.get_results()
gs.Summary(result)
gs.plot_param(result, "loc")
gs.plot_trace(result)
```

Illustration Example (Continued)

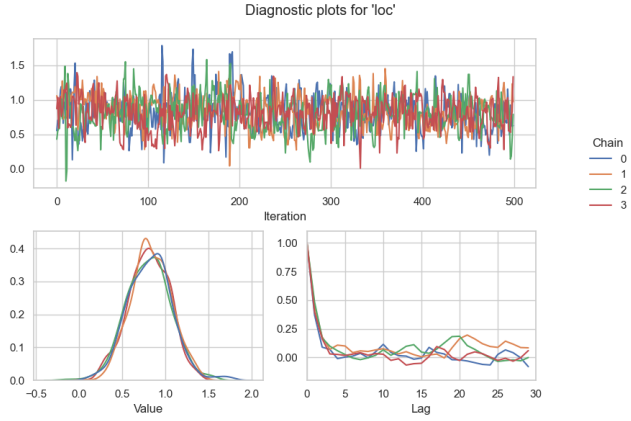


Figure 10: Trace Plot, density plot, and autocorrelation plot.

Illustration Example (Continued)

Build Model with normal distribution

```
import liesel.model as lsl
import liesel.goose as gs
dist = lsl.Dist(tfd.Normal, loc = mu, scale = sigma)
y = lsl.Var.new_obs(log_y_obs[:, 0], distribution=dist, name="y")
model = lsl.Model([y])
model.plot_vars()
```

Table 1: Comparison of parameter estimates: censored likelihood vs. Normal PDF

	Censored likelihood		Normal PDF	
Parameter	**Mean**	**SD**	**Mean**	**SD**
$\beta_{(0)}$	-1.969173	0.053905	-2.613816	0.075089
$\beta_{(1)}$	1.038472	0.044150	0.896986	0.065016

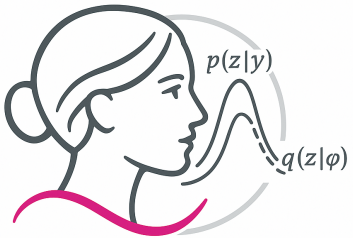
Summary

- Censoring changes how we build the likelihood
  - Uncensored $\rightarrow$ use density
  - Right-censored $\rightarrow$ use survival probability
- In PTMs, this is handled through the residuals and transformation
- The `log_prob` code mirrors the theory: density for uncensored, survival for censored
- Toy data plot showed the two groups clearly

Key point: Correct likelihood contributions are crucial for censored data.

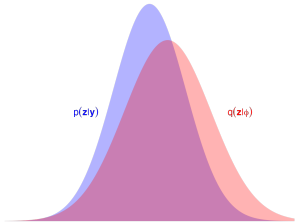
Stochastic Variational Inference with Liesel

Gianmarco Callegher



- Approximate complex probability distributions with simpler, tractable ones.
- Reformulates inference as an optimization problem.
- Preferred over MCMC for large datasets:
  - Scales efficiently to high dimensions.
  - Handles big data through stochastic optimization.

In variational inference, the posterior distribution $p(\boldsymbol{z}|\boldsymbol{y})$ is approximated by the so-called variational distribution $q(\boldsymbol{z}|\boldsymbol{\phi})$.



This can be done by solving the following optimization problem:

$$q^* = \arg\min_{\boldsymbol{\phi}} \mathcal{D}_{\mathsf{KL}}\left(p||q_{\boldsymbol{\phi}}\right).$$

Variational inference vs MCMC Sampling

Variational Inference

- global sampling scheme;
- fast mixing time;
- biased if $q(\boldsymbol{z}|\boldsymbol{\phi})$ does not approximate well $p(\boldsymbol{z}|\boldsymbol{y})$.

MCMC

- local sampling scheme;
- slow to explore the entire posterior density;
- guarantees to produce (asymptotically) exact samples from the posterior.

Usually, instead of minimizing the KL divergence, we optimize an alternative objective, called evidence lower bound (ELBO):

$$\begin{aligned}\mathrm{ELBO}(\boldsymbol{\phi}) &= \mathrm{E}_{q(\boldsymbol{z}|\boldsymbol{\phi})}\left[\ln p(\boldsymbol{z}, \boldsymbol{y})\right] - \mathrm{E}_{q(\boldsymbol{z}|\boldsymbol{\phi})}\left[\ln q(\boldsymbol{z} \mid \boldsymbol{\phi})\right] \\ &= \mathrm{E}_{q(\boldsymbol{z}|\boldsymbol{\phi})}\left[\ln p(\boldsymbol{y} \mid \boldsymbol{z})\right] - \mathcal{D}_{\mathsf{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{\phi}) \,\|\, p(\boldsymbol{z})\right).\end{aligned}$$

Maximizing the ELBO is equivalent to minimizing the KL divergence.

We estimate ϕ using stochastic optimization, which involves computing the following gradient:

$$\nabla_{\boldsymbol{\phi}} \, \mathrm{E}_{q(\boldsymbol{z}|\boldsymbol{\phi})} \left[\ln \frac{p(\boldsymbol{y}, \boldsymbol{z})}{q(\boldsymbol{z} \mid \boldsymbol{\phi})} \right] \approx \frac{1}{S} \sum_{s=1}^{S} \nabla_{\boldsymbol{\phi}} \left[\ln p(\boldsymbol{y}, \boldsymbol{z}^s) - \ln q(\boldsymbol{z}^s \mid \boldsymbol{\phi}) \right],$$

with $\boldsymbol{z}^s \sim q(\boldsymbol{z} \mid \boldsymbol{\phi})$.

Algorithm 1 Train Stochastic Variational Inference

Input: Observations $(y_1, \ldots, y_N)$ and step-size α

1: $t \leftarrow 0$
2: Initialize $\boldsymbol{\phi}$
3: **for** $t = 1, \ldots, T$ or until ELBO$(\boldsymbol{\phi})$ has not converged **do**
4: $\quad \boldsymbol{\phi}^{t+1} \leftarrow \boldsymbol{\phi}^t + \alpha \nabla_{\boldsymbol{\phi}} \text{ELBO}(\boldsymbol{\phi})$
5: $\quad t \leftarrow t + 1$
6: **end for**
7: **return** $\boldsymbol{\phi}$.

References I

Fahrmeir, L., Kneib, T., Lang, S., & Marx, B. (2021). *Regression: Models, methods and applications* (2nd ed.). Springer Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-63882-8

Ferrari, S. L. P., & Cribari-Neto, F. (2004).Beta regression for modelling rates and proportions. *Journal of Applied Statistics*, *31*(7), 799–815. https://doi.org/10.1080/0266476042000214501

Ospina, R., & Ferrari, S. L. P. (2010).Inflated beta distributions. *Statistical Papers*, *51*, 111–126. https://doi.org/10.1007/s00362-008-0125-4

Ospina, R., & Ferrari, S. L. P. (2012).A general class of zero-or-one inflated beta regression models. *Computational Statistics & Data Analysis*, *56*(6), 1609–1623. https://doi.org/10.1016/j.csda.2011.10.005