Hamiltonian Monte Carlo with Stan and Ulam

EES 5891-03

Bayesian Statistical Methods

Jonathan Gilligan

Class #12: Thursday, October 06 2022

Hamiltonian Monte Carlo Sampling

Prepare data

- `rugged` data set:
  - Each row is a different country
  - `rugged`: Topographic roughness of the terrain
  - `rpcgdp_2000`: Per-capita GDP in 2000
- Use the logarithm of GDP.
- quasi-standardize the variables.
- Model $$ \begin{align} G &\sim \text{Normal}(\mu, \sigma) \\ \mu &= \alpha_{\text{CID}} + \beta_{\text{CID}} R \\ \alpha_{\text{CID}} &\sim \text{Normal}(1, 0.1) \\ \beta_{\text{CID}} &\sim \text{Normal}(0, 0.3) \\ \sigma &\sim \text{Exponential}(1) \end{align} $$

```
library(tidyverse)
library(rethinking)

data(rugged)
d <- rugged

dd <- d %>% filter(complete.cases(rgdppc_2000)) %>%
  mutate(
    log_gdp = log(rgdppc_2000),
    log_gdp_std = log_gdp / mean(log_gdp),
    rugged_std = rugged / max(rugged),
    cid = ifelse(cont_africa == 1, 1, 2)
  )
```

```
lst_rug <- alist(
  log_gdp_std ~ dnorm(mu, sigma),
  mu <- a[cid] + b[cid] * (rugged_std - 0.215),
  a[cid] ~ dnorm(1, 0.1),
  b[cid] ~ dnorm(0, 0.3),
  sigma ~ dexp(1)
)
```

`quap` vs. `ulam`

- `quap` calculates the posterior using a *quadratic approximation*
  - Focus on finding the peak of the posterior (maximum *a-posteriori*) for each parameter.
    - Near the peak, the posterior looks like a quadratic function (parabola).
  - Approximate the width
  - Works well with Gaussian (Normal) distributions, not so well with other functions.

- `ulam` calculates the posterior using a *Hamiltonian Monte Carlo* sampler.
  - Generates samples from the posterior.
  - Doesn't directly calculate the *maximum a-posteriori*.
- Use statistical functions to analyze the posterior (mean, median, standard deviation, etc.)
- Runs slower than `quap` but works with a greater variety of prior and likelihood functions.

ulam and stan

```
lst_rug <- alist(
  log_gdp_std ~ dnorm(mu, sigma),
  mu <- a[cid] + b[cid] * (rugged_std - 0.215),
  a[cid] ~ dnorm(1, 0.1),
  b[cid] ~ dnorm(0, 0.3),
  sigma ~ dexp(1)
)
dat_slim <- list(log_gdp_std = dd$log_gdp_std,
                 rugged_std = dd$rugged_std,
                 cid = as.integer(dd$cid))
mdl_rug_ulam <- ulam(lst_rug, data = dat_slim,
                     chains = 4, cores = 4)
```

```
## Running MCMC with 4 parallel chains, with 1 thread(s)
per chain...
##
## Chain 1 Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1 Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1 Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1 Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1 Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1 Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1 Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1 Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1 Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1 Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1 Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1 Iteration: 1000 / 1000 [100%]  (Sampling)
```

Ulam creates this Stan model:

```
cat(mdl_rug_ulam@model)
```

```
## data{
##     vector[170] log_gdp_std;
##     vector[170] rugged_std;
##     int cid[170];
## }
## parameters{
##     vector[2] a;
##     vector[2] b;
##     real<lower=0> sigma;
## }
## model{
##     vector[170] mu;
##     sigma ~ exponential( 1 );
##     b ~ normal( 0 , 0.3 );
##     a ~ normal( 1 , 0.1 );
##     for ( i in 1:170 ) {
##         mu[i] = a[cid[i]] + b[cid[i]] * (rugged_std[i]
- 0.215);
##     }
##     log_gdp_std ~ normal( mu , sigma );
## }
```

Running `ulam`

- The model specification is pretty much the same as for `quap`.
- Data must **only** have the observed variables in the model
  - Do transforms (`log()`, `standardize()`) before calling `ulam`.
- When you call `ulam()` it.
  1. Generates `stan` code for the model.
  2. Calls the `stan` compiler to translate the `stan` model into C++.
  3. Compiles the C++ model.
  4. Runs the model with the data you provided.
     - Default: 1000 iterations per chain
     - Half are *warm-up* and half are samples from posterior
       - *warm-up* is where the model tunes itself
  5. Returns the results.

```
lst_rug <- alist(
  log_gdp_std ~ dnorm(mu, sigma),
  mu <- a[cid] + b[cid] * (rugged_std - 0.215),
  a[cid] ~ dnorm(1, 0.1),
  b[cid] ~ dnorm(0, 0.3),
  sigma ~ dexp(1)
)
mdl_rug_quap <- quap(lst_rug, data = dat_slim)
dat_slim <- list(log_gdp_std = dd$log_gdp_std,
                 rugged_std = dd$rugged_std,
                 cid = as.integer(dd$cid))
mdl_rug_ulam <- ulam(lst_rug, data = dat_slim,
                     chains = 4, cores = 4)
```

Running `ulam`

- `chains`: A *chain* is a sequence of samples.
  - *Markov chain*: Each sample depends on the previous sample.
  - Different chains start from different starting points.
  - *Mixing*: After a certain number of samples, chains that start at different starting points should become indistinguishable.
  - You should sample multiple chains so you can check whether the chains are well-mixed.
- `cores`: How many processor cores to use:
  - When `cores = 1`, the chains execute sequentially
  - When `cores = chains`, all chains execute in parallel (faster).
  - There's no extra benefit in having more `cores` than `chains`.

```
lst_rug <- alist(
  log_gdp_std ~ dnorm(mu, sigma),
  mu <- a[cid] + b[cid] * (rugged_std - 0.215),
  a[cid] ~ dnorm(1, 0.1),
  b[cid] ~ dnorm(0, 0.3),
  sigma ~ dexp(1)
)
dat_slim <- list(log_gdp_std = dd$log_gdp_std,
                 rugged_std = dd$rugged_std,
                 cid = as.integer(dd$cid))
mdl_rug_ulam <- ulam(lst_rug, data = dat_slim,
                     chains = 4, cores = 4)
```

Interpreting Model Results

Model Output

```
lst_rug <- alist(
  log_gdp_std ~ dnorm(mu, sigma),
  mu <- a[cid] + b[cid] * (rugged_std - 0.215),
  a[cid] ~ dnorm(1, 0.1),
  b[cid] ~ dnorm(0, 0.3),
  sigma ~ dexp(1)
)
dat_slim <- list(log_gdp_std = dd$log_gdp_std,
                 rugged_std = dd$rugged_std,
                 cid = as.integer(dd$cid))
mdl_rug_ulam <- ulam(lst_rug, data = dat_slim,
                     chains = 4, cores = 4)
```

- For each parameter, `precis` shows:
  - Statistical properties: mean, standard deviation, 5.5% and 94.5% quantiles,
  - `n_eff`: The effective number of samples (smaller than the actual number because of autocorrelation)
  - `Rhat4`: The Gelman-Rubin convergence diagnostic:
    - variance between chains / variance within chain
    - 1 when chains have converged completely
    - ≤ 1.1 indicates good convergence (well-mixed)

```
show(mdl_rug_ulam)
```

```
## Hamiltonian Monte Carlo approximation
## 2000 samples from 4 chains
##
## Sampling durations (seconds):
##         warmup sample total
## chain:1   0.08   0.05  0.13
## chain:2   0.09   0.06  0.15
## chain:3   0.08   0.05  0.13
## chain:4   0.08   0.05  0.13
##
## Formula:
## log_gdp_std ~ dnorm(mu, sigma)
## mu <- a[cid] + b[cid] * (rugged_std - 0.215)
## a[cid] ~ dnorm(1, 0.1)
## b[cid] ~ dnorm(0, 0.3)
## sigma ~ dexp(1)
```

```
precis_show(precis(mdl_rug_ulam, depth = 2, digits = 2))
```

```
##         mean   sd  5.5% 94.5% n_eff Rhat4
## a[1]    0.89 0.02  0.86  0.91  3016     1
## a[2]    1.05 0.01  1.03  1.07  2414     1
## b[1]    0.13 0.08  0.01  0.26  3307     1
## b[2]   -0.14 0.06 -0.24 -0.05  3287     1
## sigma   0.11 0.01  0.10  0.12  2306     1
```

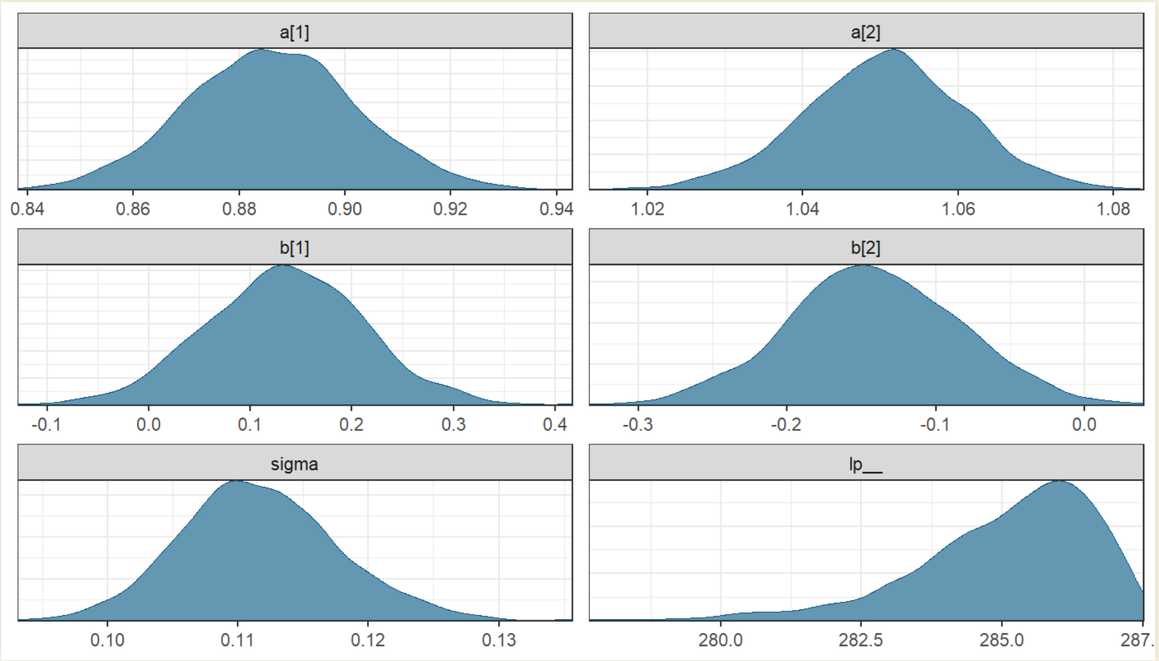
Plotting Model Results

```
library(bayesplot)

mdl_fit <- mdl_rug_ulam@stanfit

mcmc_dens(mdl_fit)
```

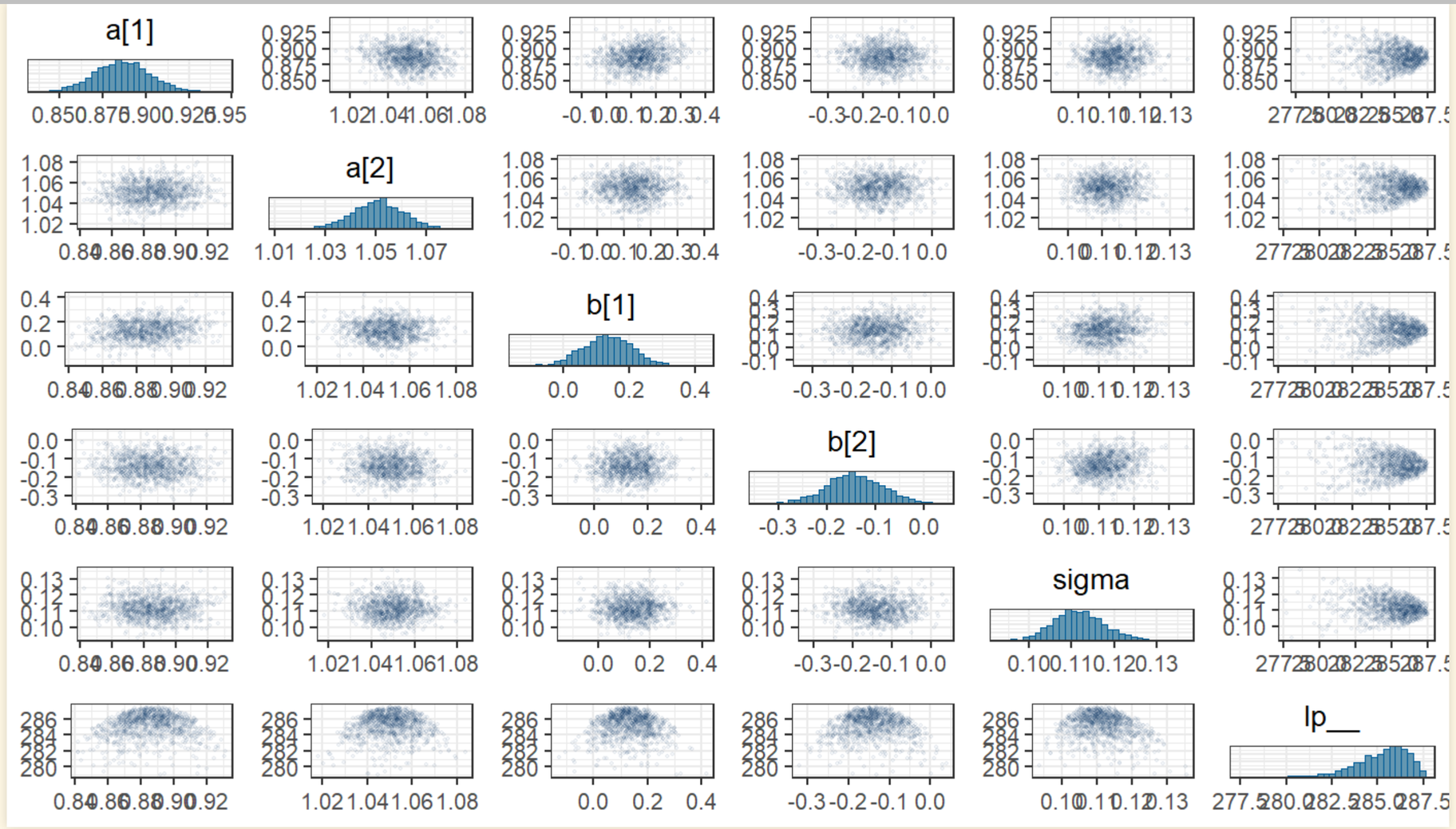
- The graphs show the probability density of each parameter.
- `lp__` is the log of the posterior probability density.
  - This graph shows how the MCMC samples are mostly drawn from higher probability regions of the posterior.



Model Diagnostics

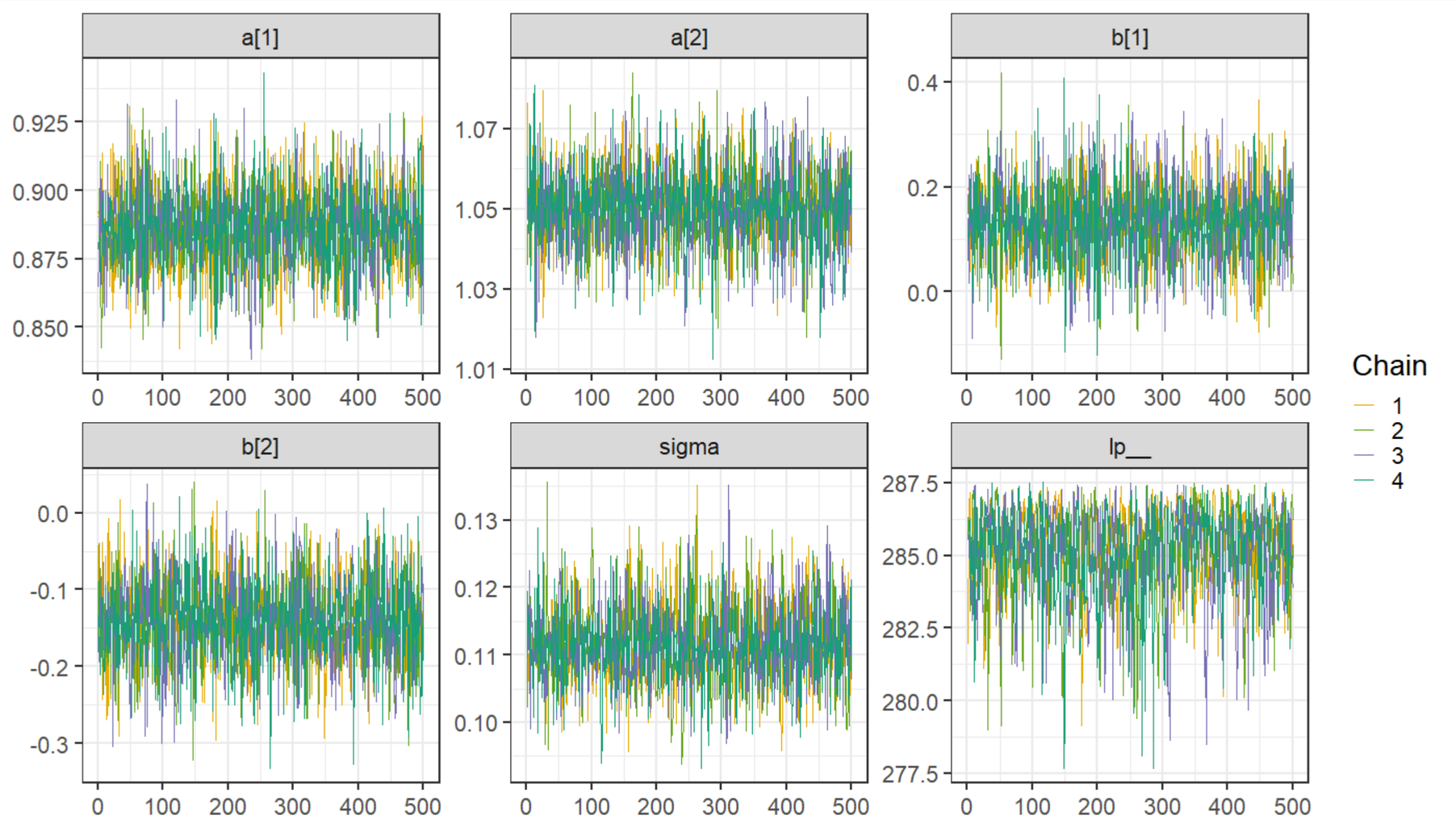
Pair-Correlation Plots

```
color_scheme_set("blue")
mcmc_pairs(mdl_fit, off_diag_args = list(size = 1, alpha = 0.05))
```



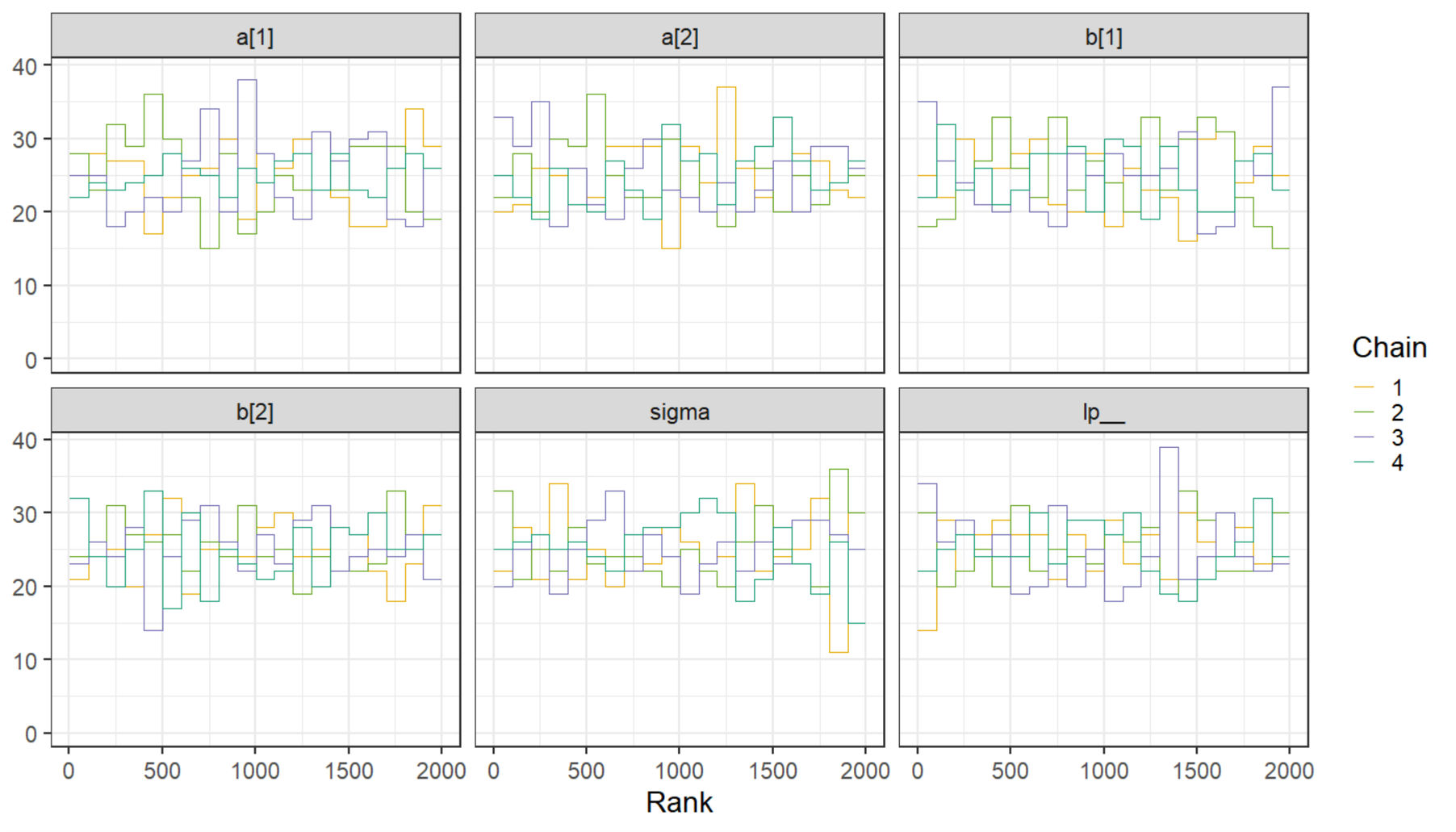
MCMC Trace Plots

```
color_scheme_set("brewer-Dark2")
mcmc_trace(mdl_fit)
```



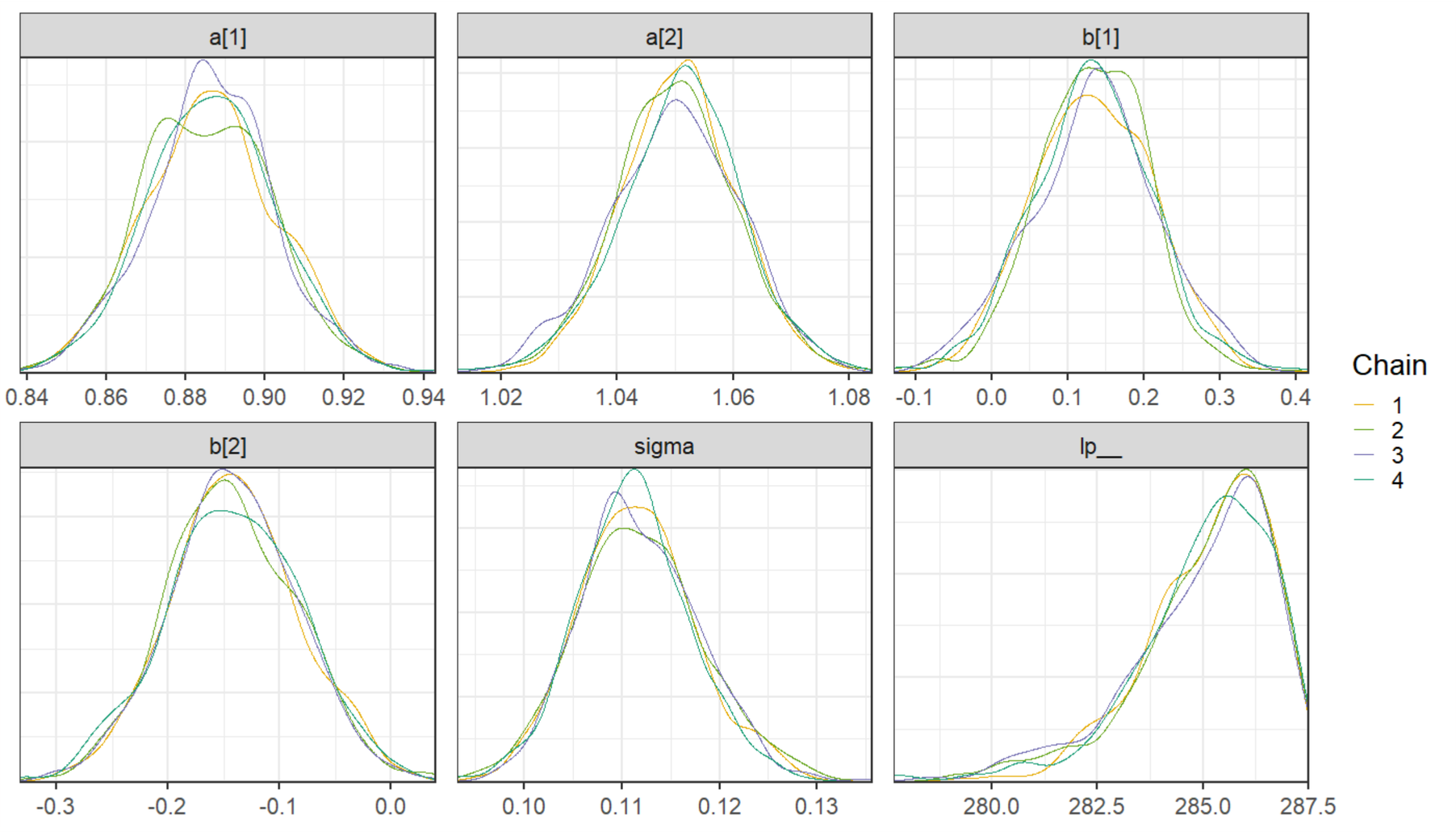
MCMC Trace-Rank Plots

```
mcmc_rank_overlay(mdl_fit)
```



Posterior Density Plots

```
mcmc_dens_overlay(mdl_fit)
```



MCMC Problems

Poorly Fit Model

```
y <- c(-1, 1)
set.seed(11)
mdl_bad <- ulam(
  alist(
    y ~ dnorm(mu, sigma),
    mu <- alpha,
    alpha ~ dnorm(0, 1000),
    sigma ~ dexp(0.0001)
  ), data = list(y = y), chains = 4, cores = 4 )
```

- 2 data observations
- Very uninformative priors
- Gives warning:

::: {bare .mtop-1 .max-listing .seventy}

```
Warning: 87 of 2000 (4.0%)
transitions ended with a
divergence.
See https://mc-
stan.org/misc/warnings for
details.
```

:::

```
show(mdl_bad)
```

```
## Hamiltonian Monte Carlo approximation
## 2000 samples from 4 chains
##
## Sampling durations (seconds):
##         warmup sample total
## chain:1   0.07   0.04  0.11
## chain:2   0.07   0.06  0.13
## chain:3   0.15   0.06  0.20
## chain:4   0.04   0.04  0.08
##
## Formula:
## y ~ dnorm(mu, sigma)
## mu <- alpha
## alpha ~ dnorm(0, 1000)
## sigma ~ dexp(1e-04)
```

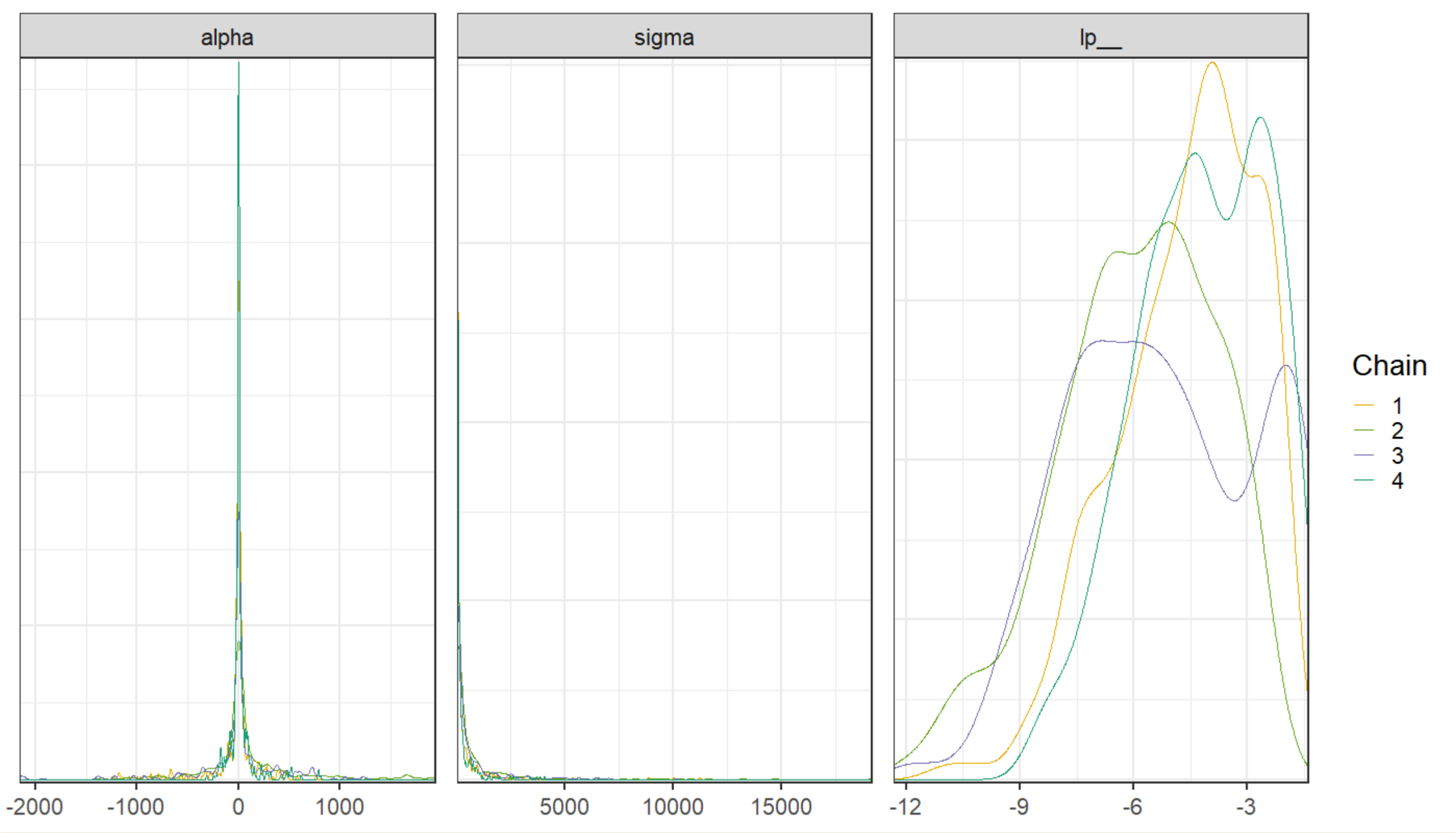
```
precis_show(precis(mdl_bad, digits=2))
```

```
##          mean      sd    5.5%   94.5% n_eff Rhat4
## alpha  16.35  334.62 -354.99  481.62   194  1.02
## sigma 492.14 1303.25    4.84 2083.80   156  1.04
```

- `n_eff`: <200 effective samples out of 2000.
- `sd`: Huge standard deviations in posteriors.

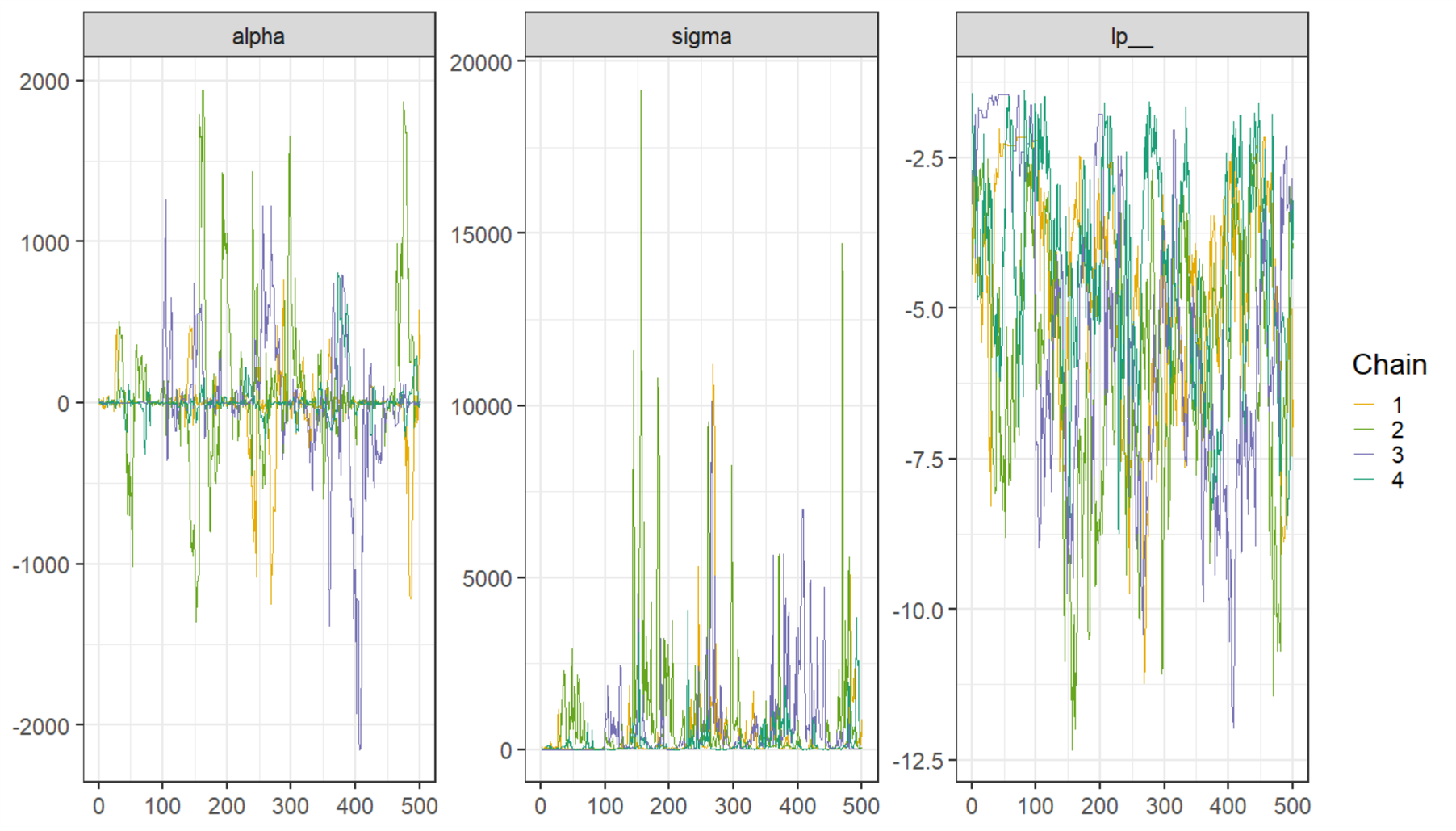
Show Posteriors

```
bad_mdl_fit <- mdl_bad@stanfit
color_scheme_set("brewer-Dark2")
mcmc_dens_overlay(bad_mdl_fit)
```



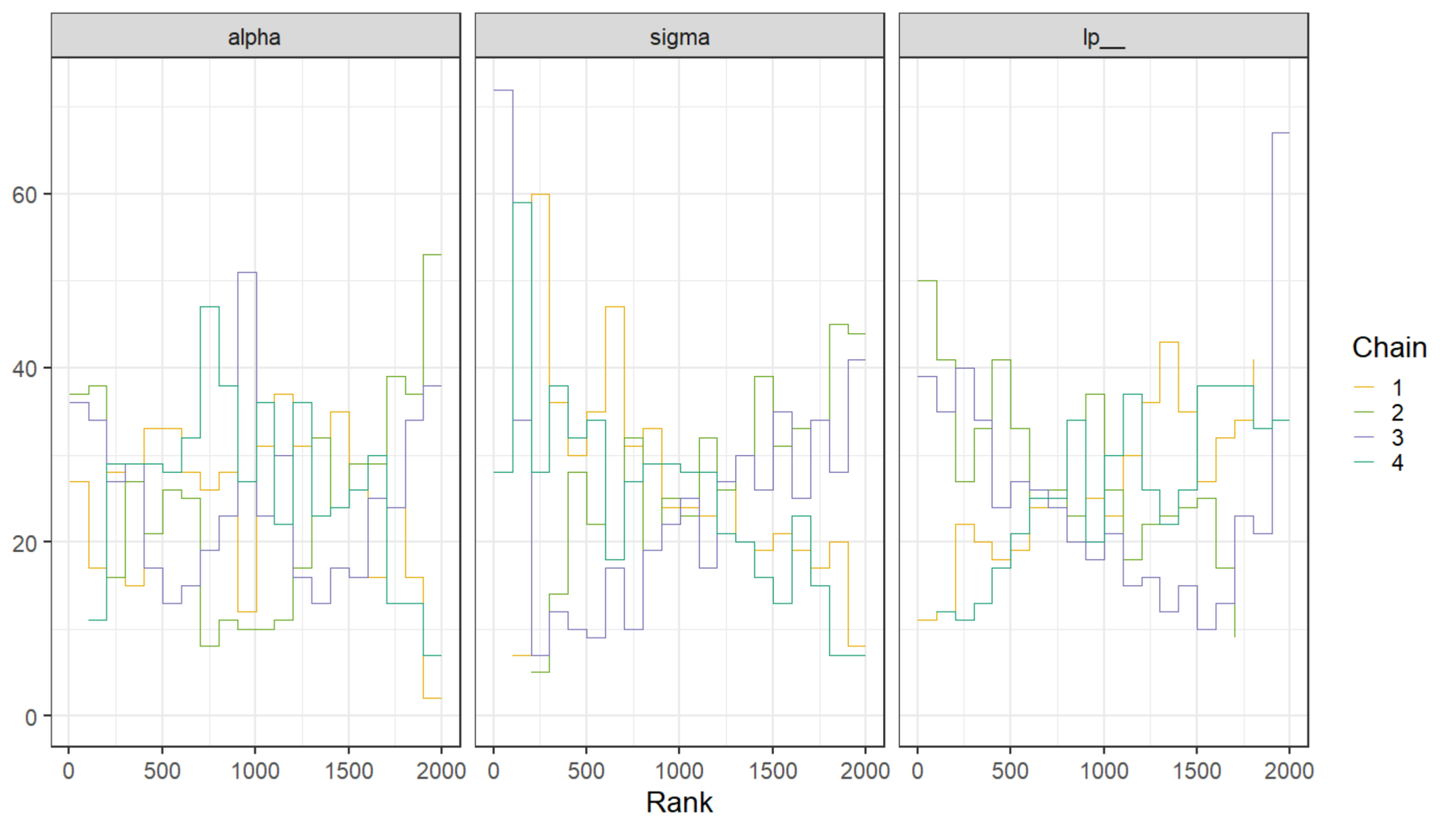
MCMC Trace Plots

```
mcmc_trace(bad_mdl_fit)
```



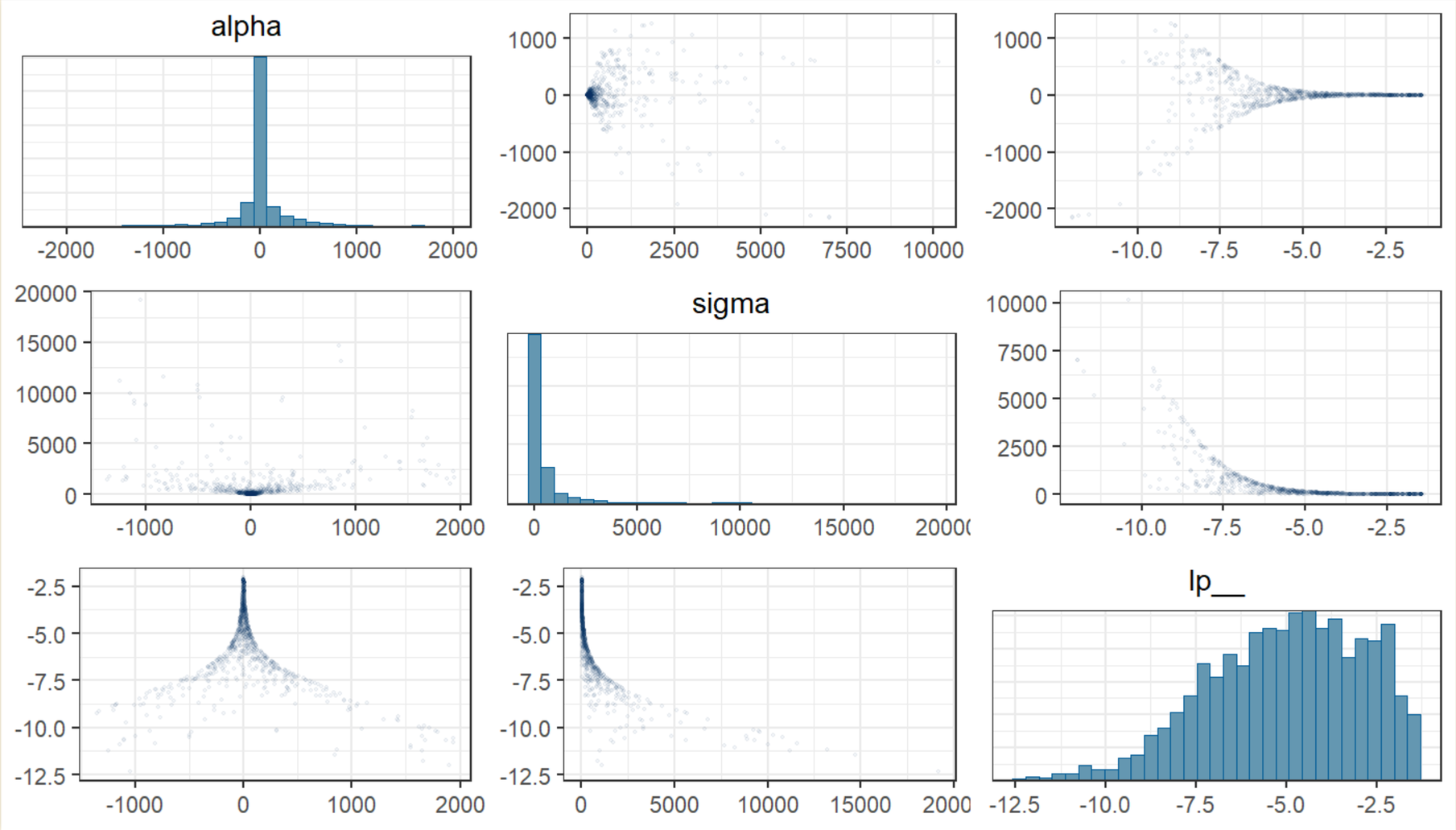
MCMC Trace-Rank Plots

```
mcmc_rank_overlay(bad_mdl_fit)
```



Pair-Correlation Plots: Funnels

```
color_scheme_set("blue")
mcmc_pairs(bad_mdl_fit, off_diag_args = list(size = 1, alpha = 0.05))
```



Better Model

```
set.seed(11)
mdl_better <- ulam(
  alist(
    y ~ dnorm(mu, sigma),
    mu <- alpha,
    alpha ~ dnorm(0, 10),
    sigma ~ dexp(1)
  ), data = list(y = y), chains = 4, cores = 4 )
```

- Replace uninformative priors with w akly informative priors.akly informative priors help a lot.

```
show(mdl_better)
```

```
## Hamiltonian Monte Carlo approximation
## 2000 samples from 4 chains
##
## Sampling durations (seconds):
##         warmup sample total
## chain:1   0.03   0.02  0.05
## chain:2   0.02   0.02  0.04
## chain:3   0.03   0.02  0.04
## chain:4   0.03   0.03  0.05
##
## Formula:
## y ~ dnorm(mu, sigma)
## mu <- alpha
## alpha ~ dnorm(0, 10)
## sigma ~ dexp(1)
```

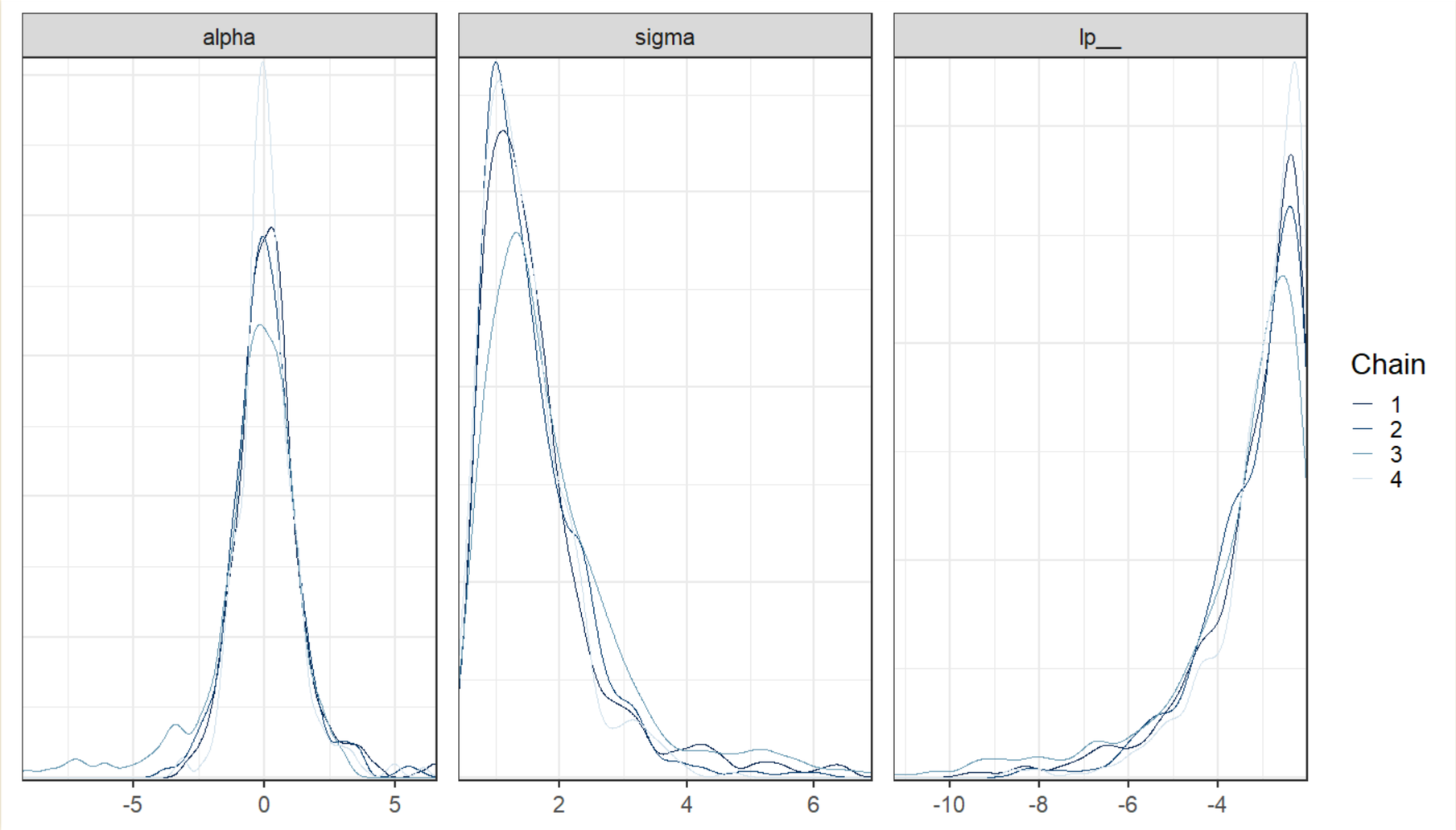
```
precis_show(precis(mdl_bad, digits=2))
```

```
##          mean       sd    5.5%   94.5% n_eff Rhat4
## alpha   16.35   334.62 -354.99  481.62   194  1.02
## sigma 492.14 1303.25    4.84 2083.80   156  1.04
```

- Even weakly informative priors help a lot!
- `n_eff`: >400 effective samples.
- `sd`: Much smaller.

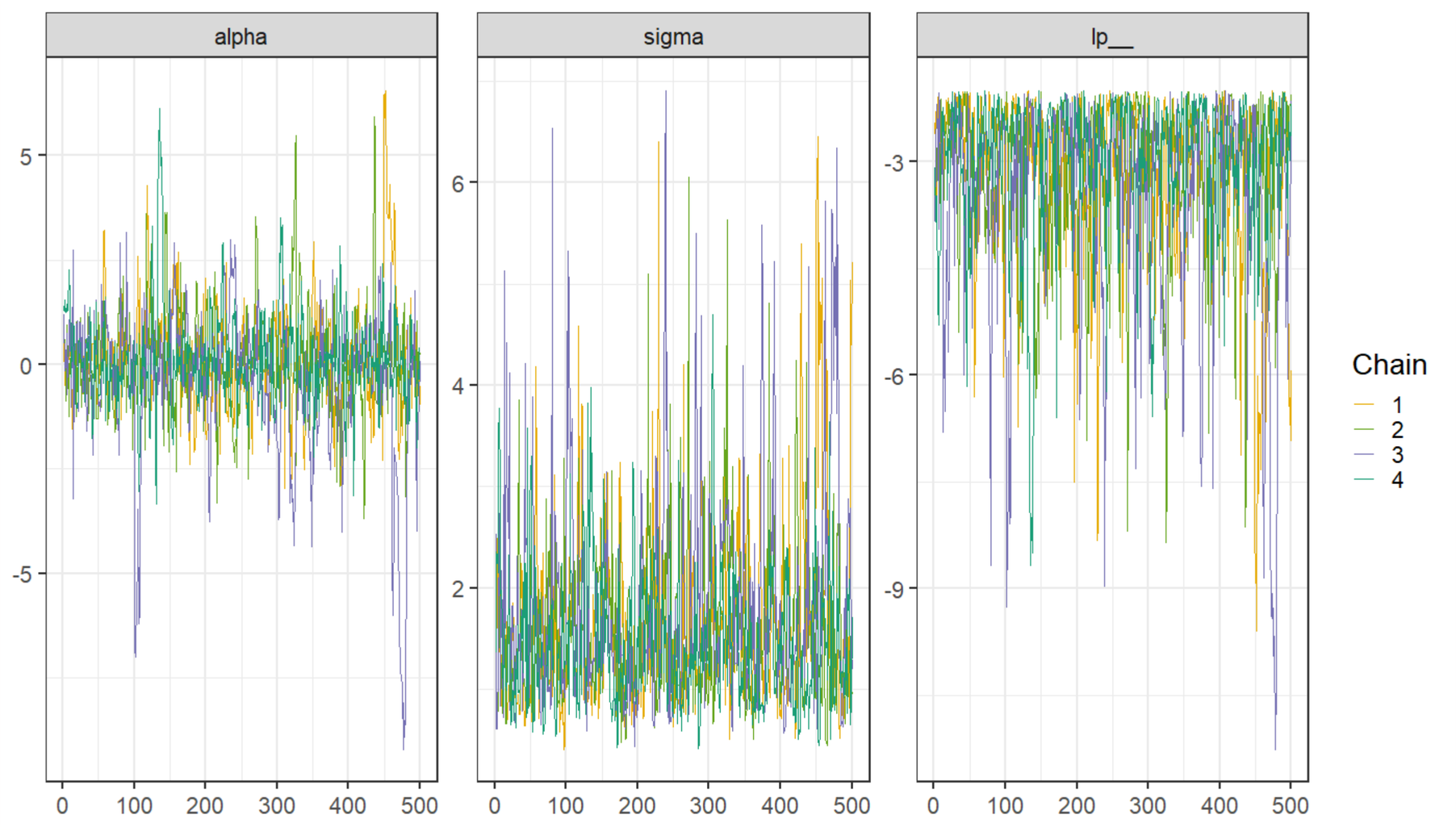
Show Posteriors

```
better_mdl_fit <- mdl_better@stanfit
mcmc_dens_overlay(better_mdl_fit)
```



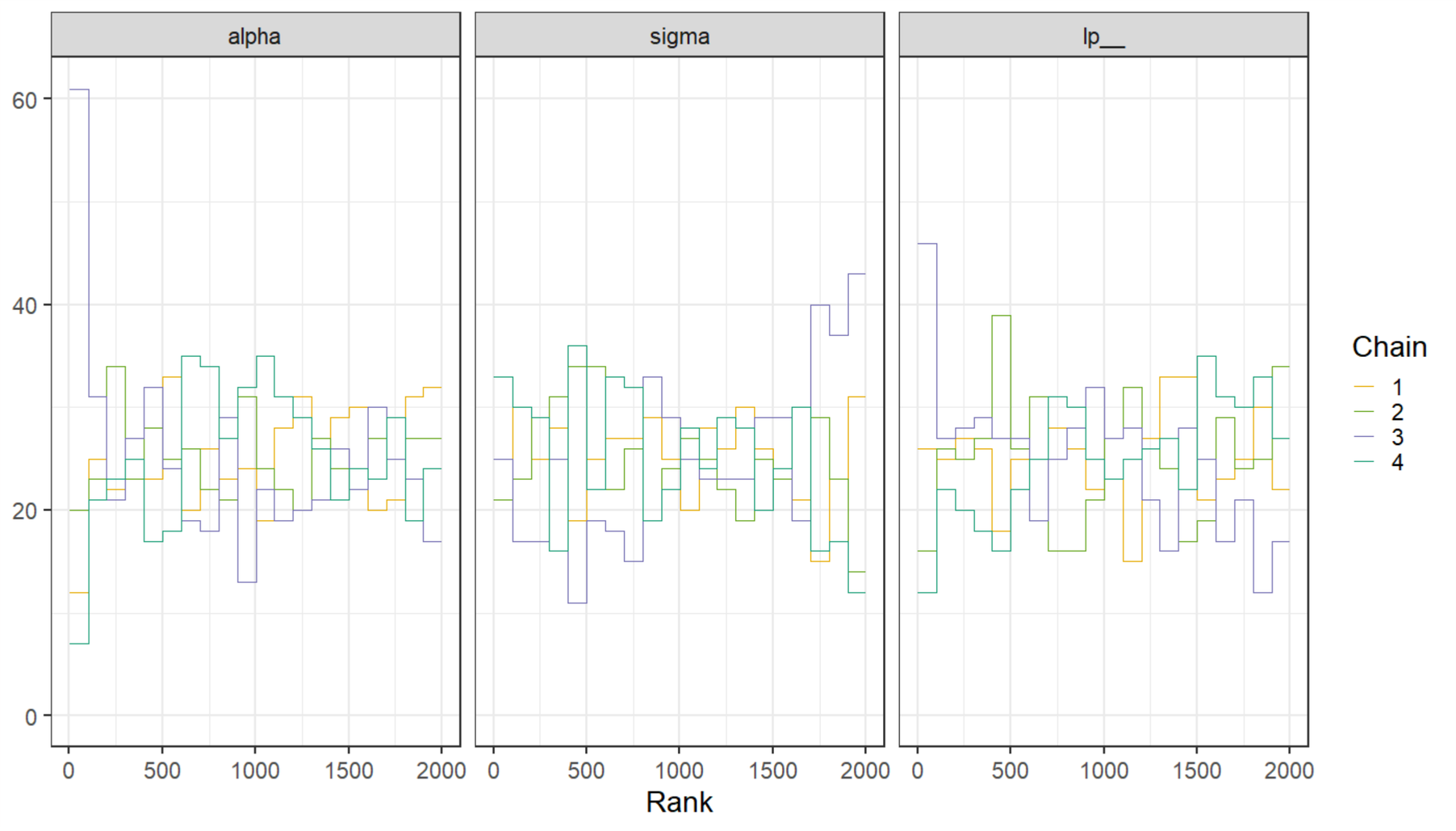
MCMC Trace Plots

```
color_scheme_set("brewer-Dark2")
mcmc_trace(better_mdl_fit)
```



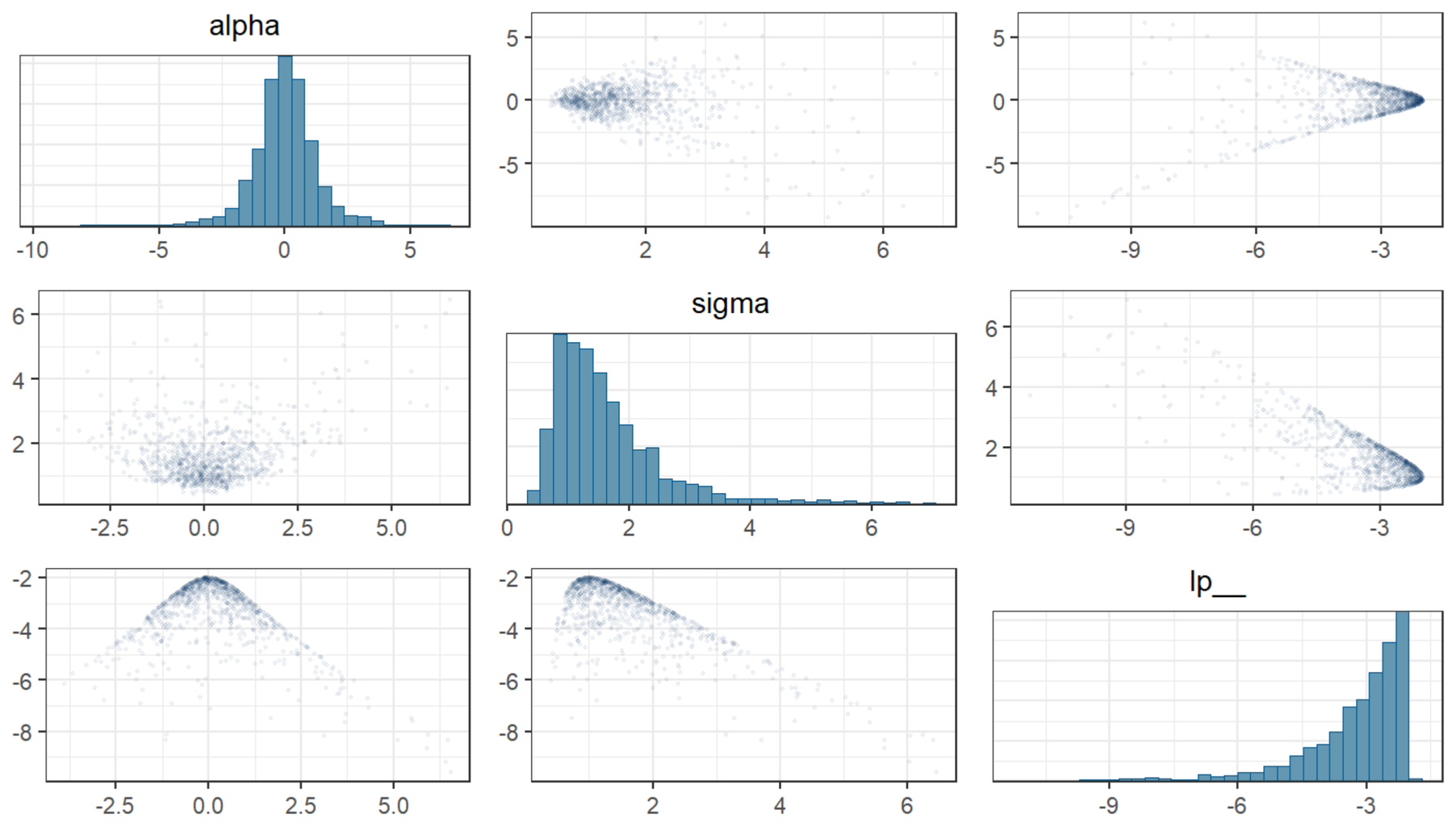
MCMC Trace-Rank Plots

```
mcmc_rank_overlay(better_mdl_fit)
```



Pair-Correlation Plots

```
color_scheme_set("blue")
mcmc_pairs(better_mdl_fit, off_diag_args = list(size = 1, alpha = 0.05))
```



Non-Identifiable Models

Non-Identifiable Model

```
set.seed(41)
y <- rnorm(100, mean = 0, sd = 1)
```

- Fit model $$ \begin{align} y &\sim \text{Normal}(\mu, \sigma) \\ \mu &= \alpha_1 + \alpha_2 \\ \alpha_1 &\sim \text{Normal}(0, 1000) \\ \alpha_2 &\sim \text{Normal}(0, 1000) \\ \sigma &\sim \text{Exponential}(1) \end{align} $$

```
set.seed(11)
mdl_non_id <- ulam(
  alist(
    y ~ dnorm(mu, sigma),
    mu <- a1 + a2,
    a1 ~ dnorm(0, 1000),
    a2 ~ dnorm(0, 1000),
    sigma ~ dexp(1)
  ), data = list(y = y), chains = 4, cores = 4 )
```

```
show(mdl_non_id)
```

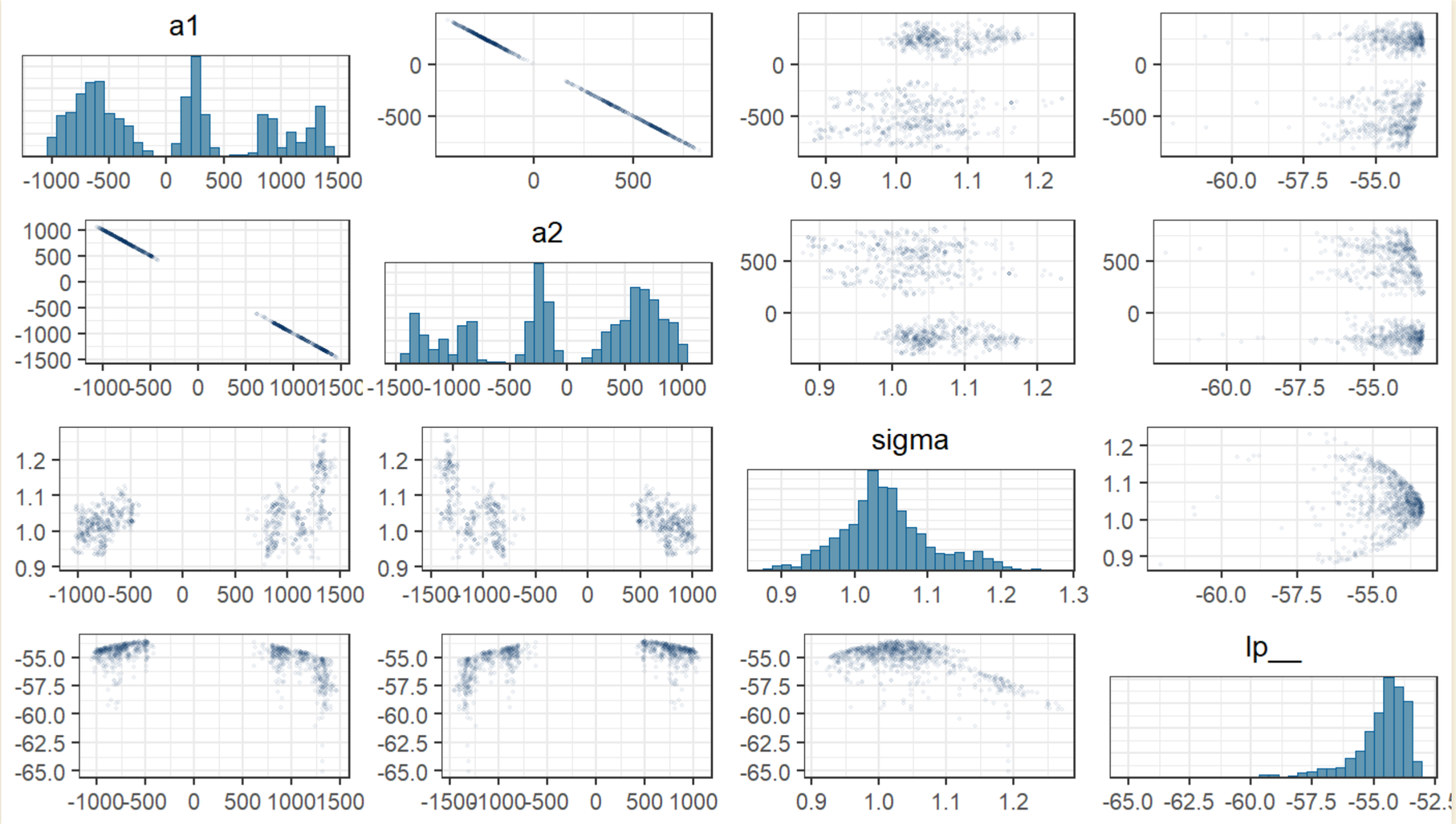
```
## Hamiltonian Monte Carlo approximation
## 2000 samples from 4 chains
##
## Sampling durations (seconds):
##         warmup sample total
## chain:1   1.20   1.30  2.50
## chain:2   1.08   1.35  2.43
## chain:3   1.12   1.33  2.45
## chain:4   1.19   1.25  2.45
##
## Formula:
## y ~ dnorm(mu, sigma)
## mu <- a1 + a2
## a1 ~ dnorm(0, 1000)
## a2 ~ dnorm(0, 1000)
## sigma ~ dexp(1)
```

```
precis_show(precis(mdl_non_id, digits=2))
```

```
##          mean     sd     5.5%   94.5% n_eff Rhat4
## a1      11.76 744.23  -907.39 1317.93     2  5.85
## a2     -11.58 744.24 -1317.69  907.51     2  5.85
## sigma    1.04   0.06     0.95    1.17    32  1.16
```

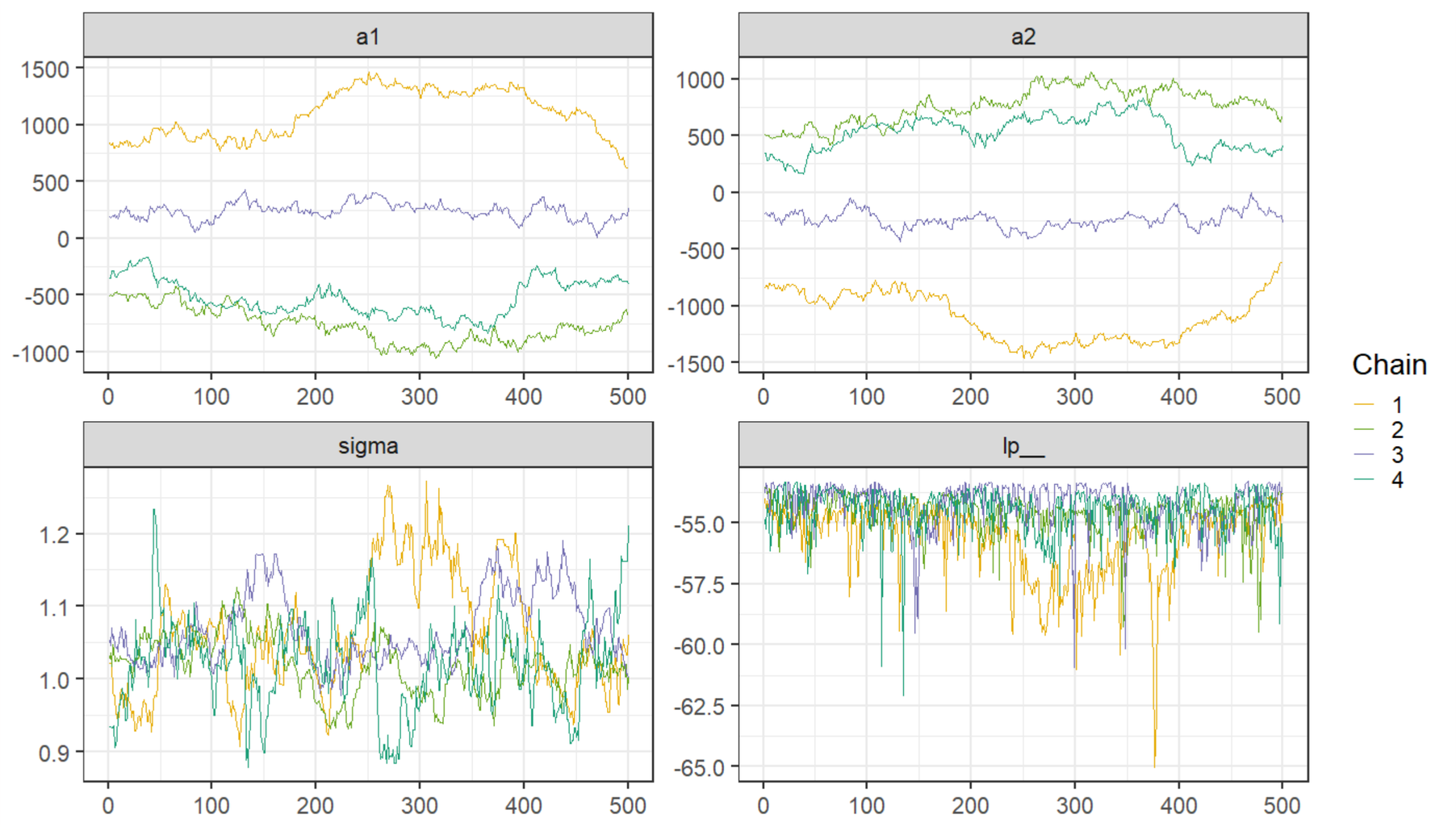
Pair-Correlation Plots

```
color_scheme_set("blue")
nid_mdl_fit <- mdl_non_id@stanfit
mcmc_pairs(nid_mdl_fit, off_diag_args = list(size = 1, alpha = 0.05))
```



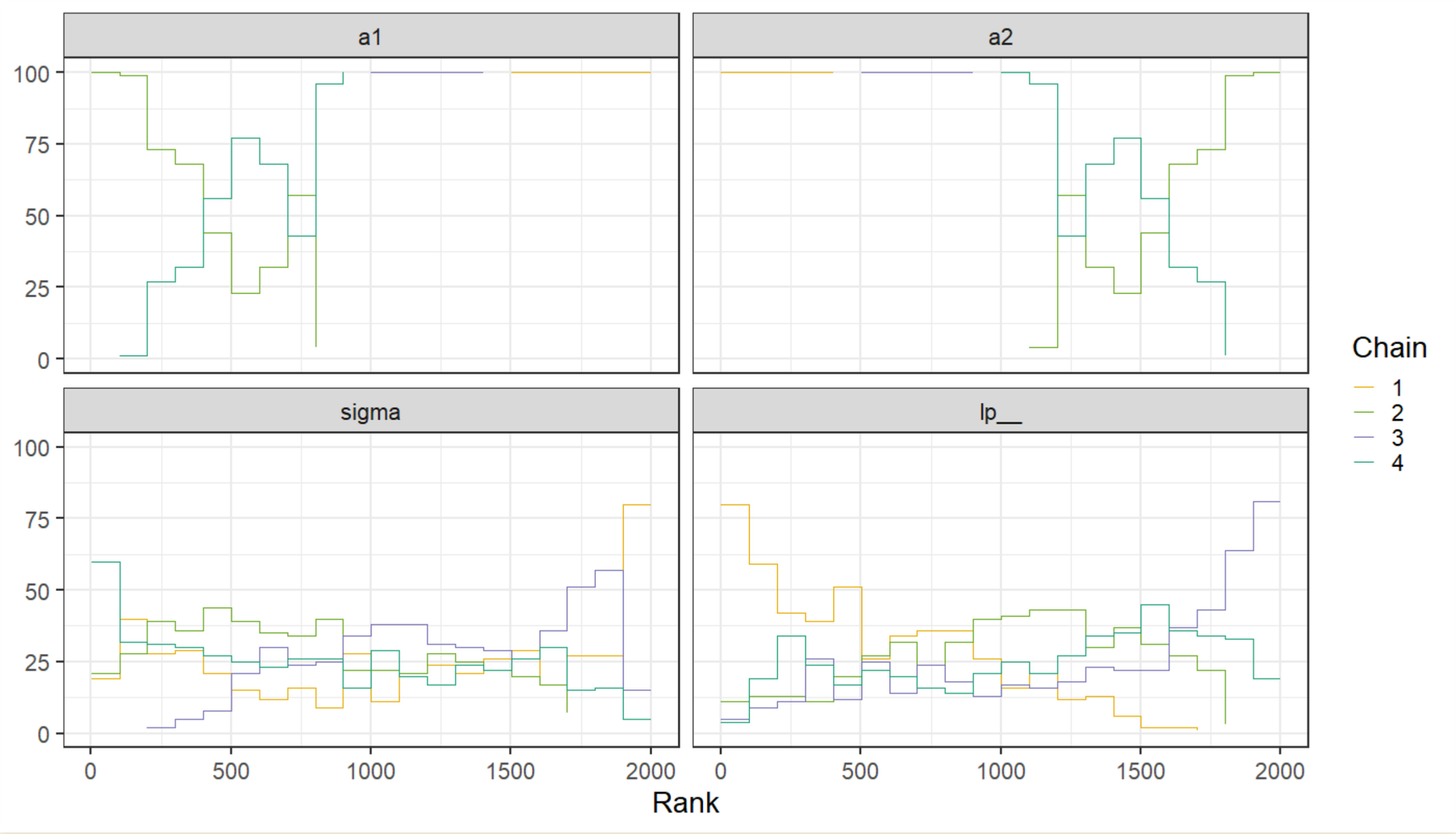
MCMC Trace Plots

```
color_scheme_set("brewer-Dark2")
mcmc_trace(nid_mdl_fit)
```



MCMC Trace-Rank Plots

```
mcmc_rank_overlay(nid_mdl_fit)
```



Regularize Model

- Model with regularizing priors $$ \begin{align} y &\sim \text{Normal}(\mu, \sigma) \\ \mu &= \alpha_1 + \alpha_2 \\ \alpha_1 &\sim \text{Normal}(0, 10) \\ \alpha_2 &\sim \text{Normal}(0, 10) \\ \sigma &\sim \text{Exponential}(1) \end{align} $$

```
set.seed(11)
mdl_reg <- ulam(
  alist(
    y ~ dnorm(mu, sigma),
    mu <- a1 + a2,
    a1 ~ dnorm(0, 10),
    a2 ~ dnorm(0, 10),
    sigma ~ dexp(1)
  ), data = list(y = y), chains = 4, cores = 4 )
```

```
show(mdl_reg)
```

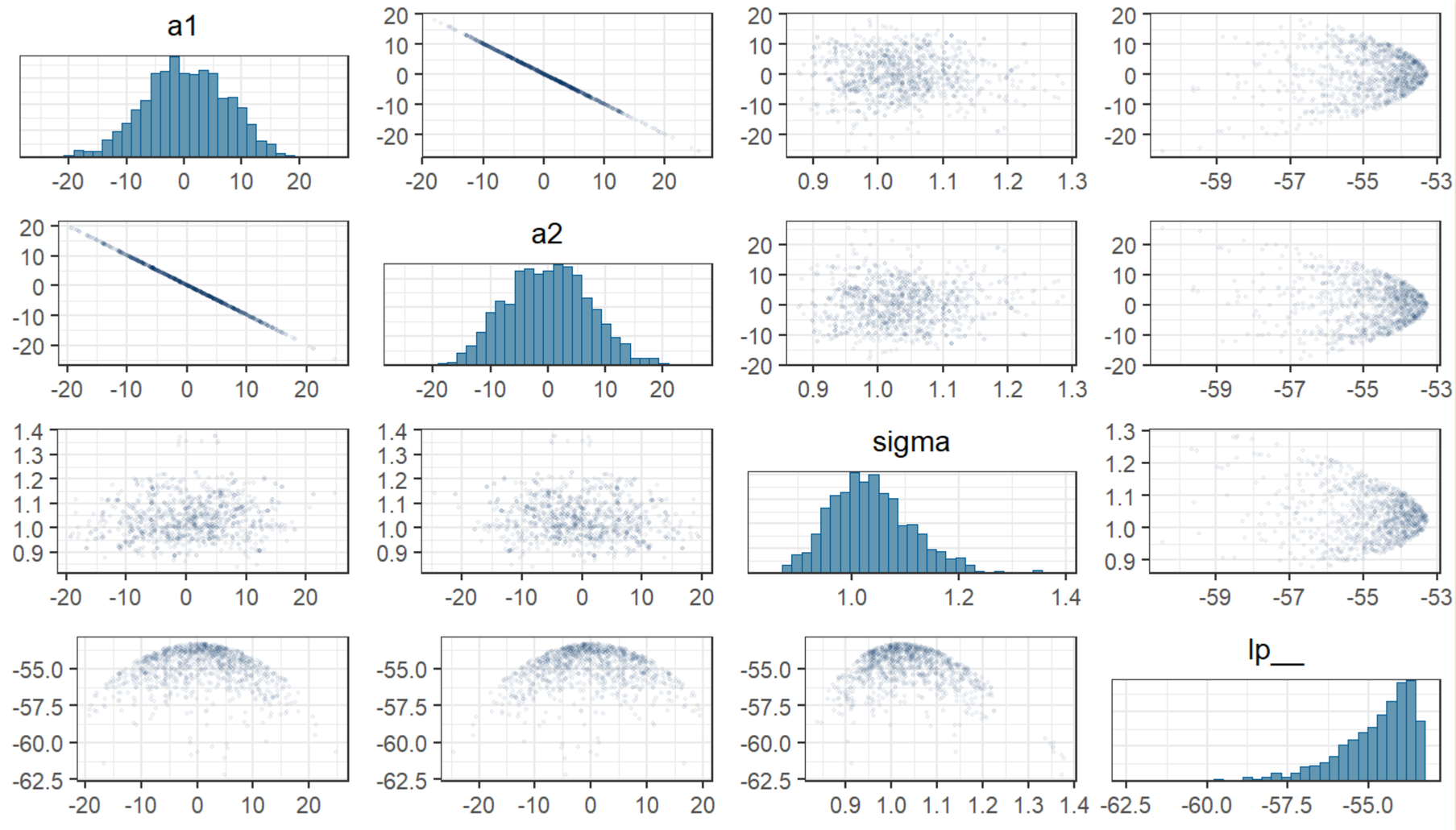
```
## Hamiltonian Monte Carlo approximation
## 2000 samples from 4 chains
##
## Sampling durations (seconds):
##         warmup sample total
## chain:1   0.46   0.45  0.91
## chain:2   0.43   0.49  0.92
## chain:3   0.39   0.42  0.81
## chain:4   0.47   0.44  0.92
##
## Formula:
## y ~ dnorm(mu, sigma)
## mu <- a1 + a2
## a1 ~ dnorm(0, 10)
## a2 ~ dnorm(0, 10)
## sigma ~ dexp(1)
```

```
precis_show(precis(mdl_reg, digits=2))
```

```
##         mean   sd   5.5% 94.5% n_eff Rhat4
## a1      0.25 7.19 -11.21 11.45   670  1.01
## a2     -0.06 7.19 -11.20 11.40   670  1.01
## sigma   1.04 0.08   0.93  1.17   616  1.00
```

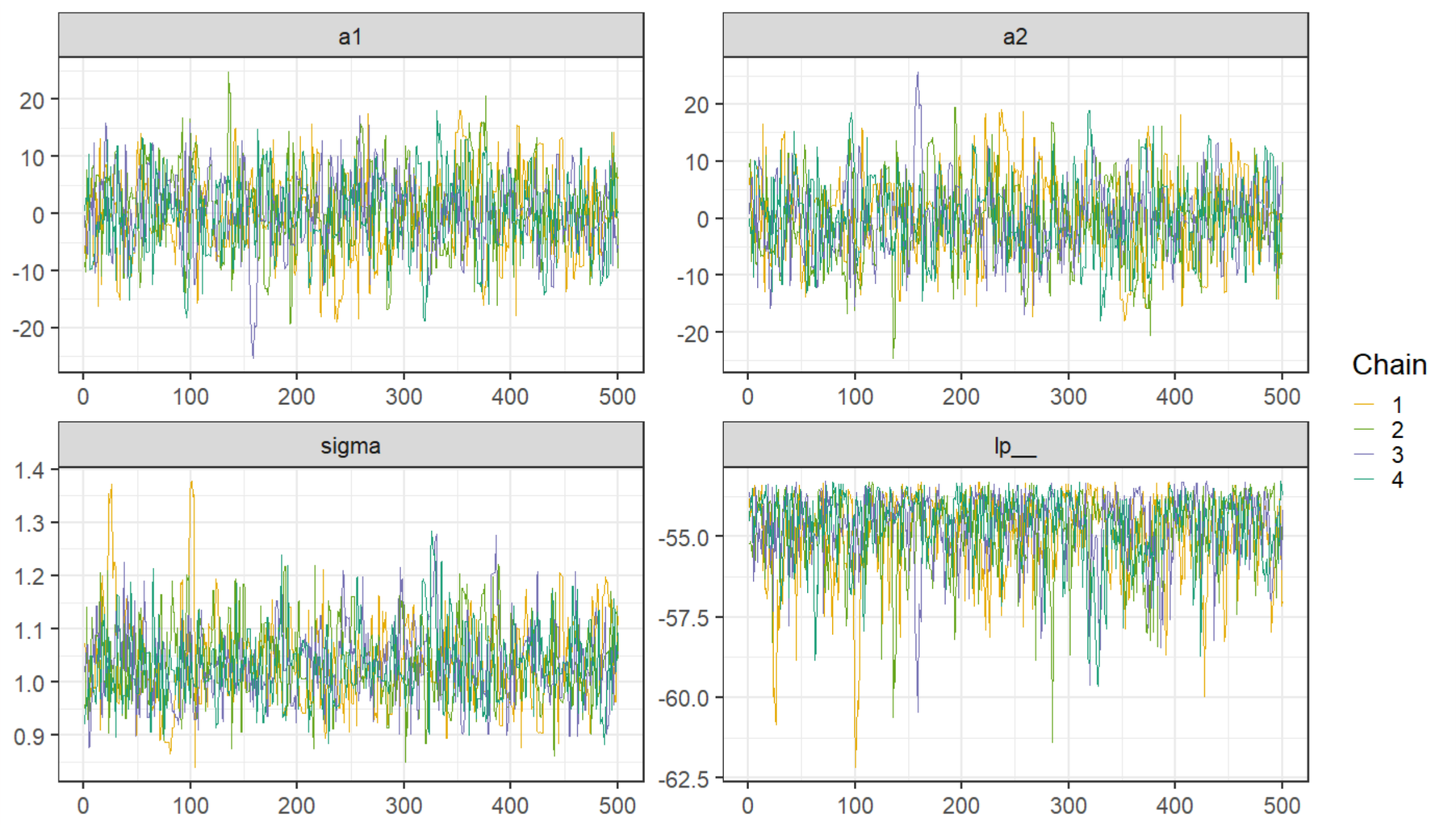
Pair-Correlation Plots

```
color_scheme_set("blue")
reg_mdl_fit <- mdl_reg@stanfit
mcmc_pairs(reg_mdl_fit, off_diag_args = list(size = 1, alpha = 0.05))
```



MCMC Trace Plots

```
color_scheme_set("brewer-Dark2")
mcmc_trace(reg_mdl_fit)
```



MCMC Trace-Rank Plots

```
mcmc_rank_overlay(reg_mdl_fit)
```