Solutions Sheet 03

On Google Colab, you can install the required libraries with the following commands:

```
!apt install libgraphviz-dev
!pip install pygraphviz
!pip install liesel
!pip install plotnine
```

Exercise 1: A Location-Scale Regression Model

We start by loading the data and importing the relevant libraries.

````
```{python}
import pandas as pd
import tensorflow_probability.substrates.jax.distributions as tfd
import tensorflow_probability.substrates.jax.bijectors as tfb
import jax.numpy as jnp
import liesel.model as lsl
import liesel.goose as gs

rent99 = pd.read_csv("https://s.gwdg.de/mzAkHV")

area = rent99.area.to_numpy("float32")
rent = rent99.rent.to_numpy("float32")
```
````

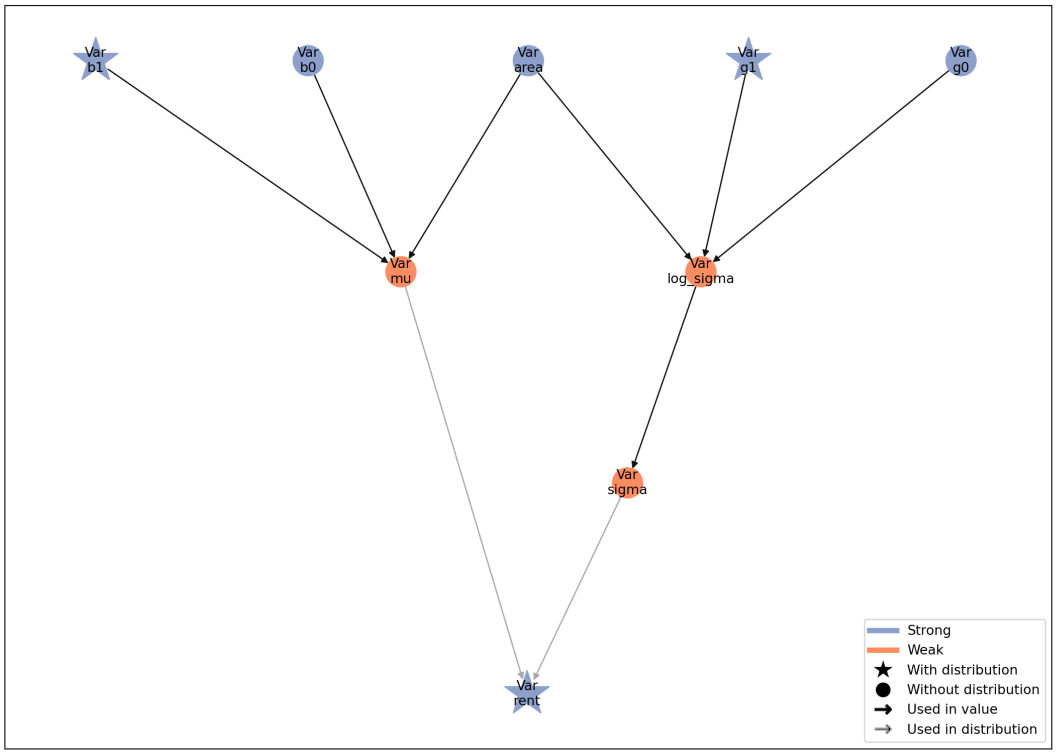
What is different about this model is the fact that we are defining a covariate model for the scale of the response.

````
```{python}
# Observed covariate values
x = lsl.obs(area, name="area")

g0 = lsl.param(0.0, name="g0")
g1 = lsl.param(0.0, lsl.Dist(tfd.Normal, loc=0., scale=10.), name="g1")

def linear_model(x, intercept, slope):
    return intercept + x*slope

log_sigma = lsl.Var(
  lsl.Calc(linear_model, x=x, intercept=g0, slope=g1),
  name="log_sigma"
)

sigma = lsl.Var(lsl.Calc(jnp.exp, log_sigma), name="sigma")
```
````

The rest works the same as before.

```{python}
# Location Model
b0 = lsl.param(0.0, name="b0")
b1 = lsl.param(0.0, lsl.Dist(tfd.Normal, loc=0., scale=10.), name="b1")

mu = lsl.Var(lsl.Calc(linear_model, x=x, intercept=b0, slope=b1), name="mu")

# Observation Model
y_dist = lsl.Dist(tfd.Normal, loc=mu, scale=sigma)
y = lsl.obs(rent, y_dist, name="rent")

# Build Graph & Plot
gb = lsl.GraphBuilder().add(y)
gb.plot_vars()
```

```
GraphBuilder(0 nodes, 1 vars)
```



Sampling from the posterior using Goose also works the same as before. The only difference here is that we have to think of sampling the regression coefficients of the scale model instead of sampling the log variance directly.

```{python}
model = gb.build_model()
interface = gs.LieselInterface(model)

eb = gs.EngineBuilder(seed=1, num_chains=4)
```

```
eb.add_kernel(gs.NUTSKernel(["b0", "b1"]))
eb.add_kernel(gs.IWLSKernel(["g0", "g1"]))

eb.set_duration(warmup_duration=1000, posterior_duration=1000)
eb.set_model(interface)
eb.set_initial_values(model.state)
eb.set_engine_seed(seed=2)

engine = eb.build()
```

```
liesel.goose.builder - WARNING - No jitter functions provided. The initial values
won't be jittered
liesel.goose.engine - INFO - Initializing kernels...
liesel.goose.engine - INFO - Done
```

After building the engine, it's time to sample and then summarize.

```{python}
#| cache: true
engine.sample_all_epochs()
results = engine.get_results()
summary = gs.Summary(results)
```

```
liesel.goose.engine - INFO - Starting epoch: FAST_ADAPTATION, 75 transitions, 25
jitted together
liesel.goose.engine - WARNING - Errors per chain for kernel_00: 2, 2, 2, 4 / 75
transitions
liesel.goose.engine - INFO - Finished epoch
liesel.goose.engine - INFO - Starting epoch: SLOW_ADAPTATION, 25 transitions, 25
jitted together
liesel.goose.engine - WARNING - Errors per chain for kernel_00: 2, 1, 1, 2 / 25
transitions
liesel.goose.engine - INFO - Finished epoch
liesel.goose.engine - INFO - Starting epoch: SLOW_ADAPTATION, 50 transitions, 25
jitted together
liesel.goose.engine - WARNING - Errors per chain for kernel_00: 1, 1, 1, 1 / 50
transitions
liesel.goose.engine - INFO - Finished epoch
liesel.goose.engine - INFO - Starting epoch: SLOW_ADAPTATION, 100 transitions, 25
jitted together
liesel.goose.engine - WARNING - Errors per chain for kernel_00: 1, 2, 2, 3 / 100
transitions
liesel.goose.engine - INFO - Finished epoch
liesel.goose.engine - INFO - Starting epoch: SLOW_ADAPTATION, 200 transitions, 25
jitted together
liesel.goose.engine - WARNING - Errors per chain for kernel_00: 3, 1, 2, 2 / 200
transitions
liesel.goose.engine - INFO - Finished epoch
liesel.goose.engine - INFO - Starting epoch: SLOW_ADAPTATION, 500 transitions, 25
jitted together
liesel.goose.engine - WARNING - Errors per chain for kernel_00: 2, 2, 1, 1 / 500
transitions
```

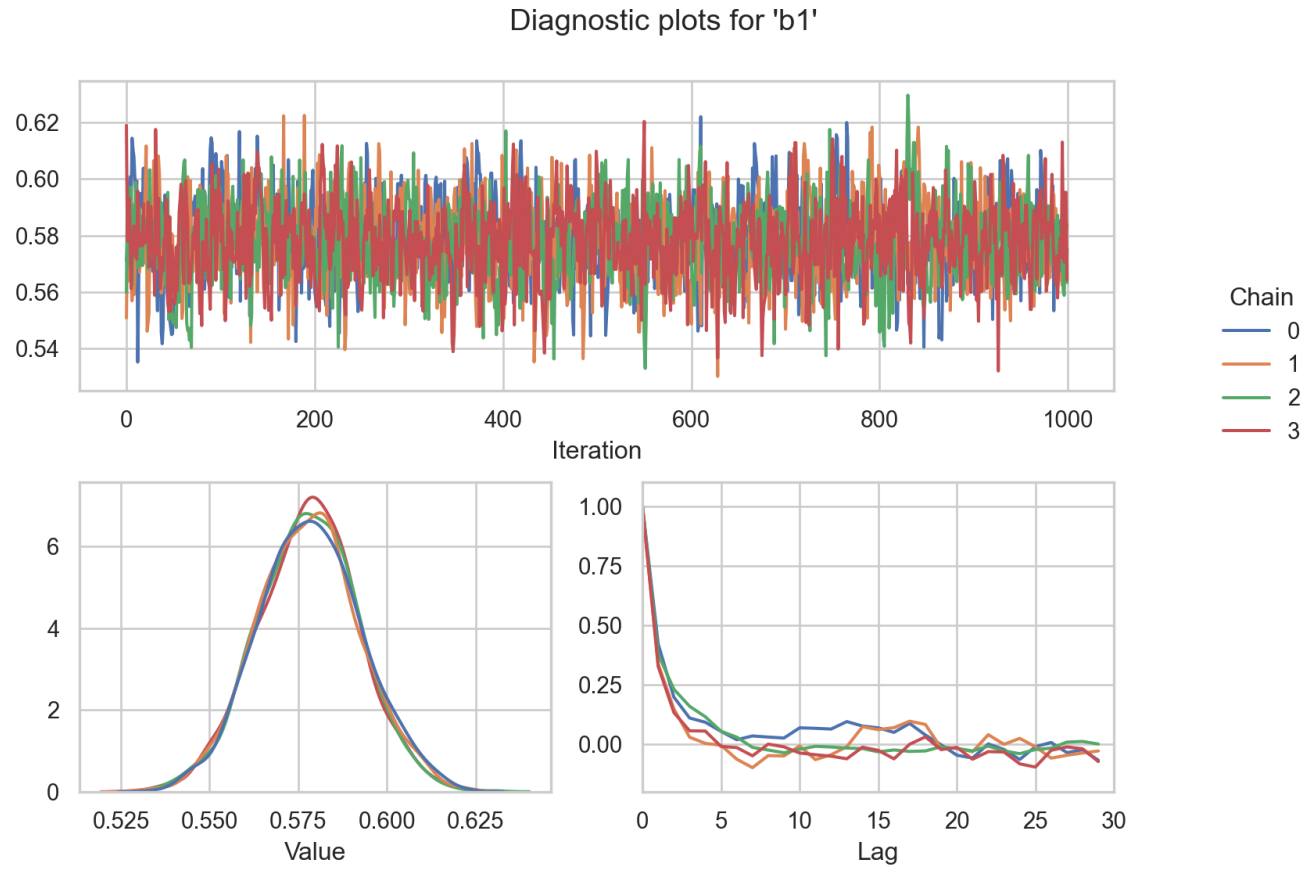
```
liesel.goose.engine - INFO - Finished epoch
liesel.goose.engine - INFO - Starting epoch: FAST_ADAPTATION, 50 transitions, 25
jitted together
liesel.goose.engine - WARNING - Errors per chain for kernel_00: 2, 2, 3, 2 / 50
transitions
liesel.goose.engine - INFO - Finished epoch
liesel.goose.engine - INFO - Finished warmup
liesel.goose.engine - INFO - Starting epoch: POSTERIOR, 1000 transitions, 25 jitted
together
liesel.goose.engine - INFO - Finished epoch
```

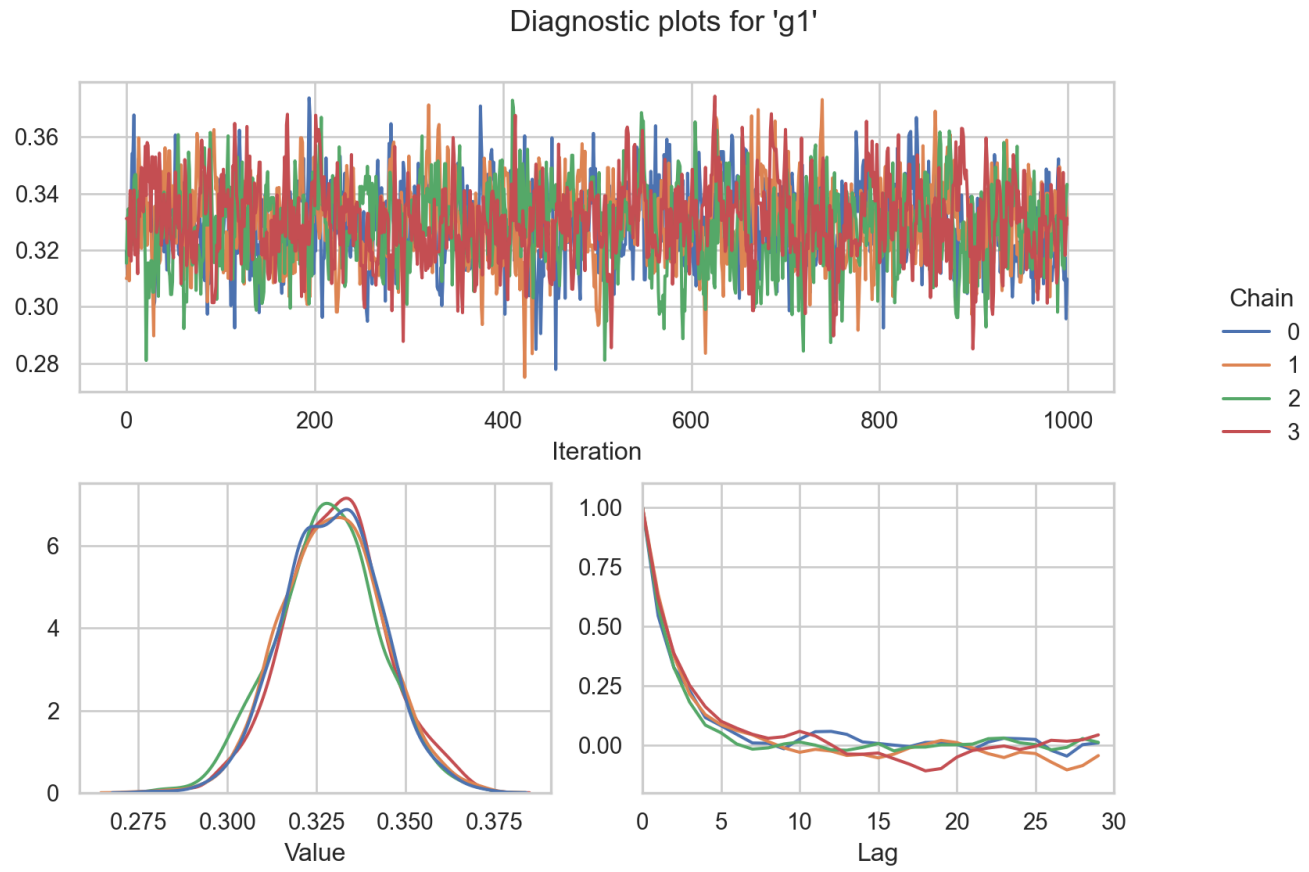
```{python}
print(summary.to_dataframe())
print(summary.error_df())
```

```
         var_fqn     kernel var_index  ...    q_0.95   hdi_low  hdi_high
variable                               ...
b0            b0  kernel_00        ()  ...  0.021832 -0.024977  0.021339
b1            b1  kernel_00        ()  ...  0.602020  0.555807  0.602602
g0            g0  kernel_01        ()  ... -0.299735 -0.341028 -0.299529
g1            g1  kernel_01        ()  ...  0.352576  0.305969  0.352205

[4 rows x 17 columns]
                                                    count relative
kernel    error_code error_msg            phase
kernel_00 1          divergent transition warmup       51  0.01275
                                          posterior     0      0.0

/Users/johannesbrachem/Documents/git/cmstats-tutorial/env/lib/python3.10/site-
packages/liesel/goose/summary_m.py:362: FutureWarning: The behavior of array
concatenation with empty entries is deprecated. In a future version, this will no
longer exclude empty items when determining the result dtype. To retain the old
behavior, exclude the empty entries before the concat operation.
  df = pd.concat({
```

Let's have a quick look at the trace plots for β_1 and γ_1.

```{python}
gs.plot_param(results, "b1")
gs.plot_param(results, "g1")
```





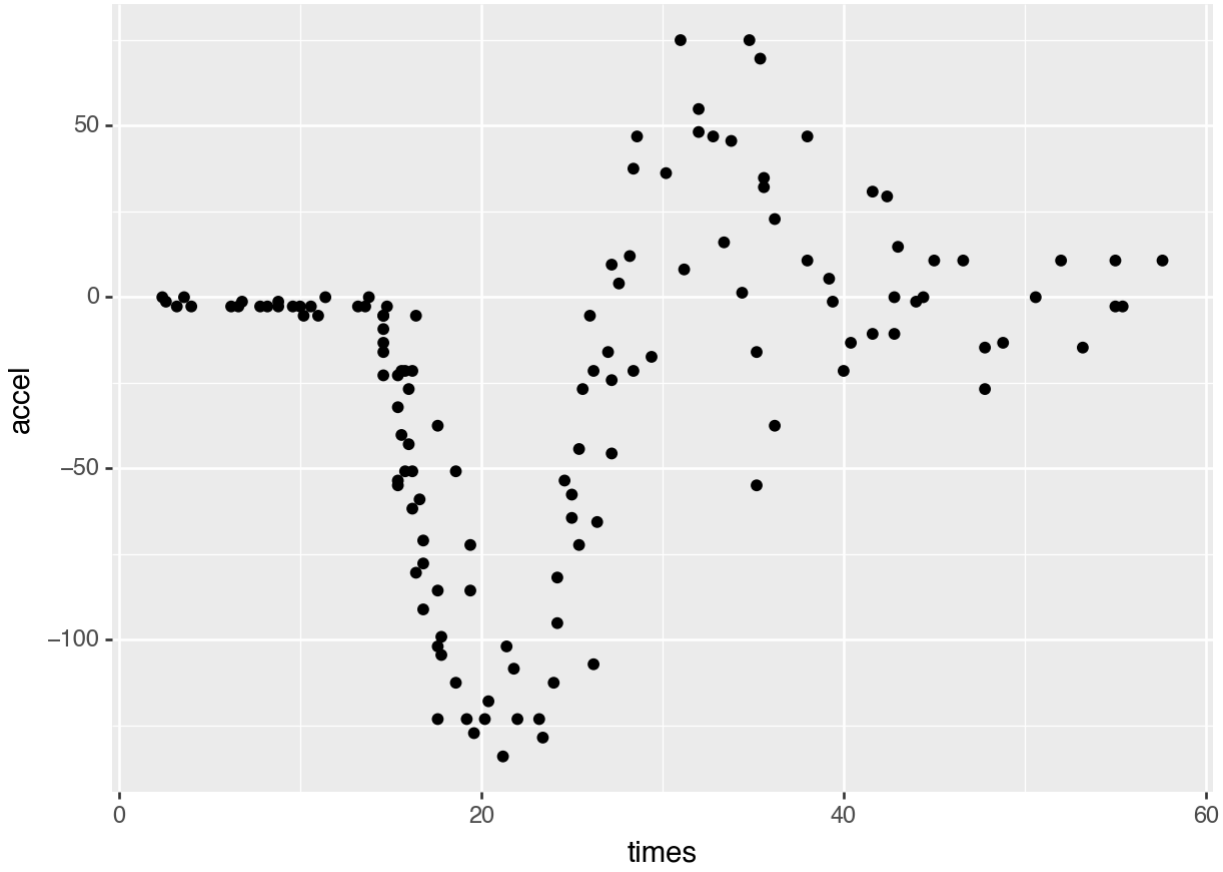
Exercise 2: A semiparametric model with `{rliesel}`

Subtask a): Plotting the dataset

We import the data and, like before, use `plotnine` for plotting.

```{python}
from plotnine import ggplot, aes, geom_line, geom_point, geom_ribbon, labs

mcycle = pd.read_csv("https://s.gwdg.de/50F2v6")

(
  ggplot(mcycle)
  + aes("times", "accel")
  + geom_point()
)
```

```
<Figure Size: (1280 x 960)>
```



Subtask b): Set up an `rliesel` model

- Rliesel offers a syntax interface that is very similar to the common modeling interfaces in R.
- Notably, we can use `mgcv` functionality to define smooth functions with the function `s()`. See `?s` for help on this function.
- Under the hood, Liesel will set up a default distributional regression configuration including priors for us.

The R code is:

```{r}
library(rliesel)
```

```
Please make sure you are using a virtual or conda environment with Liesel installed, e.g. using `reticulate::use_virtualenv()` or `reticulate::use_condaenv()`. See `vignette("versions", "reticulate")`.

After setting the environment, check if the installed versions of RLiesel and Liesel are compatible with `check_liesel_version()`.
```

```{r}
library(reticulate)
```

```{r}
mcycle <- py$mcycle
```

```{r}
model <- liesel(

  response = mcycle$accel,
  distribution = "Normal",

  predictors = list(
    loc = predictor(~s(times, bs = "ps", k=20), inverse_link = "Identity"),
    scale = predictor(~1, inverse_link = "Exp")
  ),

  data = mcycle
)
```

```
Installed Liesel version 0.2.8 is compatible, continuing to set up model
```

```{python}
#| include: false
#| echo: false
model = r.model
lsl.save_model(model, "ex2-model.pickle")
```

In pure Python, we import a prepared model object from the public tutorial repoository.

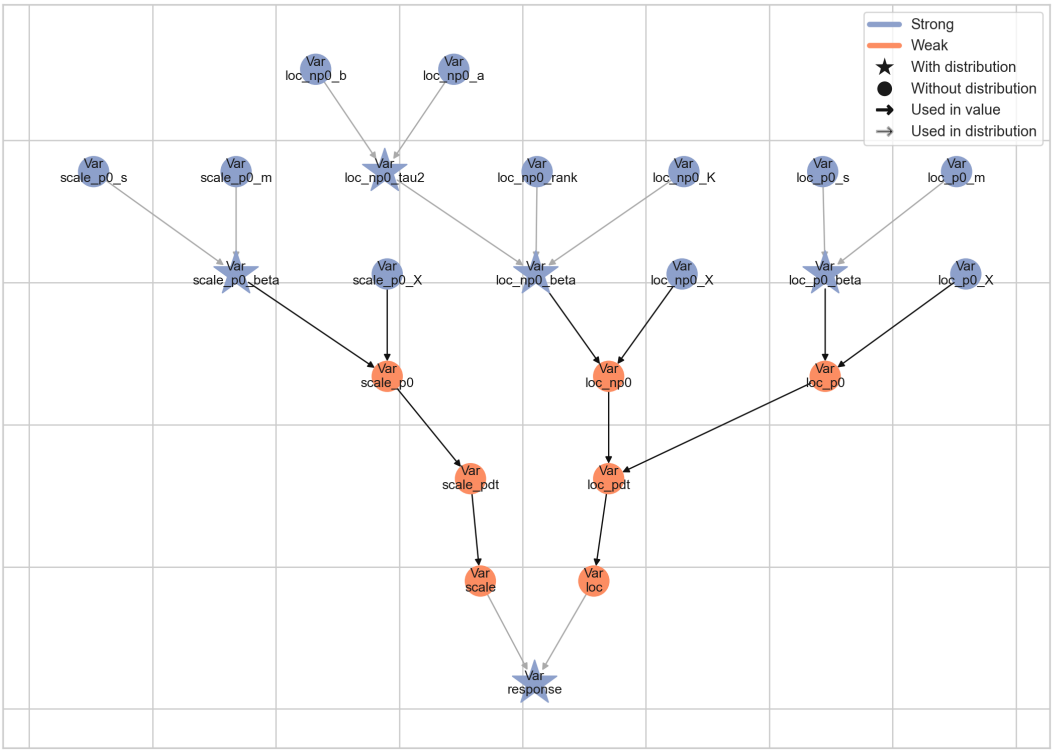
```{python}
from urllib.request import urlopen
import dill

model = dill.load(urlopen("https://s.gwdg.de/un4W29"))
```

```
# model = lsl.load_model("path/to/model.pickle")
```

Subtask c): Plot the model

Let's plot the model.

```{python}
lsl.plot_vars(model, save_path="img/loc.png")
```



Subtask d): Default sampling scheme

Describe the default sampling scheme for a semi-parametric distributional regression model, i.e. the different kernels for the different parameters.

	param	kernel_idx	kernel_cls
0	scale_p0_beta	0	IWLSKernel
1	loc_np0_tau2	1	GibbsKernel
2	loc_np0_beta	2	IWLSKernel
3	loc_p0_beta	3	IWLSKernel

Subtask e): Draw posterior samples

For a default distributional regression model, Liesel ships the convenience function `lsl.dist_reg_mcmc`, which sets up a fully prepared `gs.EngineBuilder` for us to use.

```{python}
engine_builder = lsl.dist_reg_mcmc(model, seed=1337, num_chains=4)
engine_builder.set_duration(warmup_duration=1000, posterior_duration=1000)
```

For easier plotting later on, we tell Goose to include the values of the `"loc"` node in the posterior samples.

```{python}
engine_builder.positions_included.append("loc")
```

Now we run the sampling scheme.

```{python}
#| cache: true
engine = engine_builder.build()
engine.sample_all_epochs()
```

```
liesel.goose.engine - INFO - Initializing kernels...
liesel.goose.engine - INFO - Done
liesel.goose.engine - INFO - Starting epoch: FAST_ADAPTATION, 75 transitions, 25
jitted together
liesel.goose.engine - INFO - Finished epoch
liesel.goose.engine - INFO - Starting epoch: SLOW_ADAPTATION, 25 transitions, 25
jitted together
liesel.goose.engine - INFO - Finished epoch
liesel.goose.engine - INFO - Starting epoch: SLOW_ADAPTATION, 50 transitions, 25
jitted together
liesel.goose.engine - INFO - Finished epoch
liesel.goose.engine - INFO - Starting epoch: SLOW_ADAPTATION, 100 transitions, 25
jitted together
liesel.goose.engine - INFO - Finished epoch
liesel.goose.engine - INFO - Starting epoch: SLOW_ADAPTATION, 200 transitions, 25
jitted together
liesel.goose.engine - INFO - Finished epoch
liesel.goose.engine - INFO - Starting epoch: SLOW_ADAPTATION, 500 transitions, 25
jitted together
liesel.goose.engine - INFO - Finished epoch
liesel.goose.engine - INFO - Starting epoch: FAST_ADAPTATION, 50 transitions, 25
jitted together
liesel.goose.engine - INFO - Finished epoch
liesel.goose.engine - INFO - Finished warmup
liesel.goose.engine - INFO - Starting epoch: POSTERIOR, 1000 transitions, 25 jitted
together
liesel.goose.engine - INFO - Finished epoch
```

Subtask f): Inspect results

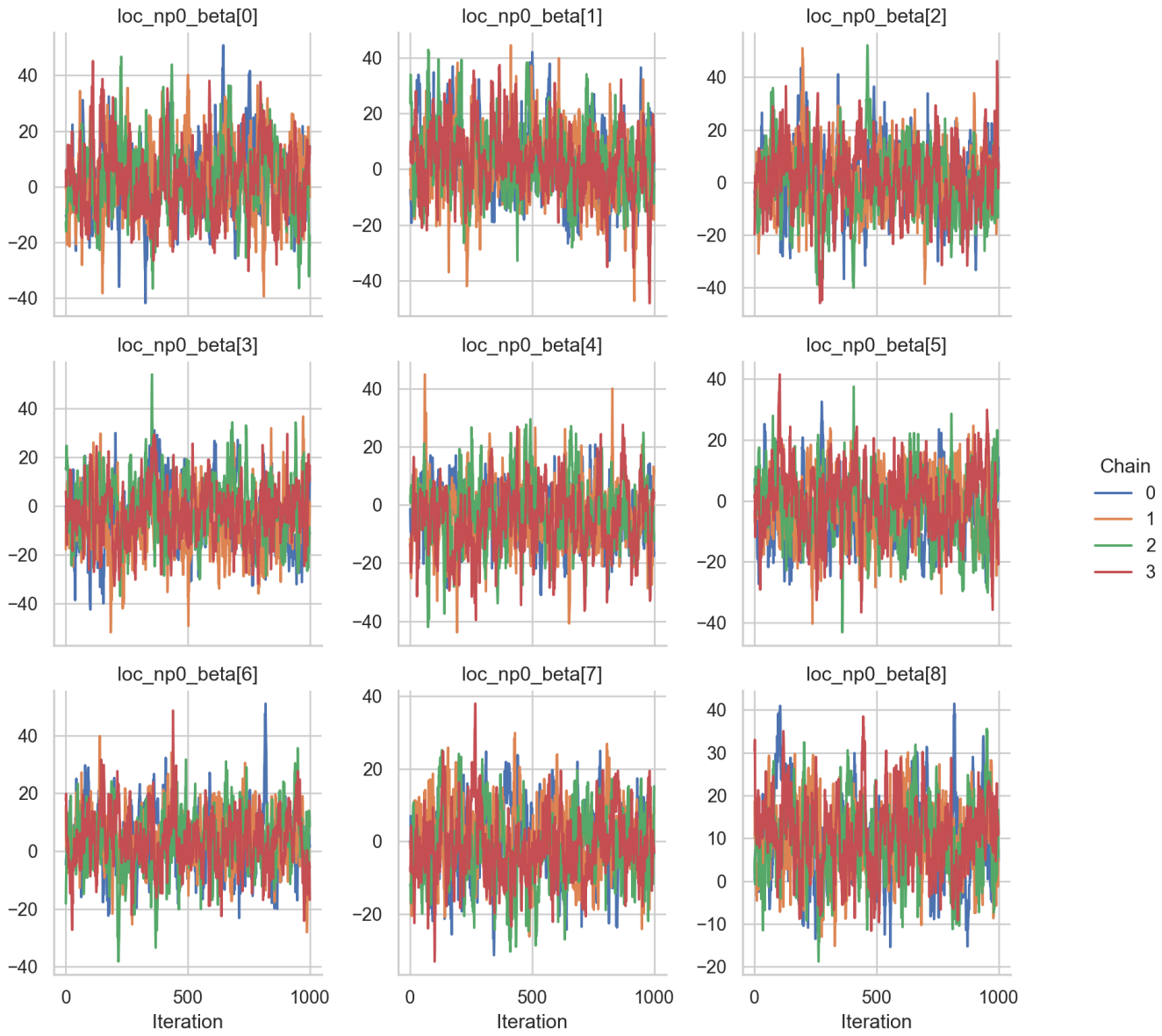
```{python}
results = engine.get_results()
summary = gs.Summary(results)
```

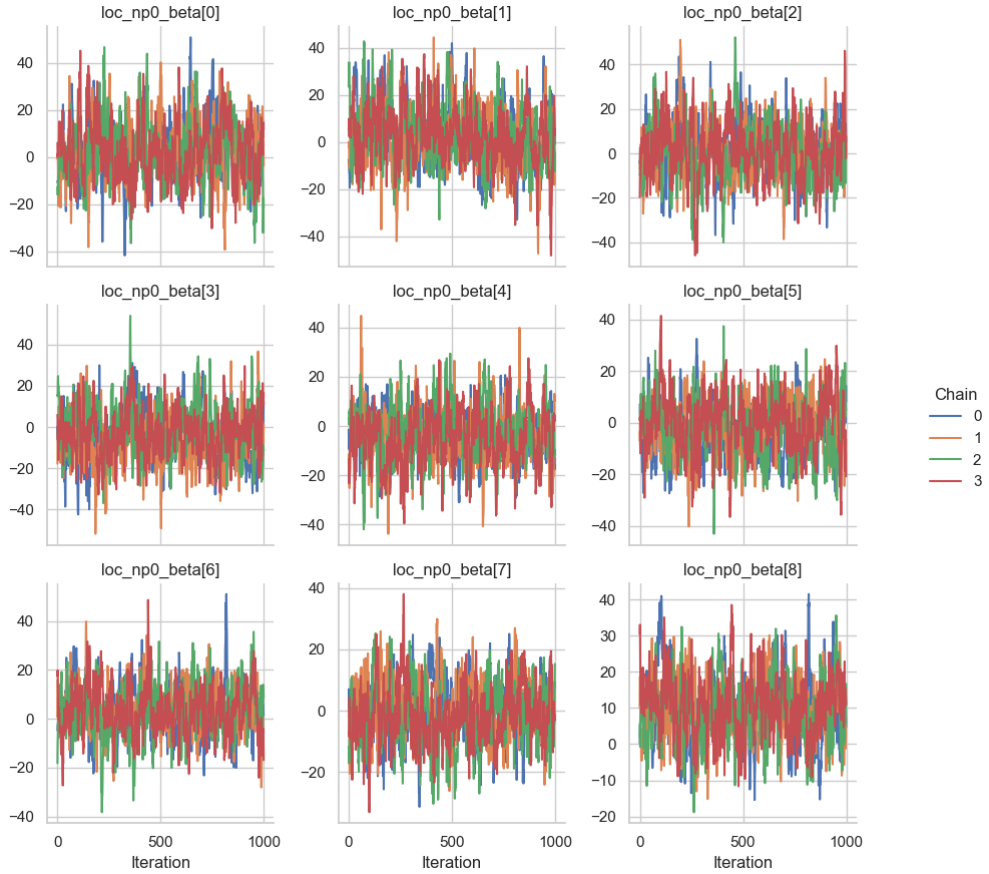
```{python}
print(summary.to_dataframe())
print(summary.error_df())
```

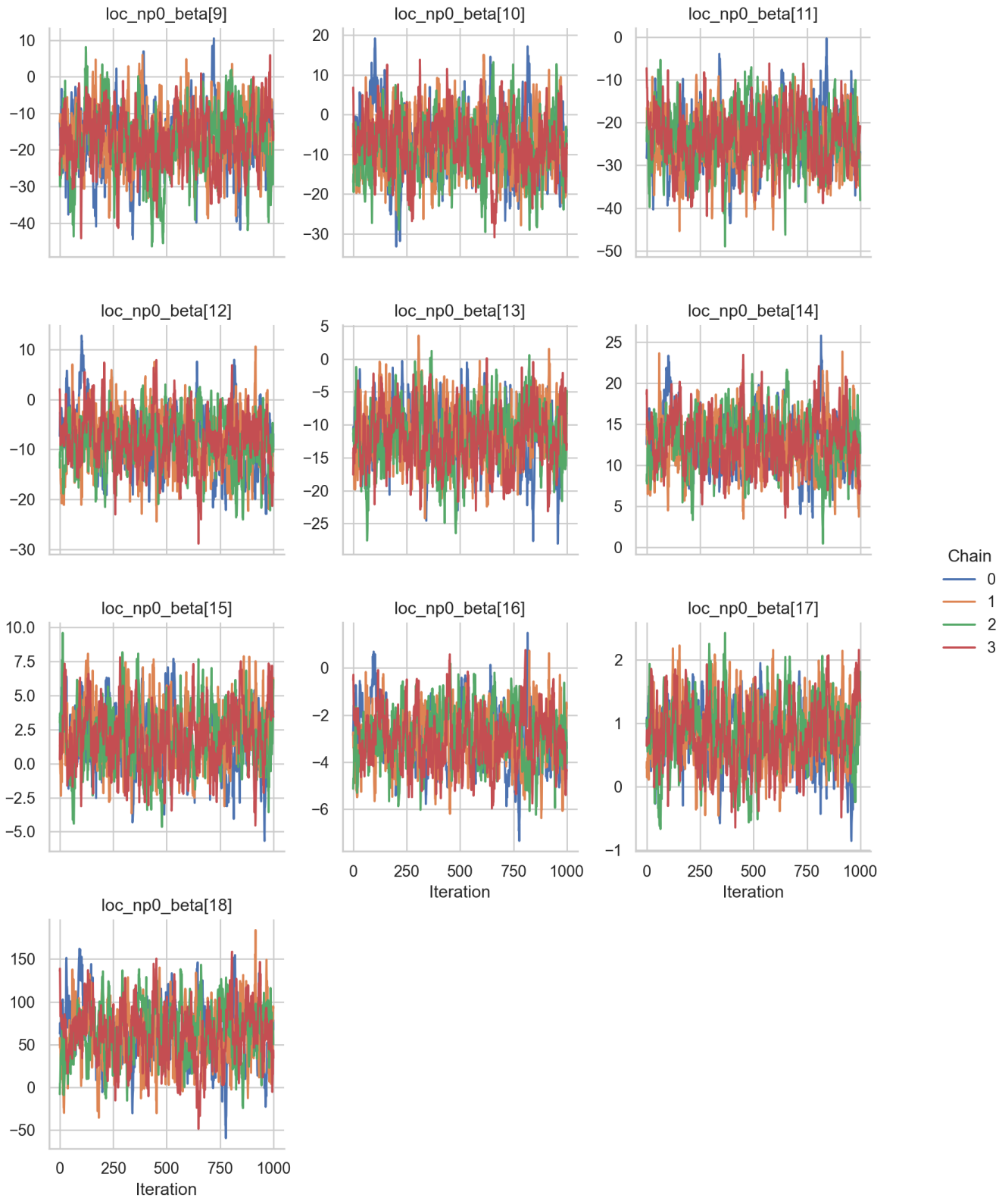
```
                       var_fqn     kernel  ...    hdi_low    hdi_high
variable                                   ...
loc                   loc[0]          -  ... -24.138468   19.828516
loc                   loc[1]          -  ... -22.207094   18.477547
loc                   loc[2]          -  ... -20.289829   13.014118
loc                   loc[3]          -  ... -18.528652   11.961233
loc                   loc[4]          -  ... -17.429401   12.263695
...                      ...        ...  ...        ...         ...
loc_np0_beta   loc_np0_beta[17]  kernel_02  ...   0.033942    1.600277
loc_np0_beta   loc_np0_beta[18]  kernel_02  ...   9.603549  111.952362
loc_np0_tau2       loc_np0_tau2  kernel_01  ...  57.376225  291.856689
loc_p0_beta      loc_p0_beta[0]  kernel_03  ... -28.737087  -22.121880
scale_p0_beta  scale_p0_beta[0]  kernel_00  ...   3.015892    3.228584

[155 rows x 17 columns]
Empty DataFrame
Columns: []
Index: []
```

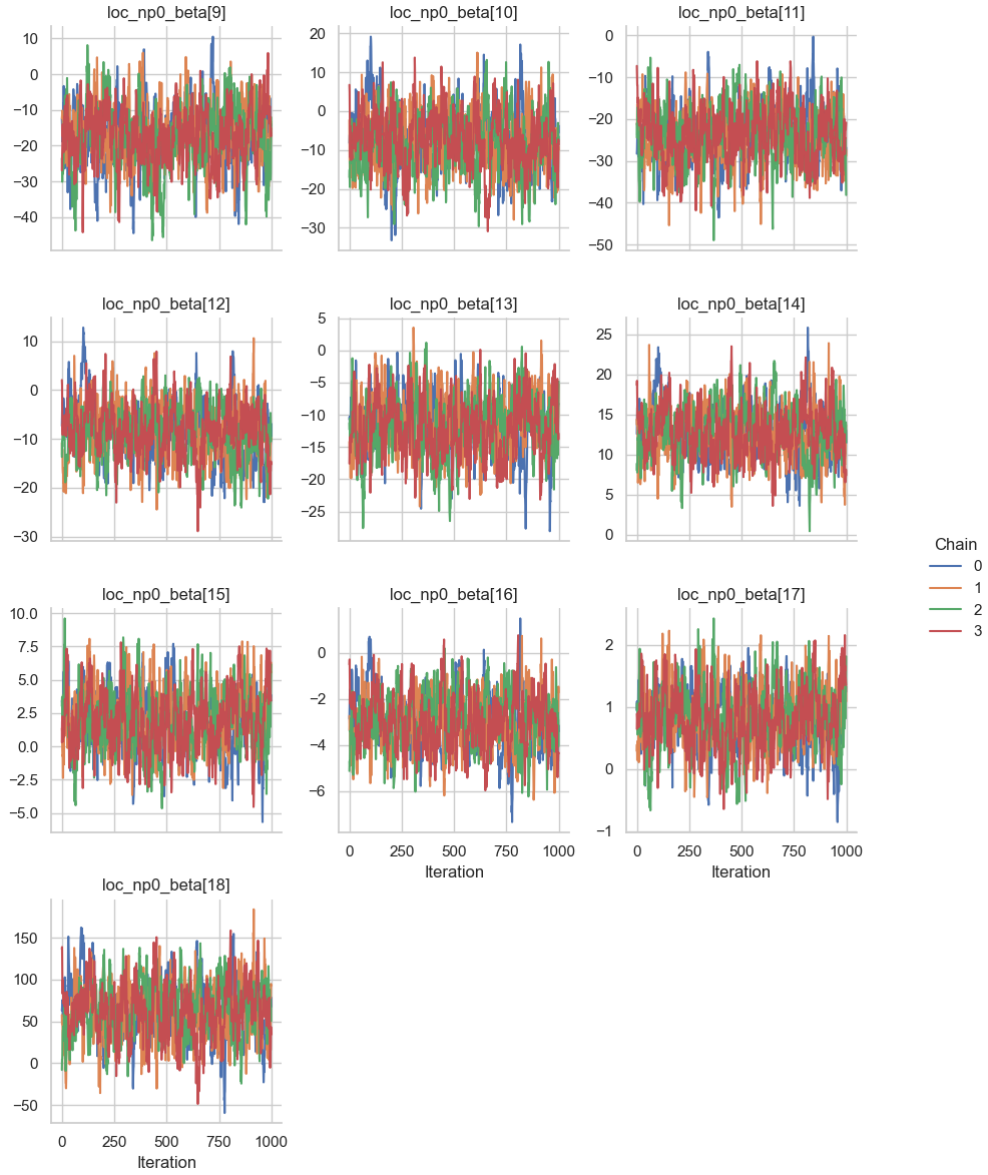
Some trace plots:

```{python}
gs.plot_trace(results, "loc_np0_beta", range(0, 9))
gs.plot_trace(results, "loc_np0_beta", range(9, 19))
```


loc_np0_beta[0]
loc_np0_beta[1]
loc_np0_beta[2]
loc_np0_beta[3]
loc_np0_beta[4]
loc_np0_beta[5]
loc_np0_beta[6]
loc_np0_beta[7]
loc_np0_beta[8]
Iteration
Iteration
Iteration
Chain
0
1
2
3


loc_np0_beta[0]
loc_np0_beta[1]
loc_np0_beta[2]
loc_np0_beta[3]
loc_np0_beta[4]
loc_np0_beta[5]
loc_np0_beta[6]
loc_np0_beta[7]
loc_np0_beta[8]
Iteration
Iteration
Iteration
Chain
0
1
2
3


loc_np0_beta[9]
loc_np0_beta[10]
loc_np0_beta[11]
loc_np0_beta[12]
loc_np0_beta[13]
loc_np0_beta[14]
loc_np0_beta[15]
loc_np0_beta[16]
loc_np0_beta[17]
loc_np0_beta[18]
Iteration
Iteration
Chain
0
1
2
3



Subtask g): Visualize estimated P-Spline

Now we make use of the fact that we tracked the value of the location. We can easily access summary statistics from the summary object.

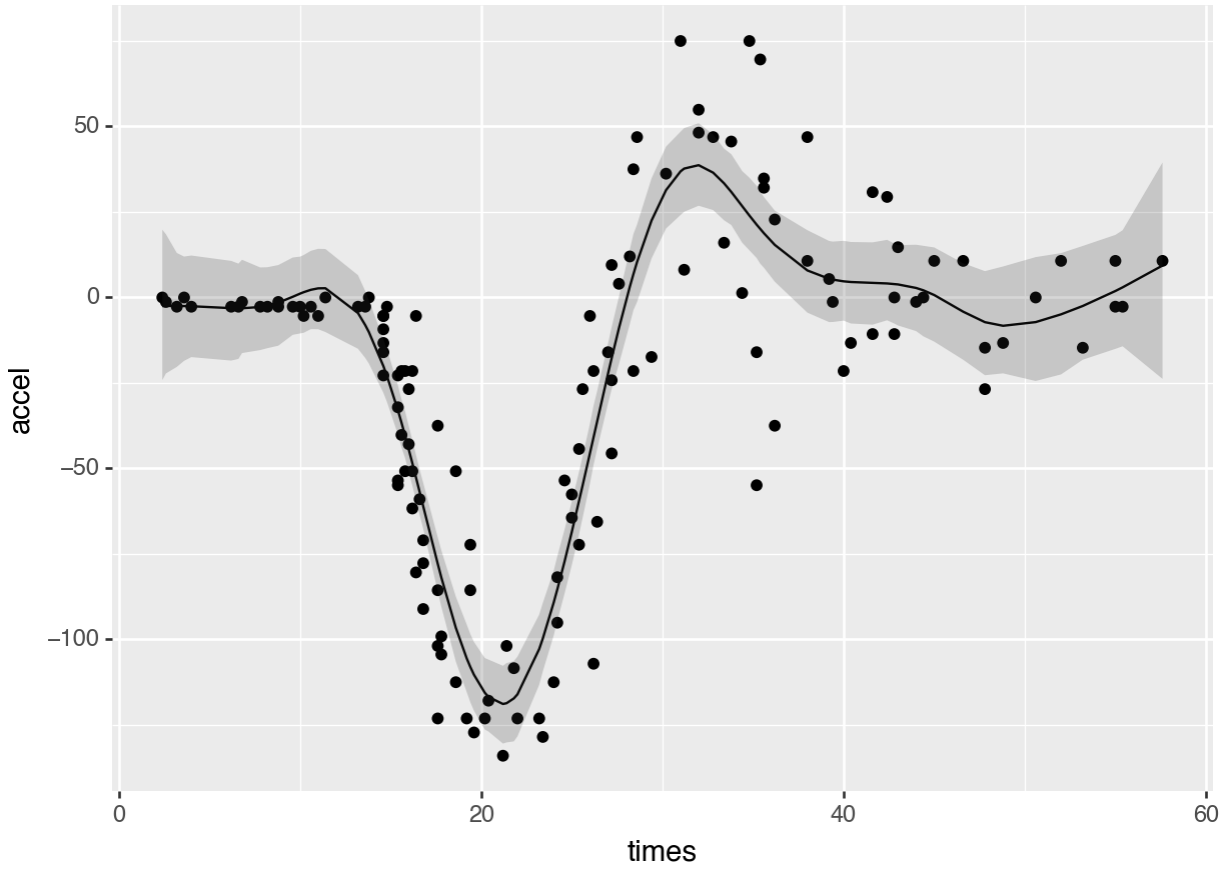
```{python}
loc = summary.quantities["mean"]["loc"]
loc_hdi = summary.quantities["hdi"]["loc"]

loc_hdi_lo = loc_hdi[0,:]
loc_hdi_hi = loc_hdi[1,:]
```

For plotting, we again use `plotnine`.

```{python}
(
  ggplot()
  + aes(x = mcycle.times)
  + geom_point(aes(y = mcycle.accel))
  + geom_line(aes(y = loc))
```

```
  + geom_ribbon(aes(ymin = loc_hdi_lo, ymax = loc_hdi_hi), alpha = 0.2)
)
```

```
<Figure Size: (1280 x 960)>
```



Exercise 3: Distributional regression with `{rliesel}`

Subtask a): Model setup

Model setup works similar - we just have to additionally define a covariate model for the scale now.

```{r}
model <- liesel(
  response = mcycle$accel,
  distribution = "Normal",
  predictors = list(
    loc = predictor(~s(times, bs = "ps", k = 20), inverse_link = "Identity"),
    scale = predictor(~s(times, bs = "ps", k = 20), inverse_link = "Exp")
  ),
  data = mcycle
)
```

```{python}
#| include: false
```

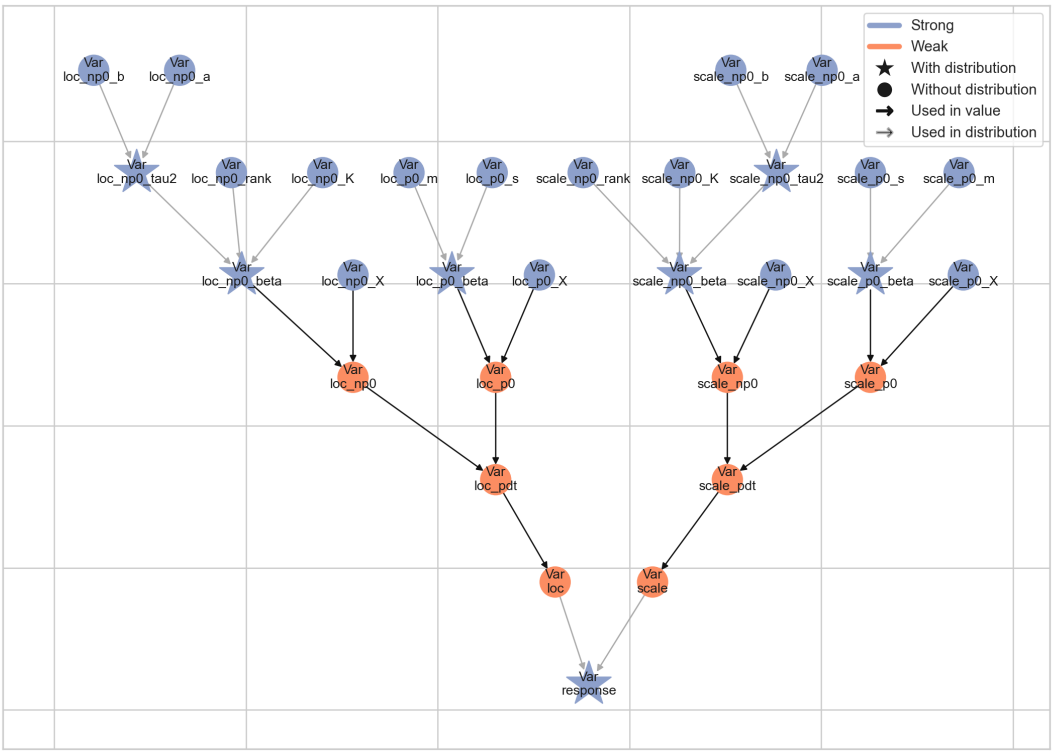
```
#| echo: false
model = r.model
lsl.save_model(model, "ex3-model.pickle")
```

For Python-only participants, we load the model from the public repository.

```{python}
model = dill.load(urlopen("https://s.gwdg.de/exn3LQ"))
```

Subtask b): Plot your model

```{python}
lsl.plot_vars(model, save_path="img/locscale.png")
```



Subtasks c), d): Set up engine builder, sample and inspect

Like before, we can quickly set up the sampling scheme with our little helper `lsl.dist_reg_mcmc`:

```{python}
engine_builder = lsl.dist_reg_mcmc(model, seed=11, num_chains=4)

engine_builder.set_duration(warmup_duration=1000, posterior_duration=1000)
```

```
engine_builder.positions_included += ["loc", "scale"]
```

There's one tricky bit here: the default jittering, which adds uniformly distributed random noise $u \sim \mathcal{U}(-2, 2)$ to the starting values, is too aggressive for the for scale regression coefficients. So we have to override the a manual jittering with a more subtle version.

```{python}
import jax

def jitter_scale_coefs(key, coef):
   jittering = jax.random.uniform(
     key, coef.shape, minval=-0.1, maxval=0.1
   )
   return coef + jittering

engine_builder.set_jitter_fns({"scale_np0_beta": jitter_scale_coefs})
```

Now we can sample successfully:

```{python}
#| cache: true
engine = engine_builder.build()
engine.sample_all_epochs()
results = engine.get_results()
summary = gs.Summary(results)
```

```
liesel.goose.builder - WARNING - No jitter functions provided for position keys
'scale_np0_tau2', 'scale_p0_beta', 'loc_np0_tau2', 'loc_np0_beta', 'loc_p0_beta'. The
initial values for these keys won't be jittered
liesel.goose.engine - INFO - Initializing kernels...
liesel.goose.engine - INFO - Done
liesel.goose.engine - INFO - Starting epoch: FAST_ADAPTATION, 75 transitions, 25
jitted together
liesel.goose.engine - WARNING - Errors per chain for kernel_01: 0, 1, 1, 0 / 75
transitions
liesel.goose.engine - WARNING - Errors per chain for kernel_02: 0, 0, 0, 1 / 75
transitions
liesel.goose.engine - INFO - Finished epoch
liesel.goose.engine - INFO - Starting epoch: SLOW_ADAPTATION, 25 transitions, 25
jitted together
liesel.goose.engine - WARNING - Errors per chain for kernel_01: 1, 0, 0, 0 / 25
transitions
liesel.goose.engine - INFO - Finished epoch
liesel.goose.engine - INFO - Starting epoch: SLOW_ADAPTATION, 50 transitions, 25
jitted together
liesel.goose.engine - WARNING - Errors per chain for kernel_01: 0, 0, 0, 1 / 50
transitions
liesel.goose.engine - INFO - Finished epoch
liesel.goose.engine - INFO - Starting epoch: SLOW_ADAPTATION, 100 transitions, 25
jitted together
```

```
liesel.goose.engine - INFO - Finished epoch
liesel.goose.engine - INFO - Starting epoch: SLOW_ADAPTATION, 200 transitions, 25
jitted together
liesel.goose.engine - WARNING - Errors per chain for kernel_01: 1, 0, 0, 0 / 200
transitions
liesel.goose.engine - INFO - Finished epoch
liesel.goose.engine - INFO - Starting epoch: SLOW_ADAPTATION, 500 transitions, 25
jitted together
liesel.goose.engine - INFO - Finished epoch
liesel.goose.engine - INFO - Starting epoch: FAST_ADAPTATION, 50 transitions, 25
jitted together
liesel.goose.engine - WARNING - Errors per chain for kernel_01: 0, 0, 1, 0 / 50
transitions
liesel.goose.engine - INFO - Finished epoch
liesel.goose.engine - INFO - Finished warmup
liesel.goose.engine - INFO - Starting epoch: POSTERIOR, 1000 transitions, 25 jitted
together
liesel.goose.engine - INFO - Finished epoch
```

```{python}
print(summary.to_dataframe())
print(summary.error_df())
```

```
                          var_fqn     kernel  ...   hdi_low  hdi_high
variable                                      ...
loc                        loc[0]          -  ... -2.844286  1.440754
loc                        loc[1]          -  ... -2.639233  0.783369
loc                        loc[2]          -  ... -2.784601 -0.230288
loc                        loc[3]          -  ... -3.095898 -0.480556
loc                        loc[4]          -  ... -3.315994 -0.641888
...                           ...        ...  ...       ...       ...
scale_np0_beta  scale_np0_beta[16]  kernel_01  ... -0.041654  0.086936
scale_np0_beta  scale_np0_beta[17]  kernel_01  ... -0.080774 -0.022098
scale_np0_beta  scale_np0_beta[18]  kernel_01  ...  0.367395  4.182557
scale_np0_tau2      scale_np0_tau2  kernel_00  ...  0.008699  0.142353
scale_p0_beta     scale_p0_beta[0]  kernel_02  ...  2.581066  2.804657

[308 rows x 17 columns]
                                                    count relative
kernel    error_code error_msg           phase
kernel_01 90         nan acceptance prob warmup         6   0.0015
                                          posterior      0      0.0
kernel_02 90         nan acceptance prob warmup         1  0.00025
                                          posterior      0      0.0

/Users/johannesbrachem/Documents/git/cmstats-tutorial/env/lib/python3.10/site-
packages/liesel/goose/summary_m.py:362: FutureWarning: The behavior of array
concatenation with empty entries is deprecated. In a future version, this will no
longer exclude empty items when determining the result dtype. To retain the old
behavior, exclude the empty entries before the concat operation.
```

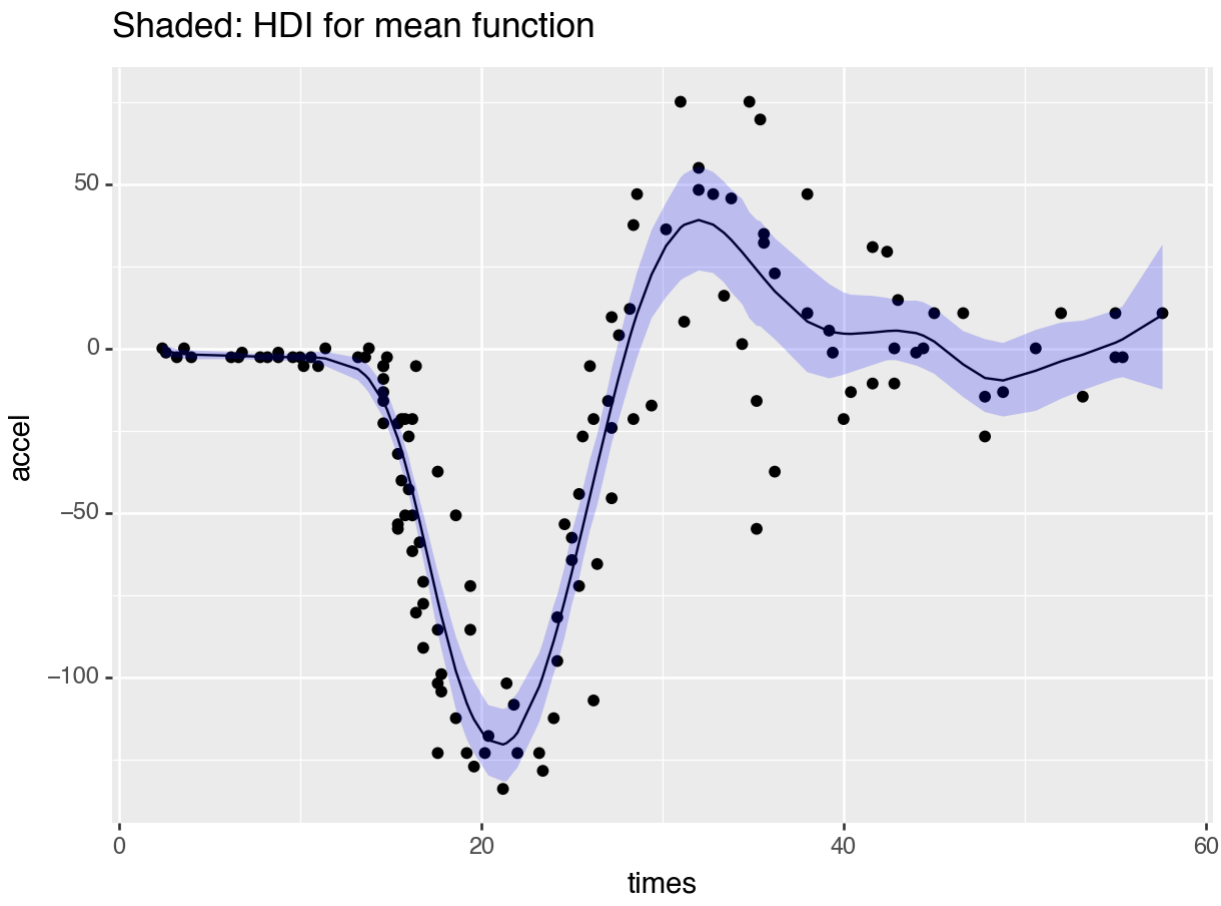
Subtask e): Plot results for mean function

And go on to plot our results:

```{python}
loc = summary.quantities["mean"]["loc"]
loc_hdi = summary.quantities["hdi"]["loc"]

loc_hdi_lo = loc_hdi[0,:]
loc_hdi_hi = loc_hdi[1,:]
```

```{python}
(
  ggplot()
  + aes(x = mcycle.times)
  + geom_point(aes(y = mcycle.accel))
  + geom_line(aes(y = loc))
  + geom_ribbon(aes(ymin = loc_hdi_lo, ymax = loc_hdi_hi), alpha = 0.2, fill = "blue")
  + labs(title = "Shaded: HDI for mean function")
)
```

```
<Figure Size: (1280 x 960)>
```



Subtask f): Plot results for scale function

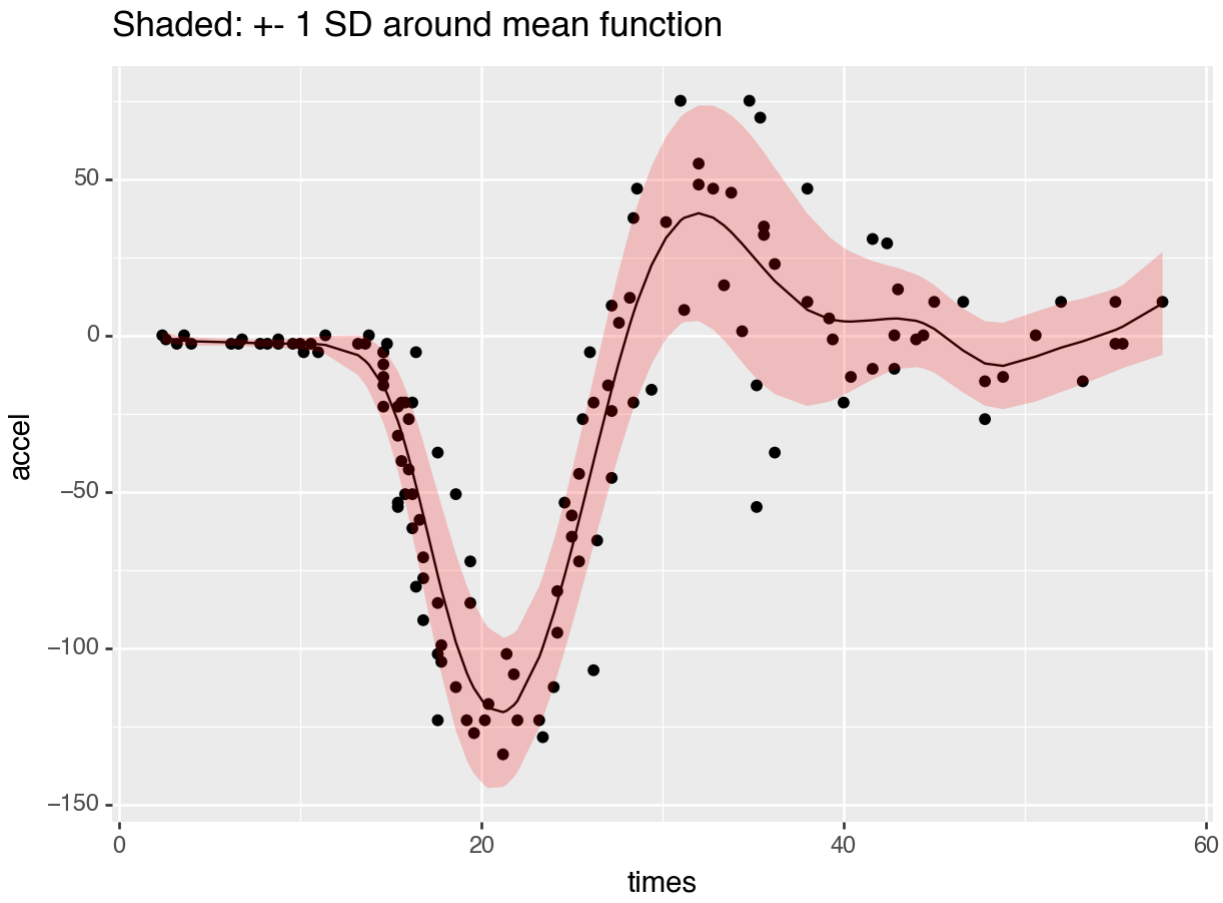
```{python}
scale = summary.quantities["mean"]["scale"]
```

```
scale_hdi = summary.quantities["hdi"]["scale"]

scale_hdi_lo = scale_hdi[0,:]
scale_hdi_hi = scale_hdi[1,:]
```

We can use the scale estimate to display one standard deviation around the mean estimate:

```{python}
(
  ggplot()
  + aes(x = mcycle.times)
  + geom_point(aes(y = mcycle.accel))
  + geom_line(aes(y = loc))
  + geom_ribbon(aes(ymin = loc - scale, ymax = loc + scale), alpha = 0.2, fill = "red"
  + labs(title = "Shaded: +- 1 SD around mean function")
)
```

```
<Figure Size: (1280 x 960)>
```

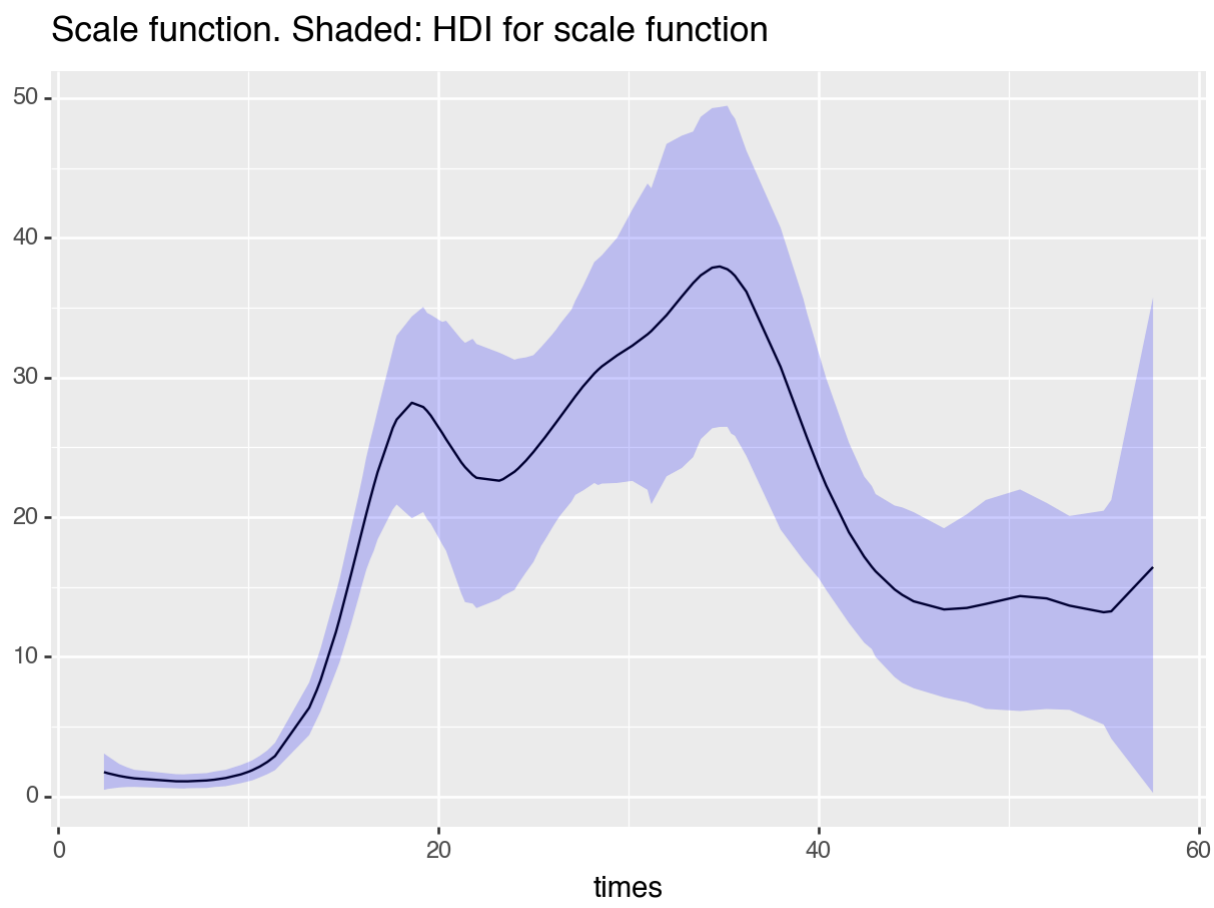


Or we can plot the scale directly as a function of our covariate, including uncertainty visualization with highest posterior density intervals:

```{python}
(
  ggplot()
  + aes(x = mcycle.times)
  + geom_line(aes(y = scale))
```

```
  + geom_ribbon(aes(ymin = scale_hdi_lo, ymax = scale_hdi_hi), alpha = 0.2, fill = "bl
  + labs(title = "Scale function. Shaded: HDI for scale function")
)
```

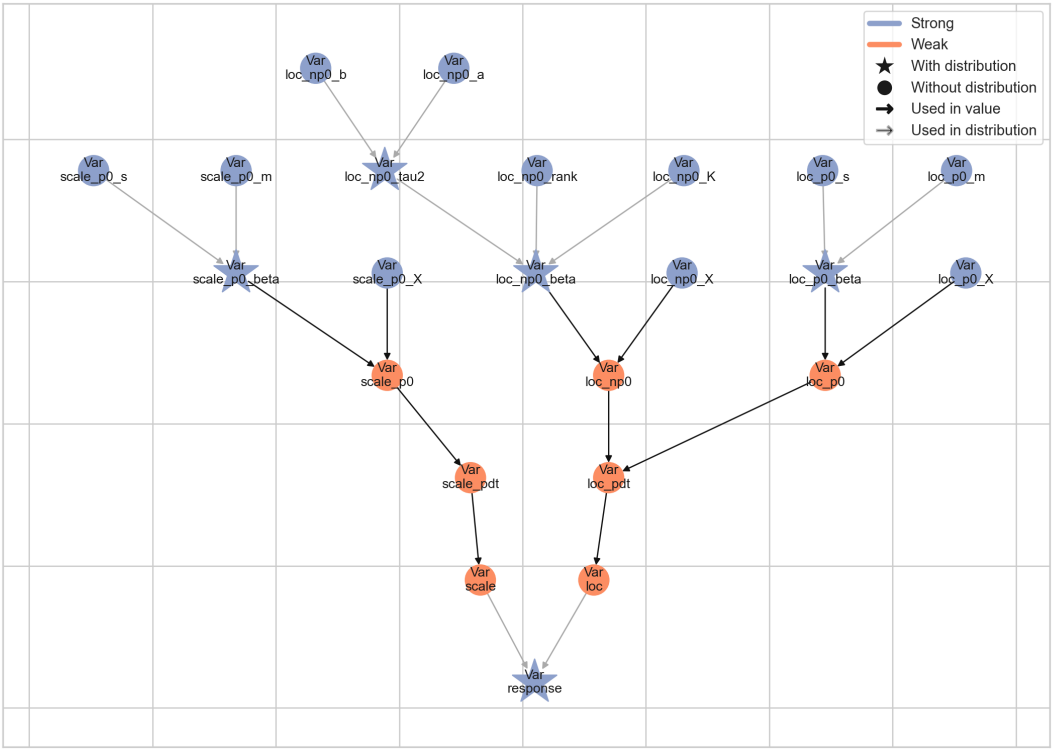
```
<Figure Size: (1280 x 960)>
```



Exercise 4: Recover the statistical model for the scale

Let's load the model again and plot the graph.

```{python}
model = dill.load(urlopen("https://s.gwdg.de/un4W29"))
lsl.plot_vars(model)
```



Information from the graph:

- `"scale_p0"` is the only term in the scale predictor.
- It has the inputs `"scale_p0_X"` and `"scale_p0_beta"`.

Let's look at these variables:

```
```{python}
model.vars["scale_p0_X"].value
model.vars["scale_p0_beta"].value
model.vars["scale_p0_beta"].value.shape
```
```

```
array([[1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
```

```
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
```

```
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
```

```
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.]], dtype=float32)
array([0.], dtype=float32)
(1,)
```

- The variable `"scale_p0_X"` is just a one-column matrix of ones, looking like the intercept-column of a design matrix.
- The variable `"scale_p0_beta"` is a scalar parameter.

Now how do the values enter `"scale_p0"`?

```{python}
model.vars["scale_p0"].weak
```

```
True
```

The variable `"scale_p0"` is weak, so we can assume that it wraps a calculator. We can access the calculator through the `lsl.Var.value_node` attribute. Then, we can access the function wrapped by the calculator through the `lsl.Calc.function` attribute. Python's `inspect.getsource()` function allows us to print the code that defined this function.

```{python}
import inspect

inspect.getsource(model.vars["scale_p0"].value_node.function)
```

```
'    calc = Calc(lambda x, beta: x @ beta, x=x, beta=beta)\n'
```

We can thus see that we have a model of the form:

$$\boldsymbol{\eta}_\sigma = \mathbf{X}_\sigma \boldsymbol{\beta}_\sigma,$$

where $\mathbf{X}_\sigma = \mathbf{1}$ and $\boldsymbol{\beta}_\sigma = \beta_\sigma$. Since we have specified the exponential function as the inverse link function in our call to `rliesel::liesel`, we now know that we have

$$\sigma = \exp(\eta_\sigma) = \exp(\beta_\sigma),$$

so β_σ in our model, represented by the variable `"scale_p0_beta"`, is the logarithm of the scale. Note that we have switched back to scalar notation here, because the model assumes a constant scale.

Now what is the prior that `rliesel` assigned for us? Let's take a closer look at the `"scale_p0_beta"` variable and its `lsl.Var.dist_node` attribute:

```{python}
model.vars["scale_p0_beta"].dist_node
model.vars["scale_p0_beta"].dist_node.distribution
model.vars["scale_p0_beta"].dist_node.distribution.__name__
```

```
Dist(name="scale_p0_beta_log_prob")
<class 'tensorflow_probability.substrates.jax.distributions.normal.Normal'>
'Normal'
```

This tells us: β_σ has a normal prior. Now what are the location and scale of this prior? We can get this insight through the `lsl.Node.kwinputs` attribute:

```{python}
model.vars["scale_p0_beta"].dist_node.kwinputs
model.vars["scale_p0_beta"].dist_node.kwinputs["loc"].value
model.vars["scale_p0_beta"].dist_node.kwinputs["scale"].value
```

```
mappingproxy({'loc': VarValue(name="scale_p0_m_var_value"), 'scale':
VarValue(name="scale_p0_s_var_value")})
array([0.], dtype=float32)
array([100.], dtype=float32)
```

So we know now that the prior is specified as

$$\beta_\sigma \sim \mathcal{N}\left(0, 100^2\right),$$

which makes the prior for σ

$$\sigma \sim \text{Lognormal}\left(0, 100^2\right).$$