aneur: comparando stan y cgam

https://mc-stan.org/docs/2_18/stan-users-guide/ordered-logistic-section.html

1. Aortic Aneurysm Progression Data

This dataset contains longitudinal measurements of grades of aortic aneurysms, measured by ultrasound examination of the diameter of the aorta.

A data frame containing 4337 rows, with each row corresponding to an ultrasound scan from one of 838 men over 65 years of age.

- ptnum (numeric) Patient identification number
- age (numeric) Recipient age at examination (years)
- diam (numeric) Aortic diameter
- state (numeric) State of aneurysm.

The states represent successive degrees of aneurysm severity, as indicated by the aortic diameter.

- State 1 Aneurysm-free < 30 cm
- State 2 Mild aneurysm 30-44 cm
- State 3 Moderate aneurysm 45-54 cm
- State 4 Severe aneurysm > 55 cm

683 of these men were aneurysm-free at age 65 and were re-screened every two years. The remaining men were aneurysmal at entry and had successive screens with frequency depending on the state of the aneurysm. Severe aneurysms are repaired by surgery.

```
data(aneur)
attach(aneur)
head(aneur)
```

```
##   ptnum      age diam state
## 1     1 60.00000   29     1
## 2     1 65.47671   29     1
## 3     1 67.50411   29     1
## 4     1 70.04384   29     1
## 5     1 72.07671   29     1
## 6     1 74.08767   29     1
```

```
tail(aneur)
```

```
##      ptnum      age diam state
## 4332   838 73.40822   43     2
## 4333   838 73.61644   43     2
## 4334   838 73.87671   42     2
## 4335   838 74.05753   43     2
## 4336   838 74.31507   41     2
## 4337   838 74.56712   40     2
```

```
#help(aneur)
dim(aneur)
```

```
## [1] 4337    4
```

```
(N = n_distinct(aneur$ptnum))   # subjects
```

```
## [1] 838
```

```
(K = max(table(aneur$ptnum)))   # times
```

```
## [1] 21
```

```
table(table(aneur$ptnum))
```

```
##
##   2   3   4   5   6   7   8   9  10  11  12  14  15  16  17  18  19  21
## 121 107  99  96 260  97  12  12   9   5   2   5   5   3   1   2   1   1
```

```
J = 4   # categories
Y_diam = array(NA,dim=c(N,K))
Y_state = array(NA,dim=c(N,K))
X_age = array(NA,dim=c(N,K))
Ki = table(aneur$ptnum)
Ni = c(0,cumsum(Ki))+1
for(i in 1:N){
    aneur_i = aneur[aneur$ptnum==i,]
    for(k in 1:Ki[i]){
        Y_diam[i,k] = aneur_i$diam[k]
        Y_state[i,k] = aneur_i$state[k]
        X_age[i,k] = aneur_i$age[k]
    }
}
```

```
(Y_diam[11:18,1:8])
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]   29   29   29   29   29   29   29   NA
## [2,]   29   29   29   29   29   29   29   NA
```

```
## [3,]   29   29   29   29   29   29   29   NA
## [4,]   29   29   NA   NA   NA   NA   NA   NA
## [5,]   29   29   29   29   29   29   29   NA
## [6,]   29   29   29   29   29   29   29   NA
## [7,]   29   29   29   29   NA   NA   NA   NA
## [8,]   29   29   34   NA   NA   NA   NA   NA
```

```
(Y_state[11:18,1:8])
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    1    1    1    1    1    1    1   NA
## [2,]    1    1    1    1    1    1    1   NA
## [3,]    1    1    1    1    1    1    1   NA
## [4,]    1    1   NA   NA   NA   NA   NA   NA
## [5,]    1    1    1    1    1    1    1   NA
## [6,]    1    1    1    1    1    1    1   NA
## [7,]    1    1    1    1   NA   NA   NA   NA
## [8,]    1    1    2   NA   NA   NA   NA   NA
```

```
(X_age[11:18,1:8])
```

```
##      [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7] [,8]
## [1,]   60 65.45205 67.45205 69.92877 72.01096 74.01096 76.00000   NA
## [2,]   60 65.44932 67.46301 69.92603 71.96986 73.96986 75.92055   NA
## [3,]   60 65.45753 67.44658 69.92329 71.96712 73.96712 75.91781   NA
## [4,]   60 65.44384       NA       NA       NA       NA       NA   NA
## [5,]   60 65.43836 67.42192 69.93699 71.94247 73.94247 75.89315   NA
## [6,]   60 65.40822 67.40822 70.04932 72.07123 74.06575 76.09041   NA
## [7,]   60 65.38082 67.38082 70.02192       NA       NA       NA   NA
## [8,]   60 65.47123 67.47123       NA       NA       NA       NA   NA
```

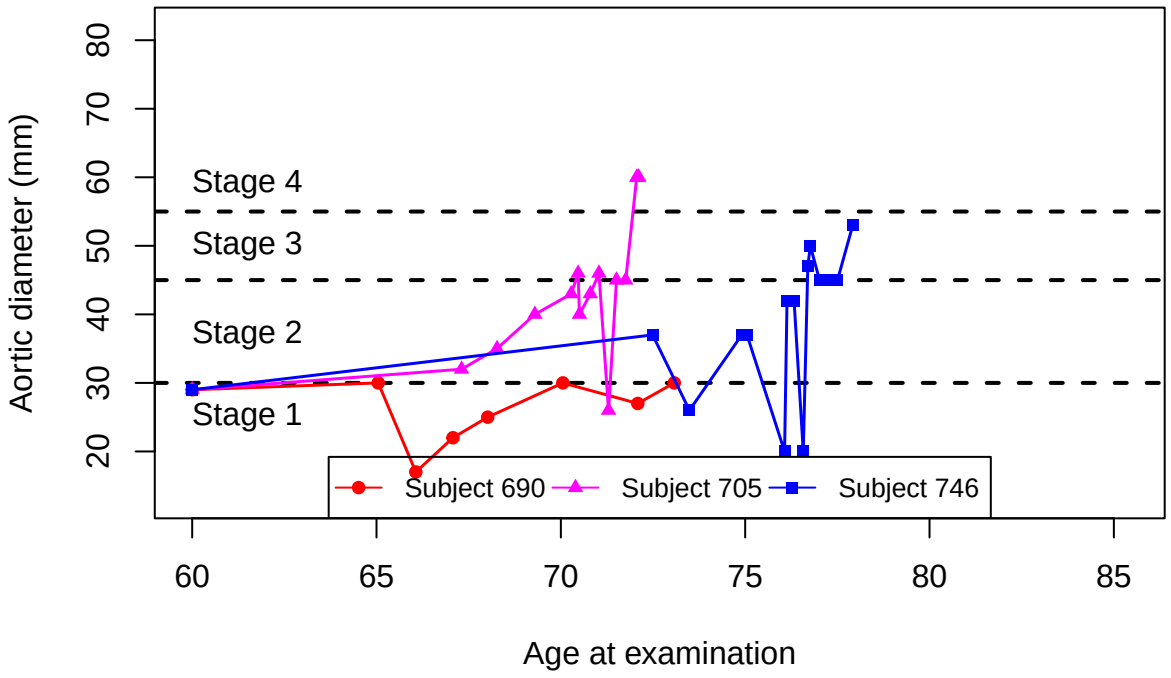
```
(Ki[11:18])
```

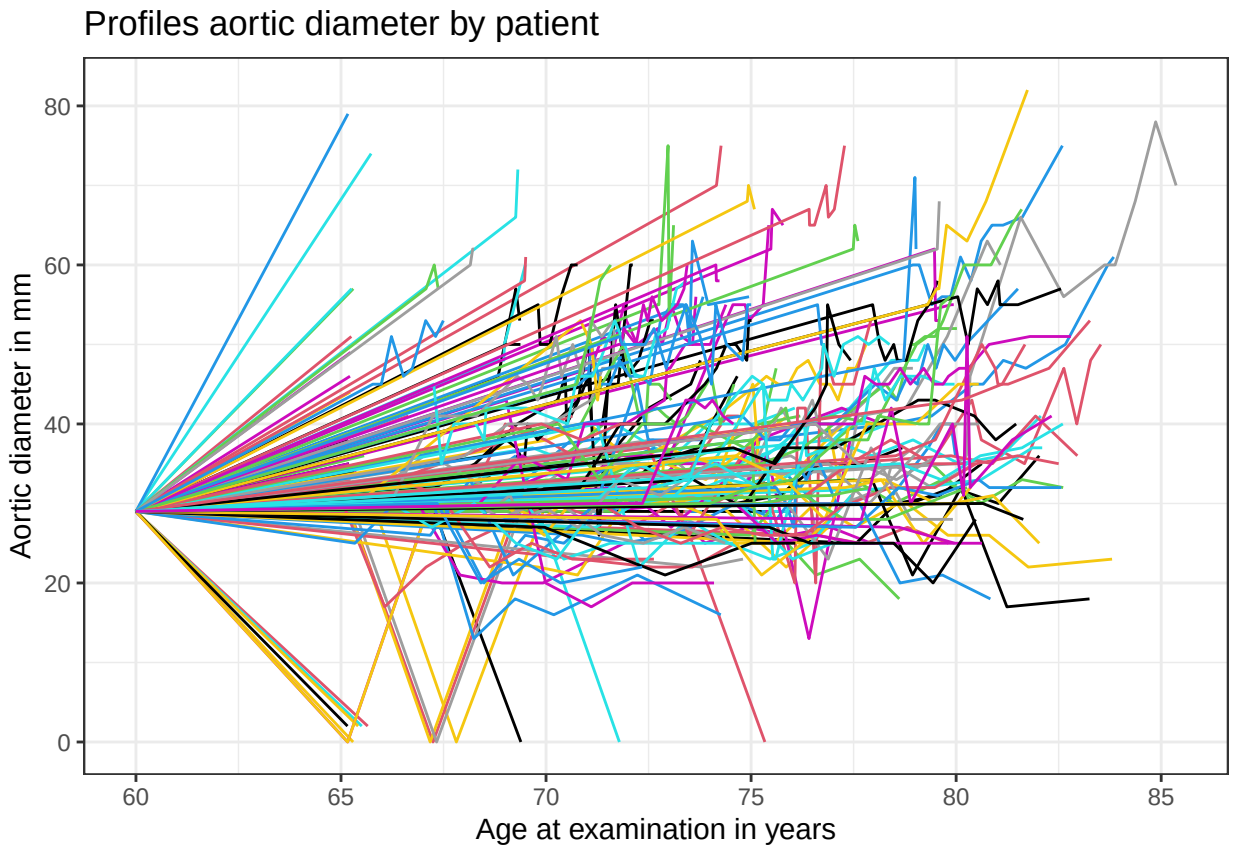
```
##
## 11 12 13 14 15 16 17 18
##  7  7  7  2  7  7  4  3
```

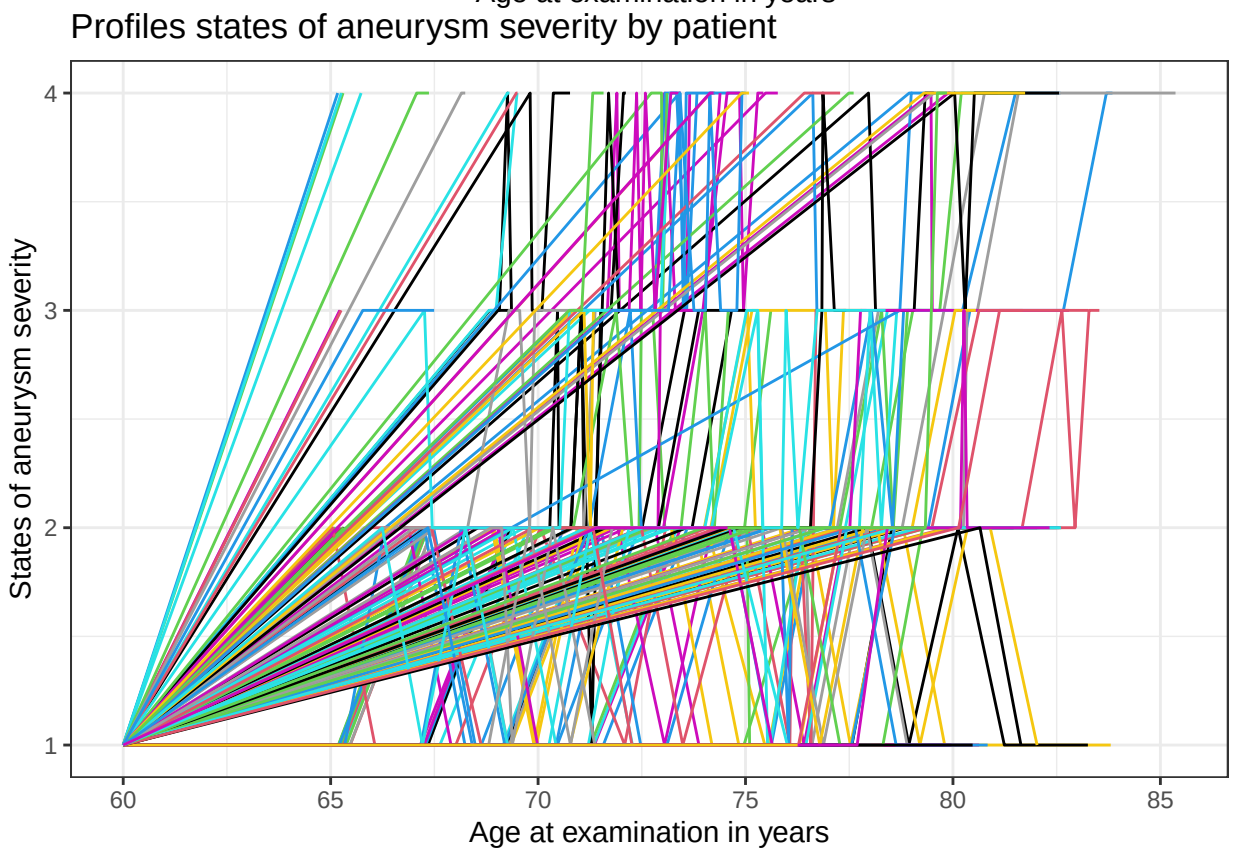
```
### Considering only data having more than one screen (state>1)
idx2 = c()
for(i in 1:N){
  if( sum(Y_state[i,1:Ki[i]])>Ki[i]){
    idx2 = c(idx2,i)
  }
}
Y2_diam = Y_diam[idx2,]
Y2_state = Y_state[idx2,]
X2_age = X_age[idx2,]
N2 = length(idx2)
Ki2 = Ki[idx2]

### Considering only data having more than one screen (diam!=29, or diam<29 & dim>29)
```

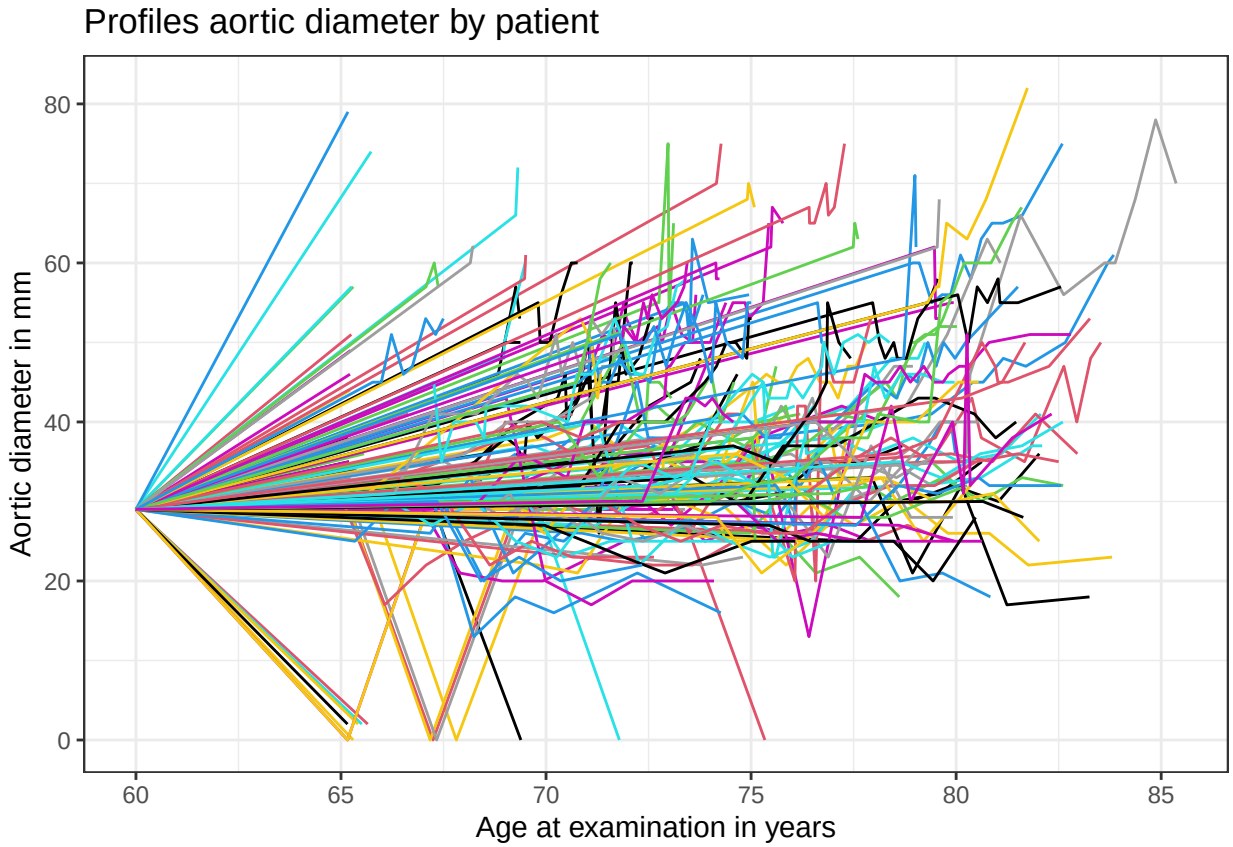
```
idx3 = c()
for(i in 1:N){
  if( min(Y_diam[i,1:Ki[i]])!=max(Y_diam[i,1:Ki[i]])){
    idx3 = c(idx3,i)
  }
}
Y3_diam = Y_diam[idx3,]
Y3_state = Y_state[idx3,]
X3_age = X_age[idx3,]
N3 = length(idx3)
Ki3 = Ki[idx3]

aneur2 = aneur%>%filter(aneur$ptnum%in%idx2)
aneur3 = aneur%>%filter(aneur$ptnum%in%idx3)
### Creo que es mejor trabajar con aneur3
```

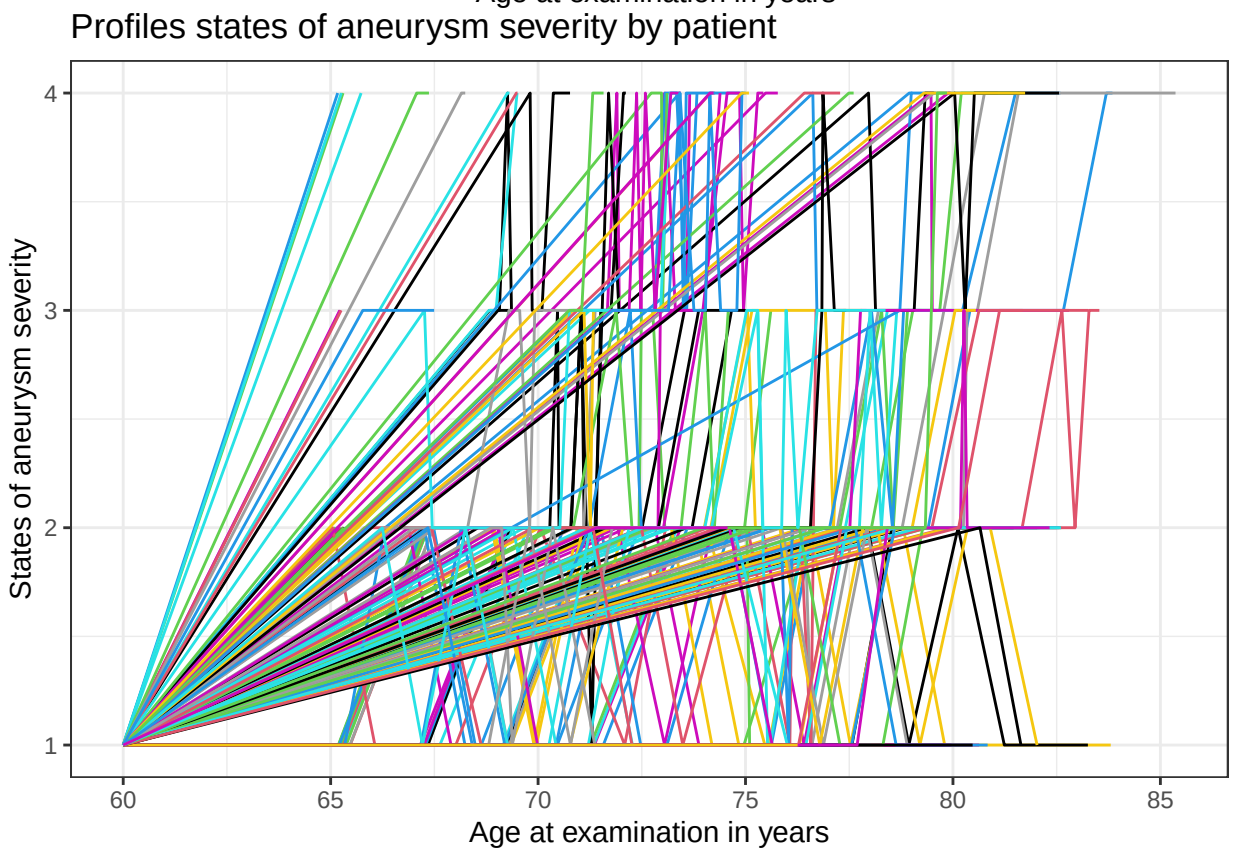
```
##
##  67  80 119
##   6   6   6
```




Profiles aortic diameter by patient
Aortic diameter in mm
0
20
40
60
80
60
65
70
75
80
85
Age at examination in years


Profiles states of aneurysm severity by patient
States of aneurysm severity
1
2
3
4
60
65
70
75
80
85
Age at examination in years





La variable respuesta puede ser continua (''diam'') u ordinal (''state''), y la unica covariable es la edad

("age") \

$$diam_{it} \quad = \quad \beta_0 + f_1(age_{it}) + b_{0i} + age_{it} \times b_{1i} + \varepsilon_{it}, \qquad b_i \sim \mathrm{N}(0, \psi), \quad \varepsilon_i \sim \mathrm{N}(0, \Lambda\sigma^2),$$

where f_1 is a non-decreasing smoothing function and $b_{1i} > 0$.

Quiza solo debemos considerar intercepto fijo, pero NO intercepto aleatorio, y SI pendiente aleatorio

$$diam_{it} \quad = \quad \beta_0 + f_1(age_{it}) + age_{it} \times b_{1i} + \varepsilon_{it}, \qquad b_{1i} \sim \mathrm{N}(0, \psi), \quad \varepsilon_i \sim \mathrm{N}(0, \Lambda\sigma^2),$$

The ordinal response $state_{it}$ is modelled in terms of the cumulative probabilities $P(state_{it} \leq j|b_i)$ by using the proportional odds model,

$$P(state_{it} \leq j|b_i) \quad = \quad \eta_{it,j},$$

subject to

$$\eta_{it,j} \quad = \quad \kappa_j + \beta_0 + f_1(age_{it}) + age_{it} \times b_{1i}, \qquad b_{1i} \sim \mathrm{N}(0, \psi),$$

where the constraints are such that f_1 is a non-decreasing smoothing function and $b_{1i} > 0$, and for the breakpoints $\kappa_j < \kappa_{j+1}$ with $j = 1, 2$.

```
y = aneur3$state
y_fact = factor(aneur3$state)
x1 = aneur3$age -60
x2 = aneur3$age -60
id = as.numeric(as.factor(aneur3$ptnum))
id_fact = as.factor(aneur3$ptnum)

n = length(y)
N = n_distinct(id)
Ni = c(0,cumsum(table(id)))+1
k1 = 3 #
k2 = 3 #
knots1 = quantile(x1, c(0.50))
knots2 = quantile(x2, c(0.50))
```

2. Generar la matriz diseño X para los B-splines

Note que $f(x)$ se representa como:

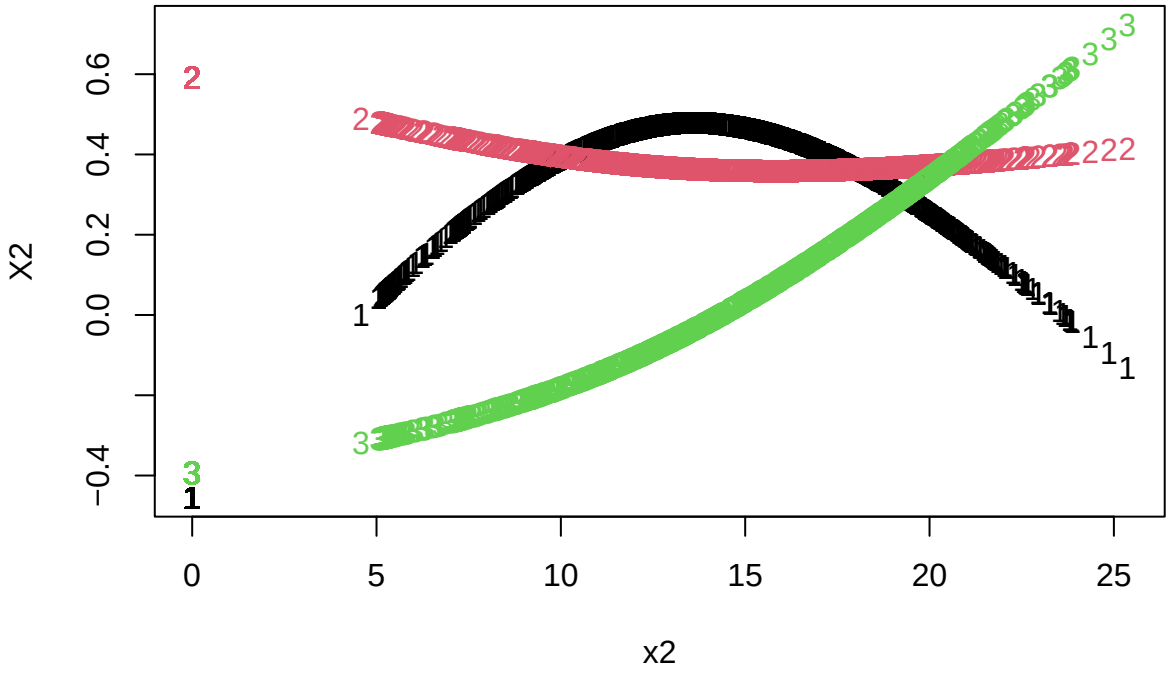
$$\begin{aligned} f(x) &= f_1(x_1) \\ &= \sum_{j=1}^{h_1} \beta_{1j} I_{1j}(x) \end{aligned}$$

para β_{1j} parámetros desconocidos, y para los $I_{1j}(x)$ se utilizar'an I-splines y B-splines.

El número de knots se elige lo suficientemente grande para evitar **over-smoothing**, pero lo suficientemente pequeño para evitar excesivo costo computacional.

El número de *knots* K es considerado a priori.

```
# Generate a basis matrix for Natural Cubic Splines
X2 <- ns(x = x2, knots = knots2, intercept = TRUE)
###X2 = (X2-mean(X2))/sd(X2)
matplot(x2, X2)
```

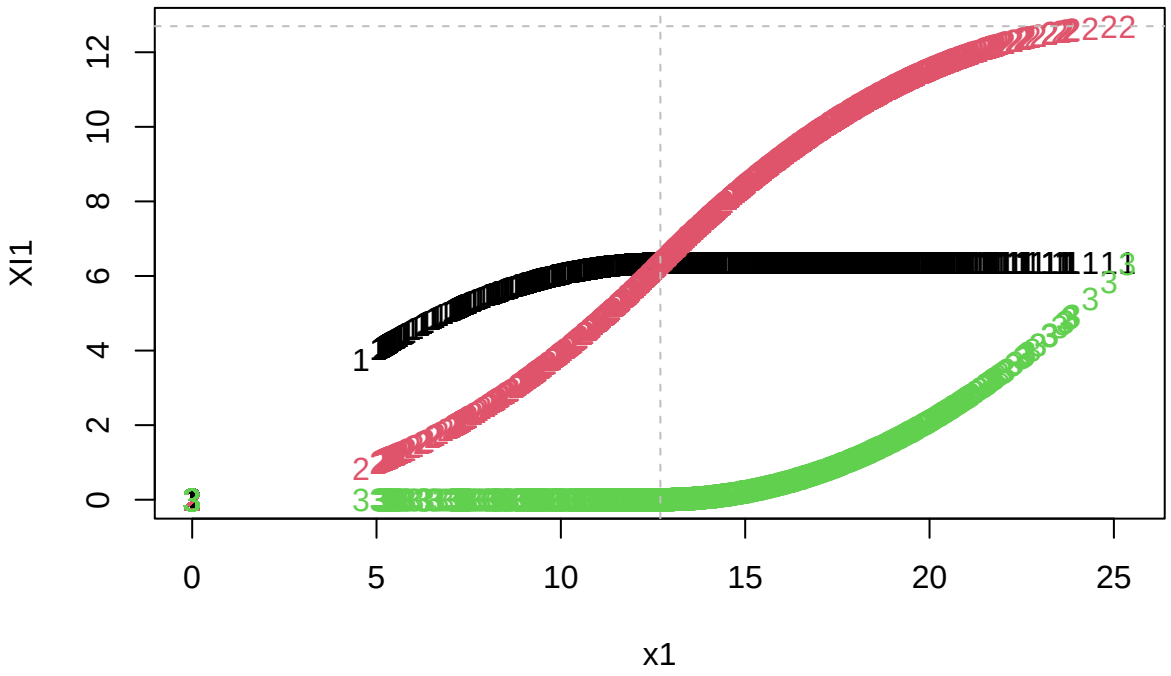


3. Generar la matriz diseño $XI1$ para los I-splines

$$f_1(x_1) \quad \sum_{j=1}^{h_1} \beta_{1j} I_{1j}(x_1)$$

$$I_{1j}(x_1) \quad = \quad \int_{x_0}^{x_1} B_{1j}(u) d_u$$

```
### ibs: integrated basis splines
### degree = 3 cubic splines
XI1 <- ibs(x1, knots = knots1, degree = 1, intercept = TRUE)
###XI1 = (XI1-mean(XI1))/sd(XI1)
matplot(x1, XI1)
abline(v = knots1, h = knots1, lty = 2, col = "gray")
```



4. Definir la penalización $S1$ y $S2$

La flexibilidad ajustada de f es controlada por K, a través de una penalización cuadrática de la forma:

$$\sum_j \lambda_j \beta^T S_j \beta$$

donde los S_j son matrices de coeficientes conocidos, y los λ_j son parámetros de suavizamiento estimados.

```
#Este es el código que produce la matriz de diferenciación.
#No es el óptimo, pero funciona.
#"k" es el número de b-splines y
#"d" el orden de la diferenciación.
#Adjunto el artículo donde discutimos esto (página 7).

diffMatrix = function(k, d = 2){
  if( (d<1) || (d %% 1 != 0) )stop("d must be a positive integer value");
  if( (k<1) || (k %% 1 != 0) )stop("k must be a positive integer value");
  if(d >= k)stop("d must be lower than k");
  out = diag(k);
  for(i in 1:d){
    out = diff(out);
  }
  return(out)
}
(D1 = diffMatrix(k=k1, d=2))
```

```
##      [,1] [,2] [,3]
## [1,]    1   -2    1
```

```
(D2 = diffMatrix(k=k2, d=2))
```

```
##      [,1] [,2] [,3]
## [1,]    1   -2    1
```

```
(S1 = t(D1)%*%D1 + diag(1,k1)*10e-4)
```

```
##        [,1]   [,2]   [,3]
## [1,]  1.001 -2.000  1.000
## [2,] -2.000  4.001 -2.000
## [3,]  1.000 -2.000  1.001
```

```
(S2 = t(D2)%*%D2 + diag(1,k2)*10e-4)
```

```
##        [,1]   [,2]   [,3]
## [1,]  1.001 -2.000  1.000
## [2,] -2.000  4.001 -2.000
## [3,]  1.000 -2.000  1.001
```

5. Lineal NO restricciones

5.1 Lineal fit without constraints:

For a fit without constraints:

```
set.seed(123)
# mod.lm <- clm(y_fact ~ x1 )
mod.lm <- polr(y_fact ~ x1 )
summary(mod.lm)
```

```
## Call:
## polr(formula = y_fact ~ x1)
##
## Coefficients:
##     Value Std. Error t value
## x1 0.1984   0.009568   20.74
##
## Intercepts:
##     Value   Std. Error t value
## 1|2  1.5692  0.1254    12.5146
## 2|3  3.6805  0.1531    24.0384
## 3|4  5.0534  0.1745    28.9524
##
## Residual Deviance: 2986.832
## AIC: 2994.832
```

```
datos.lin <- list( y = y ,
                   n = length(y) ,
                   x1 = x1 )
param.lin = c("b1", "kappa")
```

```
stancode <- readLines("jagam_10_aneur_ordinal_lin_non.stan")
# writeLines(stancode)
```

```
mod <- stan_model(model_code=stancode,
                                    verbose=TRUE)
```

```
##
## TRANSLATING MODEL '20a961012d9d9a04013dc735a1515910' FROM Stan CODE TO C++ CODE NOW.
## successful in parsing the Stan model '20a961012d9d9a04013dc735a1515910'.
## OS: x86_64, darwin17.0; rstan: 2.21.3; Rcpp: 1.0.10; inline: 0.3.19
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -mmacosx-version-min=10.13 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/inclu
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/include/Eigen/src/Core/util/
```

```
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/include/Eigen/Core:96:10: fa
## #include <complex>
##          ^~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##  >> setting environment variables:
## PKG_LIBS =  '/Library/Frameworks/R.framework/Versions/4.1/Resources/library/rstan/lib//libStanServic
## PKG_CPPFLAGS =   -I"/Library/Frameworks/R.framework/Versions/4.1/Resources/library/Rcpp/include/"  -I
##  >> Program source :
##
##    1 :
##    2 : // includes from the plugin
##    3 : // [[Rcpp::plugins(cpp14)]]
##    4 :
##    5 :
##    6 : // user includes
##    7 : #include <Rcpp.h>
##    8 : #include <rstan/io/rlist_ref_var_context.hpp>
##    9 : #include <rstan/io/r_ostream.hpp>
##   10 : #include <rstan/stan_args.hpp>
##   11 : #include <boost/integer/integer_log2.hpp>
##   12 : // Code generated by Stan version 2.21.0
##   13 :
##   14 : #include <stan/model/model_header.hpp>
##   15 :
##   16 : namespace model399138178fc_20a961012d9d9a04013dc735a1515910_namespace {
##   17 :
##   18 : using std::istream;
##   19 : using std::string;
##   20 : using std::stringstream;
##   21 : using std::vector;
##   22 : using stan::io::dump;
##   23 : using stan::math::lgamma;
##   24 : using stan::model::prob_grad;
##   25 : using namespace stan::math;
##   26 :
##   27 : static int current_statement_begin__;
##   28 :
##   29 : stan::io::program_reader prog_reader__() {
##   30 :     stan::io::program_reader reader;
##   31 :     reader.add_event(0, 0, "start", "model399138178fc_20a961012d9d9a04013dc735a1515910");
##   32 :     reader.add_event(39, 37, "end", "model399138178fc_20a961012d9d9a04013dc735a1515910");
##   33 :     return reader;
##   34 : }
##   35 :
##   36 : class model399138178fc_20a961012d9d9a04013dc735a1515910
##   37 :   : public stan::model::model_base_crtp<model399138178fc_20a961012d9d9a04013dc735a1515910> {
##   38 : private:
```

```
##   39 :         int n;
##   40 :         std::vector<int> y;
##   41 :         std::vector<double> x1;
##   42 : public:
##   43 :     model399138178fc_20a961012d9d9a04013dc735a1515910(rstan::io::rlist_ref_var_context& conte
##   44 :         std::ostream* pstream__ = 0)
##   45 :         : model_base_crtp(0) {
##   46 :         ctor_body(context__, 0, pstream__);
##   47 :     }
##   48 :
##   49 :     model399138178fc_20a961012d9d9a04013dc735a1515910(stan::io::var_context& context__,
##   50 :         unsigned int random_seed__,
##   51 :         std::ostream* pstream__ = 0)
##   52 :         : model_base_crtp(0) {
##   53 :         ctor_body(context__, random_seed__, pstream__);
##   54 :     }
##   55 :
##   56 :     void ctor_body(stan::io::var_context& context__,
##   57 :                    unsigned int random_seed__,
##   58 :                    std::ostream* pstream__) {
##   59 :         typedef double local_scalar_t__;
##   60 :
##   61 :         boost::ecuyer1988 base_rng__ =
##   62 :           stan::services::util::create_rng(random_seed__, 0);
##   63 :         (void) base_rng__;  // suppress unused var warning
##   64 :
##   65 :         current_statement_begin__ = -1;
##   66 :
##   67 :         static const char* function__ = "model399138178fc_20a961012d9d9a04013dc735a1515910_na
##   68 :         (void) function__;  // dummy to suppress unused var warning
##   69 :         size_t pos__;
##   70 :         (void) pos__;  // dummy to suppress unused var warning
##   71 :         std::vector<int> vals_i__;
##   72 :         std::vector<double> vals_r__;
##   73 :         local_scalar_t__ DUMMY_VAR__(std::numeric_limits<double>::quiet_NaN());
##   74 :         (void) DUMMY_VAR__;  // suppress unused var warning
##   75 :
##   76 :         try {
##   77 :             // initialize data block variables from context__
##   78 :             current_statement_begin__ = 8;
##   79 :             context__.validate_dims("data initialization", "n", "int", context__.to_vec());
##   80 :             n = int(0);
##   81 :             vals_i__ = context__.vals_i("n");
##   82 :             pos__ = 0;
##   83 :             n = vals_i__[pos__++];
##   84 :             check_greater_or_equal(function__, "n", n, 0);
##   85 :
##   86 :             current_statement_begin__ = 9;
##   87 :             validate_non_negative_index("y", "n", n);
##   88 :             context__.validate_dims("data initialization", "y", "int", context__.to_vec(n));
##   89 :             y = std::vector<int>(n, int(0));
##   90 :             vals_i__ = context__.vals_i("y");
##   91 :             pos__ = 0;
##   92 :             size_t y_k_0_max__ = n;
```

```
##   93 :             for (size_t k_0__ = 0; k_0__ < y_k_0_max__; ++k_0__) {
##   94 :                 y[k_0__] = vals_i__[pos__++];
##   95 :             }
##   96 :             size_t y_i_0_max__ = n;
##   97 :             for (size_t i_0__ = 0; i_0__ < y_i_0_max__; ++i_0__) {
##   98 :                 check_greater_or_equal(function__, "y[i_0__]", y[i_0__], 1);
##   99 :                 check_less_or_equal(function__, "y[i_0__]", y[i_0__], 4);
##  100 :             }
##  101 :
##  102 :             current_statement_begin__ = 10;
##  103 :             validate_non_negative_index("x1", "n", n);
##  104 :             context__.validate_dims("data initialization", "x1", "double", context__.to_vec(n
##  105 :             x1 = std::vector<double>(n, double(0));
##  106 :             vals_r__ = context__.vals_r("x1");
##  107 :             pos__ = 0;
##  108 :             size_t x1_k_0_max__ = n;
##  109 :             for (size_t k_0__ = 0; k_0__ < x1_k_0_max__; ++k_0__) {
##  110 :                 x1[k_0__] = vals_r__[pos__++];
##  111 :             }
##  112 :
##  113 :
##  114 :             // initialize transformed data variables
##  115 :             // execute transformed data statements
##  116 :
##  117 :             // validate transformed data
##  118 :
##  119 :             // validate, set parameter ranges
##  120 :             num_params_r__ = 0U;
##  121 :             param_ranges_i__.clear();
##  122 :             current_statement_begin__ = 14;
##  123 :             validate_non_negative_index("kappa", "3", 3);
##  124 :             num_params_r__ += 3;
##  125 :             current_statement_begin__ = 15;
##  126 :             num_params_r__ += 1;
##  127 :         } catch (const std::exception& e) {
##  128 :             stan::lang::rethrow_located(e, current_statement_begin__, prog_reader__());
##  129 :             // Next line prevents compiler griping about no return
##  130 :             throw std::runtime_error("*** IF YOU SEE THIS, PLEASE REPORT A BUG ***");
##  131 :         }
##  132 :     }
##  133 :
##  134 :     ~model399138178fc_20a961012d9d9a04013dc735a1515910() { }
##  135 :
##  136 :
##  137 :     void transform_inits(const stan::io::var_context& context__,
##  138 :                          std::vector<int>& params_i__,
##  139 :                          std::vector<double>& params_r__,
##  140 :                          std::ostream* pstream__) const {
##  141 :         typedef double local_scalar_t__;
##  142 :         stan::io::writer<double> writer__(params_r__, params_i__);
##  143 :         size_t pos__;
##  144 :         (void) pos__; // dummy call to supress warning
##  145 :         std::vector<double> vals_r__;
##  146 :         std::vector<int> vals_i__;
```

```
##  147 :
##  148 :         current_statement_begin__ = 14;
##  149 :         if (!(context__.contains_r("kappa")))
##  150 :             stan::lang::rethrow_located(std::runtime_error(std::string("Variable kappa missing
##  151 :         vals_r__ = context__.vals_r("kappa");
##  152 :         pos__ = 0U;
##  153 :         validate_non_negative_index("kappa", "3", 3);
##  154 :         context__.validate_dims("parameter initialization", "kappa", "vector_d", context__.to_
##  155 :         Eigen::Matrix<double, Eigen::Dynamic, 1> kappa(3);
##  156 :         size_t kappa_j_1_max__ = 3;
##  157 :         for (size_t j_1__ = 0; j_1__ < kappa_j_1_max__; ++j_1__) {
##  158 :             kappa(j_1__) = vals_r__[pos__++];
##  159 :         }
##  160 :         try {
##  161 :             writer__.ordered_unconstrain(kappa);
##  162 :         } catch (const std::exception& e) {
##  163 :             stan::lang::rethrow_located(std::runtime_error(std::string("Error transforming var
##  164 :         }
##  165 :
##  166 :         current_statement_begin__ = 15;
##  167 :         if (!(context__.contains_r("b1")))
##  168 :             stan::lang::rethrow_located(std::runtime_error(std::string("Variable b1 missing"))
##  169 :         vals_r__ = context__.vals_r("b1");
##  170 :         pos__ = 0U;
##  171 :         context__.validate_dims("parameter initialization", "b1", "double", context__.to_vec())
##  172 :         double b1(0);
##  173 :         b1 = vals_r__[pos__++];
##  174 :         try {
##  175 :             writer__.scalar_unconstrain(b1);
##  176 :         } catch (const std::exception& e) {
##  177 :             stan::lang::rethrow_located(std::runtime_error(std::string("Error transforming var
##  178 :         }
##  179 :
##  180 :         params_r__ = writer__.data_r();
##  181 :         params_i__ = writer__.data_i();
##  182 :     }
##  183 :
##  184 :     void transform_inits(const stan::io::var_context& context,
##  185 :                          Eigen::Matrix<double, Eigen::Dynamic, 1>& params_r,
##  186 :                          std::ostream* pstream__) const {
##  187 :       std::vector<double> params_r_vec;
##  188 :       std::vector<int> params_i_vec;
##  189 :       transform_inits(context, params_i_vec, params_r_vec, pstream__);
##  190 :       params_r.resize(params_r_vec.size());
##  191 :       for (int i = 0; i < params_r.size(); ++i)
##  192 :         params_r(i) = params_r_vec[i];
##  193 :     }
##  194 :
##  195 :
##  196 :     template <bool propto__, bool jacobian__, typename T__>
##  197 :     T__ log_prob(std::vector<T__>& params_r__,
##  198 :                  std::vector<int>& params_i__,
##  199 :                  std::ostream* pstream__ = 0) const {
##  200 :
```

```
##  201 :         typedef T__ local_scalar_t__;
##  202 :
##  203 :         local_scalar_t__ DUMMY_VAR__(std::numeric_limits<double>::quiet_NaN());
##  204 :         (void) DUMMY_VAR__;  // dummy to suppress unused var warning
##  205 :
##  206 :         T__ lp__(0.0);
##  207 :         stan::math::accumulator<T__> lp_accum__;
##  208 :         try {
##  209 :             stan::io::reader<local_scalar_t__> in__(params_r__, params_i__);
##  210 :
##  211 :             // model parameters
##  212 :             current_statement_begin__ = 14;
##  213 :             Eigen::Matrix<local_scalar_t__, Eigen::Dynamic, 1> kappa;
##  214 :             (void) kappa;  // dummy to suppress unused var warning
##  215 :             if (jacobian__)
##  216 :                 kappa = in__.ordered_constrain(3, lp__);
##  217 :             else
##  218 :                 kappa = in__.ordered_constrain(3);
##  219 :
##  220 :             current_statement_begin__ = 15;
##  221 :             local_scalar_t__ b1;
##  222 :             (void) b1;  // dummy to suppress unused var warning
##  223 :             if (jacobian__)
##  224 :                 b1 = in__.scalar_constrain(lp__);
##  225 :             else
##  226 :                 b1 = in__.scalar_constrain();
##  227 :
##  228 :             // transformed parameters
##  229 :             current_statement_begin__ = 19;
##  230 :             validate_non_negative_index("mu", "n", n);
##  231 :             Eigen::Matrix<local_scalar_t__, Eigen::Dynamic, 1> mu(n);
##  232 :             stan::math::initialize(mu, DUMMY_VAR__);
##  233 :             stan::math::fill(mu, DUMMY_VAR__);
##  234 :
##  235 :             // transformed parameters block statements
##  236 :             current_statement_begin__ = 20;
##  237 :             for (int i = 1; i <= n; ++i) {
##  238 :
##  239 :                 current_statement_begin__ = 21;
##  240 :                 stan::model::assign(mu,
##  241 :                             stan::model::cons_list(stan::model::index_uni(i), stan::model::nil
##  242 :                             (get_base1(x1, i, "x1", 1) * b1),
##  243 :                             "assigning variable mu");
##  244 :             }
##  245 :
##  246 :             // validate transformed parameters
##  247 :             const char* function__ = "validate transformed params";
##  248 :             (void) function__;  // dummy to suppress unused var warning
##  249 :
##  250 :             current_statement_begin__ = 19;
##  251 :             size_t mu_j_1_max__ = n;
##  252 :             for (size_t j_1__ = 0; j_1__ < mu_j_1_max__; ++j_1__) {
##  253 :                 if (stan::math::is_uninitialized(mu(j_1__))) {
##  254 :                     std::stringstream msg__;
```

```
##  255 :                     msg__ << "Undefined transformed parameter: mu" << "(" << j_1__ << ")";
##  256 :                     stan::lang::rethrow_located(std::runtime_error(std::string("Error initial
##  257 :                 }
##  258 :             }
##  259 :
##  260 :             // model body
##  261 :
##  262 :             current_statement_begin__ = 26;
##  263 :             lp_accum__.add(normal_log<propto__>(b1, 0, 9.9e+06));
##  264 :             current_statement_begin__ = 27;
##  265 :             for (int i = 1; i <= n; ++i) {
##  266 :
##  267 :                 current_statement_begin__ = 28;
##  268 :                 lp_accum__.add(ordered_logistic_log<propto__>(get_base1(y, i, "y", 1), get_bas
##  269 :             }
##  270 :
##  271 :         } catch (const std::exception& e) {
##  272 :             stan::lang::rethrow_located(e, current_statement_begin__, prog_reader__());
##  273 :             // Next line prevents compiler griping about no return
##  274 :             throw std::runtime_error("*** IF YOU SEE THIS, PLEASE REPORT A BUG ***");
##  275 :         }
##  276 :
##  277 :         lp_accum__.add(lp__);
##  278 :         return lp_accum__.sum();
##  279 :
##  280 :     } // log_prob()
##  281 :
##  282 :     template <bool propto, bool jacobian, typename T_>
##  283 :     T_ log_prob(Eigen::Matrix<T_,Eigen::Dynamic,1>& params_r,
##  284 :                 std::ostream* pstream = 0) const {
##  285 :       std::vector<T_> vec_params_r;
##  286 :       vec_params_r.reserve(params_r.size());
##  287 :       for (int i = 0; i < params_r.size(); ++i)
##  288 :         vec_params_r.push_back(params_r(i));
##  289 :       std::vector<int> vec_params_i;
##  290 :       return log_prob<propto,jacobian,T_>(vec_params_r, vec_params_i, pstream);
##  291 :     }
##  292 :
##  293 :
##  294 :     void get_param_names(std::vector<std::string>& names__) const {
##  295 :         names__.resize(0);
##  296 :         names__.push_back("kappa");
##  297 :         names__.push_back("b1");
##  298 :         names__.push_back("mu");
##  299 :         names__.push_back("log_lik");
##  300 :     }
##  301 :
##  302 :
##  303 :     void get_dims(std::vector<std::vector<size_t> >& dimss__) const {
##  304 :         dimss__.resize(0);
##  305 :         std::vector<size_t> dims__;
##  306 :         dims__.resize(0);
##  307 :         dims__.push_back(3);
##  308 :         dimss__.push_back(dims__);
```

```
##  309 :         dims__.resize(0);
##  310 :         dimss__.push_back(dims__);
##  311 :         dims__.resize(0);
##  312 :         dims__.push_back(n);
##  313 :         dimss__.push_back(dims__);
##  314 :         dims__.resize(0);
##  315 :         dims__.push_back(n);
##  316 :         dimss__.push_back(dims__);
##  317 :     }
##  318 :
##  319 :     template <typename RNG>
##  320 :     void write_array(RNG& base_rng__,
##  321 :                      std::vector<double>& params_r__,
##  322 :                      std::vector<int>& params_i__,
##  323 :                      std::vector<double>& vars__,
##  324 :                      bool include_tparams__ = true,
##  325 :                      bool include_gqs__ = true,
##  326 :                      std::ostream* pstream__ = 0) const {
##  327 :         typedef double local_scalar_t__;
##  328 :
##  329 :         vars__.resize(0);
##  330 :         stan::io::reader<local_scalar_t__> in__(params_r__, params_i__);
##  331 :         static const char* function__ = "model399138178fc_20a961012d9d9a04013dc735a1515910_nam
##  332 :         (void) function__;  // dummy to suppress unused var warning
##  333 :
##  334 :         // read-transform, write parameters
##  335 :         Eigen::Matrix<double, Eigen::Dynamic, 1> kappa = in__.ordered_constrain(3);
##  336 :         size_t kappa_j_1_max__ = 3;
##  337 :         for (size_t j_1__ = 0; j_1__ < kappa_j_1_max__; ++j_1__) {
##  338 :             vars__.push_back(kappa(j_1__));
##  339 :         }
##  340 :
##  341 :         double b1 = in__.scalar_constrain();
##  342 :         vars__.push_back(b1);
##  343 :
##  344 :         double lp__ = 0.0;
##  345 :         (void) lp__;  // dummy to suppress unused var warning
##  346 :         stan::math::accumulator<double> lp_accum__;
##  347 :
##  348 :         local_scalar_t__ DUMMY_VAR__(std::numeric_limits<double>::quiet_NaN());
##  349 :         (void) DUMMY_VAR__;  // suppress unused var warning
##  350 :
##  351 :         if (!include_tparams__ && !include_gqs__) return;
##  352 :
##  353 :         try {
##  354 :             // declare and define transformed parameters
##  355 :             current_statement_begin__ = 19;
##  356 :             validate_non_negative_index("mu", "n", n);
##  357 :             Eigen::Matrix<double, Eigen::Dynamic, 1> mu(n);
##  358 :             stan::math::initialize(mu, DUMMY_VAR__);
##  359 :             stan::math::fill(mu, DUMMY_VAR__);
##  360 :
##  361 :             // do transformed parameters statements
##  362 :             current_statement_begin__ = 20;
```

```
##  363 :             for (int i = 1; i <= n; ++i) {
##  364 :
##  365 :                 current_statement_begin__ = 21;
##  366 :                 stan::model::assign(mu,
##  367 :                             stan::model::cons_list(stan::model::index_uni(i), stan::model::nil
##  368 :                             (get_base1(x1, i, "x1", 1) * b1),
##  369 :                             "assigning variable mu");
##  370 :             }
##  371 :
##  372 :             if (!include_gqs__ && !include_tparams__) return;
##  373 :             // validate transformed parameters
##  374 :             const char* function__ = "validate transformed params";
##  375 :             (void) function__;  // dummy to suppress unused var warning
##  376 :
##  377 :             // write transformed parameters
##  378 :             if (include_tparams__) {
##  379 :                 size_t mu_j_1_max__ = n;
##  380 :                 for (size_t j_1__ = 0; j_1__ < mu_j_1_max__; ++j_1__) {
##  381 :                     vars__.push_back(mu(j_1__));
##  382 :                 }
##  383 :             }
##  384 :             if (!include_gqs__) return;
##  385 :             // declare and define generated quantities
##  386 :             current_statement_begin__ = 33;
##  387 :             validate_non_negative_index("log_lik", "n", n);
##  388 :             Eigen::Matrix<double, Eigen::Dynamic, 1> log_lik(n);
##  389 :             stan::math::initialize(log_lik, DUMMY_VAR__);
##  390 :             stan::math::fill(log_lik, DUMMY_VAR__);
##  391 :
##  392 :             // generated quantities statements
##  393 :             current_statement_begin__ = 34;
##  394 :             for (int i = 1; i <= n; ++i) {
##  395 :
##  396 :                 current_statement_begin__ = 35;
##  397 :                 stan::model::assign(log_lik,
##  398 :                             stan::model::cons_list(stan::model::index_uni(i), stan::model::nil
##  399 :                             ordered_logistic_log(get_base1(y, i, "y", 1), get_base1(mu, i, "mu
##  400 :                             "assigning variable log_lik");
##  401 :             }
##  402 :
##  403 :             // validate, write generated quantities
##  404 :             current_statement_begin__ = 33;
##  405 :             size_t log_lik_j_1_max__ = n;
##  406 :             for (size_t j_1__ = 0; j_1__ < log_lik_j_1_max__; ++j_1__) {
##  407 :                 vars__.push_back(log_lik(j_1__));
##  408 :             }
##  409 :
##  410 :         } catch (const std::exception& e) {
##  411 :             stan::lang::rethrow_located(e, current_statement_begin__, prog_reader__());
##  412 :             // Next line prevents compiler griping about no return
##  413 :             throw std::runtime_error("*** IF YOU SEE THIS, PLEASE REPORT A BUG ***");
##  414 :         }
##  415 :     }
##  416 :
```

```
##  417 :     template <typename RNG>
##  418 :     void write_array(RNG& base_rng,
##  419 :                      Eigen::Matrix<double,Eigen::Dynamic,1>& params_r,
##  420 :                      Eigen::Matrix<double,Eigen::Dynamic,1>& vars,
##  421 :                      bool include_tparams = true,
##  422 :                      bool include_gqs = true,
##  423 :                      std::ostream* pstream = 0) const {
##  424 :       std::vector<double> params_r_vec(params_r.size());
##  425 :       for (int i = 0; i < params_r.size(); ++i)
##  426 :         params_r_vec[i] = params_r(i);
##  427 :       std::vector<double> vars_vec;
##  428 :       std::vector<int> params_i_vec;
##  429 :       write_array(base_rng, params_r_vec, params_i_vec, vars_vec, include_tparams, include_gqs
##  430 :       vars.resize(vars_vec.size());
##  431 :       for (int i = 0; i < vars.size(); ++i)
##  432 :         vars(i) = vars_vec[i];
##  433 :     }
##  434 :
##  435 :     std::string model_name() const {
##  436 :         return "model399138178fc_20a961012d9d9a04013dc735a1515910";
##  437 :     }
##  438 :
##  439 :
##  440 :     void constrained_param_names(std::vector<std::string>& param_names__,
##  441 :                                  bool include_tparams__ = true,
##  442 :                                  bool include_gqs__ = true) const {
##  443 :         std::stringstream param_name_stream__;
##  444 :         size_t kappa_j_1_max__ = 3;
##  445 :         for (size_t j_1__ = 0; j_1__ < kappa_j_1_max__; ++j_1__) {
##  446 :             param_name_stream__.str(std::string());
##  447 :             param_name_stream__ << "kappa" << '.' << j_1__ + 1;
##  448 :             param_names__.push_back(param_name_stream__.str());
##  449 :         }
##  450 :         param_name_stream__.str(std::string());
##  451 :         param_name_stream__ << "b1";
##  452 :         param_names__.push_back(param_name_stream__.str());
##  453 :
##  454 :         if (!include_gqs__ && !include_tparams__) return;
##  455 :
##  456 :         if (include_tparams__) {
##  457 :             size_t mu_j_1_max__ = n;
##  458 :             for (size_t j_1__ = 0; j_1__ < mu_j_1_max__; ++j_1__) {
##  459 :                 param_name_stream__.str(std::string());
##  460 :                 param_name_stream__ << "mu" << '.' << j_1__ + 1;
##  461 :                 param_names__.push_back(param_name_stream__.str());
##  462 :             }
##  463 :         }
##  464 :
##  465 :         if (!include_gqs__) return;
##  466 :         size_t log_lik_j_1_max__ = n;
##  467 :         for (size_t j_1__ = 0; j_1__ < log_lik_j_1_max__; ++j_1__) {
##  468 :             param_name_stream__.str(std::string());
##  469 :             param_name_stream__ << "log_lik" << '.' << j_1__ + 1;
##  470 :             param_names__.push_back(param_name_stream__.str());
```

```
##  471 :         }
##  472 :     }
##  473 :
##  474 :
##  475 :     void unconstrained_param_names(std::vector<std::string>& param_names__,
##  476 :                                    bool include_tparams__ = true,
##  477 :                                    bool include_gqs__ = true) const {
##  478 :         std::stringstream param_name_stream__;
##  479 :         size_t kappa_j_1_max__ = 3;
##  480 :         for (size_t j_1__ = 0; j_1__ < kappa_j_1_max__; ++j_1__) {
##  481 :             param_name_stream__.str(std::string());
##  482 :             param_name_stream__ << "kappa" << '.' << j_1__ + 1;
##  483 :             param_names__.push_back(param_name_stream__.str());
##  484 :         }
##  485 :         param_name_stream__.str(std::string());
##  486 :         param_name_stream__ << "b1";
##  487 :         param_names__.push_back(param_name_stream__.str());
##  488 :
##  489 :         if (!include_gqs__ && !include_tparams__) return;
##  490 :
##  491 :         if (include_tparams__) {
##  492 :             size_t mu_j_1_max__ = n;
##  493 :             for (size_t j_1__ = 0; j_1__ < mu_j_1_max__; ++j_1__) {
##  494 :                 param_name_stream__.str(std::string());
##  495 :                 param_name_stream__ << "mu" << '.' << j_1__ + 1;
##  496 :                 param_names__.push_back(param_name_stream__.str());
##  497 :             }
##  498 :         }
##  499 :
##  500 :         if (!include_gqs__) return;
##  501 :         size_t log_lik_j_1_max__ = n;
##  502 :         for (size_t j_1__ = 0; j_1__ < log_lik_j_1_max__; ++j_1__) {
##  503 :             param_name_stream__.str(std::string());
##  504 :             param_name_stream__ << "log_lik" << '.' << j_1__ + 1;
##  505 :             param_names__.push_back(param_name_stream__.str());
##  506 :         }
##  507 :     }
##  508 :
##  509 : }; // model
##  510 :
##  511 : }  // namespace
##  512 :
##  513 : typedef model399138178fc_20a961012d9d9a04013dc735a1515910_namespace::model399138178fc_20a9610
##  514 :
##  515 : #ifndef USING_R
##  516 :
##  517 : stan::model::model_base& new_model(
##  518 :         stan::io::var_context& data_context,
##  519 :         unsigned int seed,
##  520 :         std::ostream* msg_stream) {
##  521 :   stan_model* m = new stan_model(data_context, seed, msg_stream);
##  522 :   return *m;
##  523 : }
##  524 :
```

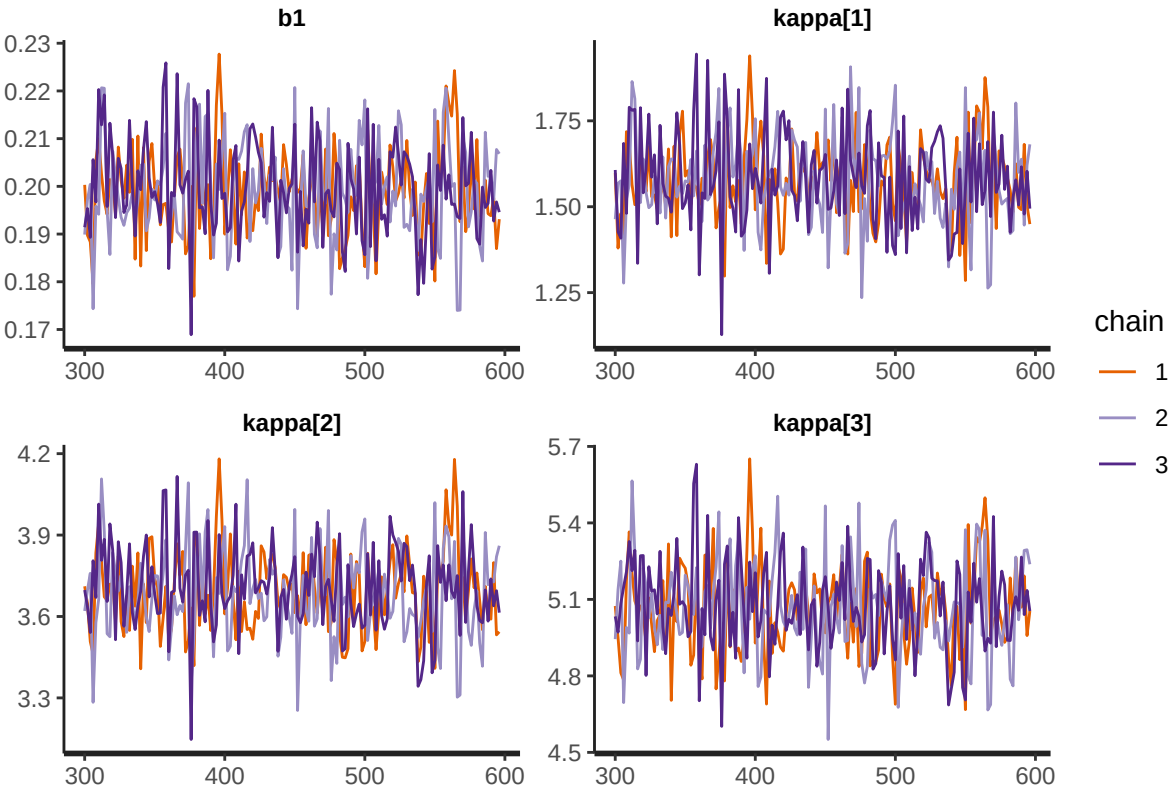
```
##  525 : #endif
##  526 :
##  527 :
##  528 :
##  529 : #include <rstan_next/stan_fit.hpp>
##  530 :
##  531 : struct stan_model_holder {
##  532 :     stan_model_holder(rstan::io::rlist_ref_var_context rcontext,
##  533 :                       unsigned int random_seed)
##  534 :     : rcontext_(rcontext), random_seed_(random_seed)
##  535 :      {
##  536 :      }
##  537 :
##  538 :    //stan::math::ChainableStack ad_stack;
##  539 :    rstan::io::rlist_ref_var_context rcontext_;
##  540 :    unsigned int random_seed_;
##  541 : };
##  542 :
##  543 : Rcpp::XPtr<stan::model::model_base> model_ptr(stan_model_holder* smh) {
##  544 :   Rcpp::XPtr<stan::model::model_base> model_instance(new stan_model(smh->rcontext_, smh->rando
##  545 :   return model_instance;
##  546 : }
##  547 :
##  548 : Rcpp::XPtr<rstan::stan_fit_base> fit_ptr(stan_model_holder* smh) {
##  549 :   return Rcpp::XPtr<rstan::stan_fit_base>(new rstan::stan_fit(model_ptr(smh), smh->random_seed
##  550 : }
##  551 :
##  552 : std::string model_name(stan_model_holder* smh) {
##  553 :   return model_ptr(smh).get()->model_name();
##  554 : }
##  555 :
##  556 : RCPP_MODULE(stan_fit4model399138178fc_20a961012d9d9a04013dc735a1515910_mod){
##  557 :   Rcpp::class_<stan_model_holder>("stan_fit4model399138178fc_20a961012d9d9a04013dc735a151591
##  558 :   .constructor<rstan::io::rlist_ref_var_context, unsigned int>()
##  559 :   .method("model_ptr", &model_ptr)
##  560 :   .method("fit_ptr", &fit_ptr)
##  561 :   .method("model_name", &model_name)
##  562 :   ;
##  563 : }
##  564 :
##  565 :
##  566 : // declarations
##  567 : extern "C" {
##  568 : SEXP file399117b5a45f( ) ;
##  569 : }
##  570 :
##  571 : // definition
##  572 : SEXP file399117b5a45f() {
##  573 :  return Rcpp::wrap("20a961012d9d9a04013dc735a1515910");
##  574 : }
```

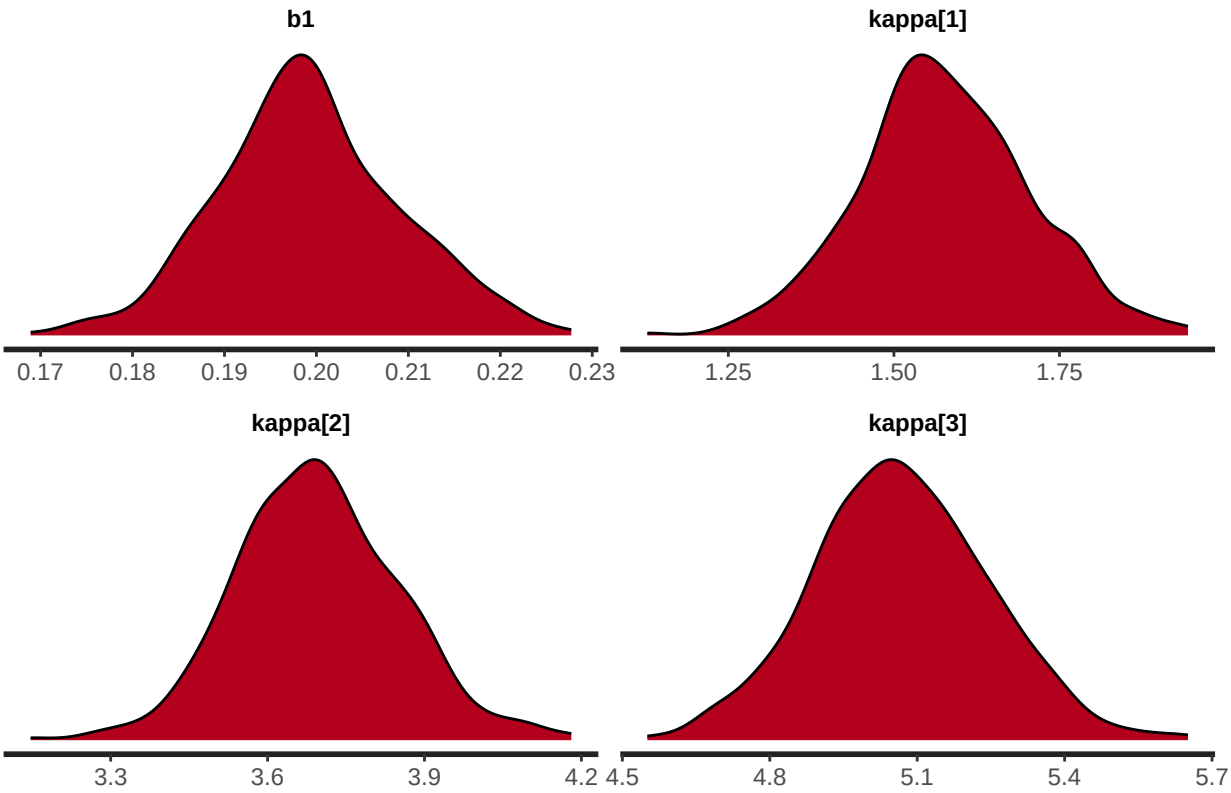
```
fit.lin.non <- sampling(mod,
                                data=datos.lin,
                                chains=3,warmup=300,iter=600,thin=2,cores=4 )
```

```
print(fit.lin.non, pars=param.lin)
```

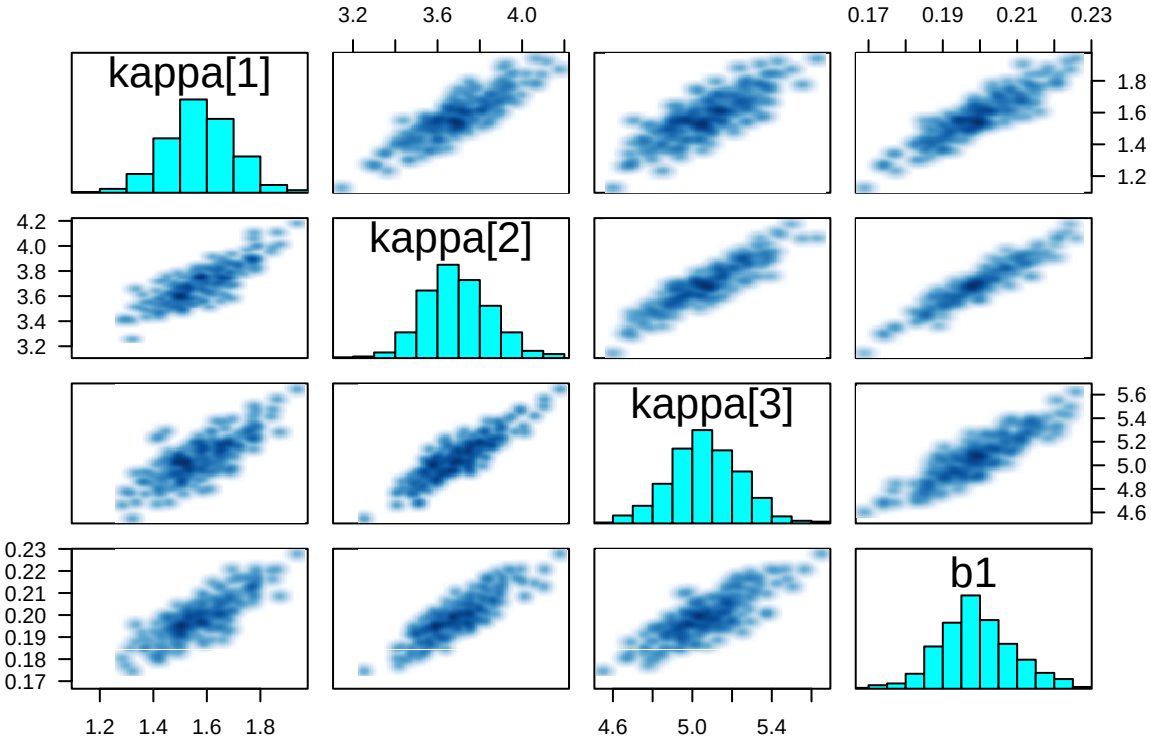
```
## Inference for Stan model: 20a961012d9d9a04013dc735a1515910.
## 3 chains, each with iter=600; warmup=300; thin=2;
## post-warmup draws per chain=150, total post-warmup draws=450.
##
##          mean se_mean   sd 2.5%  25%  50%  75% 97.5% n_eff Rhat
## b1       0.20    0.00 0.01 0.18 0.19 0.20 0.21  0.22   404    1
## kappa[1] 1.58    0.01 0.13 1.33 1.50 1.57 1.66  1.84   473    1
## kappa[2] 3.69    0.01 0.16 3.41 3.59 3.69 3.80  4.01   367    1
## kappa[3] 5.07    0.01 0.18 4.70 4.94 5.06 5.18  5.42   390    1
##
## Samples were drawn using NUTS(diag_e) at Sun Aug 13 15:38:00 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
stan_trace(fit.lin.non,pars=param.lin)
stan_dens(fit.lin.non,pars=param.lin)
```

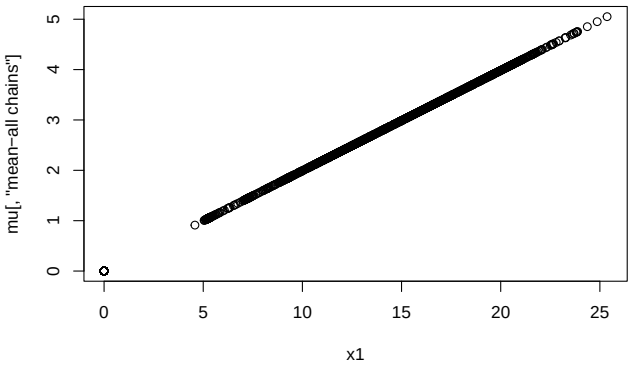





```
pairs(fit.lin.non, pars = c("kappa","b1"), las = 1)
```




```
mu=get_posterior_mean(fit.lin.non,"mu")
plot(x1,mu[,"mean-all chains"])
```


mu[, "mean−all chains"]
0
1
2
3
4
5
0
5
10
15
20
25
x1

5.2 LME: Lineal fit without constraints:

```
set.seed(123)

#mod.lme <- clmm( y_fact  ~ x1  + (1|id_fact) )

mod.lme <- clmm2( y_fact  ~ x1 , random=id_fact,
                  Hess=TRUE)
summary(mod.lme)
```

```
## Cumulative Link Mixed Model fitted with the Laplace approximation
##
## Call:
## clmm2(location = y_fact ~ x1, random = id_fact, Hess = TRUE)
##
## Random effects:
##              Var  Std.Dev
## id_fact 23.39041 4.836363
##
## Location coefficients:
##    Estimate Std. Error z value Pr(>|z|)
## x1  0.7649   0.0468    16.3538 < 2.22e-16
##
## No scale coefficients
##
## Threshold coefficients:
##     Estimate Std. Error z value
## 1|2  7.0564   0.5863    12.0347
## 2|3 12.8051   0.7878    16.2547
## 3|4 16.0283   0.8726    18.3693
##
## log-likelihood: -1034.07
## AIC: 2078.139
## Condition number of Hessian: 4541.717
```

```
datos.lme <- list( y = y ,
                   n = length(y) , N = N , Ni = Ni,
                   x1 = x1 , id = id )
param.lme = c("b1", "kappa", "invsig2","sig2","sigma")
```

```
stancode <- readLines("jagam_10_aneur_ordinal_lme_non_reintrcpt.stan")
# writeLines(stancode)
```

```
mod <- stan_model(model_code=stancode,
                                  verbose=TRUE)
```

```
##
## TRANSLATING MODEL '77f5ac182ddee7e88190776d47ea5b5b' FROM Stan CODE TO C++ CODE NOW.
## successful in parsing the Stan model '77f5ac182ddee7e88190776d47ea5b5b'.
## OS: x86_64, darwin17.0; rstan: 2.21.3; Rcpp: 1.0.10; inline: 0.3.19
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
```

```
## clang -mmacosx-version-min=10.13 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/inclu
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/include/Eigen/Core:96:10: fa
## #include <complex>
##          ^~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##  >> setting environment variables:
## PKG_LIBS =  '/Library/Frameworks/R.framework/Versions/4.1/Resources/library/rstan/lib//libStanServic
## PKG_CPPFLAGS =   -I"/Library/Frameworks/R.framework/Versions/4.1/Resources/library/Rcpp/include/"  -I
##  >> Program source :
##
##    1 :
##    2 : // includes from the plugin
##    3 : // [[Rcpp::plugins(cpp14)]]
##    4 :
##    5 :
##    6 : // user includes
##    7 : #include <Rcpp.h>
##    8 : #include <rstan/io/rlist_ref_var_context.hpp>
##    9 : #include <rstan/io/r_ostream.hpp>
##   10 : #include <rstan/stan_args.hpp>
##   11 : #include <boost/integer/integer_log2.hpp>
##   12 : // Code generated by Stan version 2.21.0
##   13 :
##   14 : #include <stan/model/model_header.hpp>
##   15 :
##   16 : namespace model39915d1b3b7b_77f5ac182ddee7e88190776d47ea5b5b_namespace {
##   17 :
##   18 : using std::istream;
##   19 : using std::string;
##   20 : using std::stringstream;
##   21 : using std::vector;
##   22 : using stan::io::dump;
##   23 : using stan::math::lgamma;
##   24 : using stan::model::prob_grad;
##   25 : using namespace stan::math;
##   26 :
##   27 : static int current_statement_begin__;
##   28 :
##   29 : stan::io::program_reader prog_reader__() {
```

```
##   30 :     stan::io::program_reader reader;
##   31 :     reader.add_event(0, 0, "start", "model39915d1b3b7b_77f5ac182ddee7e88190776d47ea5b5b");
##   32 :     reader.add_event(57, 55, "end", "model39915d1b3b7b_77f5ac182ddee7e88190776d47ea5b5b");
##   33 :     return reader;
##   34 : }
##   35 :
##   36 : class model39915d1b3b7b_77f5ac182ddee7e88190776d47ea5b5b
##   37 :   : public stan::model::model_base_crtp<model39915d1b3b7b_77f5ac182ddee7e88190776d47ea5b5b> {
##   38 : private:
##   39 :         int N;
##   40 :         int n;
##   41 :         std::vector<int> Ni;
##   42 :         std::vector<int> y;
##   43 :         std::vector<double> x1;
##   44 :         std::vector<int> id;
##   45 : public:
##   46 :     model39915d1b3b7b_77f5ac182ddee7e88190776d47ea5b5b(rstan::io::rlist_ref_var_context& cont
##   47 :         std::ostream* pstream__ = 0)
##   48 :         : model_base_crtp(0) {
##   49 :         ctor_body(context__, 0, pstream__);
##   50 :     }
##   51 :
##   52 :     model39915d1b3b7b_77f5ac182ddee7e88190776d47ea5b5b(stan::io::var_context& context__,
##   53 :         unsigned int random_seed__,
##   54 :         std::ostream* pstream__ = 0)
##   55 :         : model_base_crtp(0) {
##   56 :         ctor_body(context__, random_seed__, pstream__);
##   57 :     }
##   58 :
##   59 :     void ctor_body(stan::io::var_context& context__,
##   60 :                    unsigned int random_seed__,
##   61 :                    std::ostream* pstream__) {
##   62 :         typedef double local_scalar_t__;
##   63 :
##   64 :         boost::ecuyer1988 base_rng__ =
##   65 :           stan::services::util::create_rng(random_seed__, 0);
##   66 :         (void) base_rng__;  // suppress unused var warning
##   67 :
##   68 :         current_statement_begin__ = -1;
##   69 :
##   70 :         static const char* function__ = "model39915d1b3b7b_77f5ac182ddee7e88190776d47ea5b5b_na
##   71 :         (void) function__;  // dummy to suppress unused var warning
##   72 :         size_t pos__;
##   73 :         (void) pos__;  // dummy to suppress unused var warning
##   74 :         std::vector<int> vals_i__;
##   75 :         std::vector<double> vals_r__;
##   76 :         local_scalar_t__ DUMMY_VAR__(std::numeric_limits<double>::quiet_NaN());
##   77 :         (void) DUMMY_VAR__;  // suppress unused var warning
##   78 :
##   79 :         try {
##   80 :             // initialize data block variables from context__
##   81 :             current_statement_begin__ = 8;
##   82 :             context__.validate_dims("data initialization", "N", "int", context__.to_vec());
##   83 :             N = int(0);
```

```
##   84 :             vals_i__ = context__.vals_i("N");
##   85 :             pos__ = 0;
##   86 :             N = vals_i__[pos__++];
##   87 :             check_greater_or_equal(function__, "N", N, 0);
##   88 :
##   89 :             current_statement_begin__ = 9;
##   90 :             context__.validate_dims("data initialization", "n", "int", context__.to_vec());
##   91 :             n = int(0);
##   92 :             vals_i__ = context__.vals_i("n");
##   93 :             pos__ = 0;
##   94 :             n = vals_i__[pos__++];
##   95 :             check_greater_or_equal(function__, "n", n, 0);
##   96 :
##   97 :             current_statement_begin__ = 10;
##   98 :             validate_non_negative_index("Ni", "(N + 1)", (N + 1));
##   99 :             context__.validate_dims("data initialization", "Ni", "int", context__.to_vec((N +
##  100 :             Ni = std::vector<int>((N + 1), int(0));
##  101 :             vals_i__ = context__.vals_i("Ni");
##  102 :             pos__ = 0;
##  103 :             size_t Ni_k_0_max__ = (N + 1);
##  104 :             for (size_t k_0__ = 0; k_0__ < Ni_k_0_max__; ++k_0__) {
##  105 :                 Ni[k_0__] = vals_i__[pos__++];
##  106 :             }
##  107 :             size_t Ni_i_0_max__ = (N + 1);
##  108 :             for (size_t i_0__ = 0; i_0__ < Ni_i_0_max__; ++i_0__) {
##  109 :                 check_greater_or_equal(function__, "Ni[i_0__]", Ni[i_0__], 0);
##  110 :             }
##  111 :
##  112 :             current_statement_begin__ = 11;
##  113 :             validate_non_negative_index("y", "n", n);
##  114 :             context__.validate_dims("data initialization", "y", "int", context__.to_vec(n));
##  115 :             y = std::vector<int>(n, int(0));
##  116 :             vals_i__ = context__.vals_i("y");
##  117 :             pos__ = 0;
##  118 :             size_t y_k_0_max__ = n;
##  119 :             for (size_t k_0__ = 0; k_0__ < y_k_0_max__; ++k_0__) {
##  120 :                 y[k_0__] = vals_i__[pos__++];
##  121 :             }
##  122 :             size_t y_i_0_max__ = n;
##  123 :             for (size_t i_0__ = 0; i_0__ < y_i_0_max__; ++i_0__) {
##  124 :                 check_greater_or_equal(function__, "y[i_0__]", y[i_0__], 1);
##  125 :                 check_less_or_equal(function__, "y[i_0__]", y[i_0__], 4);
##  126 :             }
##  127 :
##  128 :             current_statement_begin__ = 12;
##  129 :             validate_non_negative_index("x1", "n", n);
##  130 :             context__.validate_dims("data initialization", "x1", "double", context__.to_vec(n
##  131 :             x1 = std::vector<double>(n, double(0));
##  132 :             vals_r__ = context__.vals_r("x1");
##  133 :             pos__ = 0;
##  134 :             size_t x1_k_0_max__ = n;
##  135 :             for (size_t k_0__ = 0; k_0__ < x1_k_0_max__; ++k_0__) {
##  136 :                 x1[k_0__] = vals_r__[pos__++];
##  137 :             }
```

```
##  138 :
##  139 :             current_statement_begin__ = 13;
##  140 :             validate_non_negative_index("id", "n", n);
##  141 :             context__.validate_dims("data initialization", "id", "int", context__.to_vec(n));
##  142 :             id = std::vector<int>(n, int(0));
##  143 :             vals_i__ = context__.vals_i("id");
##  144 :             pos__ = 0;
##  145 :             size_t id_k_0_max__ = n;
##  146 :             for (size_t k_0__ = 0; k_0__ < id_k_0_max__; ++k_0__) {
##  147 :                 id[k_0__] = vals_i__[pos__++];
##  148 :             }
##  149 :             size_t id_i_0_max__ = n;
##  150 :             for (size_t i_0__ = 0; i_0__ < id_i_0_max__; ++i_0__) {
##  151 :                 check_greater_or_equal(function__, "id[i_0__]", id[i_0__], 1);
##  152 :             }
##  153 :
##  154 :
##  155 :             // initialize transformed data variables
##  156 :             // execute transformed data statements
##  157 :
##  158 :             // validate transformed data
##  159 :
##  160 :             // validate, set parameter ranges
##  161 :             num_params_r__ = 0U;
##  162 :             param_ranges_i__.clear();
##  163 :             current_statement_begin__ = 17;
##  164 :             validate_non_negative_index("kappa", "3", 3);
##  165 :             num_params_r__ += 3;
##  166 :             current_statement_begin__ = 18;
##  167 :             num_params_r__ += 1;
##  168 :             current_statement_begin__ = 19;
##  169 :             validate_non_negative_index("bre0", "N", N);
##  170 :             num_params_r__ += (1 * N);
##  171 :             current_statement_begin__ = 20;
##  172 :             num_params_r__ += 1;
##  173 :         } catch (const std::exception& e) {
##  174 :             stan::lang::rethrow_located(e, current_statement_begin__, prog_reader__());
##  175 :             // Next line prevents compiler griping about no return
##  176 :             throw std::runtime_error("*** IF YOU SEE THIS, PLEASE REPORT A BUG ***");
##  177 :         }
##  178 :     }
##  179 :
##  180 :     ~model39915d1b3b7b_77f5ac182ddee7e88190776d47ea5b5b() { }
##  181 :
##  182 :
##  183 :     void transform_inits(const stan::io::var_context& context__,
##  184 :                          std::vector<int>& params_i__,
##  185 :                          std::vector<double>& params_r__,
##  186 :                          std::ostream* pstream__) const {
##  187 :         typedef double local_scalar_t__;
##  188 :         stan::io::writer<double> writer__(params_r__, params_i__);
##  189 :         size_t pos__;
##  190 :         (void) pos__; // dummy call to supress warning
##  191 :         std::vector<double> vals_r__;
```

```
##  192 :         std::vector<int> vals_i__;
##  193 :
##  194 :         current_statement_begin__ = 17;
##  195 :         if (!(context__.contains_r("kappa")))
##  196 :             stan::lang::rethrow_located(std::runtime_error(std::string("Variable kappa missing
##  197 :         vals_r__ = context__.vals_r("kappa");
##  198 :         pos__ = 0U;
##  199 :         validate_non_negative_index("kappa", "3", 3);
##  200 :         context__.validate_dims("parameter initialization", "kappa", "vector_d", context__.to_
##  201 :         Eigen::Matrix<double, Eigen::Dynamic, 1> kappa(3);
##  202 :         size_t kappa_j_1_max__ = 3;
##  203 :         for (size_t j_1__ = 0; j_1__ < kappa_j_1_max__; ++j_1__) {
##  204 :             kappa(j_1__) = vals_r__[pos__++];
##  205 :         }
##  206 :         try {
##  207 :             writer__.ordered_unconstrain(kappa);
##  208 :         } catch (const std::exception& e) {
##  209 :             stan::lang::rethrow_located(std::runtime_error(std::string("Error transforming var
##  210 :         }
##  211 :
##  212 :         current_statement_begin__ = 18;
##  213 :         if (!(context__.contains_r("b1")))
##  214 :             stan::lang::rethrow_located(std::runtime_error(std::string("Variable b1 missing"))
##  215 :         vals_r__ = context__.vals_r("b1");
##  216 :         pos__ = 0U;
##  217 :         context__.validate_dims("parameter initialization", "b1", "double", context__.to_vec()
##  218 :         double b1(0);
##  219 :         b1 = vals_r__[pos__++];
##  220 :         try {
##  221 :             writer__.scalar_unconstrain(b1);
##  222 :         } catch (const std::exception& e) {
##  223 :             stan::lang::rethrow_located(std::runtime_error(std::string("Error transforming var
##  224 :         }
##  225 :
##  226 :         current_statement_begin__ = 19;
##  227 :         if (!(context__.contains_r("bre0")))
##  228 :             stan::lang::rethrow_located(std::runtime_error(std::string("Variable bre0 missing"
##  229 :         vals_r__ = context__.vals_r("bre0");
##  230 :         pos__ = 0U;
##  231 :         validate_non_negative_index("bre0", "N", N);
##  232 :         context__.validate_dims("parameter initialization", "bre0", "double", context__.to_vec
##  233 :         std::vector<double> bre0(N, double(0));
##  234 :         size_t bre0_k_0_max__ = N;
##  235 :         for (size_t k_0__ = 0; k_0__ < bre0_k_0_max__; ++k_0__) {
##  236 :             bre0[k_0__] = vals_r__[pos__++];
##  237 :         }
##  238 :         size_t bre0_i_0_max__ = N;
##  239 :         for (size_t i_0__ = 0; i_0__ < bre0_i_0_max__; ++i_0__) {
##  240 :             try {
##  241 :                 writer__.scalar_unconstrain(bre0[i_0__]);
##  242 :             } catch (const std::exception& e) {
##  243 :                 stan::lang::rethrow_located(std::runtime_error(std::string("Error transforming
##  244 :             }
##  245 :         }
```

```
##  246 :
##  247 :         current_statement_begin__ = 20;
##  248 :         if (!(context__.contains_r("invsig2")))
##  249 :             stan::lang::rethrow_located(std::runtime_error(std::string("Variable invsig2 missi
##  250 :         vals_r__ = context__.vals_r("invsig2");
##  251 :         pos__ = 0U;
##  252 :         context__.validate_dims("parameter initialization", "invsig2", "double", context__.to_
##  253 :         double invsig2(0);
##  254 :         invsig2 = vals_r__[pos__++];
##  255 :         try {
##  256 :             writer__.scalar_lb_unconstrain(0, invsig2);
##  257 :         } catch (const std::exception& e) {
##  258 :             stan::lang::rethrow_located(std::runtime_error(std::string("Error transforming va
##  259 :         }
##  260 :
##  261 :         params_r__ = writer__.data_r();
##  262 :         params_i__ = writer__.data_i();
##  263 :     }
##  264 :
##  265 :     void transform_inits(const stan::io::var_context& context,
##  266 :                          Eigen::Matrix<double, Eigen::Dynamic, 1>& params_r,
##  267 :                          std::ostream* pstream__) const {
##  268 :       std::vector<double> params_r_vec;
##  269 :       std::vector<int> params_i_vec;
##  270 :       transform_inits(context, params_i_vec, params_r_vec, pstream__);
##  271 :       params_r.resize(params_r_vec.size());
##  272 :       for (int i = 0; i < params_r.size(); ++i)
##  273 :         params_r(i) = params_r_vec[i];
##  274 :     }
##  275 :
##  276 :
##  277 :     template <bool propto__, bool jacobian__, typename T__>
##  278 :     T__ log_prob(std::vector<T__>& params_r__,
##  279 :                  std::vector<int>& params_i__,
##  280 :                  std::ostream* pstream__ = 0) const {
##  281 :
##  282 :         typedef T__ local_scalar_t__;
##  283 :
##  284 :         local_scalar_t__ DUMMY_VAR__(std::numeric_limits<double>::quiet_NaN());
##  285 :         (void) DUMMY_VAR__;  // dummy to suppress unused var warning
##  286 :
##  287 :         T__ lp__(0.0);
##  288 :         stan::math::accumulator<T__> lp_accum__;
##  289 :         try {
##  290 :             stan::io::reader<local_scalar_t__> in__(params_r__, params_i__);
##  291 :
##  292 :             // model parameters
##  293 :             current_statement_begin__ = 17;
##  294 :             Eigen::Matrix<local_scalar_t__, Eigen::Dynamic, 1> kappa;
##  295 :             (void) kappa;  // dummy to suppress unused var warning
##  296 :             if (jacobian__)
##  297 :                 kappa = in__.ordered_constrain(3, lp__);
##  298 :             else
##  299 :                 kappa = in__.ordered_constrain(3);
```

```
##  300 :
##  301 :             current_statement_begin__ = 18;
##  302 :             local_scalar_t__ b1;
##  303 :             (void) b1;  // dummy to suppress unused var warning
##  304 :             if (jacobian__)
##  305 :                 b1 = in__.scalar_constrain(lp__);
##  306 :             else
##  307 :                 b1 = in__.scalar_constrain();
##  308 :
##  309 :             current_statement_begin__ = 19;
##  310 :             std::vector<local_scalar_t__> bre0;
##  311 :             size_t bre0_d_0_max__ = N;
##  312 :             bre0.reserve(bre0_d_0_max__);
##  313 :             for (size_t d_0__ = 0; d_0__ < bre0_d_0_max__; ++d_0__) {
##  314 :                 if (jacobian__)
##  315 :                     bre0.push_back(in__.scalar_constrain(lp__));
##  316 :                 else
##  317 :                     bre0.push_back(in__.scalar_constrain());
##  318 :             }
##  319 :
##  320 :             current_statement_begin__ = 20;
##  321 :             local_scalar_t__ invsig2;
##  322 :             (void) invsig2;  // dummy to suppress unused var warning
##  323 :             if (jacobian__)
##  324 :                 invsig2 = in__.scalar_lb_constrain(0, lp__);
##  325 :             else
##  326 :                 invsig2 = in__.scalar_lb_constrain(0);
##  327 :
##  328 :             // transformed parameters
##  329 :             current_statement_begin__ = 24;
##  330 :             validate_non_negative_index("mu", "n", n);
##  331 :             Eigen::Matrix<local_scalar_t__, Eigen::Dynamic, 1> mu(n);
##  332 :             stan::math::initialize(mu, DUMMY_VAR__);
##  333 :             stan::math::fill(mu, DUMMY_VAR__);
##  334 :
##  335 :             current_statement_begin__ = 25;
##  336 :             local_scalar_t__ sig2;
##  337 :             (void) sig2;  // dummy to suppress unused var warning
##  338 :             stan::math::initialize(sig2, DUMMY_VAR__);
##  339 :             stan::math::fill(sig2, DUMMY_VAR__);
##  340 :
##  341 :             current_statement_begin__ = 26;
##  342 :             local_scalar_t__ sigma;
##  343 :             (void) sigma;  // dummy to suppress unused var warning
##  344 :             stan::math::initialize(sigma, DUMMY_VAR__);
##  345 :             stan::math::fill(sigma, DUMMY_VAR__);
##  346 :
##  347 :             // transformed parameters block statements
##  348 :             current_statement_begin__ = 27;
##  349 :             for (int i = 1; i <= N; ++i) {
##  350 :
##  351 :                 current_statement_begin__ = 28;
##  352 :                 for (int t = get_base1(Ni, i, "Ni", 1); t <= (get_base1(Ni, (i + 1), "Ni", 1)
##  353 :
```

```
##  354 :                     current_statement_begin__ = 29;
##  355 :                     stan::model::assign(mu,
##  356 :                                 stan::model::cons_list(stan::model::index_uni(t), stan::model
##  357 :                                 ((get_base1(x1, t, "x1", 1) * b1) + get_base1(bre0, i, "bre0"
##  358 :                                 "assigning variable mu");
##  359 :                 }
##  360 :             }
##  361 :             current_statement_begin__ = 32;
##  362 :             stan::math::assign(sig2, (1 / invsig2));
##  363 :             current_statement_begin__ = 33;
##  364 :             stan::math::assign(sigma, pow(sig2, 0.5));
##  365 :
##  366 :             // validate transformed parameters
##  367 :             const char* function__ = "validate transformed params";
##  368 :             (void) function__;  // dummy to suppress unused var warning
##  369 :
##  370 :             current_statement_begin__ = 24;
##  371 :             size_t mu_j_1_max__ = n;
##  372 :             for (size_t j_1__ = 0; j_1__ < mu_j_1_max__; ++j_1__) {
##  373 :                 if (stan::math::is_uninitialized(mu(j_1__))) {
##  374 :                     std::stringstream msg__;
##  375 :                     msg__ << "Undefined transformed parameter: mu" << "(" << j_1__ << ")";
##  376 :                     stan::lang::rethrow_located(std::runtime_error(std::string("Error initial
##  377 :                 }
##  378 :             }
##  379 :             current_statement_begin__ = 25;
##  380 :             if (stan::math::is_uninitialized(sig2)) {
##  381 :                 std::stringstream msg__;
##  382 :                 msg__ << "Undefined transformed parameter: sig2";
##  383 :                 stan::lang::rethrow_located(std::runtime_error(std::string("Error initializing
##  384 :             }
##  385 :             check_greater_or_equal(function__, "sig2", sig2, 0);
##  386 :
##  387 :             current_statement_begin__ = 26;
##  388 :             if (stan::math::is_uninitialized(sigma)) {
##  389 :                 std::stringstream msg__;
##  390 :                 msg__ << "Undefined transformed parameter: sigma";
##  391 :                 stan::lang::rethrow_located(std::runtime_error(std::string("Error initializing
##  392 :             }
##  393 :             check_greater_or_equal(function__, "sigma", sigma, 0);
##  394 :
##  395 :
##  396 :             // model body
##  397 :
##  398 :             current_statement_begin__ = 37;
##  399 :             lp_accum__.add(gamma_log<propto__>(invsig2, .05, .005));
##  400 :             current_statement_begin__ = 38;
##  401 :             lp_accum__.add(normal_log<propto__>(b1, 0, 9.9e+06));
##  402 :             current_statement_begin__ = 39;
##  403 :             for (int i = 1; i <= N; ++i) {
##  404 :
##  405 :                 current_statement_begin__ = 40;
##  406 :                 lp_accum__.add(normal_log<propto__>(get_base1(bre0, i, "bre0", 1), 0, sigma))
##  407 :                 current_statement_begin__ = 41;
```

```
##  408 :                 for (int t = get_base1(Ni, i, "Ni", 1); t <= (get_base1(Ni, (i + 1), "Ni", 1)
##  409 :
##  410 :                     current_statement_begin__ = 42;
##  411 :                     lp_accum__.add(ordered_logistic_log<propto__>(get_base1(y, t, "y", 1), ge
##  412 :                 }
##  413 :             }
##  414 :
##  415 :         } catch (const std::exception& e) {
##  416 :             stan::lang::rethrow_located(e, current_statement_begin__, prog_reader__());
##  417 :             // Next line prevents compiler griping about no return
##  418 :             throw std::runtime_error("*** IF YOU SEE THIS, PLEASE REPORT A BUG ***");
##  419 :         }
##  420 :
##  421 :         lp_accum__.add(lp__);
##  422 :         return lp_accum__.sum();
##  423 :
##  424 :     } // log_prob()
##  425 :
##  426 :     template <bool propto, bool jacobian, typename T_>
##  427 :     T_ log_prob(Eigen::Matrix<T_,Eigen::Dynamic,1>& params_r,
##  428 :                 std::ostream* pstream = 0) const {
##  429 :       std::vector<T_> vec_params_r;
##  430 :       vec_params_r.reserve(params_r.size());
##  431 :       for (int i = 0; i < params_r.size(); ++i)
##  432 :         vec_params_r.push_back(params_r(i));
##  433 :       std::vector<int> vec_params_i;
##  434 :       return log_prob<propto,jacobian,T_>(vec_params_r, vec_params_i, pstream);
##  435 :     }
##  436 :
##  437 :
##  438 :     void get_param_names(std::vector<std::string>& names__) const {
##  439 :         names__.resize(0);
##  440 :         names__.push_back("kappa");
##  441 :         names__.push_back("b1");
##  442 :         names__.push_back("bre0");
##  443 :         names__.push_back("invsig2");
##  444 :         names__.push_back("mu");
##  445 :         names__.push_back("sig2");
##  446 :         names__.push_back("sigma");
##  447 :         names__.push_back("log_lik");
##  448 :     }
##  449 :
##  450 :
##  451 :     void get_dims(std::vector<std::vector<size_t> >& dimss__) const {
##  452 :         dimss__.resize(0);
##  453 :         std::vector<size_t> dims__;
##  454 :         dims__.resize(0);
##  455 :         dims__.push_back(3);
##  456 :         dimss__.push_back(dims__);
##  457 :         dims__.resize(0);
##  458 :         dimss__.push_back(dims__);
##  459 :         dims__.resize(0);
##  460 :         dims__.push_back(N);
##  461 :         dimss__.push_back(dims__);
```

```
##  462 :         dims__.resize(0);
##  463 :         dimss__.push_back(dims__);
##  464 :         dims__.resize(0);
##  465 :         dims__.push_back(n);
##  466 :         dimss__.push_back(dims__);
##  467 :         dims__.resize(0);
##  468 :         dimss__.push_back(dims__);
##  469 :         dims__.resize(0);
##  470 :         dimss__.push_back(dims__);
##  471 :         dims__.resize(0);
##  472 :         dims__.push_back(n);
##  473 :         dimss__.push_back(dims__);
##  474 :     }
##  475 :
##  476 :     template <typename RNG>
##  477 :     void write_array(RNG& base_rng__,
##  478 :                      std::vector<double>& params_r__,
##  479 :                      std::vector<int>& params_i__,
##  480 :                      std::vector<double>& vars__,
##  481 :                      bool include_tparams__ = true,
##  482 :                      bool include_gqs__ = true,
##  483 :                      std::ostream* pstream__ = 0) const {
##  484 :         typedef double local_scalar_t__;
##  485 :
##  486 :         vars__.resize(0);
##  487 :         stan::io::reader<local_scalar_t__> in__(params_r__, params_i__);
##  488 :         static const char* function__ = "model39915d1b3b7b_77f5ac182ddee7e88190776d47ea5b5b_na
##  489 :         (void) function__;  // dummy to suppress unused var warning
##  490 :
##  491 :         // read-transform, write parameters
##  492 :         Eigen::Matrix<double, Eigen::Dynamic, 1> kappa = in__.ordered_constrain(3);
##  493 :         size_t kappa_j_1_max__ = 3;
##  494 :         for (size_t j_1__ = 0; j_1__ < kappa_j_1_max__; ++j_1__) {
##  495 :             vars__.push_back(kappa(j_1__));
##  496 :         }
##  497 :
##  498 :         double b1 = in__.scalar_constrain();
##  499 :         vars__.push_back(b1);
##  500 :
##  501 :         std::vector<double> bre0;
##  502 :         size_t bre0_d_0_max__ = N;
##  503 :         bre0.reserve(bre0_d_0_max__);
##  504 :         for (size_t d_0__ = 0; d_0__ < bre0_d_0_max__; ++d_0__) {
##  505 :             bre0.push_back(in__.scalar_constrain());
##  506 :         }
##  507 :         size_t bre0_k_0_max__ = N;
##  508 :         for (size_t k_0__ = 0; k_0__ < bre0_k_0_max__; ++k_0__) {
##  509 :             vars__.push_back(bre0[k_0__]);
##  510 :         }
##  511 :
##  512 :         double invsig2 = in__.scalar_lb_constrain(0);
##  513 :         vars__.push_back(invsig2);
##  514 :
##  515 :         double lp__ = 0.0;
```

```
##  516 :         (void) lp__;  // dummy to suppress unused var warning
##  517 :         stan::math::accumulator<double> lp_accum__;
##  518 :
##  519 :         local_scalar_t__ DUMMY_VAR__(std::numeric_limits<double>::quiet_NaN());
##  520 :         (void) DUMMY_VAR__;  // suppress unused var warning
##  521 :
##  522 :         if (!include_tparams__ && !include_gqs__) return;
##  523 :
##  524 :         try {
##  525 :             // declare and define transformed parameters
##  526 :             current_statement_begin__ = 24;
##  527 :             validate_non_negative_index("mu", "n", n);
##  528 :             Eigen::Matrix<double, Eigen::Dynamic, 1> mu(n);
##  529 :             stan::math::initialize(mu, DUMMY_VAR__);
##  530 :             stan::math::fill(mu, DUMMY_VAR__);
##  531 :
##  532 :             current_statement_begin__ = 25;
##  533 :             double sig2;
##  534 :             (void) sig2;  // dummy to suppress unused var warning
##  535 :             stan::math::initialize(sig2, DUMMY_VAR__);
##  536 :             stan::math::fill(sig2, DUMMY_VAR__);
##  537 :
##  538 :             current_statement_begin__ = 26;
##  539 :             double sigma;
##  540 :             (void) sigma;  // dummy to suppress unused var warning
##  541 :             stan::math::initialize(sigma, DUMMY_VAR__);
##  542 :             stan::math::fill(sigma, DUMMY_VAR__);
##  543 :
##  544 :             // do transformed parameters statements
##  545 :             current_statement_begin__ = 27;
##  546 :             for (int i = 1; i <= N; ++i) {
##  547 :
##  548 :                 current_statement_begin__ = 28;
##  549 :                 for (int t = get_base1(Ni, i, "Ni", 1); t <= (get_base1(Ni, (i + 1), "Ni", 1)
##  550 :
##  551 :                     current_statement_begin__ = 29;
##  552 :                     stan::model::assign(mu,
##  553 :                                 stan::model::cons_list(stan::model::index_uni(t), stan::model
##  554 :                                 ((get_base1(x1, t, "x1", 1) * b1) + get_base1(bre0, i, "bre0"
##  555 :                                 "assigning variable mu");
##  556 :                 }
##  557 :             }
##  558 :             current_statement_begin__ = 32;
##  559 :             stan::math::assign(sig2, (1 / invsig2));
##  560 :             current_statement_begin__ = 33;
##  561 :             stan::math::assign(sigma, pow(sig2, 0.5));
##  562 :
##  563 :             if (!include_gqs__ && !include_tparams__) return;
##  564 :             // validate transformed parameters
##  565 :             const char* function__ = "validate transformed params";
##  566 :             (void) function__;  // dummy to suppress unused var warning
##  567 :
##  568 :             current_statement_begin__ = 25;
##  569 :             check_greater_or_equal(function__, "sig2", sig2, 0);
```

```
##  570 :
##  571 :             current_statement_begin__ = 26;
##  572 :             check_greater_or_equal(function__, "sigma", sigma, 0);
##  573 :
##  574 :             // write transformed parameters
##  575 :             if (include_tparams__) {
##  576 :                 size_t mu_j_1_max__ = n;
##  577 :                 for (size_t j_1__ = 0; j_1__ < mu_j_1_max__; ++j_1__) {
##  578 :                     vars__.push_back(mu(j_1__));
##  579 :                 }
##  580 :                 vars__.push_back(sig2);
##  581 :                 vars__.push_back(sigma);
##  582 :             }
##  583 :             if (!include_gqs__) return;
##  584 :             // declare and define generated quantities
##  585 :             current_statement_begin__ = 48;
##  586 :             validate_non_negative_index("log_lik", "n", n);
##  587 :             Eigen::Matrix<double, Eigen::Dynamic, 1> log_lik(n);
##  588 :             stan::math::initialize(log_lik, DUMMY_VAR__);
##  589 :             stan::math::fill(log_lik, DUMMY_VAR__);
##  590 :
##  591 :             // generated quantities statements
##  592 :             current_statement_begin__ = 49;
##  593 :             for (int i = 1; i <= N; ++i) {
##  594 :
##  595 :                 current_statement_begin__ = 50;
##  596 :                 for (int t = get_base1(Ni, i, "Ni", 1); t <= (get_base1(Ni, (i + 1), "Ni", 1)
##  597 :
##  598 :                     current_statement_begin__ = 51;
##  599 :                     stan::model::assign(log_lik,
##  600 :                                 stan::model::cons_list(stan::model::index_uni(t), stan::model
##  601 :                                 ordered_logistic_log(get_base1(y, t, "y", 1), get_base1(mu, t
##  602 :                                 "assigning variable log_lik");
##  603 :                 }
##  604 :             }
##  605 :
##  606 :             // validate, write generated quantities
##  607 :             current_statement_begin__ = 48;
##  608 :             size_t log_lik_j_1_max__ = n;
##  609 :             for (size_t j_1__ = 0; j_1__ < log_lik_j_1_max__; ++j_1__) {
##  610 :                 vars__.push_back(log_lik(j_1__));
##  611 :             }
##  612 :
##  613 :         } catch (const std::exception& e) {
##  614 :             stan::lang::rethrow_located(e, current_statement_begin__, prog_reader__());
##  615 :             // Next line prevents compiler griping about no return
##  616 :             throw std::runtime_error("*** IF YOU SEE THIS, PLEASE REPORT A BUG ***");
##  617 :         }
##  618 :     }
##  619 :
##  620 :     template <typename RNG>
##  621 :     void write_array(RNG& base_rng,
##  622 :                      Eigen::Matrix<double,Eigen::Dynamic,1>& params_r,
##  623 :                      Eigen::Matrix<double,Eigen::Dynamic,1>& vars,
```

```
##  624 :                     bool include_tparams = true,
##  625 :                     bool include_gqs = true,
##  626 :                     std::ostream* pstream = 0) const {
##  627 :       std::vector<double> params_r_vec(params_r.size());
##  628 :       for (int i = 0; i < params_r.size(); ++i)
##  629 :         params_r_vec[i] = params_r(i);
##  630 :       std::vector<double> vars_vec;
##  631 :       std::vector<int> params_i_vec;
##  632 :       write_array(base_rng, params_r_vec, params_i_vec, vars_vec, include_tparams, include_gqs
##  633 :       vars.resize(vars_vec.size());
##  634 :       for (int i = 0; i < vars.size(); ++i)
##  635 :         vars(i) = vars_vec[i];
##  636 :     }
##  637 :
##  638 :     std::string model_name() const {
##  639 :         return "model39915d1b3b7b_77f5ac182ddee7e88190776d47ea5b5b";
##  640 :     }
##  641 :
##  642 :
##  643 :     void constrained_param_names(std::vector<std::string>& param_names__,
##  644 :                                  bool include_tparams__ = true,
##  645 :                                  bool include_gqs__ = true) const {
##  646 :         std::stringstream param_name_stream__;
##  647 :         size_t kappa_j_1_max__ = 3;
##  648 :         for (size_t j_1__ = 0; j_1__ < kappa_j_1_max__; ++j_1__) {
##  649 :             param_name_stream__.str(std::string());
##  650 :             param_name_stream__ << "kappa" << '.' << j_1__ + 1;
##  651 :             param_names__.push_back(param_name_stream__.str());
##  652 :         }
##  653 :         param_name_stream__.str(std::string());
##  654 :         param_name_stream__ << "b1";
##  655 :         param_names__.push_back(param_name_stream__.str());
##  656 :         size_t bre0_k_0_max__ = N;
##  657 :         for (size_t k_0__ = 0; k_0__ < bre0_k_0_max__; ++k_0__) {
##  658 :             param_name_stream__.str(std::string());
##  659 :             param_name_stream__ << "bre0" << '.' << k_0__ + 1;
##  660 :             param_names__.push_back(param_name_stream__.str());
##  661 :         }
##  662 :         param_name_stream__.str(std::string());
##  663 :         param_name_stream__ << "invsig2";
##  664 :         param_names__.push_back(param_name_stream__.str());
##  665 :
##  666 :         if (!include_gqs__ && !include_tparams__) return;
##  667 :
##  668 :         if (include_tparams__) {
##  669 :             size_t mu_j_1_max__ = n;
##  670 :             for (size_t j_1__ = 0; j_1__ < mu_j_1_max__; ++j_1__) {
##  671 :                 param_name_stream__.str(std::string());
##  672 :                 param_name_stream__ << "mu" << '.' << j_1__ + 1;
##  673 :                 param_names__.push_back(param_name_stream__.str());
##  674 :             }
##  675 :             param_name_stream__.str(std::string());
##  676 :             param_name_stream__ << "sig2";
##  677 :             param_names__.push_back(param_name_stream__.str());
```

```
##  678 :             param_name_stream__.str(std::string());
##  679 :             param_name_stream__ << "sigma";
##  680 :             param_names__.push_back(param_name_stream__.str());
##  681 :         }
##  682 :
##  683 :         if (!include_gqs__) return;
##  684 :         size_t log_lik_j_1_max__ = n;
##  685 :         for (size_t j_1__ = 0; j_1__ < log_lik_j_1_max__; ++j_1__) {
##  686 :             param_name_stream__.str(std::string());
##  687 :             param_name_stream__ << "log_lik" << '.' << j_1__ + 1;
##  688 :             param_names__.push_back(param_name_stream__.str());
##  689 :         }
##  690 :     }
##  691 :
##  692 :
##  693 :     void unconstrained_param_names(std::vector<std::string>& param_names__,
##  694 :                                    bool include_tparams__ = true,
##  695 :                                    bool include_gqs__ = true) const {
##  696 :         std::stringstream param_name_stream__;
##  697 :         size_t kappa_j_1_max__ = 3;
##  698 :         for (size_t j_1__ = 0; j_1__ < kappa_j_1_max__; ++j_1__) {
##  699 :             param_name_stream__.str(std::string());
##  700 :             param_name_stream__ << "kappa" << '.' << j_1__ + 1;
##  701 :             param_names__.push_back(param_name_stream__.str());
##  702 :         }
##  703 :         param_name_stream__.str(std::string());
##  704 :         param_name_stream__ << "b1";
##  705 :         param_names__.push_back(param_name_stream__.str());
##  706 :         size_t bre0_k_0_max__ = N;
##  707 :         for (size_t k_0__ = 0; k_0__ < bre0_k_0_max__; ++k_0__) {
##  708 :             param_name_stream__.str(std::string());
##  709 :             param_name_stream__ << "bre0" << '.' << k_0__ + 1;
##  710 :             param_names__.push_back(param_name_stream__.str());
##  711 :         }
##  712 :         param_name_stream__.str(std::string());
##  713 :         param_name_stream__ << "invsig2";
##  714 :         param_names__.push_back(param_name_stream__.str());
##  715 :
##  716 :         if (!include_gqs__ && !include_tparams__) return;
##  717 :
##  718 :         if (include_tparams__) {
##  719 :             size_t mu_j_1_max__ = n;
##  720 :             for (size_t j_1__ = 0; j_1__ < mu_j_1_max__; ++j_1__) {
##  721 :                 param_name_stream__.str(std::string());
##  722 :                 param_name_stream__ << "mu" << '.' << j_1__ + 1;
##  723 :                 param_names__.push_back(param_name_stream__.str());
##  724 :             }
##  725 :             param_name_stream__.str(std::string());
##  726 :             param_name_stream__ << "sig2";
##  727 :             param_names__.push_back(param_name_stream__.str());
##  728 :             param_name_stream__.str(std::string());
##  729 :             param_name_stream__ << "sigma";
##  730 :             param_names__.push_back(param_name_stream__.str());
##  731 :         }
```

```
##  732 :
##  733 :         if (!include_gqs__) return;
##  734 :         size_t log_lik_j_1_max__ = n;
##  735 :         for (size_t j_1__ = 0; j_1__ < log_lik_j_1_max__; ++j_1__) {
##  736 :             param_name_stream__.str(std::string());
##  737 :             param_name_stream__ << "log_lik" << '.' << j_1__ + 1;
##  738 :             param_names__.push_back(param_name_stream__.str());
##  739 :         }
##  740 :     }
##  741 :
##  742 : }; // model
##  743 :
##  744 : }  // namespace
##  745 :
##  746 : typedef model39915d1b3b7b_77f5ac182ddee7e88190776d47ea5b5b_namespace::model39915d1b3b7b_77f5a
##  747 :
##  748 : #ifndef USING_R
##  749 :
##  750 : stan::model::model_base& new_model(
##  751 :         stan::io::var_context& data_context,
##  752 :         unsigned int seed,
##  753 :         std::ostream* msg_stream) {
##  754 :   stan_model* m = new stan_model(data_context, seed, msg_stream);
##  755 :   return *m;
##  756 : }
##  757 :
##  758 : #endif
##  759 :
##  760 :
##  761 :
##  762 : #include <rstan_next/stan_fit.hpp>
##  763 :
##  764 : struct stan_model_holder {
##  765 :     stan_model_holder(rstan::io::rlist_ref_var_context rcontext,
##  766 :                       unsigned int random_seed)
##  767 :     : rcontext_(rcontext), random_seed_(random_seed)
##  768 :      {
##  769 :      }
##  770 :
##  771 :    //stan::math::ChainableStack ad_stack;
##  772 :    rstan::io::rlist_ref_var_context rcontext_;
##  773 :    unsigned int random_seed_;
##  774 : };
##  775 :
##  776 : Rcpp::XPtr<stan::model::model_base> model_ptr(stan_model_holder* smh) {
##  777 :   Rcpp::XPtr<stan::model::model_base> model_instance(new stan_model(smh->rcontext_, smh->rand
##  778 :   return model_instance;
##  779 : }
##  780 :
##  781 : Rcpp::XPtr<rstan::stan_fit_base> fit_ptr(stan_model_holder* smh) {
##  782 :   return Rcpp::XPtr<rstan::stan_fit_base>(new rstan::stan_fit(model_ptr(smh), smh->random_see
##  783 : }
##  784 :
##  785 : std::string model_name(stan_model_holder* smh) {
```

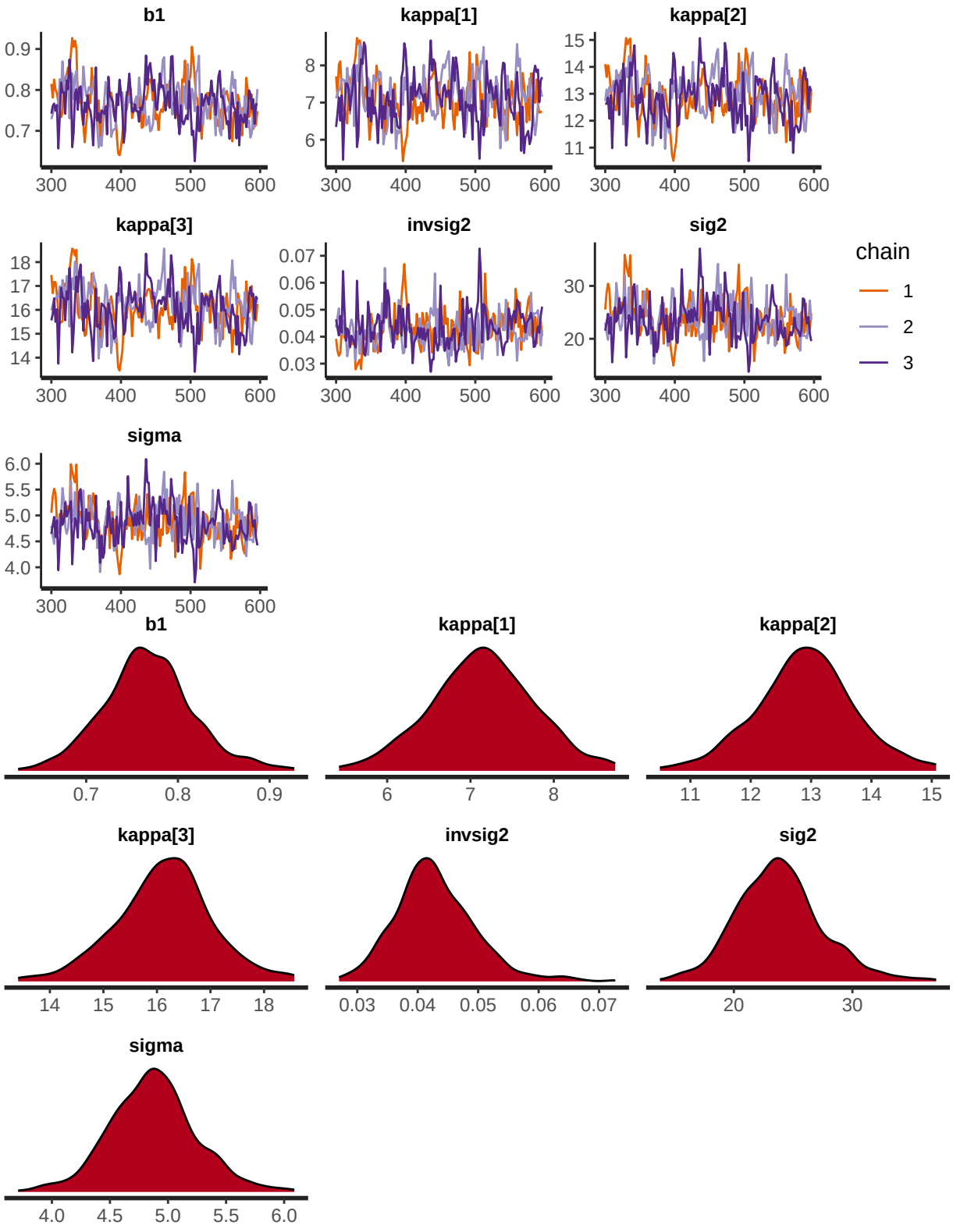
```
##  786 :   return model_ptr(smh).get()->model_name();
##  787 : }
##  788 :
##  789 : RCPP_MODULE(stan_fit4model39915d1b3b7b_77f5ac182ddee7e88190776d47ea5b5b_mod){
##  790 :   Rcpp::class_<stan_model_holder>("stan_fit4model39915d1b3b7b_77f5ac182ddee7e88190776d47ea5b5
##  791 :   .constructor<rstan::io::rlist_ref_var_context, unsigned int>()
##  792 :   .method("model_ptr", &model_ptr)
##  793 :   .method("fit_ptr", &fit_ptr)
##  794 :   .method("model_name", &model_name)
##  795 :   ;
##  796 : }
##  797 :
##  798 :
##  799 : // declarations
##  800 : extern "C" {
##  801 : SEXP file39915b322785( ) ;
##  802 : }
##  803 :
##  804 : // definition
##  805 : SEXP file39915b322785() {
##  806 :  return Rcpp::wrap("77f5ac182ddee7e88190776d47ea5b5b");
##  807 : }
```

```
fit.lme.non.reintrcpt <- sampling(mod,
                                  data=datos.lme,
                                  chains=3,warmup=300,iter=600,thin=2,cores=4 )
```

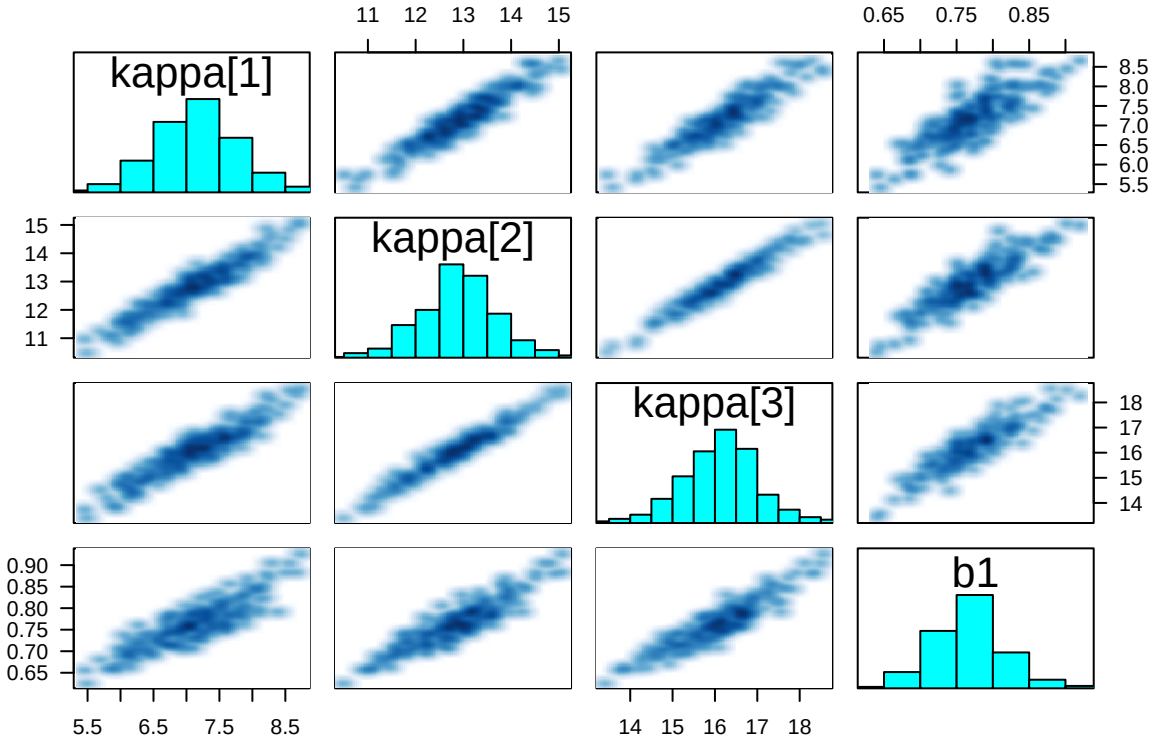
```
print(fit.lme.non.reintrcpt, pars=param.lme)
```

```
## Inference for Stan model: 77f5ac182ddee7e88190776d47ea5b5b.
## 3 chains, each with iter=600; warmup=300; thin=2;
## post-warmup draws per chain=150, total post-warmup draws=450.
##
##           mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## b1        0.77    0.00 0.05  0.67  0.74  0.77  0.79  0.87   127 1.00
## kappa[1]  7.14    0.05 0.62  5.92  6.74  7.15  7.56  8.39   139 1.01
## kappa[2] 12.90    0.07 0.81 11.22 12.39 12.91 13.38 14.52   131 1.00
## kappa[3] 16.13    0.08 0.89 14.28 15.62 16.15 16.67 17.92   126 1.00
## invsig2   0.04    0.00 0.01  0.03  0.04  0.04  0.05  0.06   149 1.00
## sig2     23.83    0.32 3.65 17.23 21.36 23.64 25.73 31.80   130 1.00
## sigma     4.87    0.03 0.37  4.15  4.62  4.86  5.07  5.64   134 1.00
##
## Samples were drawn using NUTS(diag_e) at Sun Aug 13 15:42:07 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

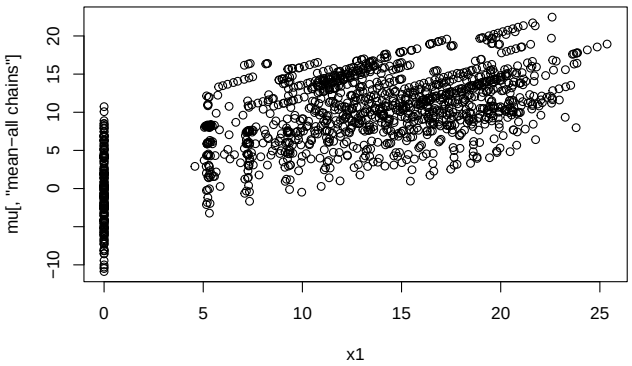
```
stan_trace(fit.lme.non.reintrcpt,pars=param.lme)
stan_dens(fit.lme.non.reintrcpt,pars=param.lme)
```




```
pairs(fit.lme.non.reintrcpt, pars = c("kappa","b1"), las = 1)
```




```
mu=get_posterior_mean(fit.lme.non.reintrcpt,"mu")
plot(x1,mu[,"mean-all chains"])
```




```
stancode <- readLines("jagam_10_aneur_ordinal_lme_non_reslope.stan")
# writeLines(stancode)
```

```
mod <- stan_model(model_code=stancode,
                                  verbose=TRUE)
```

```
##
## TRANSLATING MODEL '24b938afe5b29efe8d963b6430c621be' FROM Stan CODE TO C++ CODE NOW.
## successful in parsing the Stan model '24b938afe5b29efe8d963b6430c621be'.
## OS: x86_64, darwin17.0; rstan: 2.21.3; Rcpp: 1.0.10; inline: 0.3.19
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -mmacosx-version-min=10.13 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/StanHeaders/incl
```

```
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/inclu
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/include/Eigen/Core:96:10: fa
## #include <complex>
##          ^~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##  >> setting environment variables:
## PKG_LIBS =  '/Library/Frameworks/R.framework/Versions/4.1/Resources/library/rstan/lib//libStanServic
## PKG_CPPFLAGS =   -I"/Library/Frameworks/R.framework/Versions/4.1/Resources/library/Rcpp/include/"  -I
##  >> Program source :
##
##    1 :
##    2 : // includes from the plugin
##    3 : // [[Rcpp::plugins(cpp14)]]
##    4 :
##    5 :
##    6 : // user includes
##    7 : #include <Rcpp.h>
##    8 : #include <rstan/io/rlist_ref_var_context.hpp>
##    9 : #include <rstan/io/r_ostream.hpp>
##   10 : #include <rstan/stan_args.hpp>
##   11 : #include <boost/integer/integer_log2.hpp>
##   12 : // Code generated by Stan version 2.21.0
##   13 :
##   14 : #include <stan/model/model_header.hpp>
##   15 :
##   16 : namespace model39916aa5bb6f_24b938afe5b29efe8d963b6430c621be_namespace {
##   17 :
##   18 : using std::istream;
##   19 : using std::string;
##   20 : using std::stringstream;
##   21 : using std::vector;
##   22 : using stan::io::dump;
##   23 : using stan::math::lgamma;
##   24 : using stan::model::prob_grad;
##   25 : using namespace stan::math;
##   26 :
##   27 : static int current_statement_begin__;
##   28 :
##   29 : stan::io::program_reader prog_reader__() {
##   30 :     stan::io::program_reader reader;
##   31 :     reader.add_event(0, 0, "start", "model39916aa5bb6f_24b938afe5b29efe8d963b6430c621be");
##   32 :     reader.add_event(57, 55, "end", "model39916aa5bb6f_24b938afe5b29efe8d963b6430c621be");
```

```
##   33 :     return reader;
##   34 : }
##   35 :
##   36 : class model39916aa5bb6f_24b938afe5b29efe8d963b6430c621be
##   37 :   : public stan::model::model_base_crtp<model39916aa5bb6f_24b938afe5b29efe8d963b6430c621be> {
##   38 : private:
##   39 :         int N;
##   40 :         int n;
##   41 :         std::vector<int> Ni;
##   42 :         std::vector<int> y;
##   43 :         std::vector<double> x1;
##   44 :         std::vector<int> id;
##   45 : public:
##   46 :     model39916aa5bb6f_24b938afe5b29efe8d963b6430c621be(rstan::io::rlist_ref_var_context& cont
##   47 :         std::ostream* pstream__ = 0)
##   48 :         : model_base_crtp(0) {
##   49 :         ctor_body(context__, 0, pstream__);
##   50 :     }
##   51 :
##   52 :     model39916aa5bb6f_24b938afe5b29efe8d963b6430c621be(stan::io::var_context& context__,
##   53 :         unsigned int random_seed__,
##   54 :         std::ostream* pstream__ = 0)
##   55 :         : model_base_crtp(0) {
##   56 :         ctor_body(context__, random_seed__, pstream__);
##   57 :     }
##   58 :
##   59 :     void ctor_body(stan::io::var_context& context__,
##   60 :                    unsigned int random_seed__,
##   61 :                    std::ostream* pstream__) {
##   62 :         typedef double local_scalar_t__;
##   63 :
##   64 :         boost::ecuyer1988 base_rng__ =
##   65 :           stan::services::util::create_rng(random_seed__, 0);
##   66 :         (void) base_rng__;  // suppress unused var warning
##   67 :
##   68 :         current_statement_begin__ = -1;
##   69 :
##   70 :         static const char* function__ = "model39916aa5bb6f_24b938afe5b29efe8d963b6430c621be_na
##   71 :         (void) function__;  // dummy to suppress unused var warning
##   72 :         size_t pos__;
##   73 :         (void) pos__;  // dummy to suppress unused var warning
##   74 :         std::vector<int> vals_i__;
##   75 :         std::vector<double> vals_r__;
##   76 :         local_scalar_t__ DUMMY_VAR__(std::numeric_limits<double>::quiet_NaN());
##   77 :         (void) DUMMY_VAR__;  // suppress unused var warning
##   78 :
##   79 :         try {
##   80 :             // initialize data block variables from context__
##   81 :             current_statement_begin__ = 8;
##   82 :             context__.validate_dims("data initialization", "N", "int", context__.to_vec());
##   83 :             N = int(0);
##   84 :             vals_i__ = context__.vals_i("N");
##   85 :             pos__ = 0;
##   86 :             N = vals_i__[pos__++];
```

```
##   87 :             check_greater_or_equal(function__, "N", N, 0);
##   88 :
##   89 :             current_statement_begin__ = 9;
##   90 :             context__.validate_dims("data initialization", "n", "int", context__.to_vec());
##   91 :             n = int(0);
##   92 :             vals_i__ = context__.vals_i("n");
##   93 :             pos__ = 0;
##   94 :             n = vals_i__[pos__++];
##   95 :             check_greater_or_equal(function__, "n", n, 0);
##   96 :
##   97 :             current_statement_begin__ = 10;
##   98 :             validate_non_negative_index("Ni", "(N + 1)", (N + 1));
##   99 :             context__.validate_dims("data initialization", "Ni", "int", context__.to_vec((N +
##  100 :             Ni = std::vector<int>((N + 1), int(0));
##  101 :             vals_i__ = context__.vals_i("Ni");
##  102 :             pos__ = 0;
##  103 :             size_t Ni_k_0_max__ = (N + 1);
##  104 :             for (size_t k_0__ = 0; k_0__ < Ni_k_0_max__; ++k_0__) {
##  105 :                 Ni[k_0__] = vals_i__[pos__++];
##  106 :             }
##  107 :             size_t Ni_i_0_max__ = (N + 1);
##  108 :             for (size_t i_0__ = 0; i_0__ < Ni_i_0_max__; ++i_0__) {
##  109 :                 check_greater_or_equal(function__, "Ni[i_0__]", Ni[i_0__], 0);
##  110 :             }
##  111 :
##  112 :             current_statement_begin__ = 11;
##  113 :             validate_non_negative_index("y", "n", n);
##  114 :             context__.validate_dims("data initialization", "y", "int", context__.to_vec(n));
##  115 :             y = std::vector<int>(n, int(0));
##  116 :             vals_i__ = context__.vals_i("y");
##  117 :             pos__ = 0;
##  118 :             size_t y_k_0_max__ = n;
##  119 :             for (size_t k_0__ = 0; k_0__ < y_k_0_max__; ++k_0__) {
##  120 :                 y[k_0__] = vals_i__[pos__++];
##  121 :             }
##  122 :             size_t y_i_0_max__ = n;
##  123 :             for (size_t i_0__ = 0; i_0__ < y_i_0_max__; ++i_0__) {
##  124 :                 check_greater_or_equal(function__, "y[i_0__]", y[i_0__], 1);
##  125 :                 check_less_or_equal(function__, "y[i_0__]", y[i_0__], 4);
##  126 :             }
##  127 :
##  128 :             current_statement_begin__ = 12;
##  129 :             validate_non_negative_index("x1", "n", n);
##  130 :             context__.validate_dims("data initialization", "x1", "double", context__.to_vec(n
##  131 :             x1 = std::vector<double>(n, double(0));
##  132 :             vals_r__ = context__.vals_r("x1");
##  133 :             pos__ = 0;
##  134 :             size_t x1_k_0_max__ = n;
##  135 :             for (size_t k_0__ = 0; k_0__ < x1_k_0_max__; ++k_0__) {
##  136 :                 x1[k_0__] = vals_r__[pos__++];
##  137 :             }
##  138 :
##  139 :             current_statement_begin__ = 13;
##  140 :             validate_non_negative_index("id", "n", n);
```

```
##  141 :             context__.validate_dims("data initialization", "id", "int", context__.to_vec(n));
##  142 :             id = std::vector<int>(n, int(0));
##  143 :             vals_i__ = context__.vals_i("id");
##  144 :             pos__ = 0;
##  145 :             size_t id_k_0_max__ = n;
##  146 :             for (size_t k_0__ = 0; k_0__ < id_k_0_max__; ++k_0__) {
##  147 :                 id[k_0__] = vals_i__[pos__++];
##  148 :             }
##  149 :             size_t id_i_0_max__ = n;
##  150 :             for (size_t i_0__ = 0; i_0__ < id_i_0_max__; ++i_0__) {
##  151 :                 check_greater_or_equal(function__, "id[i_0__]", id[i_0__], 1);
##  152 :             }
##  153 :
##  154 :
##  155 :             // initialize transformed data variables
##  156 :             // execute transformed data statements
##  157 :
##  158 :             // validate transformed data
##  159 :
##  160 :             // validate, set parameter ranges
##  161 :             num_params_r__ = 0U;
##  162 :             param_ranges_i__.clear();
##  163 :             current_statement_begin__ = 17;
##  164 :             validate_non_negative_index("kappa", "3", 3);
##  165 :             num_params_r__ += 3;
##  166 :             current_statement_begin__ = 18;
##  167 :             num_params_r__ += 1;
##  168 :             current_statement_begin__ = 19;
##  169 :             validate_non_negative_index("bre1", "N", N);
##  170 :             num_params_r__ += (1 * N);
##  171 :             current_statement_begin__ = 20;
##  172 :             num_params_r__ += 1;
##  173 :         } catch (const std::exception& e) {
##  174 :             stan::lang::rethrow_located(e, current_statement_begin__, prog_reader__());
##  175 :             // Next line prevents compiler griping about no return
##  176 :             throw std::runtime_error("*** IF YOU SEE THIS, PLEASE REPORT A BUG ***");
##  177 :         }
##  178 :     }
##  179 :
##  180 :     ~model39916aa5bb6f_24b938afe5b29efe8d963b6430c621be() { }
##  181 :
##  182 :
##  183 :     void transform_inits(const stan::io::var_context& context__,
##  184 :                          std::vector<int>& params_i__,
##  185 :                          std::vector<double>& params_r__,
##  186 :                          std::ostream* pstream__) const {
##  187 :         typedef double local_scalar_t__;
##  188 :         stan::io::writer<double> writer__(params_r__, params_i__);
##  189 :         size_t pos__;
##  190 :         (void) pos__; // dummy call to supress warning
##  191 :         std::vector<double> vals_r__;
##  192 :         std::vector<int> vals_i__;
##  193 :
##  194 :         current_statement_begin__ = 17;
```

```
##  195 :         if (!(context__.contains_r("kappa")))
##  196 :             stan::lang::rethrow_located(std::runtime_error(std::string("Variable kappa missing
##  197 :         vals_r__ = context__.vals_r("kappa");
##  198 :         pos__ = 0U;
##  199 :         validate_non_negative_index("kappa", "3", 3);
##  200 :         context__.validate_dims("parameter initialization", "kappa", "vector_d", context__.to_
##  201 :         Eigen::Matrix<double, Eigen::Dynamic, 1> kappa(3);
##  202 :         size_t kappa_j_1_max__ = 3;
##  203 :         for (size_t j_1__ = 0; j_1__ < kappa_j_1_max__; ++j_1__) {
##  204 :             kappa(j_1__) = vals_r__[pos__++];
##  205 :         }
##  206 :         try {
##  207 :             writer__.ordered_unconstrain(kappa);
##  208 :         } catch (const std::exception& e) {
##  209 :             stan::lang::rethrow_located(std::runtime_error(std::string("Error transforming va
##  210 :         }
##  211 :
##  212 :         current_statement_begin__ = 18;
##  213 :         if (!(context__.contains_r("b1")))
##  214 :             stan::lang::rethrow_located(std::runtime_error(std::string("Variable b1 missing"))
##  215 :         vals_r__ = context__.vals_r("b1");
##  216 :         pos__ = 0U;
##  217 :         context__.validate_dims("parameter initialization", "b1", "double", context__.to_vec()
##  218 :         double b1(0);
##  219 :         b1 = vals_r__[pos__++];
##  220 :         try {
##  221 :             writer__.scalar_unconstrain(b1);
##  222 :         } catch (const std::exception& e) {
##  223 :             stan::lang::rethrow_located(std::runtime_error(std::string("Error transforming va
##  224 :         }
##  225 :
##  226 :         current_statement_begin__ = 19;
##  227 :         if (!(context__.contains_r("bre1")))
##  228 :             stan::lang::rethrow_located(std::runtime_error(std::string("Variable bre1 missing
##  229 :         vals_r__ = context__.vals_r("bre1");
##  230 :         pos__ = 0U;
##  231 :         validate_non_negative_index("bre1", "N", N);
##  232 :         context__.validate_dims("parameter initialization", "bre1", "double", context__.to_ve
##  233 :         std::vector<double> bre1(N, double(0));
##  234 :         size_t bre1_k_0_max__ = N;
##  235 :         for (size_t k_0__ = 0; k_0__ < bre1_k_0_max__; ++k_0__) {
##  236 :             bre1[k_0__] = vals_r__[pos__++];
##  237 :         }
##  238 :         size_t bre1_i_0_max__ = N;
##  239 :         for (size_t i_0__ = 0; i_0__ < bre1_i_0_max__; ++i_0__) {
##  240 :             try {
##  241 :                 writer__.scalar_unconstrain(bre1[i_0__]);
##  242 :             } catch (const std::exception& e) {
##  243 :                 stan::lang::rethrow_located(std::runtime_error(std::string("Error transformin
##  244 :             }
##  245 :         }
##  246 :
##  247 :         current_statement_begin__ = 20;
##  248 :         if (!(context__.contains_r("invsig2")))
```

```
##  249 :             stan::lang::rethrow_located(std::runtime_error(std::string("Variable invsig2 missi
##  250 :         vals_r__ = context__.vals_r("invsig2");
##  251 :         pos__ = 0U;
##  252 :         context__.validate_dims("parameter initialization", "invsig2", "double", context__.to
##  253 :         double invsig2(0);
##  254 :         invsig2 = vals_r__[pos__++];
##  255 :         try {
##  256 :             writer__.scalar_lb_unconstrain(0, invsig2);
##  257 :         } catch (const std::exception& e) {
##  258 :             stan::lang::rethrow_located(std::runtime_error(std::string("Error transforming va
##  259 :         }
##  260 :
##  261 :         params_r__ = writer__.data_r();
##  262 :         params_i__ = writer__.data_i();
##  263 :     }
##  264 :
##  265 :     void transform_inits(const stan::io::var_context& context,
##  266 :                          Eigen::Matrix<double, Eigen::Dynamic, 1>& params_r,
##  267 :                          std::ostream* pstream__) const {
##  268 :       std::vector<double> params_r_vec;
##  269 :       std::vector<int> params_i_vec;
##  270 :       transform_inits(context, params_i_vec, params_r_vec, pstream__);
##  271 :       params_r.resize(params_r_vec.size());
##  272 :       for (int i = 0; i < params_r.size(); ++i)
##  273 :         params_r(i) = params_r_vec[i];
##  274 :     }
##  275 :
##  276 :
##  277 :     template <bool propto__, bool jacobian__, typename T__>
##  278 :     T__ log_prob(std::vector<T__>& params_r__,
##  279 :                  std::vector<int>& params_i__,
##  280 :                  std::ostream* pstream__ = 0) const {
##  281 :
##  282 :         typedef T__ local_scalar_t__;
##  283 :
##  284 :         local_scalar_t__ DUMMY_VAR__(std::numeric_limits<double>::quiet_NaN());
##  285 :         (void) DUMMY_VAR__;  // dummy to suppress unused var warning
##  286 :
##  287 :         T__ lp__(0.0);
##  288 :         stan::math::accumulator<T__> lp_accum__;
##  289 :         try {
##  290 :             stan::io::reader<local_scalar_t__> in__(params_r__, params_i__);
##  291 :
##  292 :             // model parameters
##  293 :             current_statement_begin__ = 17;
##  294 :             Eigen::Matrix<local_scalar_t__, Eigen::Dynamic, 1> kappa;
##  295 :             (void) kappa;  // dummy to suppress unused var warning
##  296 :             if (jacobian__)
##  297 :                 kappa = in__.ordered_constrain(3, lp__);
##  298 :             else
##  299 :                 kappa = in__.ordered_constrain(3);
##  300 :
##  301 :             current_statement_begin__ = 18;
##  302 :             local_scalar_t__ b1;
```

```
##  303 :             (void) b1;  // dummy to suppress unused var warning
##  304 :             if (jacobian__)
##  305 :                 b1 = in__.scalar_constrain(lp__);
##  306 :             else
##  307 :                 b1 = in__.scalar_constrain();
##  308 :
##  309 :             current_statement_begin__ = 19;
##  310 :             std::vector<local_scalar_t__> bre1;
##  311 :             size_t bre1_d_0_max__ = N;
##  312 :             bre1.reserve(bre1_d_0_max__);
##  313 :             for (size_t d_0__ = 0; d_0__ < bre1_d_0_max__; ++d_0__) {
##  314 :                 if (jacobian__)
##  315 :                     bre1.push_back(in__.scalar_constrain(lp__));
##  316 :                 else
##  317 :                     bre1.push_back(in__.scalar_constrain());
##  318 :             }
##  319 :
##  320 :             current_statement_begin__ = 20;
##  321 :             local_scalar_t__ invsig2;
##  322 :             (void) invsig2;  // dummy to suppress unused var warning
##  323 :             if (jacobian__)
##  324 :                 invsig2 = in__.scalar_lb_constrain(0, lp__);
##  325 :             else
##  326 :                 invsig2 = in__.scalar_lb_constrain(0);
##  327 :
##  328 :             // transformed parameters
##  329 :             current_statement_begin__ = 24;
##  330 :             validate_non_negative_index("mu", "n", n);
##  331 :             Eigen::Matrix<local_scalar_t__, Eigen::Dynamic, 1> mu(n);
##  332 :             stan::math::initialize(mu, DUMMY_VAR__);
##  333 :             stan::math::fill(mu, DUMMY_VAR__);
##  334 :
##  335 :             current_statement_begin__ = 25;
##  336 :             local_scalar_t__ sig2;
##  337 :             (void) sig2;  // dummy to suppress unused var warning
##  338 :             stan::math::initialize(sig2, DUMMY_VAR__);
##  339 :             stan::math::fill(sig2, DUMMY_VAR__);
##  340 :
##  341 :             current_statement_begin__ = 26;
##  342 :             local_scalar_t__ sigma;
##  343 :             (void) sigma;  // dummy to suppress unused var warning
##  344 :             stan::math::initialize(sigma, DUMMY_VAR__);
##  345 :             stan::math::fill(sigma, DUMMY_VAR__);
##  346 :
##  347 :             // transformed parameters block statements
##  348 :             current_statement_begin__ = 27;
##  349 :             for (int i = 1; i <= N; ++i) {
##  350 :
##  351 :                 current_statement_begin__ = 28;
##  352 :                 for (int t = get_base1(Ni, i, "Ni", 1); t <= (get_base1(Ni, (i + 1), "Ni", 1)
##  353 :
##  354 :                     current_statement_begin__ = 29;
##  355 :                     stan::model::assign(mu,
##  356 :                                 stan::model::cons_list(stan::model::index_uni(t), stan::model
```

```
##  357 :                                ((get_base1(x1, t, "x1", 1) * b1) + (get_base1(x1, t, "x1", 1
##  358 :                                "assigning variable mu");
##  359 :                }
##  360 :            }
##  361 :            current_statement_begin__ = 32;
##  362 :            stan::math::assign(sig2, (1 / invsig2));
##  363 :            current_statement_begin__ = 33;
##  364 :            stan::math::assign(sigma, pow(sig2, 0.5));
##  365 :
##  366 :            // validate transformed parameters
##  367 :            const char* function__ = "validate transformed params";
##  368 :            (void) function__;  // dummy to suppress unused var warning
##  369 :
##  370 :            current_statement_begin__ = 24;
##  371 :            size_t mu_j_1_max__ = n;
##  372 :            for (size_t j_1__ = 0; j_1__ < mu_j_1_max__; ++j_1__) {
##  373 :                if (stan::math::is_uninitialized(mu(j_1__))) {
##  374 :                    std::stringstream msg__;
##  375 :                    msg__ << "Undefined transformed parameter: mu" << "(" << j_1__ << ")";
##  376 :                    stan::lang::rethrow_located(std::runtime_error(std::string("Error initial
##  377 :                }
##  378 :            }
##  379 :            current_statement_begin__ = 25;
##  380 :            if (stan::math::is_uninitialized(sig2)) {
##  381 :                std::stringstream msg__;
##  382 :                msg__ << "Undefined transformed parameter: sig2";
##  383 :                stan::lang::rethrow_located(std::runtime_error(std::string("Error initializing
##  384 :            }
##  385 :            check_greater_or_equal(function__, "sig2", sig2, 0);
##  386 :
##  387 :            current_statement_begin__ = 26;
##  388 :            if (stan::math::is_uninitialized(sigma)) {
##  389 :                std::stringstream msg__;
##  390 :                msg__ << "Undefined transformed parameter: sigma";
##  391 :                stan::lang::rethrow_located(std::runtime_error(std::string("Error initializing
##  392 :            }
##  393 :            check_greater_or_equal(function__, "sigma", sigma, 0);
##  394 :
##  395 :
##  396 :            // model body
##  397 :
##  398 :            current_statement_begin__ = 37;
##  399 :            lp_accum__.add(gamma_log<propto__>(invsig2, .05, .005));
##  400 :            current_statement_begin__ = 38;
##  401 :            lp_accum__.add(normal_log<propto__>(b1, 0, 9.9e+06));
##  402 :            current_statement_begin__ = 39;
##  403 :            for (int i = 1; i <= N; ++i) {
##  404 :
##  405 :                current_statement_begin__ = 40;
##  406 :                lp_accum__.add(normal_log<propto__>(get_base1(bre1, i, "bre1", 1), 0, sigma))
##  407 :                current_statement_begin__ = 41;
##  408 :                for (int t = get_base1(Ni, i, "Ni", 1); t <= (get_base1(Ni, (i + 1), "Ni", 1)
##  409 :
##  410 :                    current_statement_begin__ = 42;
```

```
##  411 :                     lp_accum__.add(ordered_logistic_log<propto__>(get_base1(y, t, "y", 1), ge
##  412 :                 }
##  413 :             }
##  414 :
##  415 :         } catch (const std::exception& e) {
##  416 :             stan::lang::rethrow_located(e, current_statement_begin__, prog_reader__());
##  417 :             // Next line prevents compiler griping about no return
##  418 :             throw std::runtime_error("*** IF YOU SEE THIS, PLEASE REPORT A BUG ***");
##  419 :         }
##  420 :
##  421 :         lp_accum__.add(lp__);
##  422 :         return lp_accum__.sum();
##  423 :
##  424 :     } // log_prob()
##  425 :
##  426 :     template <bool propto, bool jacobian, typename T_>
##  427 :     T_ log_prob(Eigen::Matrix<T_,Eigen::Dynamic,1>& params_r,
##  428 :                 std::ostream* pstream = 0) const {
##  429 :       std::vector<T_> vec_params_r;
##  430 :       vec_params_r.reserve(params_r.size());
##  431 :       for (int i = 0; i < params_r.size(); ++i)
##  432 :         vec_params_r.push_back(params_r(i));
##  433 :       std::vector<int> vec_params_i;
##  434 :       return log_prob<propto,jacobian,T_>(vec_params_r, vec_params_i, pstream);
##  435 :     }
##  436 :
##  437 :
##  438 :     void get_param_names(std::vector<std::string>& names__) const {
##  439 :         names__.resize(0);
##  440 :         names__.push_back("kappa");
##  441 :         names__.push_back("b1");
##  442 :         names__.push_back("bre1");
##  443 :         names__.push_back("invsig2");
##  444 :         names__.push_back("mu");
##  445 :         names__.push_back("sig2");
##  446 :         names__.push_back("sigma");
##  447 :         names__.push_back("log_lik");
##  448 :     }
##  449 :
##  450 :
##  451 :     void get_dims(std::vector<std::vector<size_t> >& dimss__) const {
##  452 :         dimss__.resize(0);
##  453 :         std::vector<size_t> dims__;
##  454 :         dims__.resize(0);
##  455 :         dims__.push_back(3);
##  456 :         dimss__.push_back(dims__);
##  457 :         dims__.resize(0);
##  458 :         dimss__.push_back(dims__);
##  459 :         dims__.resize(0);
##  460 :         dims__.push_back(N);
##  461 :         dimss__.push_back(dims__);
##  462 :         dims__.resize(0);
##  463 :         dimss__.push_back(dims__);
##  464 :         dims__.resize(0);
```

```
##  465 :         dims__.push_back(n);
##  466 :         dimss__.push_back(dims__);
##  467 :         dims__.resize(0);
##  468 :         dimss__.push_back(dims__);
##  469 :         dims__.resize(0);
##  470 :         dimss__.push_back(dims__);
##  471 :         dims__.resize(0);
##  472 :         dims__.push_back(n);
##  473 :         dimss__.push_back(dims__);
##  474 :     }
##  475 :
##  476 :     template <typename RNG>
##  477 :     void write_array(RNG& base_rng__,
##  478 :                      std::vector<double>& params_r__,
##  479 :                      std::vector<int>& params_i__,
##  480 :                      std::vector<double>& vars__,
##  481 :                      bool include_tparams__ = true,
##  482 :                      bool include_gqs__ = true,
##  483 :                      std::ostream* pstream__ = 0) const {
##  484 :         typedef double local_scalar_t__;
##  485 :
##  486 :         vars__.resize(0);
##  487 :         stan::io::reader<local_scalar_t__> in__(params_r__, params_i__);
##  488 :         static const char* function__ = "model39916aa5bb6f_24b938afe5b29efe8d963b6430c621be_na
##  489 :         (void) function__;  // dummy to suppress unused var warning
##  490 :
##  491 :         // read-transform, write parameters
##  492 :         Eigen::Matrix<double, Eigen::Dynamic, 1> kappa = in__.ordered_constrain(3);
##  493 :         size_t kappa_j_1_max__ = 3;
##  494 :         for (size_t j_1__ = 0; j_1__ < kappa_j_1_max__; ++j_1__) {
##  495 :             vars__.push_back(kappa(j_1__));
##  496 :         }
##  497 :
##  498 :         double b1 = in__.scalar_constrain();
##  499 :         vars__.push_back(b1);
##  500 :
##  501 :         std::vector<double> bre1;
##  502 :         size_t bre1_d_0_max__ = N;
##  503 :         bre1.reserve(bre1_d_0_max__);
##  504 :         for (size_t d_0__ = 0; d_0__ < bre1_d_0_max__; ++d_0__) {
##  505 :             bre1.push_back(in__.scalar_constrain());
##  506 :         }
##  507 :         size_t bre1_k_0_max__ = N;
##  508 :         for (size_t k_0__ = 0; k_0__ < bre1_k_0_max__; ++k_0__) {
##  509 :             vars__.push_back(bre1[k_0__]);
##  510 :         }
##  511 :
##  512 :         double invsig2 = in__.scalar_lb_constrain(0);
##  513 :         vars__.push_back(invsig2);
##  514 :
##  515 :         double lp__ = 0.0;
##  516 :         (void) lp__;  // dummy to suppress unused var warning
##  517 :         stan::math::accumulator<double> lp_accum__;
##  518 :
```

```
##  519 :         local_scalar_t__ DUMMY_VAR__(std::numeric_limits<double>::quiet_NaN());
##  520 :         (void) DUMMY_VAR__;  // suppress unused var warning
##  521 :
##  522 :         if (!include_tparams__ && !include_gqs__) return;
##  523 :
##  524 :         try {
##  525 :             // declare and define transformed parameters
##  526 :             current_statement_begin__ = 24;
##  527 :             validate_non_negative_index("mu", "n", n);
##  528 :             Eigen::Matrix<double, Eigen::Dynamic, 1> mu(n);
##  529 :             stan::math::initialize(mu, DUMMY_VAR__);
##  530 :             stan::math::fill(mu, DUMMY_VAR__);
##  531 :
##  532 :             current_statement_begin__ = 25;
##  533 :             double sig2;
##  534 :             (void) sig2;  // dummy to suppress unused var warning
##  535 :             stan::math::initialize(sig2, DUMMY_VAR__);
##  536 :             stan::math::fill(sig2, DUMMY_VAR__);
##  537 :
##  538 :             current_statement_begin__ = 26;
##  539 :             double sigma;
##  540 :             (void) sigma;  // dummy to suppress unused var warning
##  541 :             stan::math::initialize(sigma, DUMMY_VAR__);
##  542 :             stan::math::fill(sigma, DUMMY_VAR__);
##  543 :
##  544 :             // do transformed parameters statements
##  545 :             current_statement_begin__ = 27;
##  546 :             for (int i = 1; i <= N; ++i) {
##  547 :
##  548 :                 current_statement_begin__ = 28;
##  549 :                 for (int t = get_base1(Ni, i, "Ni", 1); t <= (get_base1(Ni, (i + 1), "Ni", 1)
##  550 :
##  551 :                     current_statement_begin__ = 29;
##  552 :                     stan::model::assign(mu,
##  553 :                                 stan::model::cons_list(stan::model::index_uni(t), stan::model
##  554 :                                 ((get_base1(x1, t, "x1", 1) * b1) + (get_base1(x1, t, "x1", 1
##  555 :                                 "assigning variable mu");
##  556 :                 }
##  557 :             }
##  558 :             current_statement_begin__ = 32;
##  559 :             stan::math::assign(sig2, (1 / invsig2));
##  560 :             current_statement_begin__ = 33;
##  561 :             stan::math::assign(sigma, pow(sig2, 0.5));
##  562 :
##  563 :             if (!include_gqs__ && !include_tparams__) return;
##  564 :             // validate transformed parameters
##  565 :             const char* function__ = "validate transformed params";
##  566 :             (void) function__;  // dummy to suppress unused var warning
##  567 :
##  568 :             current_statement_begin__ = 25;
##  569 :             check_greater_or_equal(function__, "sig2", sig2, 0);
##  570 :
##  571 :             current_statement_begin__ = 26;
##  572 :             check_greater_or_equal(function__, "sigma", sigma, 0);
```

```
##  573 :
##  574 :             // write transformed parameters
##  575 :             if (include_tparams__) {
##  576 :                 size_t mu_j_1_max__ = n;
##  577 :                 for (size_t j_1__ = 0; j_1__ < mu_j_1_max__; ++j_1__) {
##  578 :                     vars__.push_back(mu(j_1__));
##  579 :                 }
##  580 :                 vars__.push_back(sig2);
##  581 :                 vars__.push_back(sigma);
##  582 :             }
##  583 :             if (!include_gqs__) return;
##  584 :             // declare and define generated quantities
##  585 :             current_statement_begin__ = 49;
##  586 :             validate_non_negative_index("log_lik", "n", n);
##  587 :             Eigen::Matrix<double, Eigen::Dynamic, 1> log_lik(n);
##  588 :             stan::math::initialize(log_lik, DUMMY_VAR__);
##  589 :             stan::math::fill(log_lik, DUMMY_VAR__);
##  590 :
##  591 :             // generated quantities statements
##  592 :             current_statement_begin__ = 50;
##  593 :             for (int i = 1; i <= N; ++i) {
##  594 :
##  595 :                 current_statement_begin__ = 51;
##  596 :                 for (int t = get_base1(Ni, i, "Ni", 1); t <= (get_base1(Ni, (i + 1), "Ni", 1)
##  597 :
##  598 :                     current_statement_begin__ = 52;
##  599 :                     stan::model::assign(log_lik,
##  600 :                                 stan::model::cons_list(stan::model::index_uni(t), stan::model
##  601 :                                 ordered_logistic_log(get_base1(y, t, "y", 1), get_base1(mu, t
##  602 :                                 "assigning variable log_lik");
##  603 :                 }
##  604 :             }
##  605 :
##  606 :             // validate, write generated quantities
##  607 :             current_statement_begin__ = 49;
##  608 :             size_t log_lik_j_1_max__ = n;
##  609 :             for (size_t j_1__ = 0; j_1__ < log_lik_j_1_max__; ++j_1__) {
##  610 :                 vars__.push_back(log_lik(j_1__));
##  611 :             }
##  612 :
##  613 :         } catch (const std::exception& e) {
##  614 :             stan::lang::rethrow_located(e, current_statement_begin__, prog_reader__());
##  615 :             // Next line prevents compiler griping about no return
##  616 :             throw std::runtime_error("*** IF YOU SEE THIS, PLEASE REPORT A BUG ***");
##  617 :         }
##  618 :     }
##  619 :
##  620 :     template <typename RNG>
##  621 :     void write_array(RNG& base_rng,
##  622 :                      Eigen::Matrix<double,Eigen::Dynamic,1>& params_r,
##  623 :                      Eigen::Matrix<double,Eigen::Dynamic,1>& vars,
##  624 :                      bool include_tparams = true,
##  625 :                      bool include_gqs = true,
##  626 :                      std::ostream* pstream = 0) const {
```

```
##  627 :       std::vector<double> params_r_vec(params_r.size());
##  628 :       for (int i = 0; i < params_r.size(); ++i)
##  629 :         params_r_vec[i] = params_r(i);
##  630 :       std::vector<double> vars_vec;
##  631 :       std::vector<int> params_i_vec;
##  632 :       write_array(base_rng, params_r_vec, params_i_vec, vars_vec, include_tparams, include_gqs
##  633 :       vars.resize(vars_vec.size());
##  634 :       for (int i = 0; i < vars.size(); ++i)
##  635 :         vars(i) = vars_vec[i];
##  636 :     }
##  637 :
##  638 :     std::string model_name() const {
##  639 :         return "model39916aa5bb6f_24b938afe5b29efe8d963b6430c621be";
##  640 :     }
##  641 :
##  642 :
##  643 :     void constrained_param_names(std::vector<std::string>& param_names__,
##  644 :                                  bool include_tparams__ = true,
##  645 :                                  bool include_gqs__ = true) const {
##  646 :         std::stringstream param_name_stream__;
##  647 :         size_t kappa_j_1_max__ = 3;
##  648 :         for (size_t j_1__ = 0; j_1__ < kappa_j_1_max__; ++j_1__) {
##  649 :             param_name_stream__.str(std::string());
##  650 :             param_name_stream__ << "kappa" << '.' << j_1__ + 1;
##  651 :             param_names__.push_back(param_name_stream__.str());
##  652 :         }
##  653 :         param_name_stream__.str(std::string());
##  654 :         param_name_stream__ << "b1";
##  655 :         param_names__.push_back(param_name_stream__.str());
##  656 :         size_t bre1_k_0_max__ = N;
##  657 :         for (size_t k_0__ = 0; k_0__ < bre1_k_0_max__; ++k_0__) {
##  658 :             param_name_stream__.str(std::string());
##  659 :             param_name_stream__ << "bre1" << '.' << k_0__ + 1;
##  660 :             param_names__.push_back(param_name_stream__.str());
##  661 :         }
##  662 :         param_name_stream__.str(std::string());
##  663 :         param_name_stream__ << "invsig2";
##  664 :         param_names__.push_back(param_name_stream__.str());
##  665 :
##  666 :         if (!include_gqs__ && !include_tparams__) return;
##  667 :
##  668 :         if (include_tparams__) {
##  669 :             size_t mu_j_1_max__ = n;
##  670 :             for (size_t j_1__ = 0; j_1__ < mu_j_1_max__; ++j_1__) {
##  671 :                 param_name_stream__.str(std::string());
##  672 :                 param_name_stream__ << "mu" << '.' << j_1__ + 1;
##  673 :                 param_names__.push_back(param_name_stream__.str());
##  674 :             }
##  675 :             param_name_stream__.str(std::string());
##  676 :             param_name_stream__ << "sig2";
##  677 :             param_names__.push_back(param_name_stream__.str());
##  678 :             param_name_stream__.str(std::string());
##  679 :             param_name_stream__ << "sigma";
##  680 :             param_names__.push_back(param_name_stream__.str());
```

```
##  681 :         }
##  682 :
##  683 :         if (!include_gqs__) return;
##  684 :         size_t log_lik_j_1_max__ = n;
##  685 :         for (size_t j_1__ = 0; j_1__ < log_lik_j_1_max__; ++j_1__) {
##  686 :             param_name_stream__.str(std::string());
##  687 :             param_name_stream__ << "log_lik" << '.' << j_1__ + 1;
##  688 :             param_names__.push_back(param_name_stream__.str());
##  689 :         }
##  690 :     }
##  691 :
##  692 :
##  693 :     void unconstrained_param_names(std::vector<std::string>& param_names__,
##  694 :                                    bool include_tparams__ = true,
##  695 :                                    bool include_gqs__ = true) const {
##  696 :         std::stringstream param_name_stream__;
##  697 :         size_t kappa_j_1_max__ = 3;
##  698 :         for (size_t j_1__ = 0; j_1__ < kappa_j_1_max__; ++j_1__) {
##  699 :             param_name_stream__.str(std::string());
##  700 :             param_name_stream__ << "kappa" << '.' << j_1__ + 1;
##  701 :             param_names__.push_back(param_name_stream__.str());
##  702 :         }
##  703 :         param_name_stream__.str(std::string());
##  704 :         param_name_stream__ << "b1";
##  705 :         param_names__.push_back(param_name_stream__.str());
##  706 :         size_t bre1_k_0_max__ = N;
##  707 :         for (size_t k_0__ = 0; k_0__ < bre1_k_0_max__; ++k_0__) {
##  708 :             param_name_stream__.str(std::string());
##  709 :             param_name_stream__ << "bre1" << '.' << k_0__ + 1;
##  710 :             param_names__.push_back(param_name_stream__.str());
##  711 :         }
##  712 :         param_name_stream__.str(std::string());
##  713 :         param_name_stream__ << "invsig2";
##  714 :         param_names__.push_back(param_name_stream__.str());
##  715 :
##  716 :         if (!include_gqs__ && !include_tparams__) return;
##  717 :
##  718 :         if (include_tparams__) {
##  719 :             size_t mu_j_1_max__ = n;
##  720 :             for (size_t j_1__ = 0; j_1__ < mu_j_1_max__; ++j_1__) {
##  721 :                 param_name_stream__.str(std::string());
##  722 :                 param_name_stream__ << "mu" << '.' << j_1__ + 1;
##  723 :                 param_names__.push_back(param_name_stream__.str());
##  724 :             }
##  725 :             param_name_stream__.str(std::string());
##  726 :             param_name_stream__ << "sig2";
##  727 :             param_names__.push_back(param_name_stream__.str());
##  728 :             param_name_stream__.str(std::string());
##  729 :             param_name_stream__ << "sigma";
##  730 :             param_names__.push_back(param_name_stream__.str());
##  731 :         }
##  732 :
##  733 :         if (!include_gqs__) return;
##  734 :         size_t log_lik_j_1_max__ = n;
```

```
##  735 :         for (size_t j_1__ = 0; j_1__ < log_lik_j_1_max__; ++j_1__) {
##  736 :             param_name_stream__.str(std::string());
##  737 :             param_name_stream__ << "log_lik" << '.' << j_1__ + 1;
##  738 :             param_names__.push_back(param_name_stream__.str());
##  739 :         }
##  740 :     }
##  741 :
##  742 : }; // model
##  743 :
##  744 : }  // namespace
##  745 :
##  746 : typedef model39916aa5bb6f_24b938afe5b29efe8d963b6430c621be_namespace::model39916aa5bb6f_24b938
##  747 :
##  748 : #ifndef USING_R
##  749 :
##  750 : stan::model::model_base& new_model(
##  751 :         stan::io::var_context& data_context,
##  752 :         unsigned int seed,
##  753 :         std::ostream* msg_stream) {
##  754 :   stan_model* m = new stan_model(data_context, seed, msg_stream);
##  755 :   return *m;
##  756 : }
##  757 :
##  758 : #endif
##  759 :
##  760 :
##  761 :
##  762 : #include <rstan_next/stan_fit.hpp>
##  763 :
##  764 : struct stan_model_holder {
##  765 :     stan_model_holder(rstan::io::rlist_ref_var_context rcontext,
##  766 :                       unsigned int random_seed)
##  767 :     : rcontext_(rcontext), random_seed_(random_seed)
##  768 :      {
##  769 :      }
##  770 :
##  771 :    //stan::math::ChainableStack ad_stack;
##  772 :    rstan::io::rlist_ref_var_context rcontext_;
##  773 :    unsigned int random_seed_;
##  774 : };
##  775 :
##  776 : Rcpp::XPtr<stan::model::model_base> model_ptr(stan_model_holder* smh) {
##  777 :   Rcpp::XPtr<stan::model::model_base> model_instance(new stan_model(smh->rcontext_, smh->rando
##  778 :   return model_instance;
##  779 : }
##  780 :
##  781 : Rcpp::XPtr<rstan::stan_fit_base> fit_ptr(stan_model_holder* smh) {
##  782 :   return Rcpp::XPtr<rstan::stan_fit_base>(new rstan::stan_fit(model_ptr(smh), smh->random_seed
##  783 : }
##  784 :
##  785 : std::string model_name(stan_model_holder* smh) {
##  786 :   return model_ptr(smh).get()->model_name();
##  787 : }
##  788 :
```

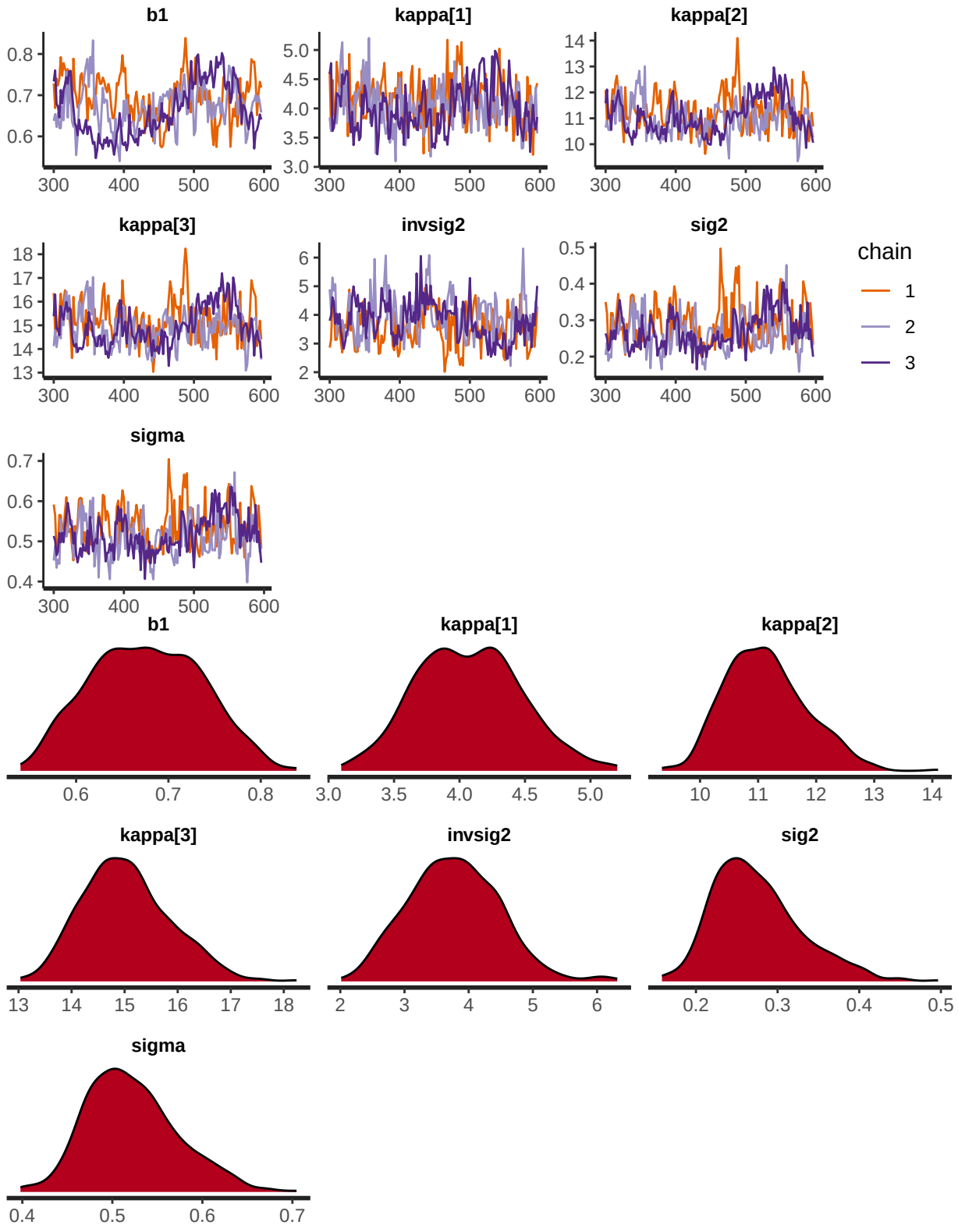
```
##  789 : RCPP_MODULE(stan_fit4model39916aa5bb6f_24b938afe5b29efe8d963b6430c621be_mod){
##  790 :   Rcpp::class_<stan_model_holder>("stan_fit4model39916aa5bb6f_24b938afe5b29efe8d963b6430c621be
##  791 :   .constructor<rstan::io::rlist_ref_var_context, unsigned int>()
##  792 :   .method("model_ptr", &model_ptr)
##  793 :   .method("fit_ptr", &fit_ptr)
##  794 :   .method("model_name", &model_name)
##  795 :   ;
##  796 : }
##  797 :
##  798 :
##  799 : // declarations
##  800 : extern "C" {
##  801 : SEXP file3991130aebc6( ) ;
##  802 : }
##  803 :
##  804 : // definition
##  805 : SEXP file3991130aebc6() {
##  806 :  return Rcpp::wrap("24b938afe5b29efe8d963b6430c621be");
##  807 : }
```

```
fit.lme.non.reslope <- sampling(mod,
                                data=datos.lme,
                                chains=3,warmup=300,iter=600,thin=2,cores=4 )
```

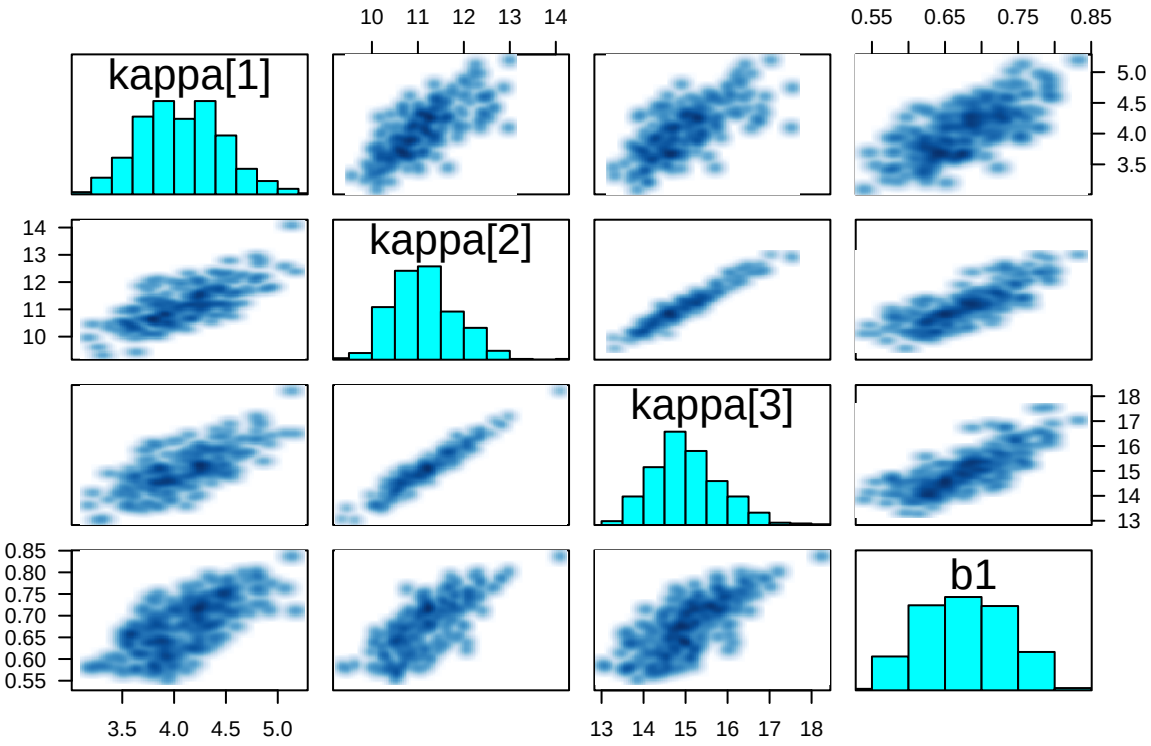
```
print(fit.lme.non.reslope, pars=param.lme)
```

```
## Inference for Stan model: 24b938afe5b29efe8d963b6430c621be.
## 3 chains, each with iter=600; warmup=300; thin=2;
## post-warmup draws per chain=150, total post-warmup draws=450.
##
##           mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## b1        0.68    0.01 0.06  0.57  0.63  0.68  0.72  0.79    25 1.14
## kappa[1]  4.07    0.04 0.40  3.34  3.78  4.06  4.34  4.86    90 1.02
## kappa[2] 11.13    0.09 0.69 10.03 10.64 11.07 11.56 12.57    53 1.04
## kappa[3] 15.03    0.12 0.82 13.63 14.48 14.96 15.51 16.76    50 1.05
## invsig2   3.78    0.08 0.71  2.49  3.30  3.78  4.27  5.18    76 1.05
## sig2      0.27    0.01 0.05  0.19  0.23  0.26  0.30  0.40    78 1.05
## sigma     0.52    0.01 0.05  0.44  0.48  0.51  0.55  0.63    76 1.05
##
## Samples were drawn using NUTS(diag_e) at Sun Aug 13 15:44:26 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

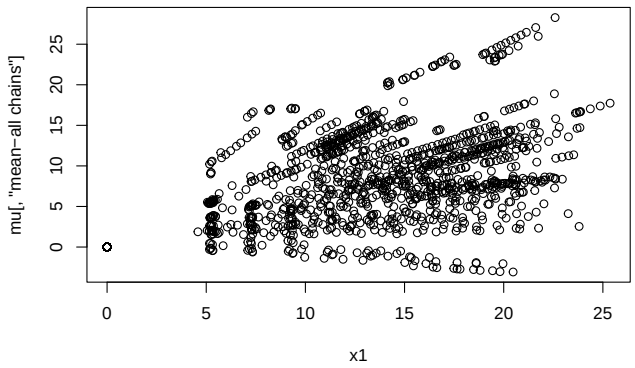
```
stan_trace(fit.lme.non.reslope,pars=param.lme)
stan_dens(fit.lme.non.reslope,pars=param.lme)
```




```
pairs(fit.lme.non.reslope, pars = c("kappa","b1"), las = 1)
```




```
mu=get_posterior_mean(fit.lme.non.reslope,"mu")
plot(x1,mu[,"mean-all chains"])
```



6. Lineal creciente

6.1. Lineal creciente

For a spline-based fit without constraints:

```
stancode <- readLines("jagam_10_aneur_ordinal_lin_incr.stan")
# writeLines(stancode)
```

```
mod <- stan_model(model_code=stancode,
                                  verbose=TRUE)
```

```
##
## TRANSLATING MODEL '0bf89011ab06da95b4084b1691d14aa4' FROM Stan CODE TO C++ CODE NOW.
## successful in parsing the Stan model '0bf89011ab06da95b4084b1691d14aa4'.
## OS: x86_64, darwin17.0; rstan: 2.21.3; Rcpp: 1.0.10; inline: 0.3.19
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -mmacosx-version-min=10.13 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/inclu
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
##                 ^
##                 ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/include/Eigen/Core:96:10: fa
## #include <complex>
##          ^~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##  >> setting environment variables:
## PKG_LIBS =  '/Library/Frameworks/R.framework/Versions/4.1/Resources/library/rstan/lib//libStanServic
## PKG_CPPFLAGS =   -I"/Library/Frameworks/R.framework/Versions/4.1/Resources/library/Rcpp/include/"  -I
##  >> Program source :
##
##    1 :
##    2 : // includes from the plugin
##    3 : // [[Rcpp::plugins(cpp14)]]
##    4 :
##    5 :
##    6 : // user includes
##    7 : #include <Rcpp.h>
##    8 : #include <rstan/io/rlist_ref_var_context.hpp>
##    9 : #include <rstan/io/r_ostream.hpp>
##   10 : #include <rstan/stan_args.hpp>
##   11 : #include <boost/integer/integer_log2.hpp>
```

```
##   12 : // Code generated by Stan version 2.21.0
##   13 :
##   14 : #include <stan/model/model_header.hpp>
##   15 :
##   16 : namespace model39917acb1865_0bf89011ab06da95b4084b1691d14aa4_namespace {
##   17 :
##   18 : using std::istream;
##   19 : using std::string;
##   20 : using std::stringstream;
##   21 : using std::vector;
##   22 : using stan::io::dump;
##   23 : using stan::math::lgamma;
##   24 : using stan::model::prob_grad;
##   25 : using namespace stan::math;
##   26 :
##   27 : static int current_statement_begin__;
##   28 :
##   29 : stan::io::program_reader prog_reader__() {
##   30 :     stan::io::program_reader reader;
##   31 :     reader.add_event(0, 0, "start", "model39917acb1865_0bf89011ab06da95b4084b1691d14aa4");
##   32 :     reader.add_event(39, 37, "end", "model39917acb1865_0bf89011ab06da95b4084b1691d14aa4");
##   33 :     return reader;
##   34 : }
##   35 :
##   36 : class model39917acb1865_0bf89011ab06da95b4084b1691d14aa4
##   37 :   : public stan::model::model_base_crtp<model39917acb1865_0bf89011ab06da95b4084b1691d14aa4> {
##   38 : private:
##   39 :         int n;
##   40 :         std::vector<int> y;
##   41 :         std::vector<double> x1;
##   42 : public:
##   43 :     model39917acb1865_0bf89011ab06da95b4084b1691d14aa4(rstan::io::rlist_ref_var_context& cont
##   44 :         std::ostream* pstream__ = 0)
##   45 :         : model_base_crtp(0) {
##   46 :         ctor_body(context__, 0, pstream__);
##   47 :     }
##   48 :
##   49 :     model39917acb1865_0bf89011ab06da95b4084b1691d14aa4(stan::io::var_context& context__,
##   50 :         unsigned int random_seed__,
##   51 :         std::ostream* pstream__ = 0)
##   52 :         : model_base_crtp(0) {
##   53 :         ctor_body(context__, random_seed__, pstream__);
##   54 :     }
##   55 :
##   56 :     void ctor_body(stan::io::var_context& context__,
##   57 :                    unsigned int random_seed__,
##   58 :                    std::ostream* pstream__) {
##   59 :         typedef double local_scalar_t__;
##   60 :
##   61 :         boost::ecuyer1988 base_rng__ =
##   62 :           stan::services::util::create_rng(random_seed__, 0);
##   63 :         (void) base_rng__;  // suppress unused var warning
##   64 :
##   65 :         current_statement_begin__ = -1;
```

```
##   66 :
##   67 :         static const char* function__ = "model39917acb1865_0bf89011ab06da95b4084b1691d14aa4_na
##   68 :         (void) function__;  // dummy to suppress unused var warning
##   69 :         size_t pos__;
##   70 :         (void) pos__;  // dummy to suppress unused var warning
##   71 :         std::vector<int> vals_i__;
##   72 :         std::vector<double> vals_r__;
##   73 :         local_scalar_t__ DUMMY_VAR__(std::numeric_limits<double>::quiet_NaN());
##   74 :         (void) DUMMY_VAR__;  // suppress unused var warning
##   75 :
##   76 :         try {
##   77 :             // initialize data block variables from context__
##   78 :             current_statement_begin__ = 8;
##   79 :             context__.validate_dims("data initialization", "n", "int", context__.to_vec());
##   80 :             n = int(0);
##   81 :             vals_i__ = context__.vals_i("n");
##   82 :             pos__ = 0;
##   83 :             n = vals_i__[pos__++];
##   84 :             check_greater_or_equal(function__, "n", n, 0);
##   85 :
##   86 :             current_statement_begin__ = 9;
##   87 :             validate_non_negative_index("y", "n", n);
##   88 :             context__.validate_dims("data initialization", "y", "int", context__.to_vec(n));
##   89 :             y = std::vector<int>(n, int(0));
##   90 :             vals_i__ = context__.vals_i("y");
##   91 :             pos__ = 0;
##   92 :             size_t y_k_0_max__ = n;
##   93 :             for (size_t k_0__ = 0; k_0__ < y_k_0_max__; ++k_0__) {
##   94 :                 y[k_0__] = vals_i__[pos__++];
##   95 :             }
##   96 :             size_t y_i_0_max__ = n;
##   97 :             for (size_t i_0__ = 0; i_0__ < y_i_0_max__; ++i_0__) {
##   98 :                 check_greater_or_equal(function__, "y[i_0__]", y[i_0__], 1);
##   99 :                 check_less_or_equal(function__, "y[i_0__]", y[i_0__], 4);
##  100 :             }
##  101 :
##  102 :             current_statement_begin__ = 10;
##  103 :             validate_non_negative_index("x1", "n", n);
##  104 :             context__.validate_dims("data initialization", "x1", "double", context__.to_vec(n
##  105 :             x1 = std::vector<double>(n, double(0));
##  106 :             vals_r__ = context__.vals_r("x1");
##  107 :             pos__ = 0;
##  108 :             size_t x1_k_0_max__ = n;
##  109 :             for (size_t k_0__ = 0; k_0__ < x1_k_0_max__; ++k_0__) {
##  110 :                 x1[k_0__] = vals_r__[pos__++];
##  111 :             }
##  112 :
##  113 :
##  114 :             // initialize transformed data variables
##  115 :             // execute transformed data statements
##  116 :
##  117 :             // validate transformed data
##  118 :
##  119 :             // validate, set parameter ranges
```

```
##  120 :             num_params_r__ = 0U;
##  121 :             param_ranges_i__.clear();
##  122 :             current_statement_begin__ = 14;
##  123 :             validate_non_negative_index("kappa", "3", 3);
##  124 :             num_params_r__ += 3;
##  125 :             current_statement_begin__ = 15;
##  126 :             num_params_r__ += 1;
##  127 :         } catch (const std::exception& e) {
##  128 :             stan::lang::rethrow_located(e, current_statement_begin__, prog_reader__());
##  129 :             // Next line prevents compiler griping about no return
##  130 :             throw std::runtime_error("*** IF YOU SEE THIS, PLEASE REPORT A BUG ***");
##  131 :         }
##  132 :     }
##  133 :
##  134 :     ~model39917acb1865_0bf89011ab06da95b4084b1691d14aa4() { }
##  135 :
##  136 :
##  137 :     void transform_inits(const stan::io::var_context& context__,
##  138 :                          std::vector<int>& params_i__,
##  139 :                          std::vector<double>& params_r__,
##  140 :                          std::ostream* pstream__) const {
##  141 :         typedef double local_scalar_t__;
##  142 :         stan::io::writer<double> writer__(params_r__, params_i__);
##  143 :         size_t pos__;
##  144 :         (void) pos__; // dummy call to supress warning
##  145 :         std::vector<double> vals_r__;
##  146 :         std::vector<int> vals_i__;
##  147 :
##  148 :         current_statement_begin__ = 14;
##  149 :         if (!(context__.contains_r("kappa")))
##  150 :             stan::lang::rethrow_located(std::runtime_error(std::string("Variable kappa missing
##  151 :         vals_r__ = context__.vals_r("kappa");
##  152 :         pos__ = 0U;
##  153 :         validate_non_negative_index("kappa", "3", 3);
##  154 :         context__.validate_dims("parameter initialization", "kappa", "vector_d", context__.to_
##  155 :         Eigen::Matrix<double, Eigen::Dynamic, 1> kappa(3);
##  156 :         size_t kappa_j_1_max__ = 3;
##  157 :         for (size_t j_1__ = 0; j_1__ < kappa_j_1_max__; ++j_1__) {
##  158 :             kappa(j_1__) = vals_r__[pos__++];
##  159 :         }
##  160 :         try {
##  161 :             writer__.ordered_unconstrain(kappa);
##  162 :         } catch (const std::exception& e) {
##  163 :             stan::lang::rethrow_located(std::runtime_error(std::string("Error transforming va
##  164 :         }
##  165 :
##  166 :         current_statement_begin__ = 15;
##  167 :         if (!(context__.contains_r("b1")))
##  168 :             stan::lang::rethrow_located(std::runtime_error(std::string("Variable b1 missing"))
##  169 :         vals_r__ = context__.vals_r("b1");
##  170 :         pos__ = 0U;
##  171 :         context__.validate_dims("parameter initialization", "b1", "double", context__.to_vec()
##  172 :         double b1(0);
##  173 :         b1 = vals_r__[pos__++];
```

```
##  174 :         try {
##  175 :             writer__.scalar_lb_unconstrain(0, b1);
##  176 :         } catch (const std::exception& e) {
##  177 :             stan::lang::rethrow_located(std::runtime_error(std::string("Error transforming va
##  178 :         }
##  179 :
##  180 :         params_r__ = writer__.data_r();
##  181 :         params_i__ = writer__.data_i();
##  182 :     }
##  183 :
##  184 :     void transform_inits(const stan::io::var_context& context,
##  185 :                          Eigen::Matrix<double, Eigen::Dynamic, 1>& params_r,
##  186 :                          std::ostream* pstream__) const {
##  187 :       std::vector<double> params_r_vec;
##  188 :       std::vector<int> params_i_vec;
##  189 :       transform_inits(context, params_i_vec, params_r_vec, pstream__);
##  190 :       params_r.resize(params_r_vec.size());
##  191 :       for (int i = 0; i < params_r.size(); ++i)
##  192 :         params_r(i) = params_r_vec[i];
##  193 :     }
##  194 :
##  195 :
##  196 :     template <bool propto__, bool jacobian__, typename T__>
##  197 :     T__ log_prob(std::vector<T__>& params_r__,
##  198 :                  std::vector<int>& params_i__,
##  199 :                  std::ostream* pstream__ = 0) const {
##  200 :
##  201 :         typedef T__ local_scalar_t__;
##  202 :
##  203 :         local_scalar_t__ DUMMY_VAR__(std::numeric_limits<double>::quiet_NaN());
##  204 :         (void) DUMMY_VAR__;  // dummy to suppress unused var warning
##  205 :
##  206 :         T__ lp__(0.0);
##  207 :         stan::math::accumulator<T__> lp_accum__;
##  208 :         try {
##  209 :             stan::io::reader<local_scalar_t__> in__(params_r__, params_i__);
##  210 :
##  211 :             // model parameters
##  212 :             current_statement_begin__ = 14;
##  213 :             Eigen::Matrix<local_scalar_t__, Eigen::Dynamic, 1> kappa;
##  214 :             (void) kappa;  // dummy to suppress unused var warning
##  215 :             if (jacobian__)
##  216 :                 kappa = in__.ordered_constrain(3, lp__);
##  217 :             else
##  218 :                 kappa = in__.ordered_constrain(3);
##  219 :
##  220 :             current_statement_begin__ = 15;
##  221 :             local_scalar_t__ b1;
##  222 :             (void) b1;  // dummy to suppress unused var warning
##  223 :             if (jacobian__)
##  224 :                 b1 = in__.scalar_lb_constrain(0, lp__);
##  225 :             else
##  226 :                 b1 = in__.scalar_lb_constrain(0);
##  227 :
```

```
##  228 :             // transformed parameters
##  229 :             current_statement_begin__ = 19;
##  230 :             validate_non_negative_index("mu", "n", n);
##  231 :             Eigen::Matrix<local_scalar_t__, Eigen::Dynamic, 1> mu(n);
##  232 :             stan::math::initialize(mu, DUMMY_VAR__);
##  233 :             stan::math::fill(mu, DUMMY_VAR__);
##  234 :
##  235 :             // transformed parameters block statements
##  236 :             current_statement_begin__ = 20;
##  237 :             for (int i = 1; i <= n; ++i) {
##  238 :
##  239 :                 current_statement_begin__ = 21;
##  240 :                 stan::model::assign(mu,
##  241 :                             stan::model::cons_list(stan::model::index_uni(i), stan::model::nil
##  242 :                             (get_base1(x1, i, "x1", 1) * b1),
##  243 :                             "assigning variable mu");
##  244 :             }
##  245 :
##  246 :             // validate transformed parameters
##  247 :             const char* function__ = "validate transformed params";
##  248 :             (void) function__;  // dummy to suppress unused var warning
##  249 :
##  250 :             current_statement_begin__ = 19;
##  251 :             size_t mu_j_1_max__ = n;
##  252 :             for (size_t j_1__ = 0; j_1__ < mu_j_1_max__; ++j_1__) {
##  253 :                 if (stan::math::is_uninitialized(mu(j_1__))) {
##  254 :                     std::stringstream msg__;
##  255 :                     msg__ << "Undefined transformed parameter: mu" << "(" << j_1__ << ")";
##  256 :                     stan::lang::rethrow_located(std::runtime_error(std::string("Error initiali
##  257 :                 }
##  258 :             }
##  259 :
##  260 :             // model body
##  261 :
##  262 :             current_statement_begin__ = 26;
##  263 :             lp_accum__.add(normal_log<propto__>(b1, 0, 9.9e+06));
##  264 :             current_statement_begin__ = 27;
##  265 :             for (int i = 1; i <= n; ++i) {
##  266 :
##  267 :                 current_statement_begin__ = 28;
##  268 :                 lp_accum__.add(ordered_logistic_log<propto__>(get_base1(y, i, "y", 1), get_bas
##  269 :             }
##  270 :
##  271 :         } catch (const std::exception& e) {
##  272 :             stan::lang::rethrow_located(e, current_statement_begin__, prog_reader__());
##  273 :             // Next line prevents compiler griping about no return
##  274 :             throw std::runtime_error("*** IF YOU SEE THIS, PLEASE REPORT A BUG ***");
##  275 :         }
##  276 :
##  277 :         lp_accum__.add(lp__);
##  278 :         return lp_accum__.sum();
##  279 :
##  280 :     } // log_prob()
##  281 :
```

```
##  282 :     template <bool propto, bool jacobian, typename T_>
##  283 :     T_ log_prob(Eigen::Matrix<T_,Eigen::Dynamic,1>& params_r,
##  284 :                 std::ostream* pstream = 0) const {
##  285 :       std::vector<T_> vec_params_r;
##  286 :       vec_params_r.reserve(params_r.size());
##  287 :       for (int i = 0; i < params_r.size(); ++i)
##  288 :         vec_params_r.push_back(params_r(i));
##  289 :       std::vector<int> vec_params_i;
##  290 :       return log_prob<propto,jacobian,T_>(vec_params_r, vec_params_i, pstream);
##  291 :     }
##  292 :
##  293 :
##  294 :     void get_param_names(std::vector<std::string>& names__) const {
##  295 :         names__.resize(0);
##  296 :         names__.push_back("kappa");
##  297 :         names__.push_back("b1");
##  298 :         names__.push_back("mu");
##  299 :         names__.push_back("log_lik");
##  300 :     }
##  301 :
##  302 :
##  303 :     void get_dims(std::vector<std::vector<size_t> >& dimss__) const {
##  304 :         dimss__.resize(0);
##  305 :         std::vector<size_t> dims__;
##  306 :         dims__.resize(0);
##  307 :         dims__.push_back(3);
##  308 :         dimss__.push_back(dims__);
##  309 :         dims__.resize(0);
##  310 :         dimss__.push_back(dims__);
##  311 :         dims__.resize(0);
##  312 :         dims__.push_back(n);
##  313 :         dimss__.push_back(dims__);
##  314 :         dims__.resize(0);
##  315 :         dims__.push_back(n);
##  316 :         dimss__.push_back(dims__);
##  317 :     }
##  318 :
##  319 :     template <typename RNG>
##  320 :     void write_array(RNG& base_rng__,
##  321 :                      std::vector<double>& params_r__,
##  322 :                      std::vector<int>& params_i__,
##  323 :                      std::vector<double>& vars__,
##  324 :                      bool include_tparams__ = true,
##  325 :                      bool include_gqs__ = true,
##  326 :                      std::ostream* pstream__ = 0) const {
##  327 :         typedef double local_scalar_t__;
##  328 :
##  329 :         vars__.resize(0);
##  330 :         stan::io::reader<local_scalar_t__> in__(params_r__, params_i__);
##  331 :         static const char* function__ = "model39917acb1865_0bf89011ab06da95b4084b1691d14aa4_na
##  332 :         (void) function__;  // dummy to suppress unused var warning
##  333 :
##  334 :         // read-transform, write parameters
##  335 :         Eigen::Matrix<double, Eigen::Dynamic, 1> kappa = in__.ordered_constrain(3);
```

```
##  336 :         size_t kappa_j_1_max__ = 3;
##  337 :         for (size_t j_1__ = 0; j_1__ < kappa_j_1_max__; ++j_1__) {
##  338 :             vars__.push_back(kappa(j_1__));
##  339 :         }
##  340 :
##  341 :         double b1 = in__.scalar_lb_constrain(0);
##  342 :         vars__.push_back(b1);
##  343 :
##  344 :         double lp__ = 0.0;
##  345 :         (void) lp__;  // dummy to suppress unused var warning
##  346 :         stan::math::accumulator<double> lp_accum__;
##  347 :
##  348 :         local_scalar_t__ DUMMY_VAR__(std::numeric_limits<double>::quiet_NaN());
##  349 :         (void) DUMMY_VAR__;  // suppress unused var warning
##  350 :
##  351 :         if (!include_tparams__ && !include_gqs__) return;
##  352 :
##  353 :         try {
##  354 :             // declare and define transformed parameters
##  355 :             current_statement_begin__ = 19;
##  356 :             validate_non_negative_index("mu", "n", n);
##  357 :             Eigen::Matrix<double, Eigen::Dynamic, 1> mu(n);
##  358 :             stan::math::initialize(mu, DUMMY_VAR__);
##  359 :             stan::math::fill(mu, DUMMY_VAR__);
##  360 :
##  361 :             // do transformed parameters statements
##  362 :             current_statement_begin__ = 20;
##  363 :             for (int i = 1; i <= n; ++i) {
##  364 :
##  365 :                 current_statement_begin__ = 21;
##  366 :                 stan::model::assign(mu,
##  367 :                             stan::model::cons_list(stan::model::index_uni(i), stan::model::nil
##  368 :                             (get_base1(x1, i, "x1", 1) * b1),
##  369 :                             "assigning variable mu");
##  370 :             }
##  371 :
##  372 :             if (!include_gqs__ && !include_tparams__) return;
##  373 :             // validate transformed parameters
##  374 :             const char* function__ = "validate transformed params";
##  375 :             (void) function__;  // dummy to suppress unused var warning
##  376 :
##  377 :             // write transformed parameters
##  378 :             if (include_tparams__) {
##  379 :                 size_t mu_j_1_max__ = n;
##  380 :                 for (size_t j_1__ = 0; j_1__ < mu_j_1_max__; ++j_1__) {
##  381 :                     vars__.push_back(mu(j_1__));
##  382 :                 }
##  383 :             }
##  384 :             if (!include_gqs__) return;
##  385 :             // declare and define generated quantities
##  386 :             current_statement_begin__ = 33;
##  387 :             validate_non_negative_index("log_lik", "n", n);
##  388 :             Eigen::Matrix<double, Eigen::Dynamic, 1> log_lik(n);
##  389 :             stan::math::initialize(log_lik, DUMMY_VAR__);
```

```
##  390 :             stan::math::fill(log_lik, DUMMY_VAR__);
##  391 :
##  392 :             // generated quantities statements
##  393 :             current_statement_begin__ = 34;
##  394 :             for (int i = 1; i <= n; ++i) {
##  395 :
##  396 :                 current_statement_begin__ = 35;
##  397 :                 stan::model::assign(log_lik,
##  398 :                             stan::model::cons_list(stan::model::index_uni(i), stan::model::nil
##  399 :                             ordered_logistic_log(get_base1(y, i, "y", 1), get_base1(mu, i, "mu
##  400 :                             "assigning variable log_lik");
##  401 :             }
##  402 :
##  403 :             // validate, write generated quantities
##  404 :             current_statement_begin__ = 33;
##  405 :             size_t log_lik_j_1_max__ = n;
##  406 :             for (size_t j_1__ = 0; j_1__ < log_lik_j_1_max__; ++j_1__) {
##  407 :                 vars__.push_back(log_lik(j_1__));
##  408 :             }
##  409 :
##  410 :         } catch (const std::exception& e) {
##  411 :             stan::lang::rethrow_located(e, current_statement_begin__, prog_reader__());
##  412 :             // Next line prevents compiler griping about no return
##  413 :             throw std::runtime_error("*** IF YOU SEE THIS, PLEASE REPORT A BUG ***");
##  414 :         }
##  415 :     }
##  416 :
##  417 :     template <typename RNG>
##  418 :     void write_array(RNG& base_rng,
##  419 :                      Eigen::Matrix<double,Eigen::Dynamic,1>& params_r,
##  420 :                      Eigen::Matrix<double,Eigen::Dynamic,1>& vars,
##  421 :                      bool include_tparams = true,
##  422 :                      bool include_gqs = true,
##  423 :                      std::ostream* pstream = 0) const {
##  424 :       std::vector<double> params_r_vec(params_r.size());
##  425 :       for (int i = 0; i < params_r.size(); ++i)
##  426 :         params_r_vec[i] = params_r(i);
##  427 :       std::vector<double> vars_vec;
##  428 :       std::vector<int> params_i_vec;
##  429 :       write_array(base_rng, params_r_vec, params_i_vec, vars_vec, include_tparams, include_gqs
##  430 :       vars.resize(vars_vec.size());
##  431 :       for (int i = 0; i < vars.size(); ++i)
##  432 :         vars(i) = vars_vec[i];
##  433 :     }
##  434 :
##  435 :     std::string model_name() const {
##  436 :         return "model39917acb1865_0bf89011ab06da95b4084b1691d14aa4";
##  437 :     }
##  438 :
##  439 :
##  440 :     void constrained_param_names(std::vector<std::string>& param_names__,
##  441 :                                  bool include_tparams__ = true,
##  442 :                                  bool include_gqs__ = true) const {
##  443 :         std::stringstream param_name_stream__;
```

```
##  444 :         size_t kappa_j_1_max__ = 3;
##  445 :         for (size_t j_1__ = 0; j_1__ < kappa_j_1_max__; ++j_1__) {
##  446 :             param_name_stream__.str(std::string());
##  447 :             param_name_stream__ << "kappa" << '.' << j_1__ + 1;
##  448 :             param_names__.push_back(param_name_stream__.str());
##  449 :         }
##  450 :         param_name_stream__.str(std::string());
##  451 :         param_name_stream__ << "b1";
##  452 :         param_names__.push_back(param_name_stream__.str());
##  453 :
##  454 :         if (!include_gqs__ && !include_tparams__) return;
##  455 :
##  456 :         if (include_tparams__) {
##  457 :             size_t mu_j_1_max__ = n;
##  458 :             for (size_t j_1__ = 0; j_1__ < mu_j_1_max__; ++j_1__) {
##  459 :                 param_name_stream__.str(std::string());
##  460 :                 param_name_stream__ << "mu" << '.' << j_1__ + 1;
##  461 :                 param_names__.push_back(param_name_stream__.str());
##  462 :             }
##  463 :         }
##  464 :
##  465 :         if (!include_gqs__) return;
##  466 :         size_t log_lik_j_1_max__ = n;
##  467 :         for (size_t j_1__ = 0; j_1__ < log_lik_j_1_max__; ++j_1__) {
##  468 :             param_name_stream__.str(std::string());
##  469 :             param_name_stream__ << "log_lik" << '.' << j_1__ + 1;
##  470 :             param_names__.push_back(param_name_stream__.str());
##  471 :         }
##  472 :     }
##  473 :
##  474 :
##  475 :     void unconstrained_param_names(std::vector<std::string>& param_names__,
##  476 :                                    bool include_tparams__ = true,
##  477 :                                    bool include_gqs__ = true) const {
##  478 :         std::stringstream param_name_stream__;
##  479 :         size_t kappa_j_1_max__ = 3;
##  480 :         for (size_t j_1__ = 0; j_1__ < kappa_j_1_max__; ++j_1__) {
##  481 :             param_name_stream__.str(std::string());
##  482 :             param_name_stream__ << "kappa" << '.' << j_1__ + 1;
##  483 :             param_names__.push_back(param_name_stream__.str());
##  484 :         }
##  485 :         param_name_stream__.str(std::string());
##  486 :         param_name_stream__ << "b1";
##  487 :         param_names__.push_back(param_name_stream__.str());
##  488 :
##  489 :         if (!include_gqs__ && !include_tparams__) return;
##  490 :
##  491 :         if (include_tparams__) {
##  492 :             size_t mu_j_1_max__ = n;
##  493 :             for (size_t j_1__ = 0; j_1__ < mu_j_1_max__; ++j_1__) {
##  494 :                 param_name_stream__.str(std::string());
##  495 :                 param_name_stream__ << "mu" << '.' << j_1__ + 1;
##  496 :                 param_names__.push_back(param_name_stream__.str());
##  497 :             }
```

```
##  498 :         }
##  499 :
##  500 :         if (!include_gqs__) return;
##  501 :         size_t log_lik_j_1_max__ = n;
##  502 :         for (size_t j_1__ = 0; j_1__ < log_lik_j_1_max__; ++j_1__) {
##  503 :             param_name_stream__.str(std::string());
##  504 :             param_name_stream__ << "log_lik" << '.' << j_1__ + 1;
##  505 :             param_names__.push_back(param_name_stream__.str());
##  506 :         }
##  507 :     }
##  508 :
##  509 : }; // model
##  510 :
##  511 : }  // namespace
##  512 :
##  513 : typedef model39917acb1865_0bf89011ab06da95b4084b1691d14aa4_namespace::model39917acb1865_0bf89
##  514 :
##  515 : #ifndef USING_R
##  516 :
##  517 : stan::model::model_base& new_model(
##  518 :         stan::io::var_context& data_context,
##  519 :         unsigned int seed,
##  520 :         std::ostream* msg_stream) {
##  521 :   stan_model* m = new stan_model(data_context, seed, msg_stream);
##  522 :   return *m;
##  523 : }
##  524 :
##  525 : #endif
##  526 :
##  527 :
##  528 :
##  529 : #include <rstan_next/stan_fit.hpp>
##  530 :
##  531 : struct stan_model_holder {
##  532 :     stan_model_holder(rstan::io::rlist_ref_var_context rcontext,
##  533 :                       unsigned int random_seed)
##  534 :     : rcontext_(rcontext), random_seed_(random_seed)
##  535 :      {
##  536 :      }
##  537 :
##  538 :    //stan::math::ChainableStack ad_stack;
##  539 :    rstan::io::rlist_ref_var_context rcontext_;
##  540 :    unsigned int random_seed_;
##  541 : };
##  542 :
##  543 : Rcpp::XPtr<stan::model::model_base> model_ptr(stan_model_holder* smh) {
##  544 :   Rcpp::XPtr<stan::model::model_base> model_instance(new stan_model(smh->rcontext_, smh->rando
##  545 :   return model_instance;
##  546 : }
##  547 :
##  548 : Rcpp::XPtr<rstan::stan_fit_base> fit_ptr(stan_model_holder* smh) {
##  549 :   return Rcpp::XPtr<rstan::stan_fit_base>(new rstan::stan_fit(model_ptr(smh), smh->random_seed
##  550 : }
##  551 :
```

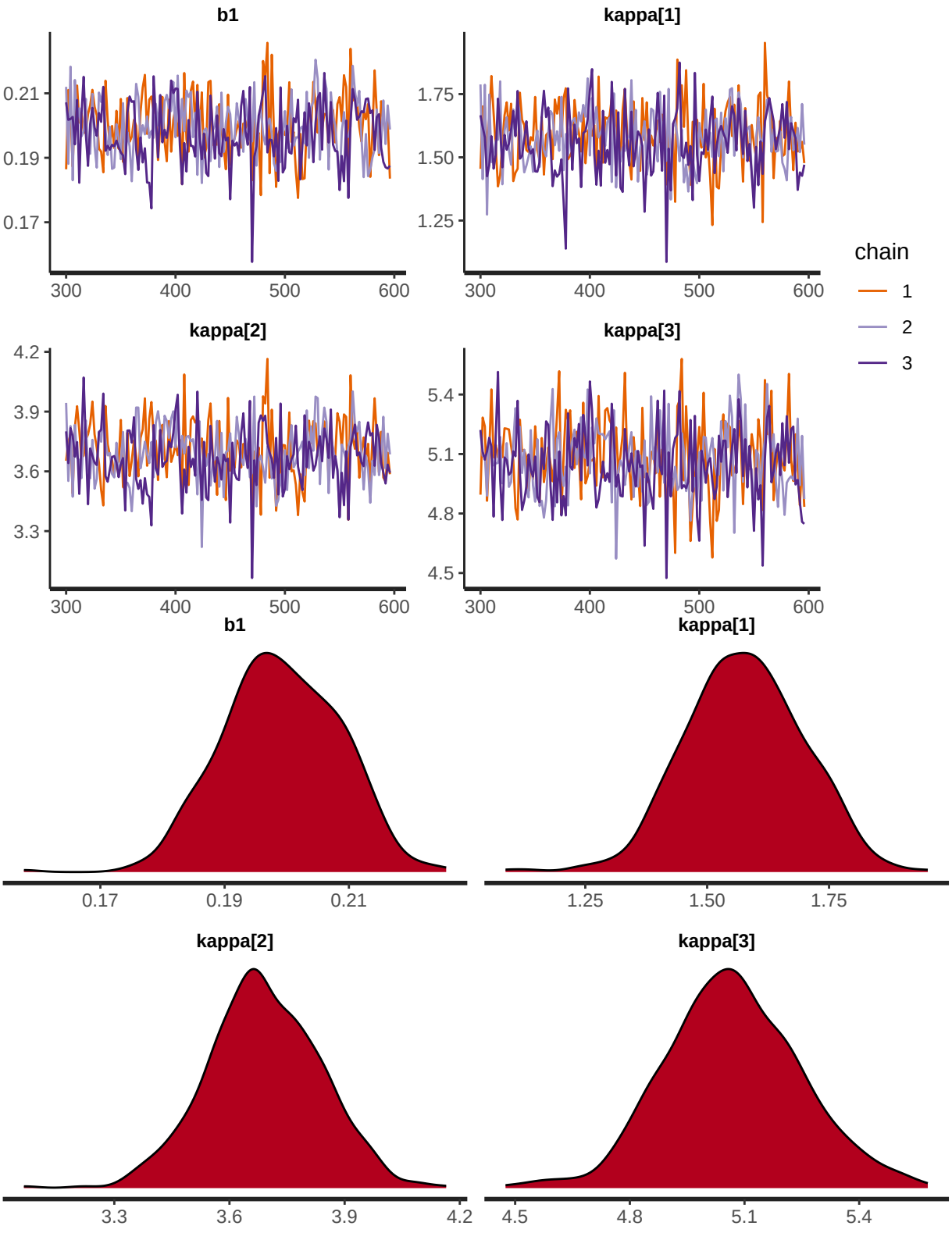
```
##  552 : std::string model_name(stan_model_holder* smh) {
##  553 :   return model_ptr(smh).get()->model_name();
##  554 : }
##  555 :
##  556 : RCPP_MODULE(stan_fit4model39917acb1865_0bf89011ab06da95b4084b1691d14aa4_mod){
##  557 :   Rcpp::class_<stan_model_holder>("stan_fit4model39917acb1865_0bf89011ab06da95b4084b1691d14aa
##  558 :   .constructor<rstan::io::rlist_ref_var_context, unsigned int>()
##  559 :   .method("model_ptr", &model_ptr)
##  560 :   .method("fit_ptr", &fit_ptr)
##  561 :   .method("model_name", &model_name)
##  562 :   ;
##  563 : }
##  564 :
##  565 :
##  566 : // declarations
##  567 : extern "C" {
##  568 : SEXP file39911ab211a5( ) ;
##  569 : }
##  570 :
##  571 : // definition
##  572 : SEXP file39911ab211a5() {
##  573 :  return Rcpp::wrap("0bf89011ab06da95b4084b1691d14aa4");
##  574 : }
```

```
fit.lin.incr <- sampling(mod,
                                data=datos.lin,
                                chains=3,warmup=300,iter=600,thin=2,cores=4 )
```

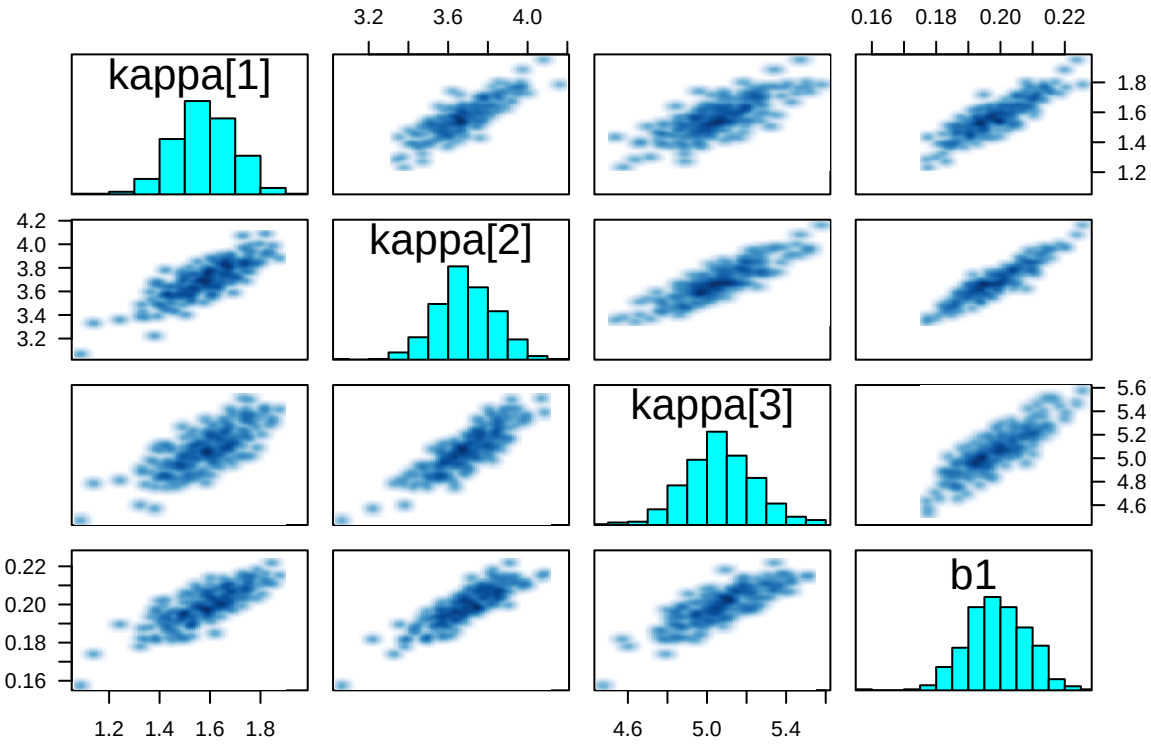
```
print(fit.lin.incr, pars=param.lin)
```

```
## Inference for Stan model: 0bf89011ab06da95b4084b1691d14aa4.
## 3 chains, each with iter=600; warmup=300; thin=2;
## post-warmup draws per chain=150, total post-warmup draws=450.
##
##          mean se_mean   sd 2.5%  25%  50%  75% 97.5% n_eff Rhat
## b1       0.20    0.00 0.01 0.18 0.19 0.20 0.21  0.22   288 1.01
## kappa[1] 1.58    0.01 0.12 1.35 1.50 1.58 1.66  1.80   359 1.01
## kappa[2] 3.69    0.01 0.15 3.40 3.59 3.68 3.79  3.97   301 1.00
## kappa[3] 5.07    0.01 0.18 4.76 4.96 5.07 5.19  5.42   342 1.00
##
## Samples were drawn using NUTS(diag_e) at Sun Aug 13 15:45:15 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

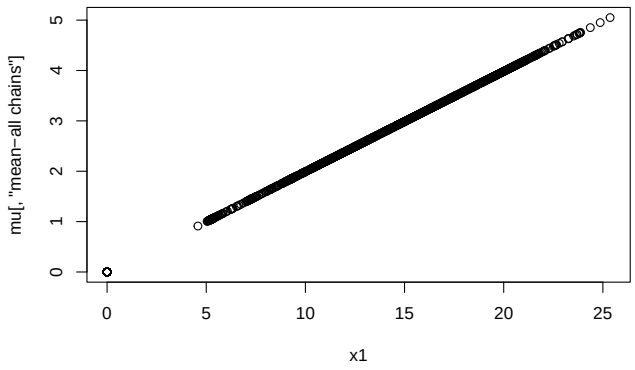
```
stan_trace(fit.lin.incr,pars=param.lin)
stan_dens(fit.lin.incr,pars=param.lin)
```




```
pairs(fit.lin.incr, pars = c("kappa","b1"), las = 1)
```




```
mu=get_posterior_mean(fit.lin.incr,"mu")
plot(x1,mu[,"mean-all chains"])
```

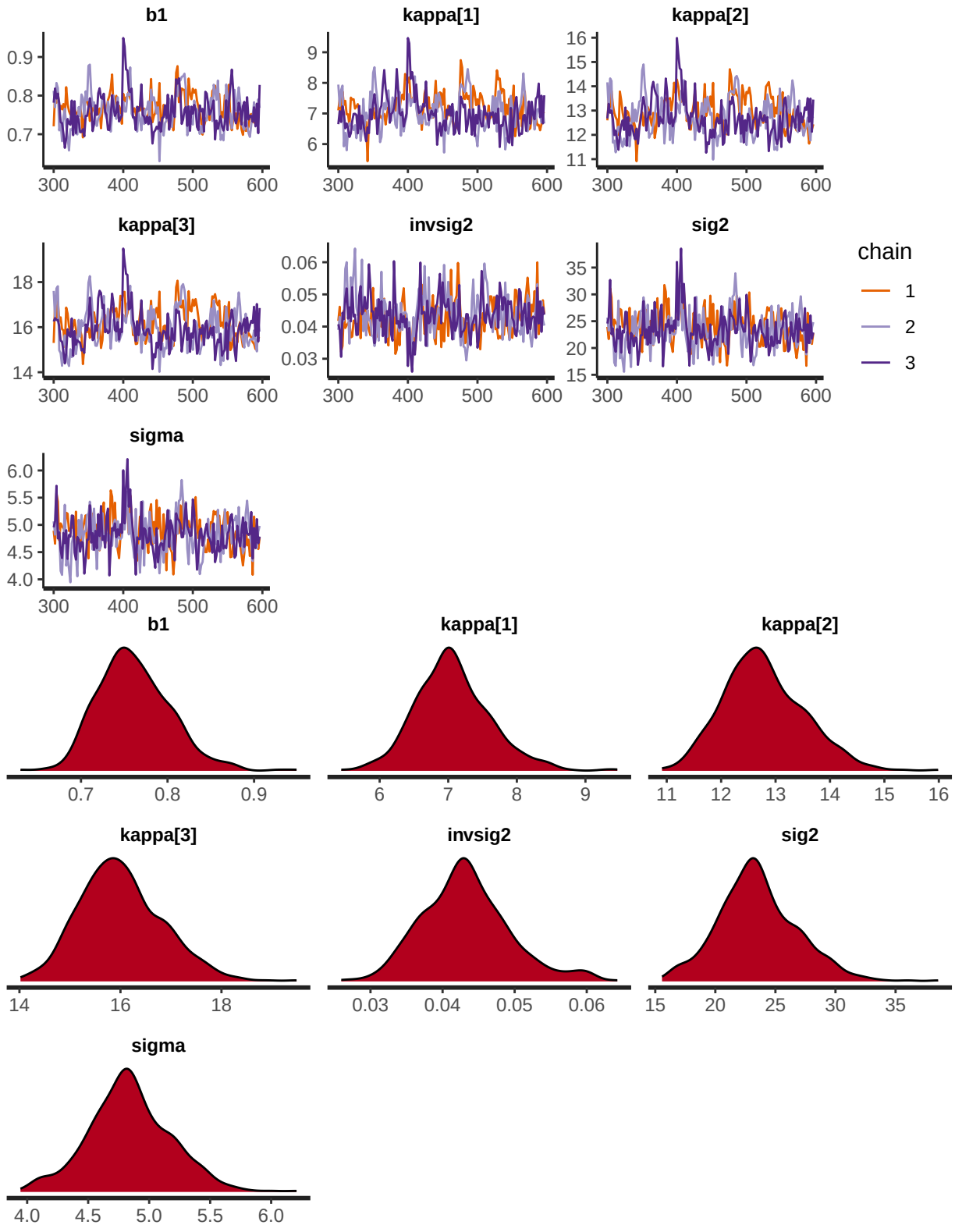


6.2. LME: Lineal creciente

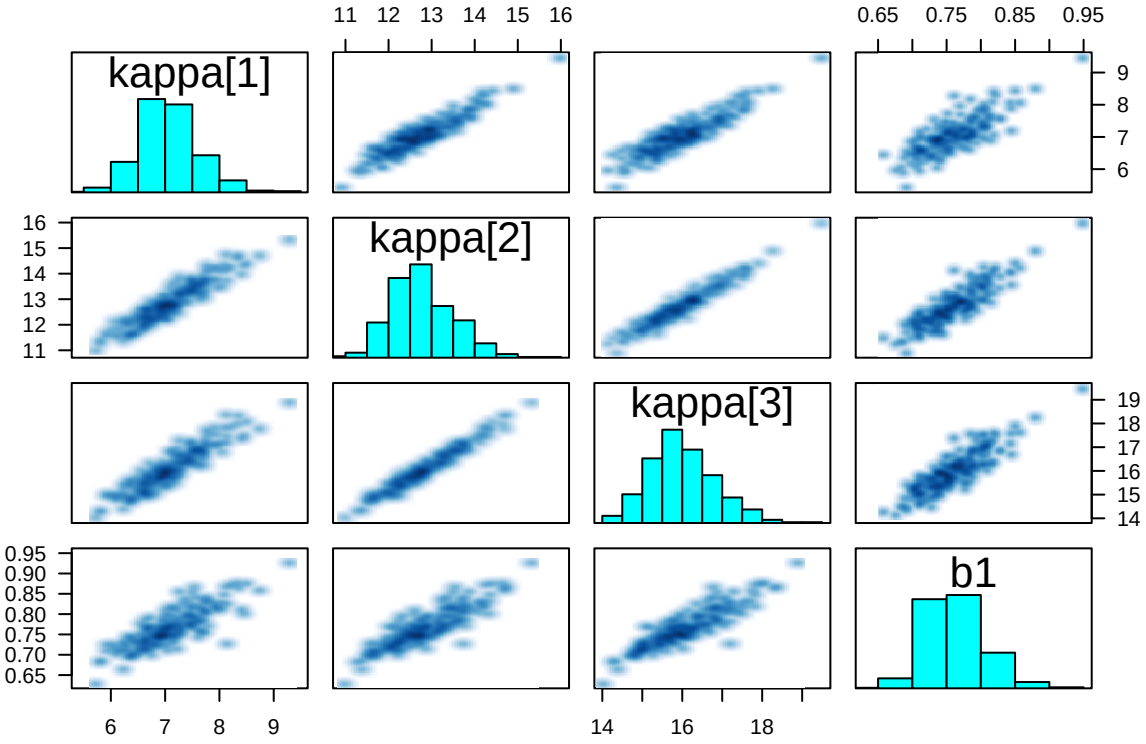
```
fit.lme.incr.reintrcpt <- stan("jagam_10_aneur_ordinal_lme_incr_reintrcpt.stan",
            data=datos.lme,
            chains=3,warmup=300,iter=600,thin=2,cores=4 )

print(fit.lme.incr.reintrcpt, pars=param.lme)
```

```
## Inference for Stan model: jagam_10_aneur_ordinal_lme_incr_reintrcpt.
## 3 chains, each with iter=600; warmup=300; thin=2;
## post-warmup draws per chain=150, total post-warmup draws=450.
##
##           mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## b1        0.76    0.00 0.04  0.69  0.73  0.76  0.79  0.86   139 1.00
## kappa[1]  7.07    0.06 0.55  6.07  6.68  7.02  7.38  8.30    99 1.02
## kappa[2] 12.78    0.08 0.74 11.53 12.26 12.71 13.25 14.31    96 1.01
## kappa[3] 16.01    0.08 0.82 14.60 15.43 15.94 16.52 17.76    95 1.01
## invsig2   0.04    0.00 0.01  0.03  0.04  0.04  0.05  0.06   173 1.01
## sig2     23.59    0.25 3.35 17.29 21.47 23.27 25.52 30.64   173 1.00
## sigma     4.84    0.03 0.34  4.16  4.63  4.82  5.05  5.54   173 1.01
##
## Samples were drawn using NUTS(diag_e) at Sun Aug 13 15:48:08 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

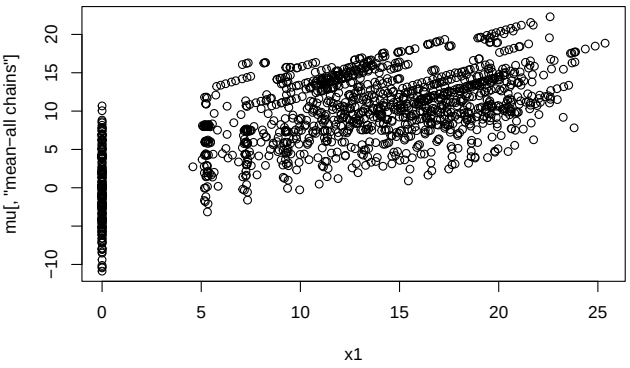
```
stan_trace(fit.lme.incr.reintrcpt,pars=param.lme)
stan_dens(fit.lme.incr.reintrcpt,pars=param.lme)
```




```
pairs(fit.lme.incr.reintrcpt, pars = c("kappa","b1"), las = 1)
```




```
mu=get_posterior_mean(fit.lme.incr.reintrcpt,"mu")
plot(x1,mu[,"mean-all chains"])
```

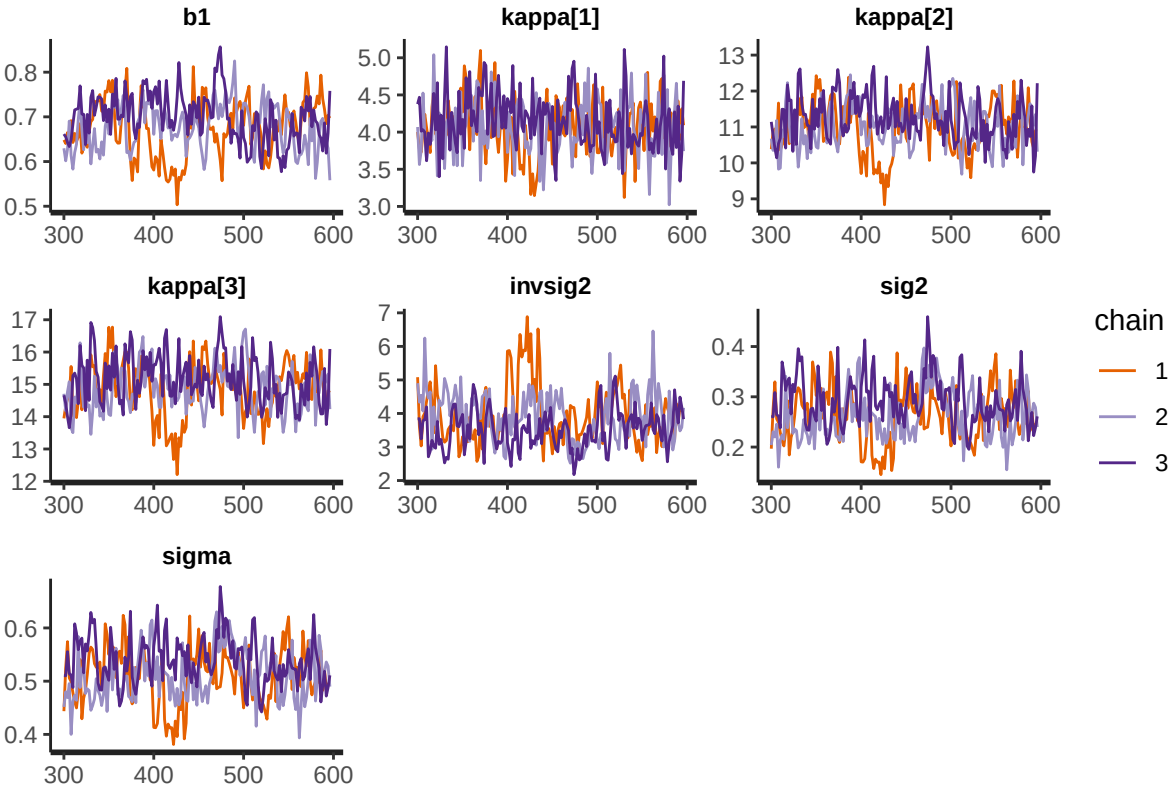


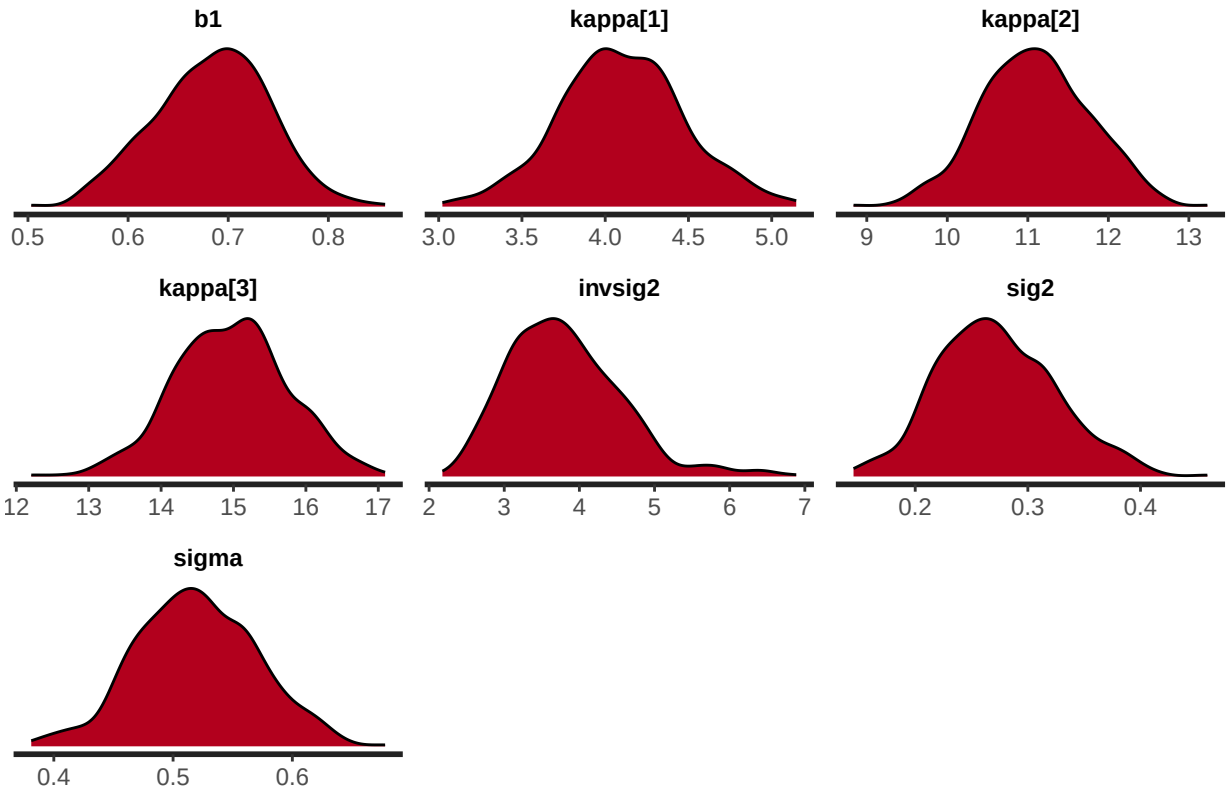

```
fit.lme.incr.reslope <- stan("jagam_10_aneur_ordinal_lme_incr_reslope.stan",
            data=datos.lme,
            chains=3,warmup=300,iter=600,thin=2,cores=4 )

print(fit.lme.incr.reslope, pars=param.lme)
```

```
## Inference for Stan model: jagam_10_aneur_ordinal_lme_incr_reslope.
## 3 chains, each with iter=600; warmup=300; thin=2;
## post-warmup draws per chain=150, total post-warmup draws=450.
##
##           mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## b1        0.68    0.01 0.06  0.57  0.65  0.69  0.72  0.79    53 1.06
## kappa[1]  4.10    0.02 0.37  3.35  3.86  4.09  4.34  4.86   317 1.01
## kappa[2] 11.10    0.07 0.67  9.76 10.61 11.07 11.55 12.38    92 1.02
```

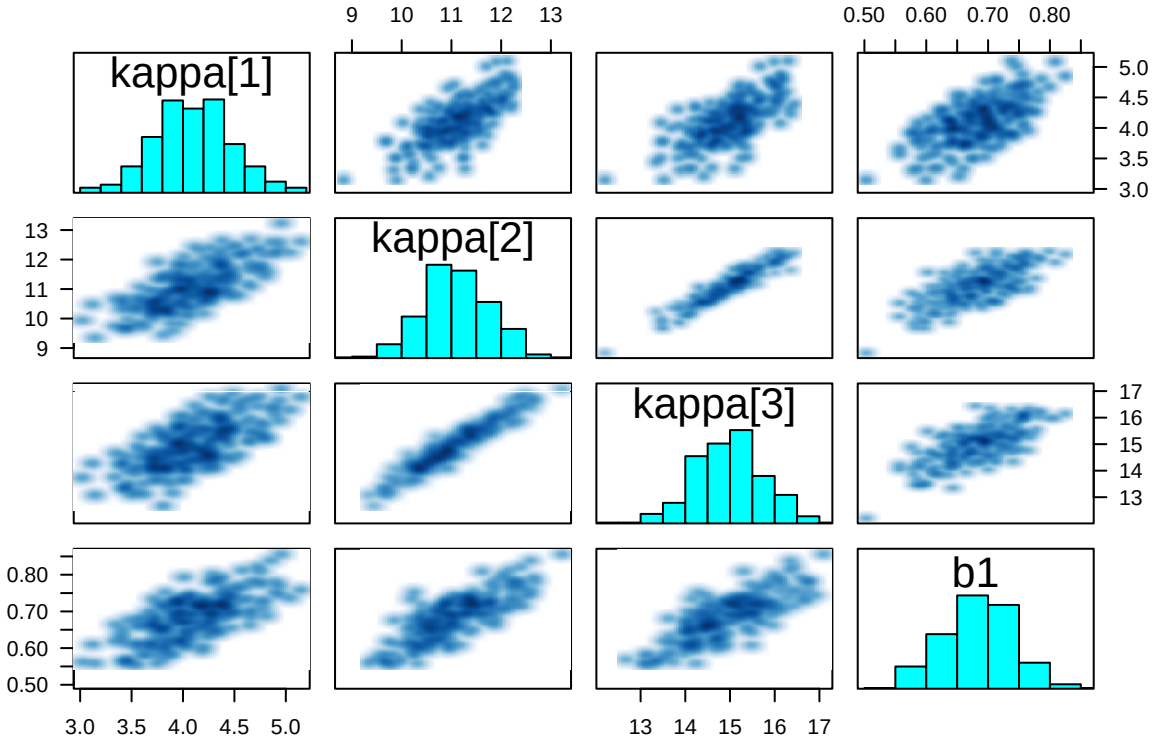
```
## kappa[3] 14.97    0.08 0.78 13.46 14.42 15.00 15.47 16.44    90 1.02
## invsig2   3.80    0.10 0.76  2.62  3.23  3.70  4.24  5.68    56 1.05
## sig2      0.27    0.01 0.05  0.18  0.24  0.27  0.31  0.38    65 1.05
## sigma     0.52    0.01 0.05  0.42  0.49  0.52  0.56  0.62    62 1.05
##
## Samples were drawn using NUTS(diag_e) at Sun Aug 13 15:49:56 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
stan_trace(fit.lme.incr.reslope,pars=param.lme)
stan_dens(fit.lme.incr.reslope,pars=param.lme)
```

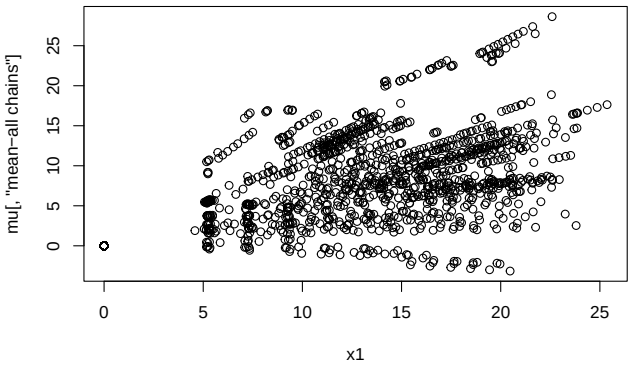





```
pairs(fit.lme.incr.reslope, pars = c("kappa","b1"), las = 1)
```




```
mu=get_posterior_mean(fit.lme.incr.reslope,"mu")
plot(x1,mu[,"mean-all chains"])
```


mu[, "mean−all chains"]
0
5
10
15
20
25
x1

7. Spline NO restricciones

7.1 For a spline-based fit without constraints:

```
#set.seed(123)
#mod.s <- cgam(y ~  s(x1) )
#summary(mod.s)
```

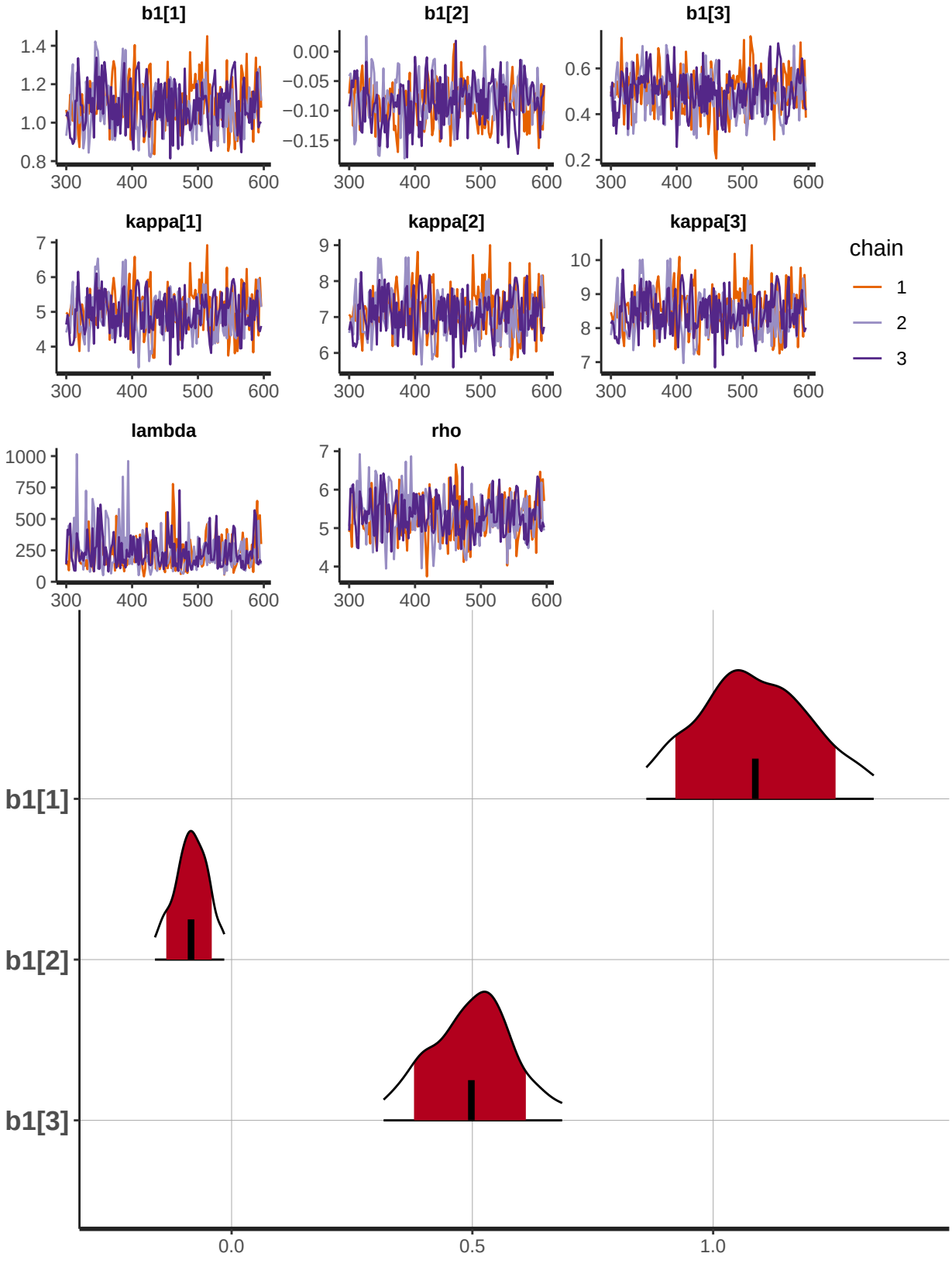
```
datos.add.non <- list(  y = y ,
               n = length(y) , k1=k1,
              XI1 = XI1,
              zero = rep(0,1+k1),
              S1=S1  )

inits.add.non <- function(){    list(
  "b1" = rnorm(k1,0,0.1) ,
  "lambda" = rgamma(1,1,1)
  ) }

param.add = c("b1", "kappa", "lambda","rho")
```

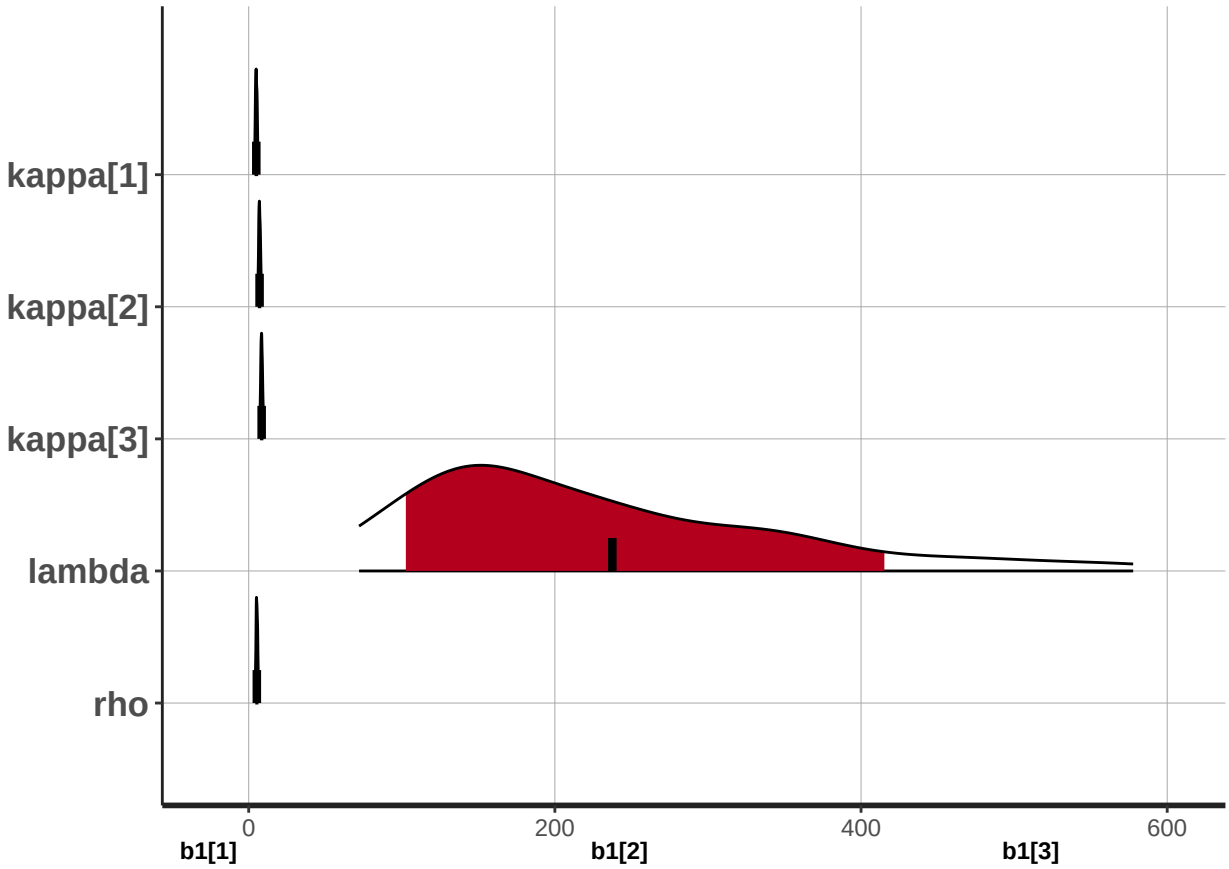
```
fit.add.non <- stan("jagam_10_aneur_ordinal_add_non.stan",
            data=datos.add.non,
            chains=3,warmup=300,iter=600,thin=2,cores=4,
            init= inits.add.non)
```

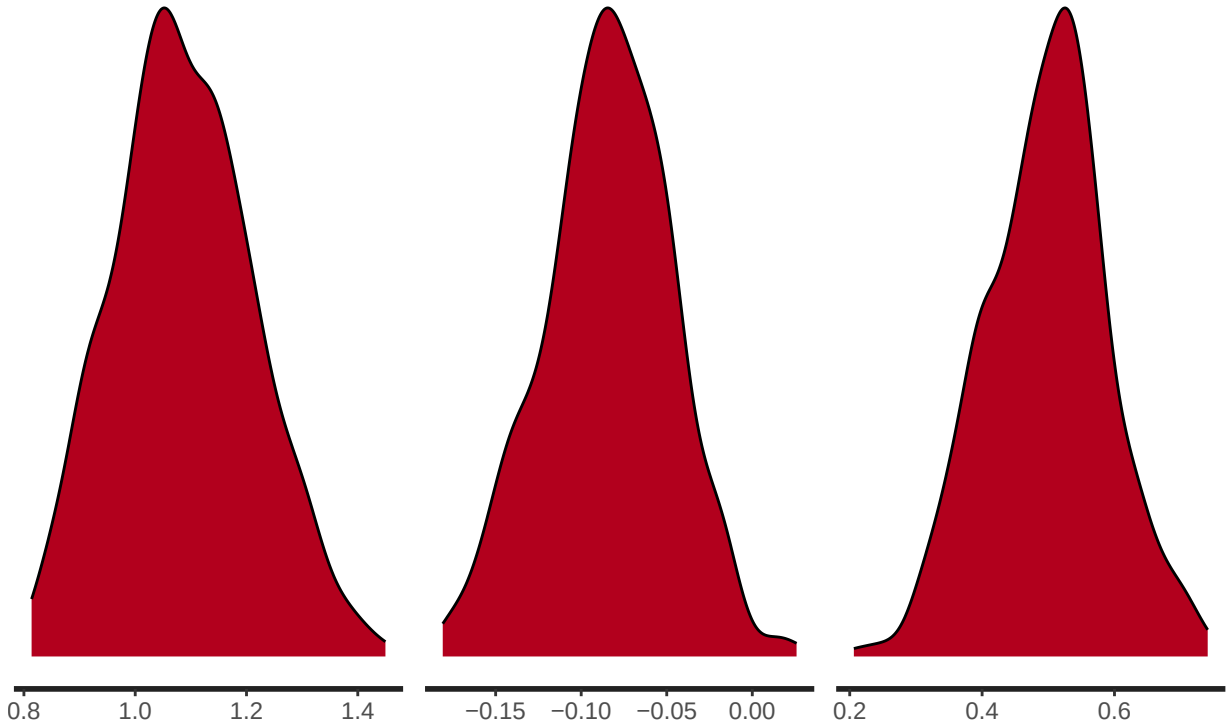
```
print(fit.add.non, pars=param.add)
```

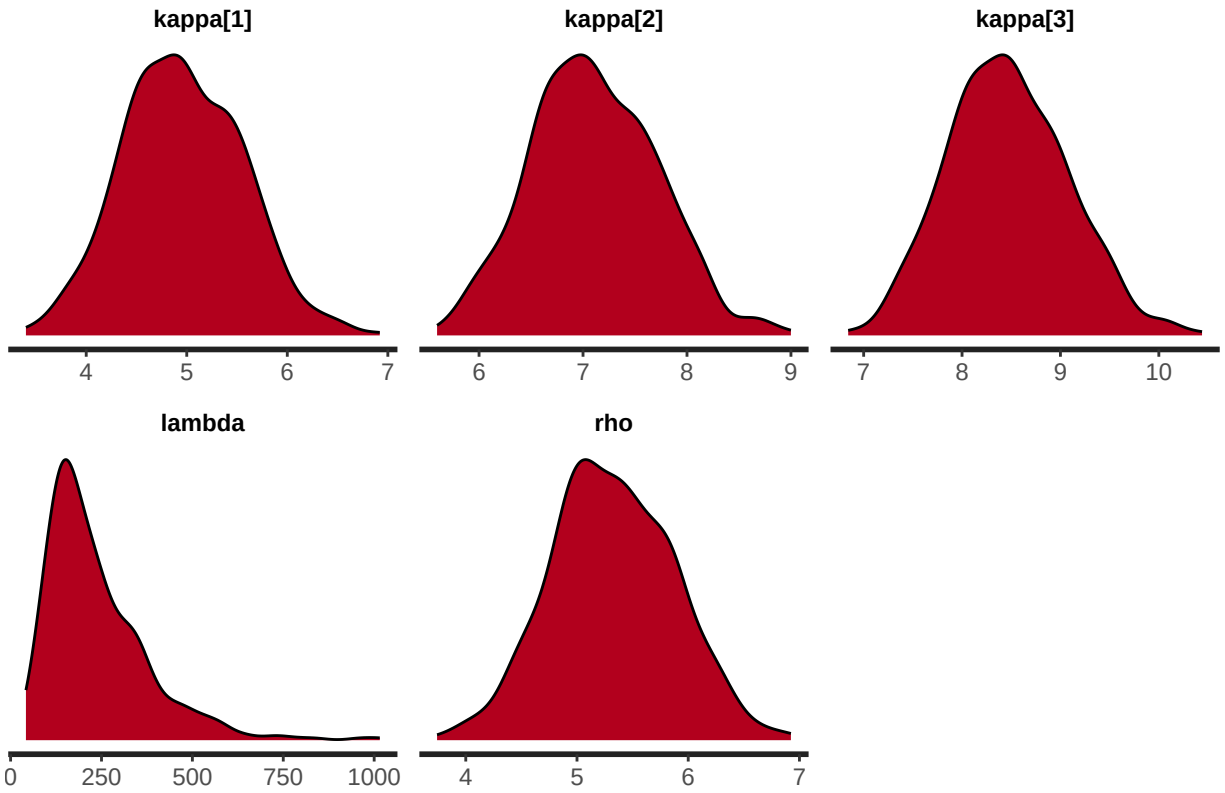
```
## Inference for Stan model: jagam_10_aneur_ordinal_add_non.
## 3 chains, each with iter=600; warmup=300; thin=2;
## post-warmup draws per chain=150, total post-warmup draws=450.
##
##            mean se_mean     sd  2.5%    25%    50%    75%  97.5% n_eff Rhat
## b1[1]      1.09    0.01   0.12  0.86   1.00   1.08   1.17   1.33   334 1.00
## b1[2]     -0.08    0.00   0.04 -0.16  -0.11  -0.08  -0.06  -0.01   323 1.00
## b1[3]      0.50    0.01   0.09  0.32   0.44   0.50   0.56   0.69   308 1.00
## kappa[1]   4.96    0.03   0.59  3.84   4.54   4.93   5.38   6.14   357 1.00
## kappa[2]   7.11    0.03   0.60  5.97   6.67   7.08   7.53   8.22   366 1.00
## kappa[3]   8.47    0.03   0.61  7.36   8.04   8.44   8.88   9.62   359 1.00
## lambda   237.37    6.98 139.99 72.44 139.72 203.30 305.55 577.08   403 1.02
## rho        5.32    0.03   0.55  4.28   4.94   5.31   5.72   6.36   395 1.01
##
## Samples were drawn using NUTS(diag_e) at Sun Aug 13 15:52:12 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
stan_trace(fit.add.non, pars=param.add)
stan_plot(fit.add.non, pars=c("b1"), point_est = "mean", show_density = TRUE)
stan_plot(fit.add.non, pars=c("kappa", "lambda","rho"), point_est = "mean", show_density = TRUE)
stan_dens(fit.add.non, pars=c("b1"))
stan_dens(fit.add.non, pars=c("kappa", "lambda","rho"))
```

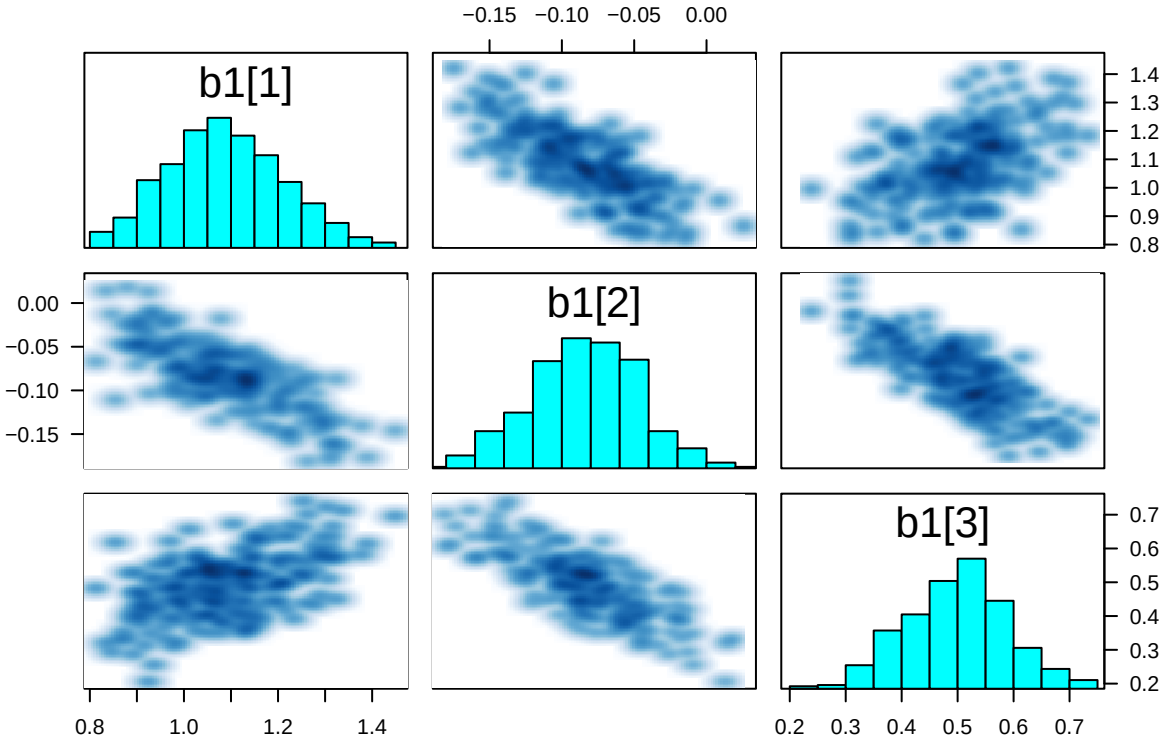

b1[1]
b1[2]
b1[3]
kappa[1]
kappa[2]
kappa[3]
lambda
rho
chain
1
2
3
300
400
500
600
0.0
0.5
1.0


kappa[1]
kappa[2]
kappa[3]
lambda
rho
0
200
400
600

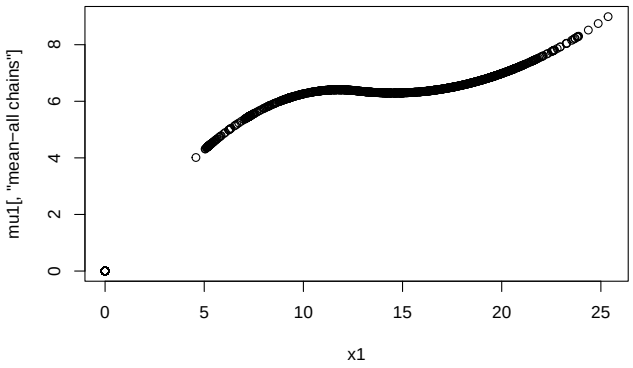

b1[1]
b1[2]
b1[3]
0.8
1.0
1.2
1.4
−0.15
−0.10
−0.05
0.00
0.2
0.4
0.6




```
pairs(fit.add.non, pars = c("b1"), las = 1)
```




```
mu1 = get_posterior_mean(fit.add.non,"mu1")
plot(x1, mu1[,"mean-all chains"])
```


mu1[, "mean−all chains"]
0
2
4
6
8
0
5
10
15
20
25
x1

7.2 LME: For a spline-based fit without constraints:

```
set.seed(123)
mod.lme.s <- gamm( y ~  s(x1,k=5) , random=list(id=~1), correlation=corAR1() )
mod.lme.s
```

```
## $lme
## Linear mixed-effects model fit by maximum likelihood
##   Data: strip.offset(mf)
##   Log-likelihood: -1201.349
##   Fixed: y.0 ~ X - 1
## X(Intercept)    Xs(x1)Fx1
##    2.0244094    0.7829412
##
## Random effects:
##  Formula: ~Xr - 1 | g
##  Structure: pdIdnot
##              Xr1      Xr2      Xr3
## StdDev: 1.477827 1.477827 1.477827
##
##  Formula: ~1 | id %in% g
##         (Intercept)  Residual
## StdDev:   0.3341393 0.6996356
##
## Correlation Structure: AR(1)
##  Formula: ~1 | g/id
##  Parameter estimate(s):
##       Phi
## 0.6517004
## Number of Observations: 1387
## Number of Groups:
##         g id %in% g
##         1       229
##
## $gam
##
## Family: gaussian
## Link function: identity
##
## Formula:
## y ~ s(x1, k = 5)
##
## Estimated degrees of freedom:
## 3.46  total = 4.46
##
##
## attr(,"class")
## [1] "gamm" "list"
```

```
summary(mod.lme.s$lme)
```

```
## Linear mixed-effects model fit by maximum likelihood
```

```
##   Data: strip.offset(mf)
##        AIC      BIC    logLik
##   2414.698 2446.107 -1201.349
##
## Random effects:
##  Formula: ~Xr - 1 | g
##  Structure: pdIdnot
##              Xr1      Xr2      Xr3
## StdDev: 1.477827 1.477827 1.477827
##
##  Formula: ~1 | id %in% g
##         (Intercept)  Residual
## StdDev:   0.3341393 0.6996356
##
## Correlation Structure: AR(1)
##  Formula: ~1 | g/id
##  Parameter estimate(s):
##       Phi
## 0.6517004
## Fixed effects:  y.0 ~ X - 1
##                  Value  Std.Error   DF  t-value p-value
## X(Intercept) 2.0244094 0.04013518 1157 50.43977       0
## Xs(x1)Fx1    0.7829412 0.12628098 1157  6.19999       0
##  Correlation:
##           X(Int)
## Xs(x1)Fx1 0.004
##
## Standardized Within-Group Residuals:
##        Min         Q1        Med         Q3        Max
## -2.1429589 -0.5564476 -0.1633574  0.4687043  2.9554799
##
## Number of Observations: 1387
## Number of Groups:
##         g id %in% g
##         1       229
```

```
summary(mod.lme.s$gam)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## y ~ s(x1, k = 5)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.02441    0.04012   50.46   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##         edf Ref.df     F p-value
## s(x1) 3.464  3.464 267.8  <2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.295
##   Scale est. = 0.48949   n = 1387
```

```
set.seed(123)
mod.lme.s <- gamm( y ~  s(x1,k=5) , random=list(id=~0+x1), correlation=corAR1() )
mod.lme.s
```

```
## $lme
## Linear mixed-effects model fit by maximum likelihood
##   Data: strip.offset(mf)
##   Log-likelihood: -977.1392
##   Fixed: y.0 ~ X - 1
## X(Intercept)    Xs(x1)Fx1
##     2.152756     0.899882
##
## Random effects:
##  Formula: ~Xr - 1 | g
##  Structure: pdIdnot
##               Xr1       Xr2       Xr3
## StdDev: 0.7564856 0.7564856 0.7564856
##
##  Formula: ~0 + x1 | id %in% g
##                 x1  Residual
## StdDev: 0.07980298 0.3972504
##
## Correlation Structure: AR(1)
##  Formula: ~1 | g/id
##  Parameter estimate(s):
##       Phi
## 0.2542532
## Number of Observations: 1387
## Number of Groups:
##         g id %in% g
##         1       229
##
## $gam
##
## Family: gaussian
## Link function: identity
##
## Formula:
## y ~ s(x1, k = 5)
##
## Estimated degrees of freedom:
## 3.12  total = 4.12
##
##
## attr(,"class")
## [1] "gamm" "list"
```

```
summary(mod.lme.s$lme)
```

```
## Linear mixed-effects model fit by maximum likelihood
##   Data: strip.offset(mf)
##        AIC      BIC    logLik
##   1966.278 1997.688 -977.1392
##
## Random effects:
##  Formula: ~Xr - 1 | g
##  Structure: pdIdnot
##               Xr1       Xr2       Xr3
## StdDev: 0.7564856 0.7564856 0.7564856
##
##  Formula: ~0 + x1 | id %in% g
##                 x1  Residual
## StdDev: 0.07980298 0.3972504
##
## Correlation Structure: AR(1)
##  Formula: ~1 | g/id
##  Parameter estimate(s):
##       Phi
## 0.2542532
## Fixed effects:  y.0 ~ X - 1
##                 Value  Std.Error   DF  t-value p-value
## X(Intercept) 2.152757 0.06775890 1157 31.77083       0
## Xs(x1)Fx1    0.899882 0.08960451 1157 10.04282       0
##  Correlation:
##           X(Int)
## Xs(x1)Fx1 0.438
##
## Standardized Within-Group Residuals:
##         Min          Q1         Med          Q3         Max
## -4.51277087 -0.55805871  0.03587902  0.44572661  3.55585080
##
## Number of Observations: 1387
## Number of Groups:
##         g id %in% g
##         1       229
```

```
summary(mod.lme.s$gam)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## y ~ s(x1, k = 5)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.15276    0.06773   31.78   <2e-16 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##         edf Ref.df    F p-value
## s(x1) 3.121  3.121 91.5  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.229
##   Scale est. = 0.15781   n = 1387
```

```
set.seed(123)
mod.lme.s <- clmm2( y_fact ~  XI1 , random=id_fact,
                    Hess=TRUE)
mod.lme.s
```

```
## Cumulative Link Mixed Model fitted with the Laplace approximation
##
## Call:
## clmm2(location = y_fact ~ XI1, random = id_fact, Hess = TRUE)
##
## Random effects:
##              Var  Std.Dev
## id_fact 25.10254 5.010244
##
## Location coefficients:
##      XI11      XI12      XI13
## 2.3521860 0.1025951 1.2356484
##
## No Scale coefficients
##
## Threshold coefficients:
##      1|2      2|3      3|4
## 11.71361 18.07009 21.48100
##
## log-likelihood: -962.3698
## AIC: 1938.74
```

```
summary(mod.lme.s)
```

```
## Cumulative Link Mixed Model fitted with the Laplace approximation
##
## Call:
## clmm2(location = y_fact ~ XI1, random = id_fact, Hess = TRUE)
##
## Random effects:
##              Var  Std.Dev
## id_fact 25.10254 5.010244
##
## Location coefficients:
##      Estimate   Std. Error z value    Pr(>|z|)
## XI11     2.3522     0.0021  1115.3832 < 2.22e-16
## XI12     0.1026     0.0021    48.7109 < 2.22e-16
```

```
## XI13     1.2356     0.0021   585.0572 < 2.22e-16
##
## No scale coefficients
##
## Threshold coefficients:
##     Estimate   Std. Error z value
## 1|2    11.7136     0.0021  5545.9187
## 2|3    18.0701     0.0021  8555.0873
## 3|4    21.4810     0.0021 10171.1007
##
## log-likelihood: -962.3698
## AIC: 1938.74
## Condition number of Hessian: 1.00795
```

```
datos.lme.add.non <- list( y = y ,
                           id = id ,
               n = length(y) ,
               N = N , Ni = Ni,
               k1=k1,
              XI1 = XI1,
              x1 = x1,
              zero = rep(0,1+k1),
              S1=S1  )
inits.lme.add.non <- function(){    list(
  "b1" = rnorm(k1,0,0.1) ,
  "lambda" = rgamma(1,1,1) ,
  "invsig2" = rgamma(1,1,1) ,
  "bre0" = rnorm(N,0,0.1)
  ) }
param.lme.add = c("b1", "kappa", "lambda","rho", "invsig2","sig2","sigma")
```

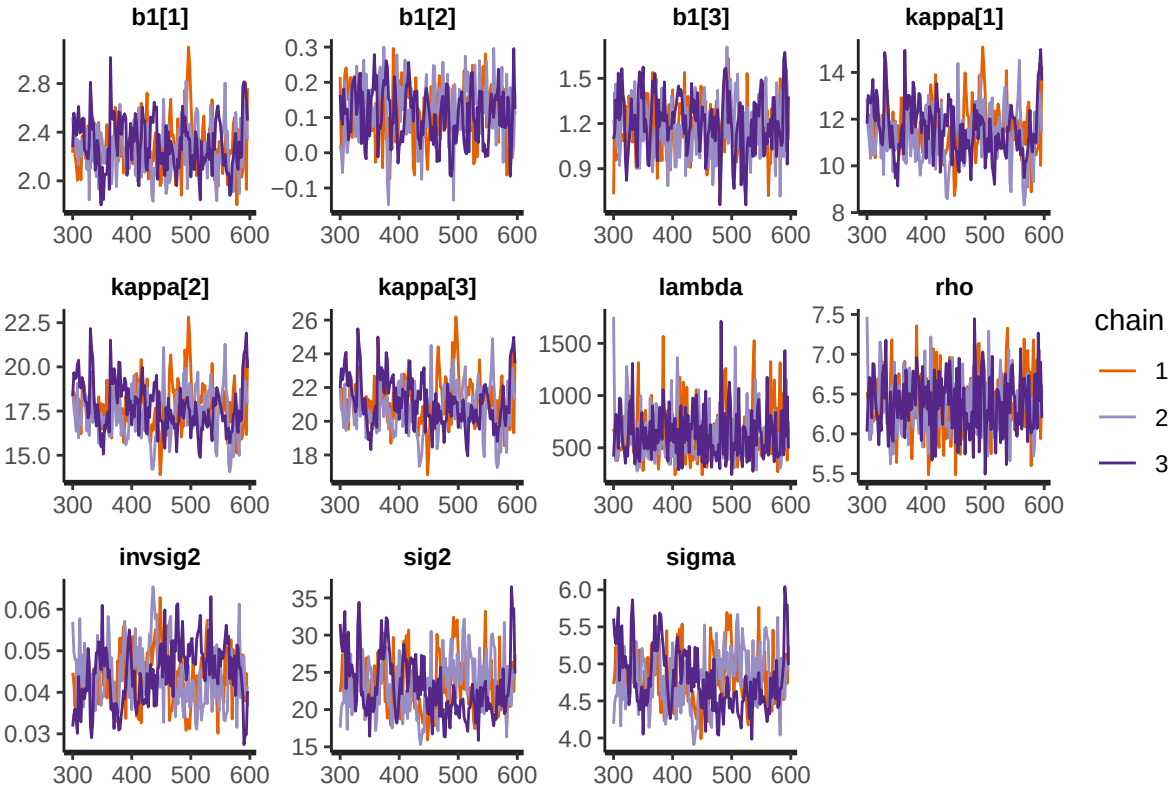
```
fit.lme.add.non.reintrcpt <- stan("jagam_10_aneur_ordinal_lme_add_non_reintrcpt.stan",
            data=datos.lme.add.non,
            chains=3,warmup=300,iter=600,thin=2,cores=4,
            init= inits.lme.add.non)
```

```
print(fit.lme.add.non.reintrcpt, pars=param.lme.add)
```

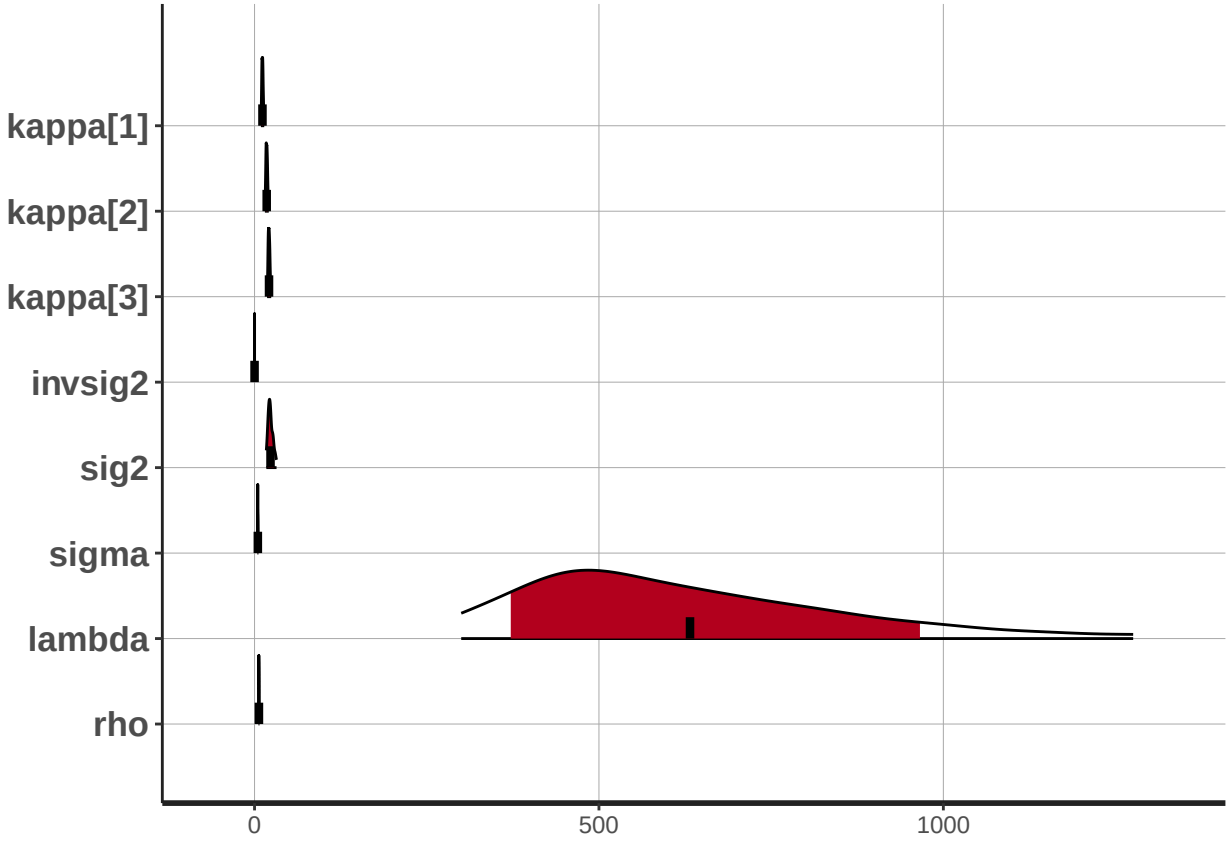
```
## Inference for Stan model: jagam_10_aneur_ordinal_lme_add_non_reintrcpt.
## 3 chains, each with iter=600; warmup=300; thin=2;
## post-warmup draws per chain=150, total post-warmup draws=450.
##
##            mean se_mean     sd   2.5%    25%    50%    75%   97.5% n_eff Rhat
## b1[1]      2.28    0.02   0.21   1.90   2.14   2.27   2.41    2.77   190 1.01
## b1[2]      0.12    0.00   0.08  -0.05   0.06   0.12   0.17    0.26   266 1.01
## b1[3]      1.19    0.01   0.19   0.84   1.07   1.18   1.32    1.54   279 1.01
## kappa[1]  11.50    0.10   1.16   9.22  10.74  11.47  12.18   14.13   139 1.04
## kappa[2]  17.67    0.13   1.40  15.01  16.74  17.60  18.54   20.74   121 1.05
## kappa[3]  21.03    0.13   1.46  18.20  20.01  20.94  21.94   24.36   120 1.05
## lambda   628.99   11.13 246.22 297.90 451.09 574.57 755.98 1256.35   490 1.00
## rho        6.37    0.02   0.37   5.70   6.11   6.35   6.63    7.14   507 1.00
## invsig2    0.04    0.00   0.01   0.03   0.04   0.04   0.05    0.06   105 1.05
## sig2      23.23    0.36   3.78  17.10  20.57  22.65  25.80   31.91   110 1.05
```

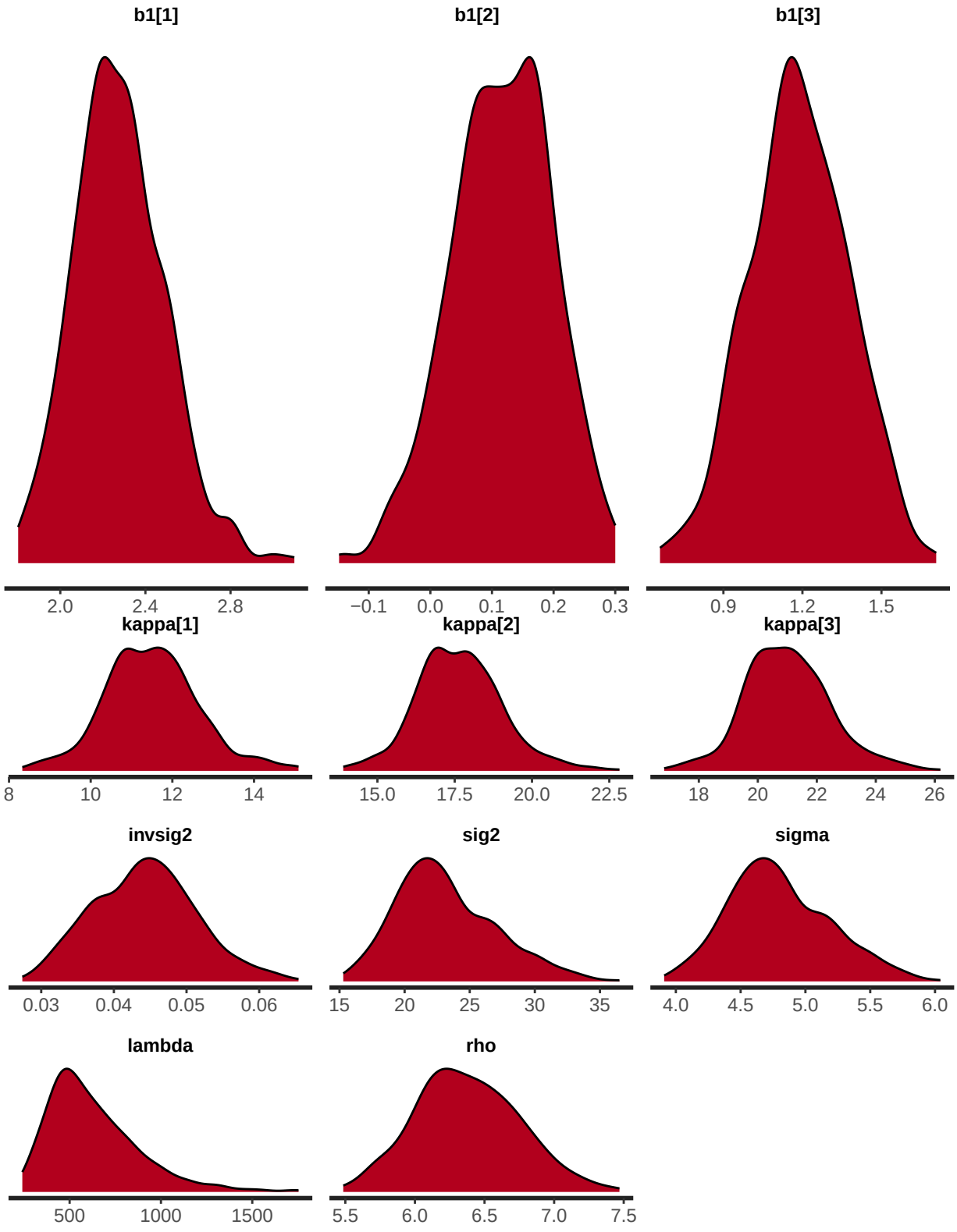
```
## sigma      4.80    0.04   0.39   4.13   4.54   4.76   5.08    5.65   108 1.05
##
## Samples were drawn using NUTS(diag_e) at Sun Aug 13 15:58:31 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
stan_trace(fit.lme.add.non.reintrcpt, pars=param.lme.add)
stan_plot(fit.lme.add.non.reintrcpt, pars=c("b1"), point_est = "mean", show_density = TRUE)
stan_plot(fit.lme.add.non.reintrcpt, pars=c("kappa", "invsig2","sig2","sigma",  "lambda","rho"), point_
stan_dens(fit.lme.add.non.reintrcpt, pars=c("b1"))
stan_dens(fit.lme.add.non.reintrcpt, pars=c("kappa", "invsig2","sig2","sigma", "lambda","rho"))
```

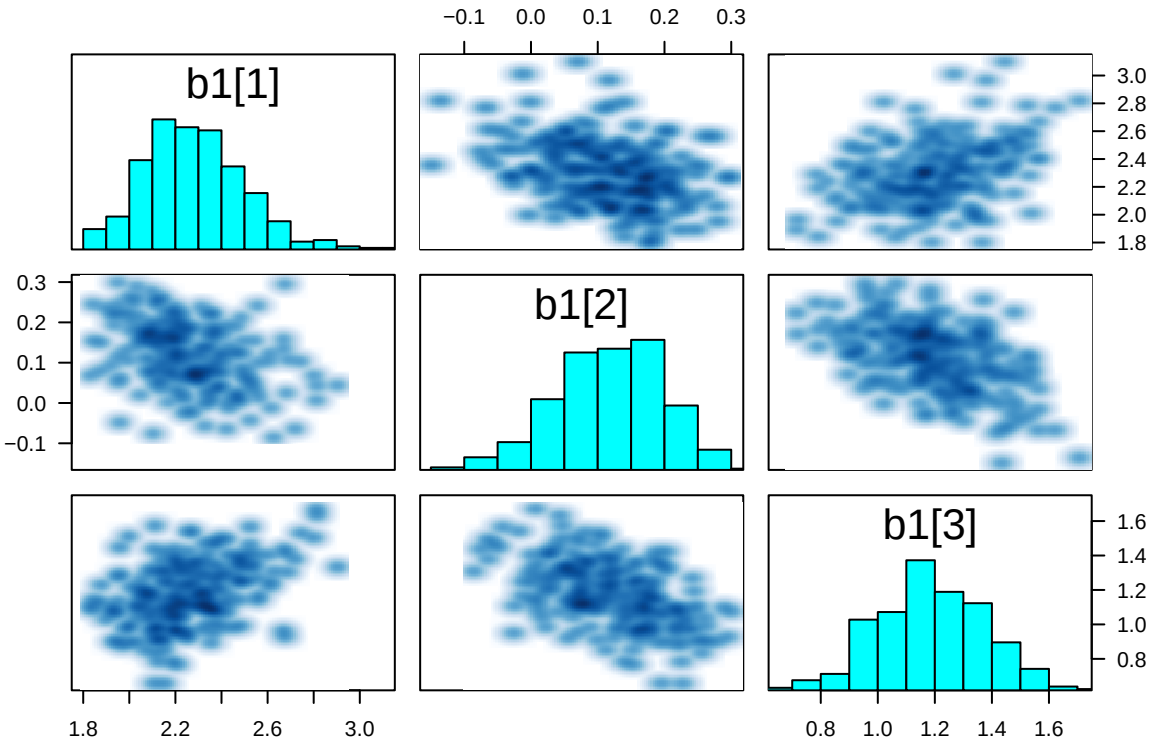



b1[1]
b1[2]
b1[3]
0
1
2
3

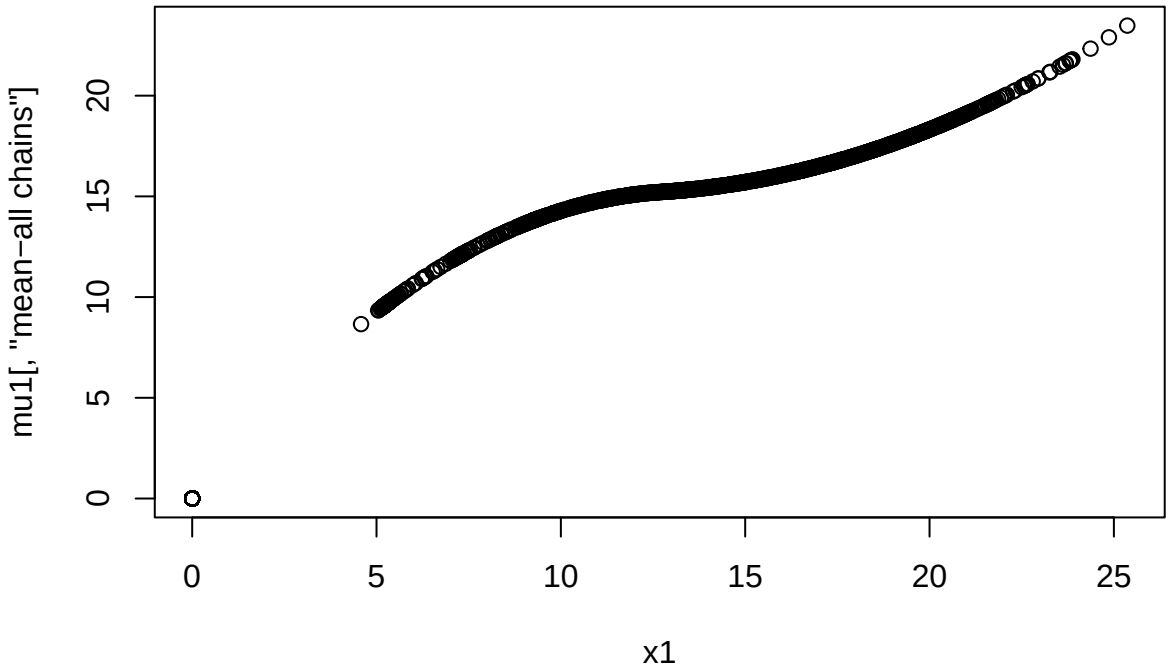

kappa[1]
kappa[2]
kappa[3]
invsig2
sig2
sigma
lambda
rho
0
500
1000



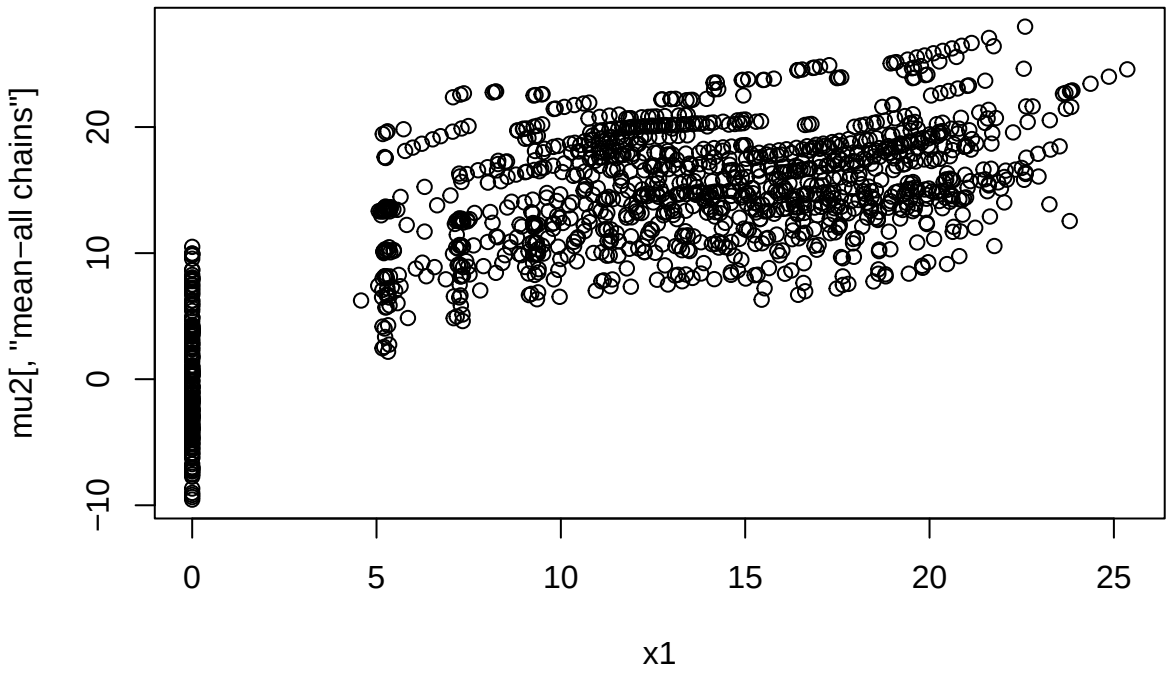

```
pairs(fit.lme.add.non.reintrcpt, pars = c("b1"), las = 1)
```




```
mu1 = get_posterior_mean(fit.lme.add.non.reintrcpt,"mu1")
plot(x1,mu1[,"mean-all chains"])
```




```
mu2 = get_posterior_mean(fit.lme.add.non.reintrcpt,"mu2")
plot(x1,mu2[,"mean-all chains"])
```



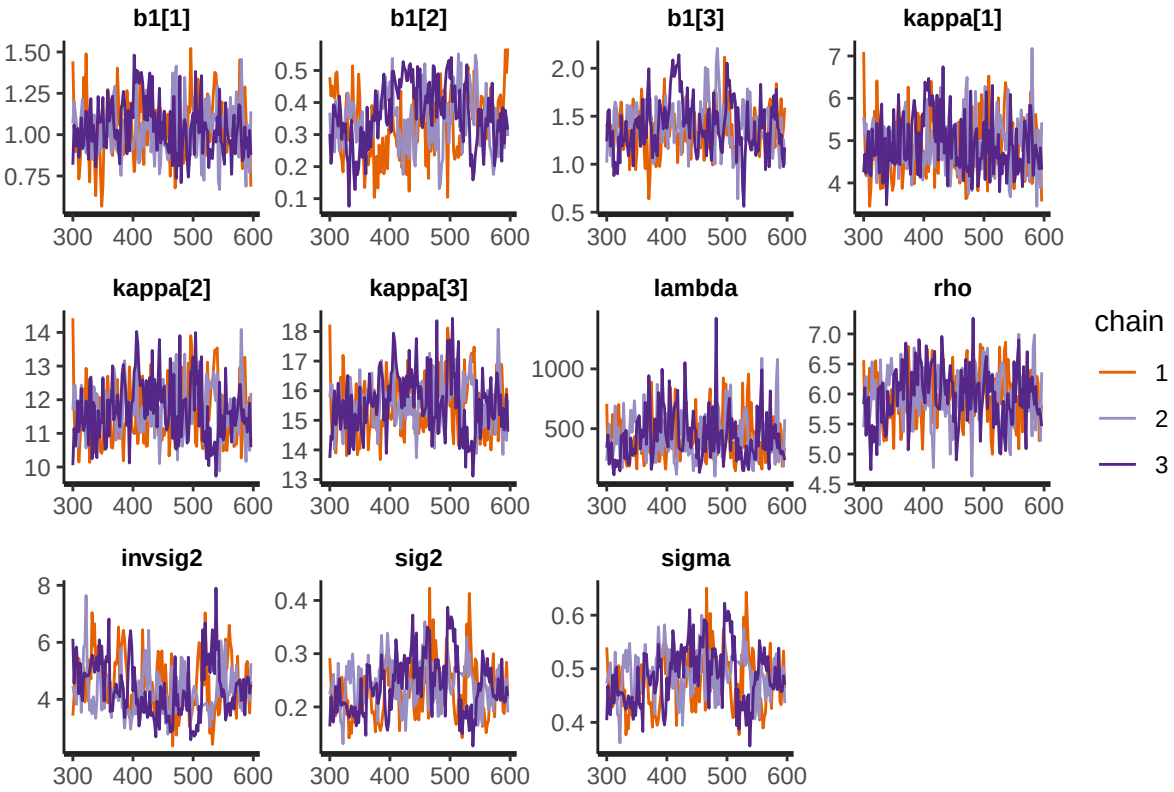

```
fit.lme.add.non.reslope <- stan("jagam_10_aneur_ordinal_lme_add_non_reslope.stan",
            data=datos.lme.add.non,
            chains=3,warmup=300,iter=600,thin=2,cores=4,
            init= inits.lme.add.non)
```

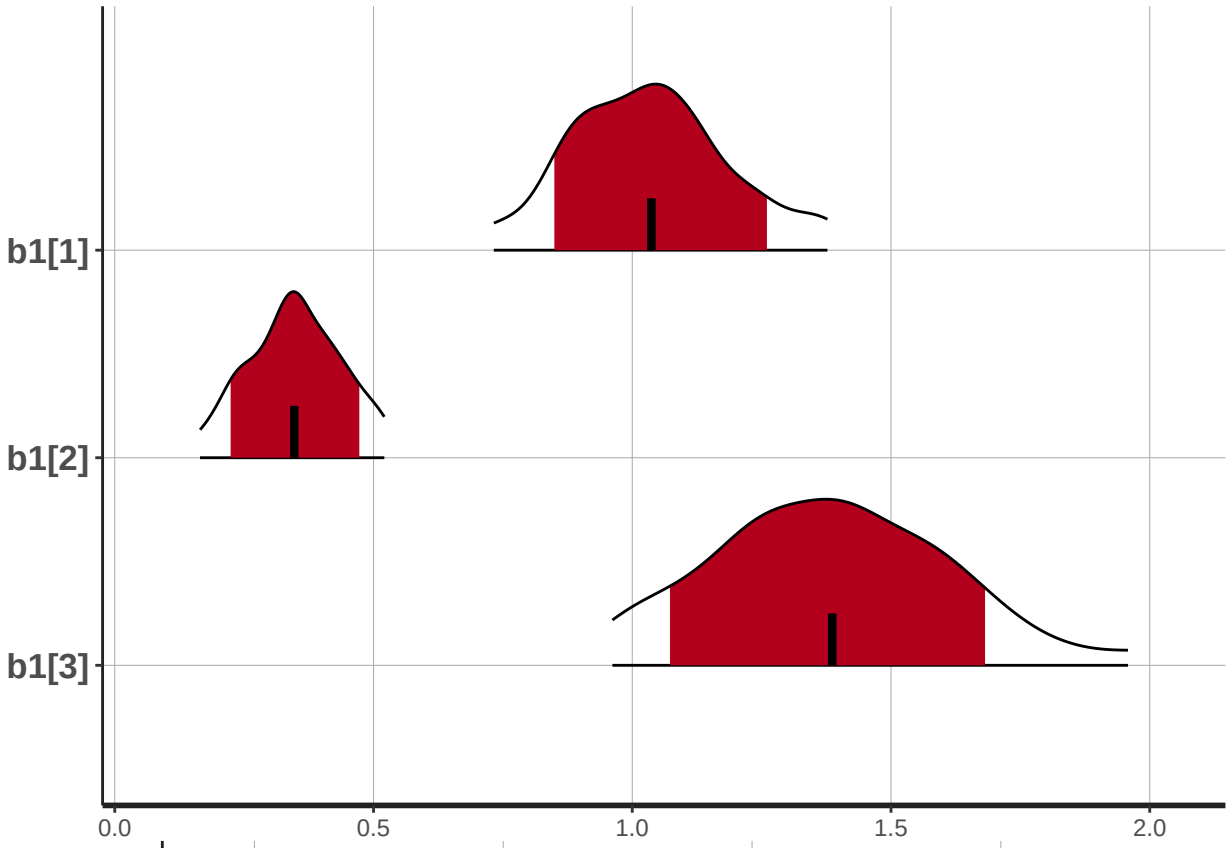
```
print(fit.lme.add.non.reslope, pars=param.lme.add)
```

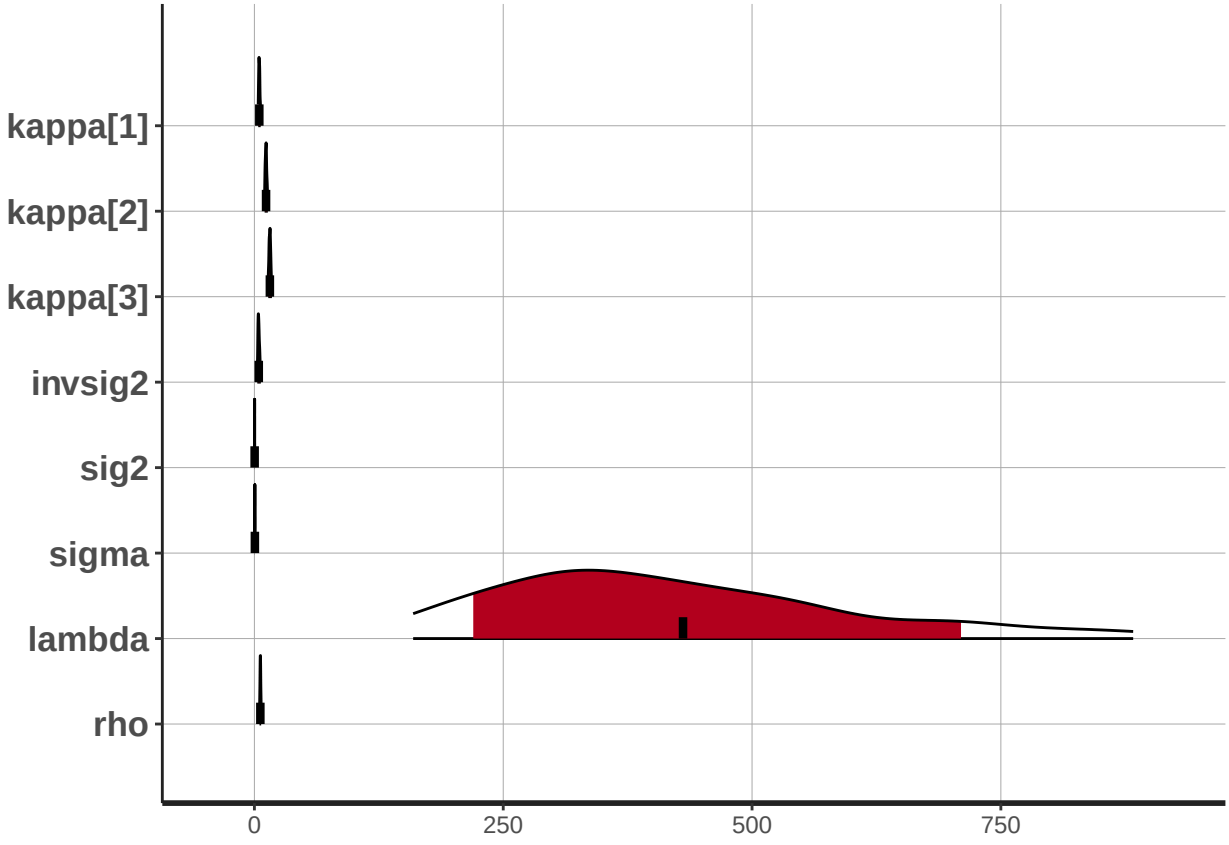
```
## Inference for Stan model: jagam_10_aneur_ordinal_lme_add_non_reslope.
## 3 chains, each with iter=600; warmup=300; thin=2;
## post-warmup draws per chain=150, total post-warmup draws=450.
##
##            mean se_mean     sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
## b1[1]      1.04    0.01   0.16   0.73   0.92   1.03   1.13   1.38   162 1.02
## b1[2]      0.35    0.01   0.09   0.17   0.28   0.35   0.41   0.52    43 1.03
## b1[3]      1.39    0.02   0.25   0.96   1.22   1.38   1.55   1.96   120 1.01
## kappa[1]   4.92    0.04   0.66   3.82   4.44   4.87   5.35   6.26   295 1.01
## kappa[2]  11.69    0.06   0.83  10.25  11.06  11.66  12.21  13.41   180 1.01
## kappa[3]  15.54    0.08   0.93  13.94  14.83  15.53  16.17  17.46   138 1.01
## lambda   431.41   12.95 193.42 159.60 293.96 394.43 533.91 882.85   223 1.01
## rho        5.97    0.03   0.45   5.07   5.68   5.98   6.28   6.78   200 1.01
## invsig2    4.39    0.09   0.90   2.88   3.75   4.26   4.94   6.44    92 1.03
## sig2       0.24    0.01   0.05   0.16   0.20   0.23   0.27   0.35    87 1.03
## sigma      0.48    0.01   0.05   0.39   0.45   0.48   0.52   0.59    87 1.03
##
## Samples were drawn using NUTS(diag_e) at Sun Aug 13 16:01:47 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

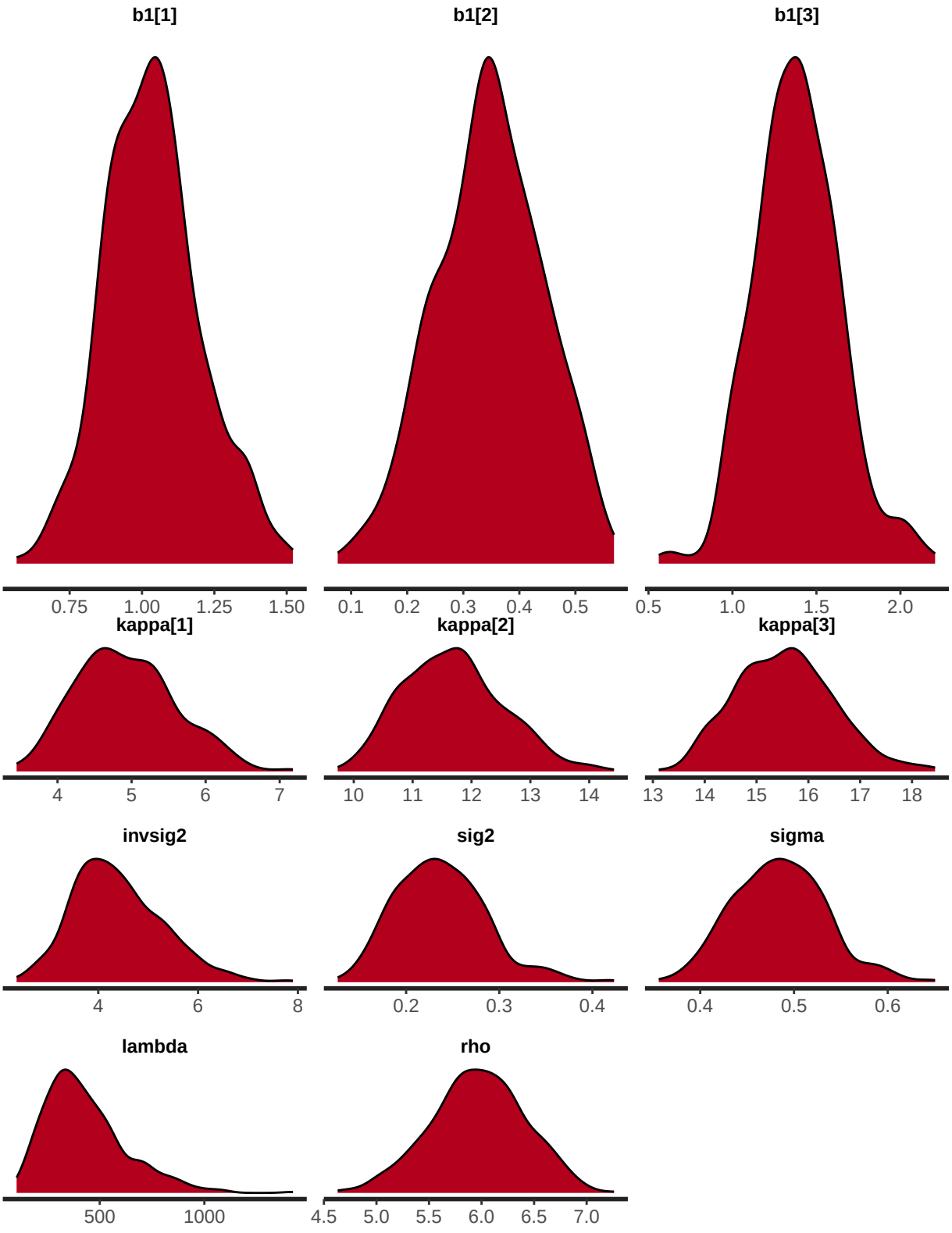
```
stan_trace(fit.lme.add.non.reslope, pars=param.lme.add)
stan_plot(fit.lme.add.non.reslope, pars=c("b1"), point_est = "mean", show_density = TRUE)
```

```
stan_plot(fit.lme.add.non.reslope, pars=c("kappa", "invsig2","sig2","sigma",  "lambda","rho"), point_es
stan_dens(fit.lme.add.non.reslope, pars=c("b1"))
stan_dens(fit.lme.add.non.reslope, pars=c("kappa", "invsig2","sig2","sigma", "lambda","rho"))
```

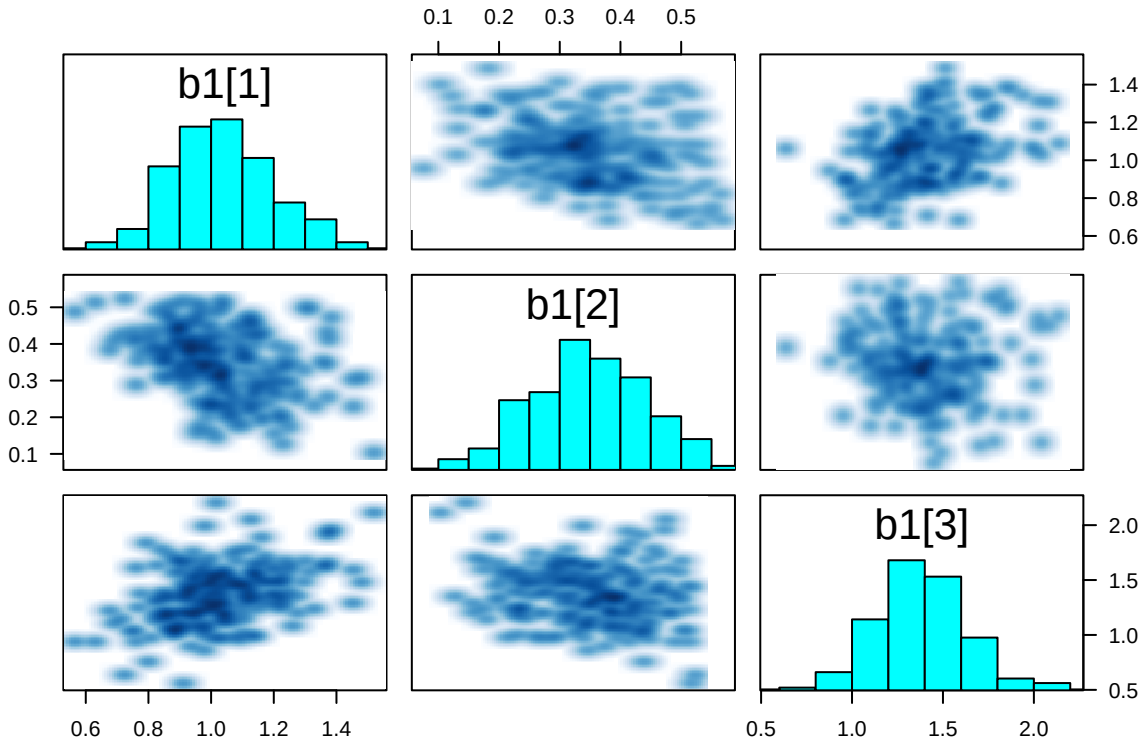



b1[1]
b1[2]
b1[3]
0.0
0.5
1.0
1.5
2.0

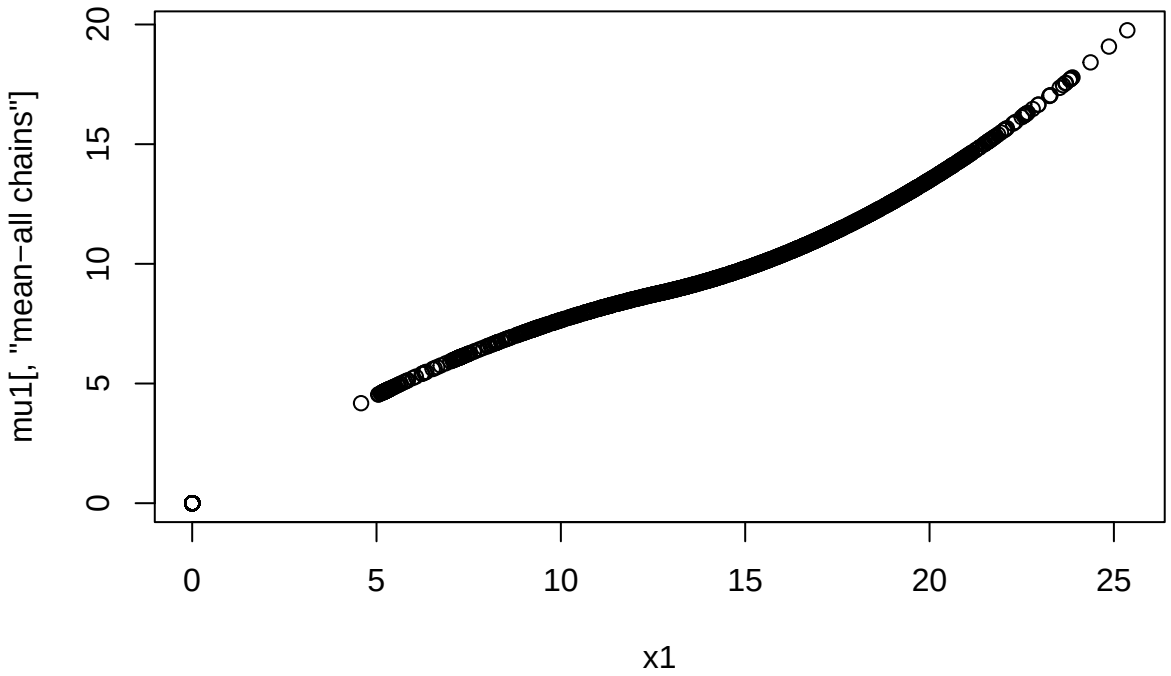

kappa[1]
kappa[2]
kappa[3]
invsig2
sig2
sigma
lambda
rho
0
250
500
750



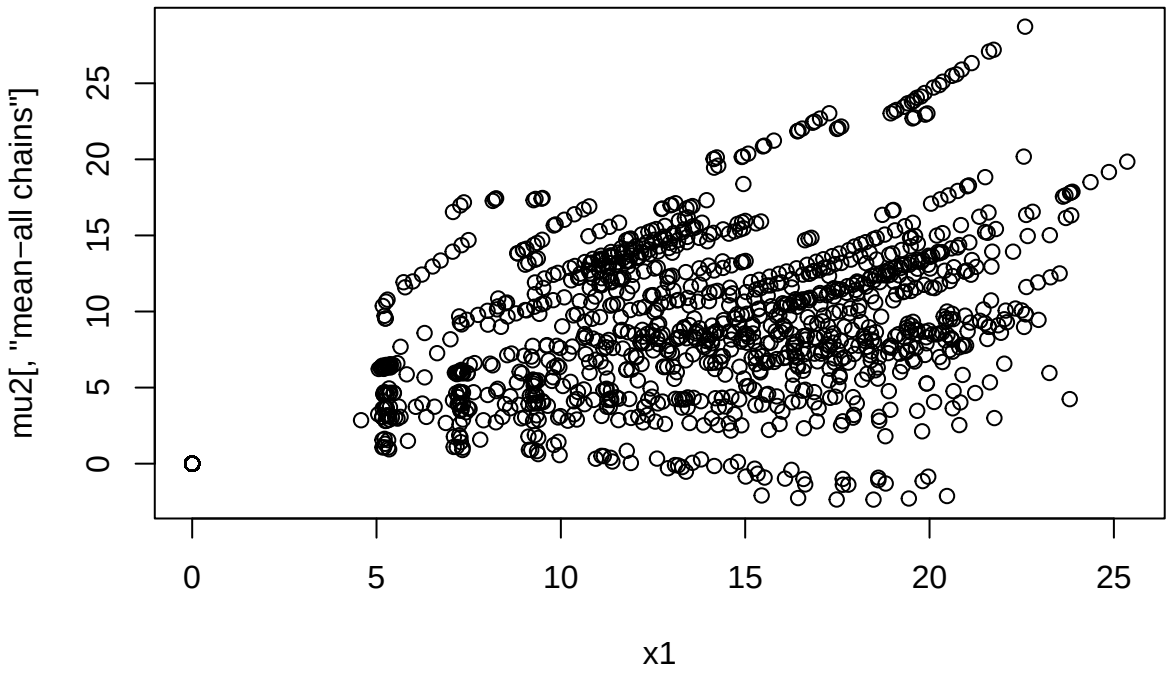

```
pairs(fit.lme.add.non.reslope, pars = c("b1"), las = 1)
```




```
mu1 = get_posterior_mean(fit.lme.add.non.reslope,"mu1")
plot(x1,mu1[,"mean-all chains"])
```




```
mu2 = get_posterior_mean(fit.lme.add.non.reslope,"mu2")
plot(x1,mu2[,"mean-all chains"])
```


mu2[, "mean−all chains"]
0
5
10
15
20
25
x1

8. Spline con restricciones creciente

8.1. LIN: Spline con restricciones creciente

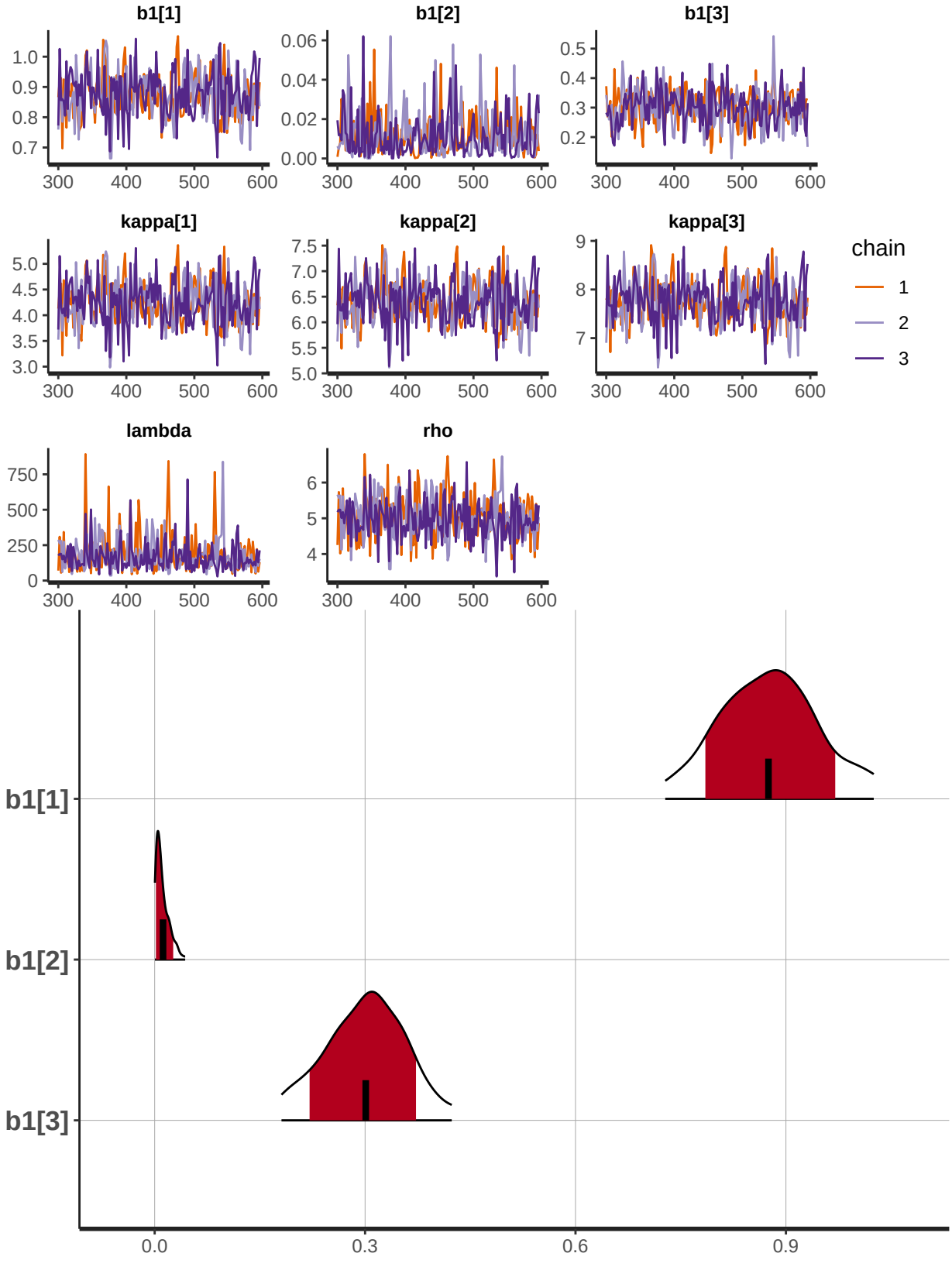
We can model the relationship using I-spline basis functions:

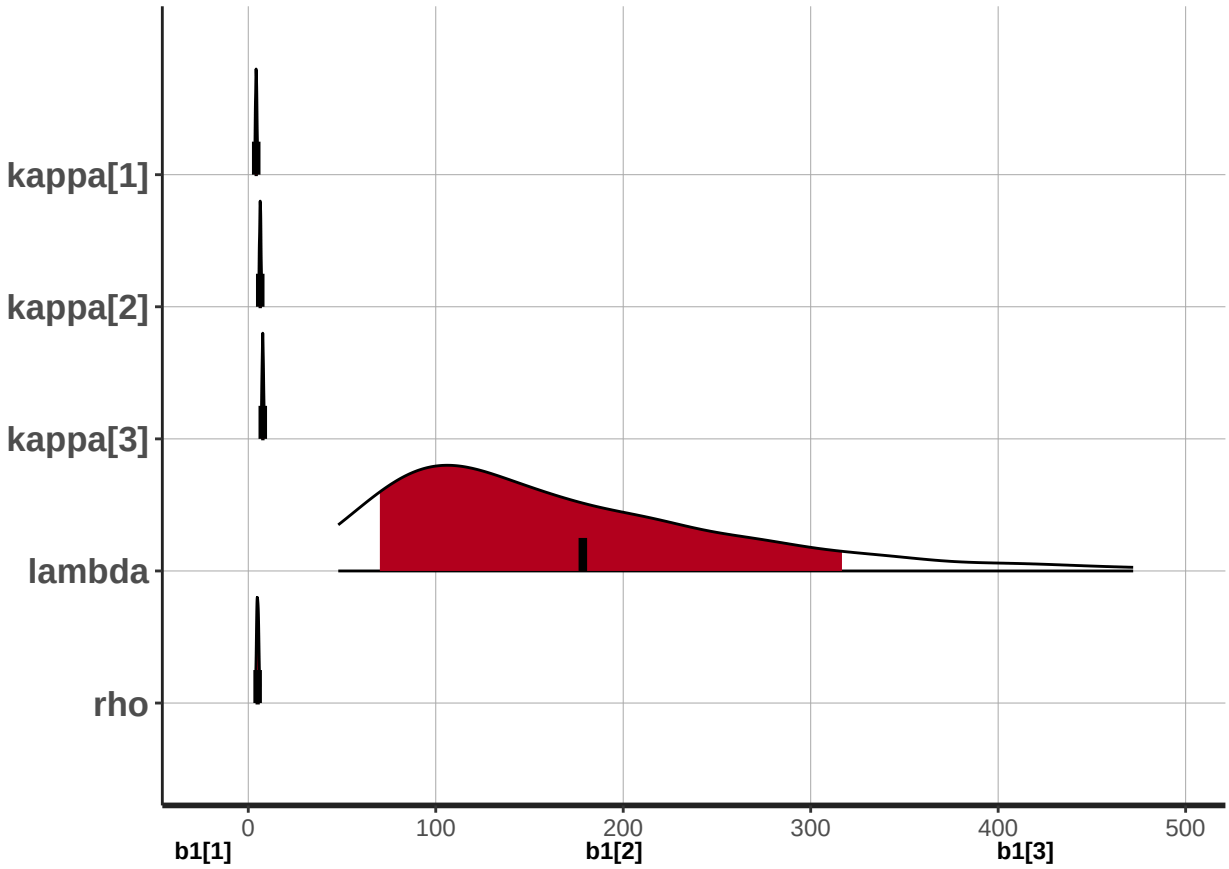
```
datos.add.incr <- list( y = y ,
                n = length(y) , k1=k1,
               XI1 = XI1,
               zero = rep(0,1+k1),
               S1=S1  )

inits.add.incr <- function(){   list(
  "b1" = abs(rnorm(k1,0,0.1)) ,
  "lambda" = rgamma(1,1,1)
  ) }
```

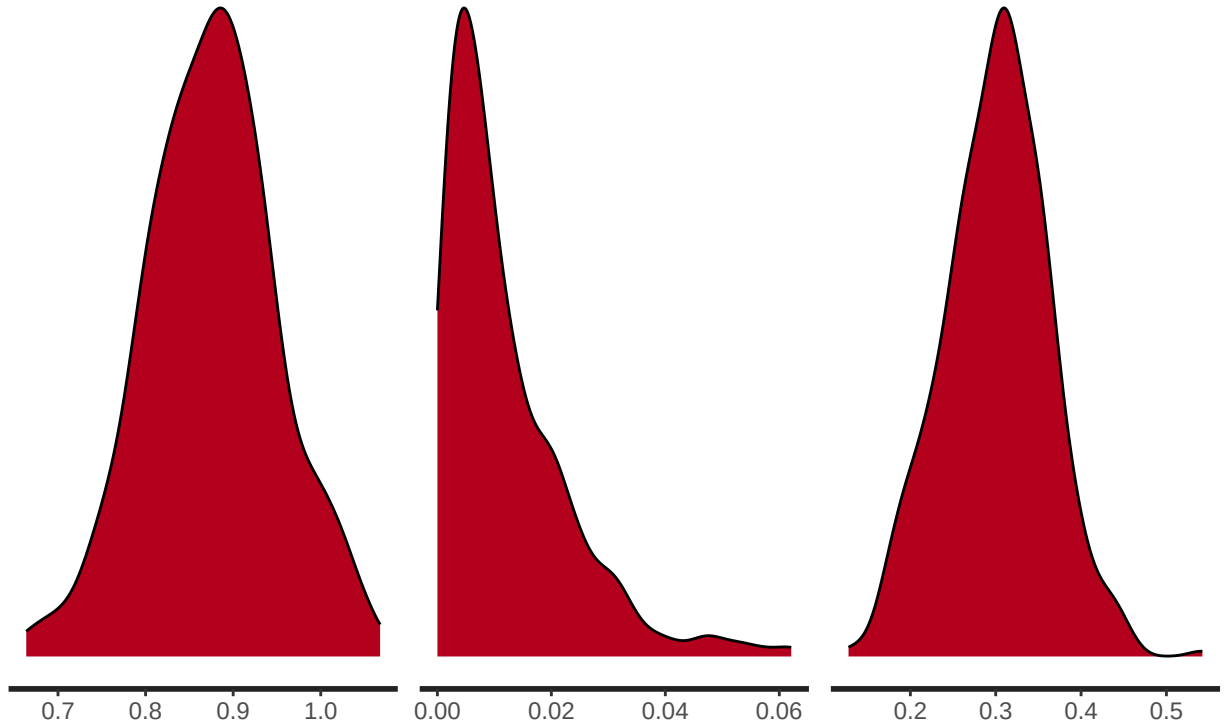
```
fit.add.incr <- stan("jagam_10_aneur_ordinal_add_incr.stan",
             data=datos.add.incr,
             chains=3,warmup=300,iter=600,thin=2,cores=4,
             init= inits.add.incr )

print(fit.add.incr, pars=param.add)
```

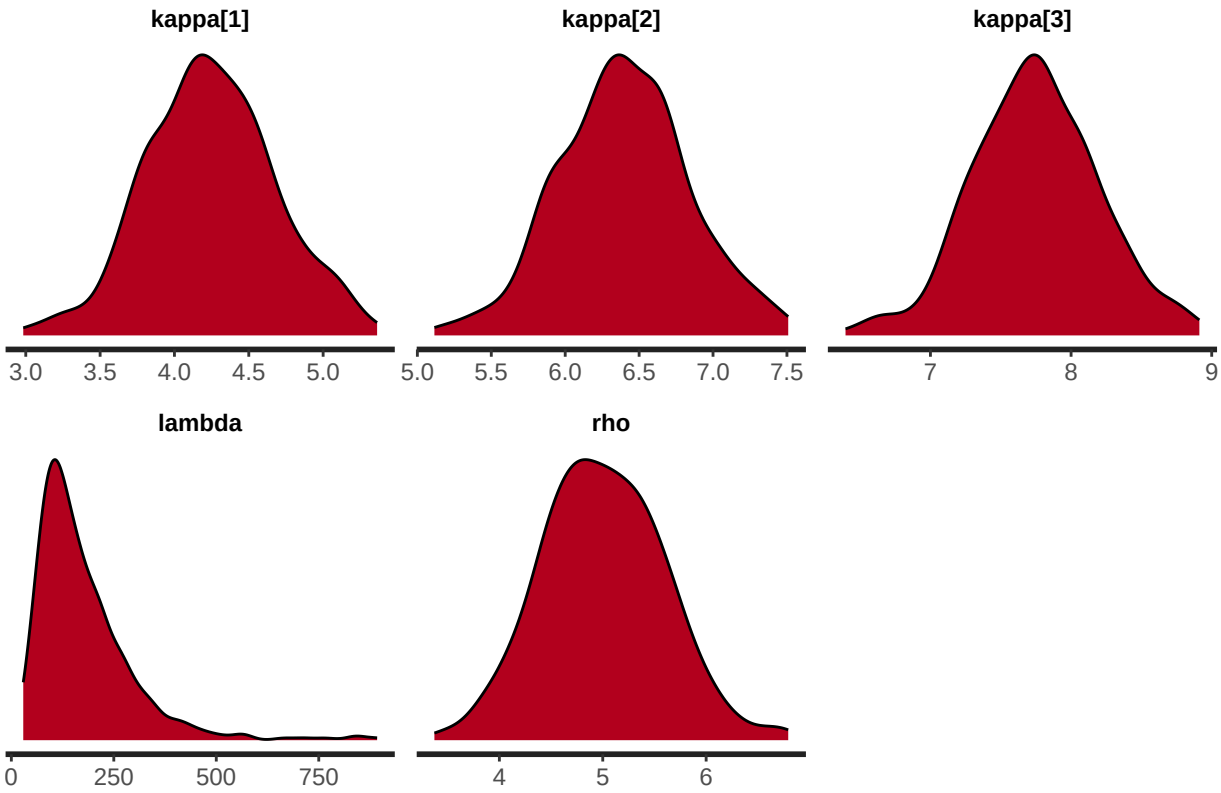
```
## Inference for Stan model: jagam_10_aneur_ordinal_add_incr.
## 3 chains, each with iter=600; warmup=300; thin=2;
## post-warmup draws per chain=150, total post-warmup draws=450.
##
##            mean se_mean     sd  2.5%   25%    50%    75%  97.5% n_eff Rhat
## b1[1]      0.88    0.00   0.08  0.73  0.83   0.88   0.93   1.03   365    1
## b1[2]      0.01    0.00   0.01  0.00  0.00   0.01   0.02   0.04   395    1
## b1[3]      0.30    0.00   0.06  0.18  0.26   0.30   0.34   0.42   392    1
## kappa[1]   4.25    0.02   0.43  3.37  3.96   4.23   4.54   5.12   365    1
## kappa[2]   6.40    0.02   0.44  5.50  6.10   6.41   6.68   7.30   353    1
## kappa[3]   7.75    0.02   0.45  6.87  7.45   7.74   8.06   8.69   359    1
## lambda   177.71    6.19 120.70 47.30 99.63 147.41 219.93 471.94   381    1
## rho        5.00    0.03   0.59  3.86  4.60   4.99   5.39   6.16   439    1
##
## Samples were drawn using NUTS(diag_e) at Sun Aug 13 16:03:39 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
stan_trace(fit.add.incr,pars=param.add)
stan_plot(fit.add.incr,pars=c("b1"), point_est = "mean", show_density = TRUE)
stan_plot(fit.add.incr,pars=c("kappa", "lambda","rho"), point_est = "mean", show_density = TRUE)
stan_dens(fit.add.incr,pars=c("b1"))
stan_dens(fit.add.incr,pars=c("kappa", "lambda","rho"))
```

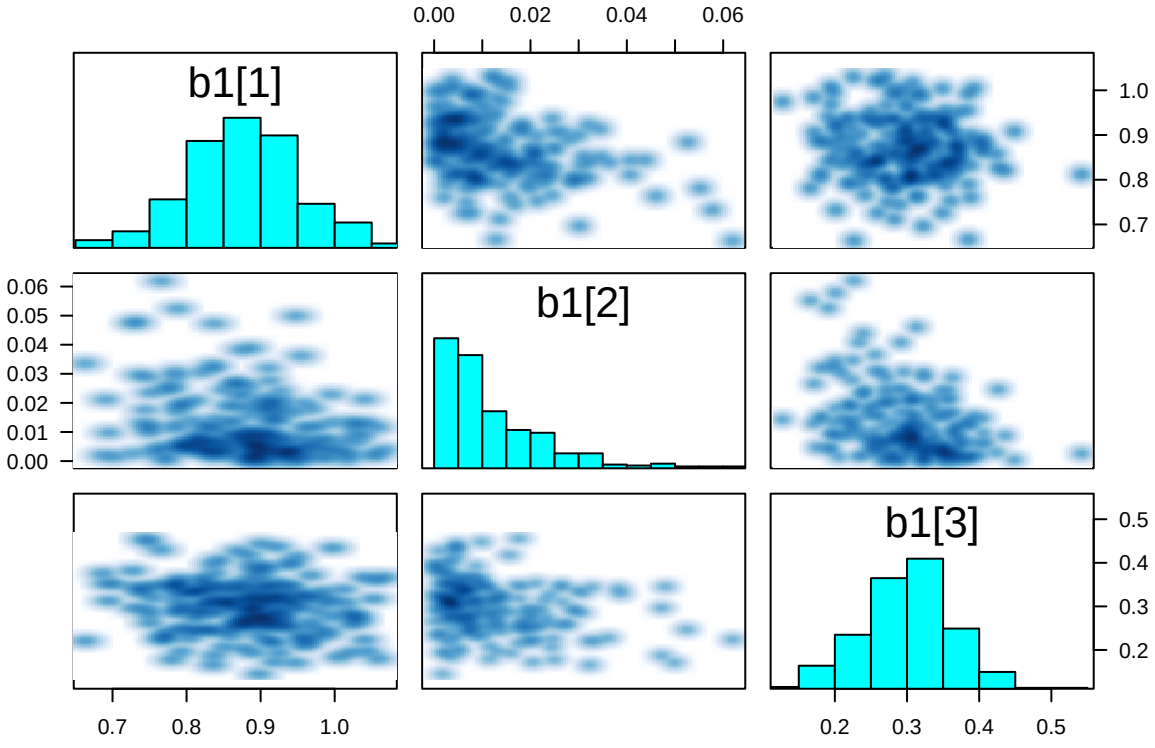

b1[1]
b1[2]
b1[3]
kappa[1]
kappa[2]
kappa[3]
lambda
rho
chain
1
2
3
b1[1]
b1[2]
b1[3]
0.0
0.3
0.6
0.9


kappa[1]
kappa[2]
kappa[3]
lambda
rho
0
100
200
300
400
500
b1[1]
b1[2]
b1[3]

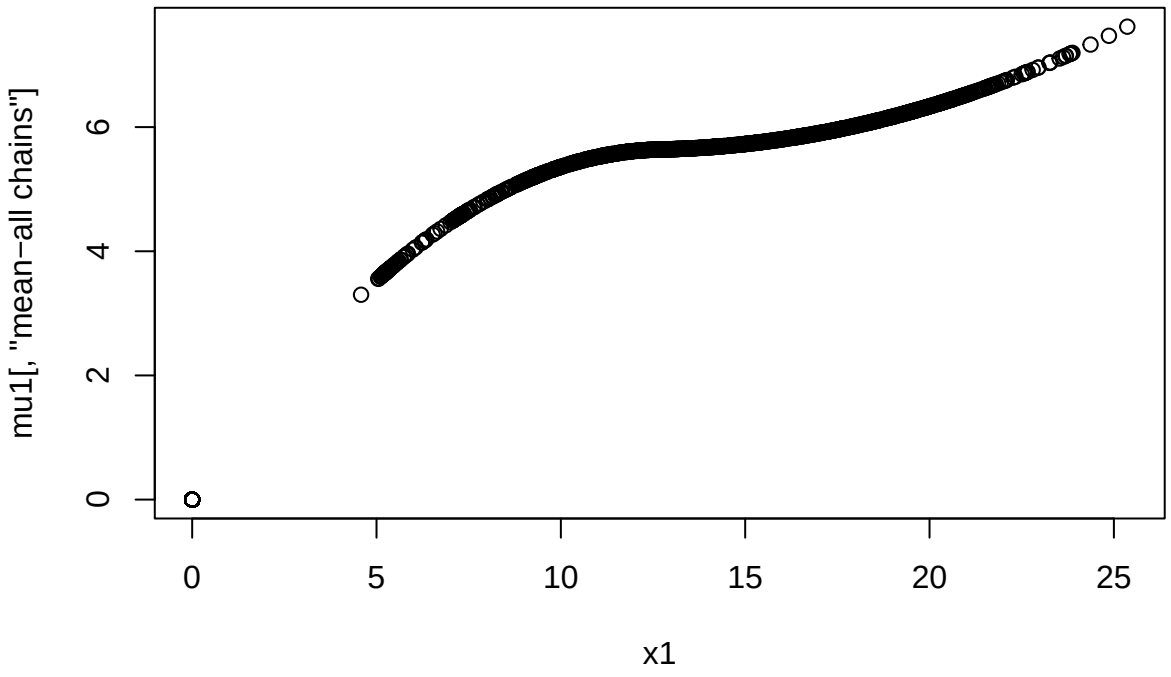

0.7
0.8
0.9
1.0
0.00
0.02
0.04
0.06
0.2
0.3
0.4
0.5




```
pairs(fit.add.incr, pars = c("b1"), las = 1)
```




```
mu1 = get_posterior_mean(fit.add.incr,"mu1")
plot(x1,mu1[,"mean-all chains"])
```


mu1[, "mean−all chains"]
0
2
4
6
0
5
10
15
20
25
x1

8.2. LME: Spline con restricciones creciente

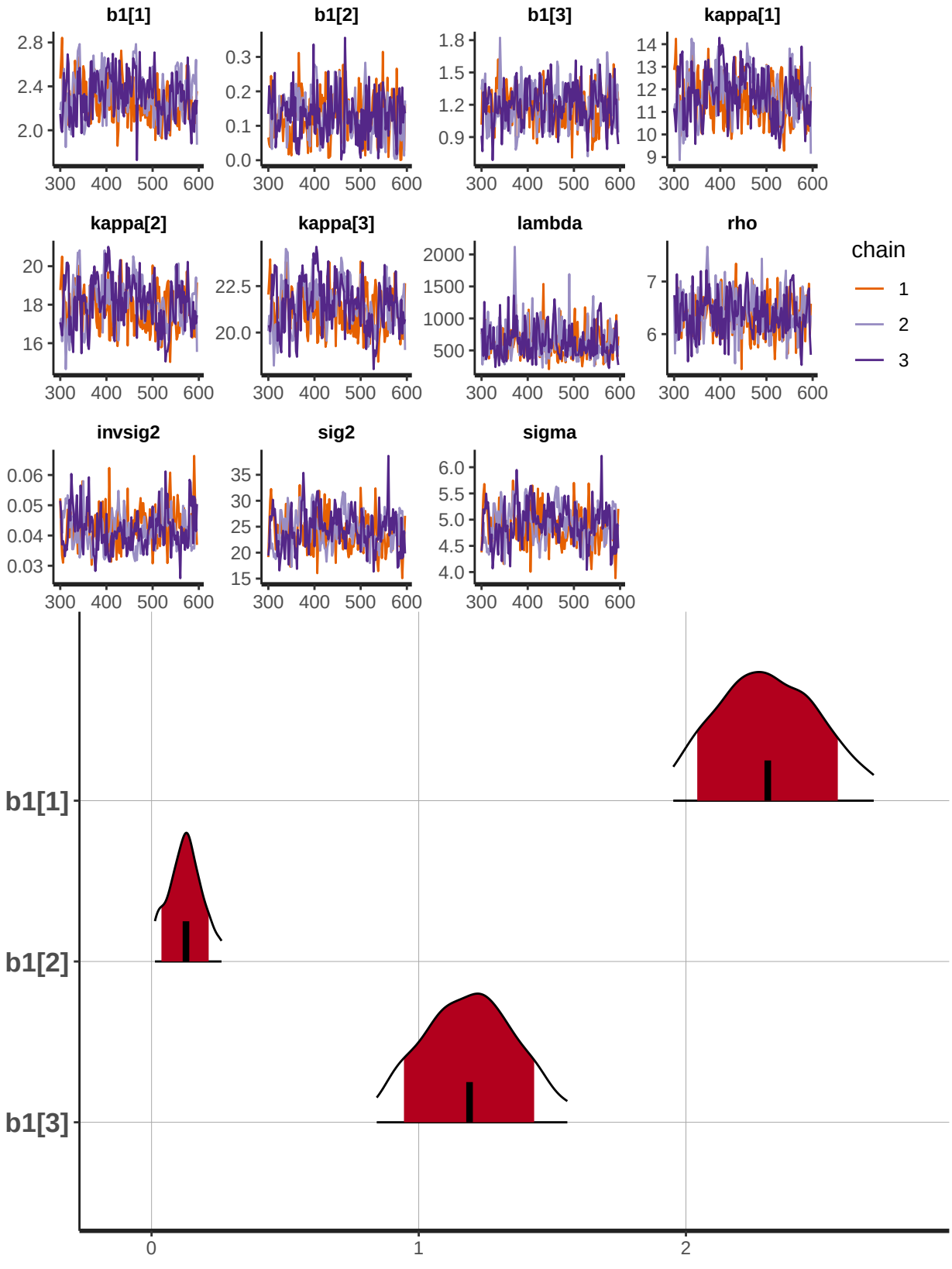
```
datos.lme.add.incr <- list( y = y ,
                            id = id ,
               n = length(y) ,
               N = N , Ni = Ni,
               k1=k1,
              XI1 = XI1,
              x1 = x1,
              zero = rep(0,1+k1),
              S1=S1  )
inits.lme.add.incr <- function(){   list(
  "b1" = abs(rnorm(k1,0,0.1)),
  "lambda" = rgamma(1,1,1) ,
  "invsig2" = rgamma(1,1,1)
  ) }
```

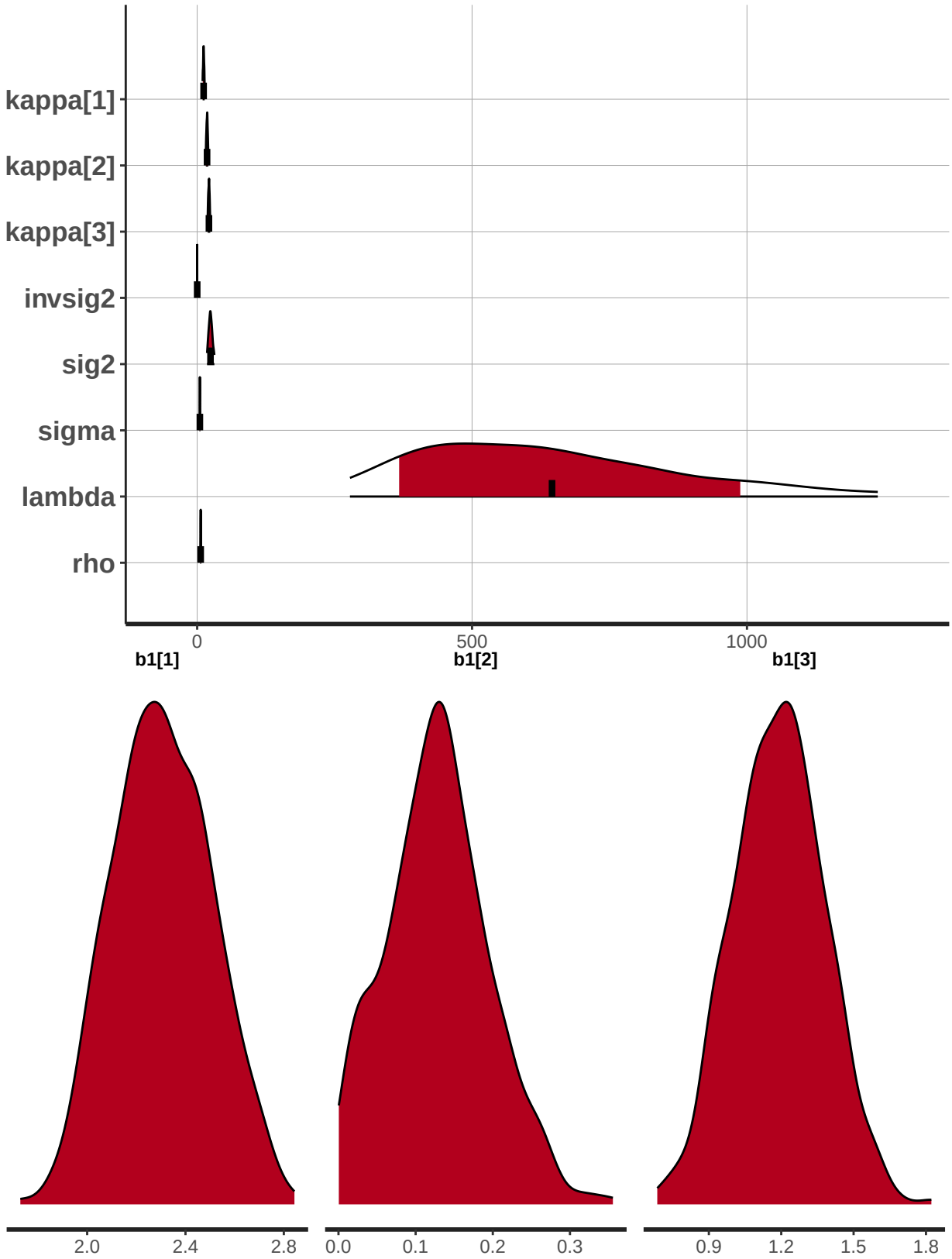
```
fit.lme.add.incr.reintrcpt <- stan("jagam_10_aneur_ordinal_lme_add_incr_reintrcpt.stan",
            data=datos.lme.add.incr,
            chains=3,warmup=300,iter=600,thin=2,cores=4,
            init= inits.lme.add.incr)
```

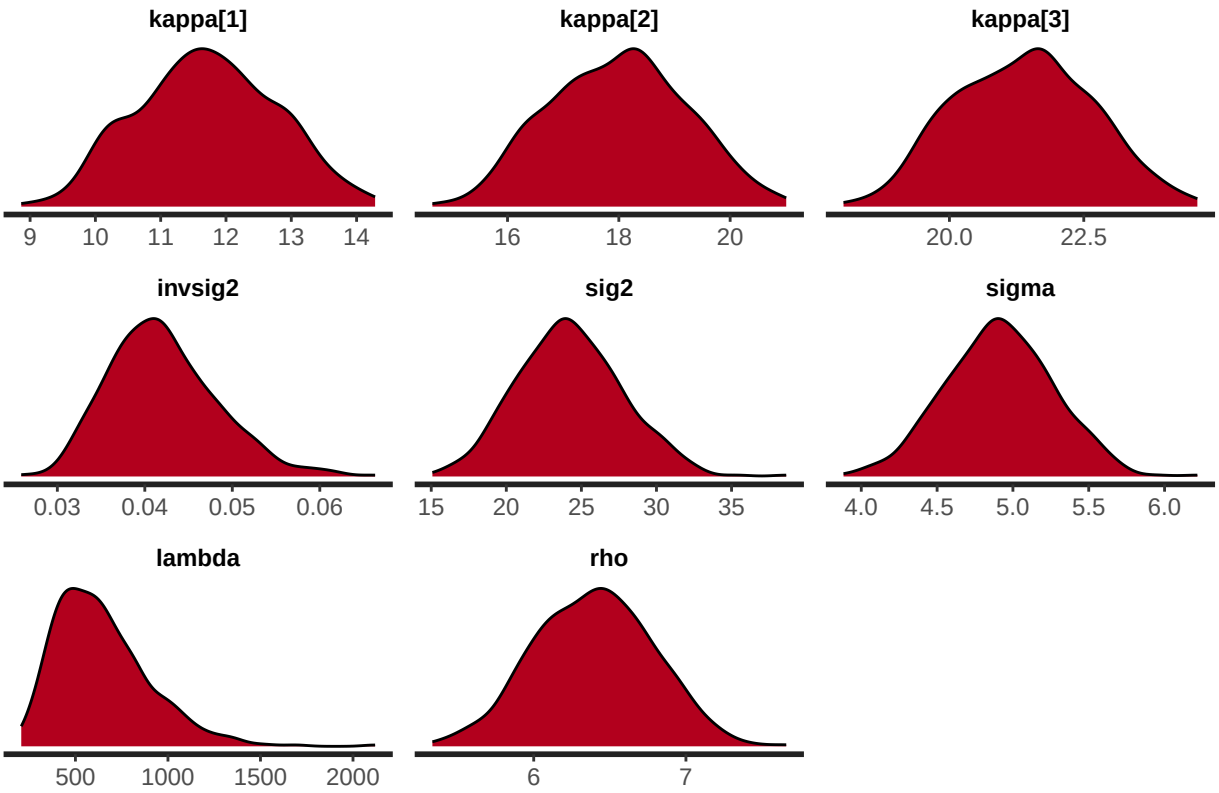
```
print(fit.lme.add.incr.reintrcpt, pars=param.lme.add)
```

```
## Inference for Stan model: jagam_10_aneur_ordinal_lme_add_incr_reintrcpt.
## 3 chains, each with iter=600; warmup=300; thin=2;
## post-warmup draws per chain=150, total post-warmup draws=450.
##
##            mean se_mean     sd   2.5%    25%    50%    75%   97.5% n_eff Rhat
## b1[1]      2.31    0.02   0.20   1.95   2.16   2.30   2.45    2.70   169 1.01
## b1[2]      0.13    0.00   0.07   0.01   0.08   0.13   0.17    0.26   278 1.00
## b1[3]      1.19    0.01   0.19   0.84   1.07   1.19   1.31    1.57   291 1.00
## kappa[1]  11.71    0.08   1.05   9.91  10.99  11.67  12.44   13.69   159 1.02
## kappa[2]  17.99    0.10   1.22  15.79  17.09  18.07  18.84   20.22   139 1.02
## kappa[3]  21.39    0.11   1.27  19.07  20.38  21.44  22.35   23.81   135 1.02
## lambda   643.51   14.19 257.41 278.00 459.12 601.30 780.49 1237.61   329 1.00
## rho        6.39    0.02   0.39   5.63   6.13   6.40   6.66    7.12   353 1.00
## invsig2    0.04    0.00   0.01   0.03   0.04   0.04   0.05    0.06   164 1.01
## sig2      24.28    0.28   3.49  18.00  21.83  24.14  26.62   31.25   161 1.01
## sigma      4.91    0.03   0.35   4.24   4.67   4.91   5.16    5.59   161 1.01
##
## Samples were drawn using NUTS(diag_e) at Sun Aug 13 16:09:43 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

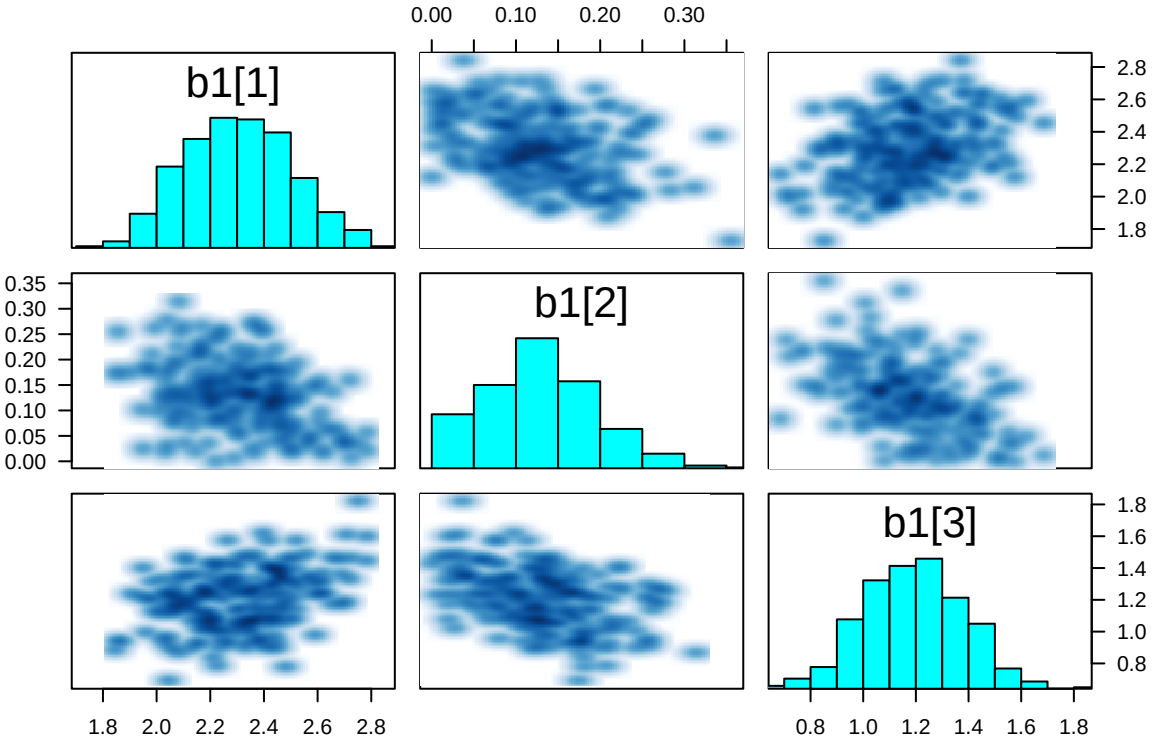
```
stan_trace(fit.lme.add.incr.reintrcpt, pars=param.lme.add)
stan_plot(fit.lme.add.incr.reintrcpt, pars=c("b1"), point_est = "mean", show_density = TRUE)
stan_plot(fit.lme.add.incr.reintrcpt, pars=c("kappa", "invsig2","sig2","sigma",  "lambda","rho"), point
stan_dens(fit.lme.add.incr.reintrcpt, pars=c("b1"))
stan_dens(fit.lme.add.incr.reintrcpt, pars=c("kappa", "invsig2","sig2","sigma", "lambda","rho"))
```


b1[1]
b1[2]
b1[3]
kappa[1]
kappa[2]
kappa[3]
lambda
rho
invsig2
sig2
sigma
chain
1
2
3
b1[1]
b1[2]
b1[3]
0
1
2

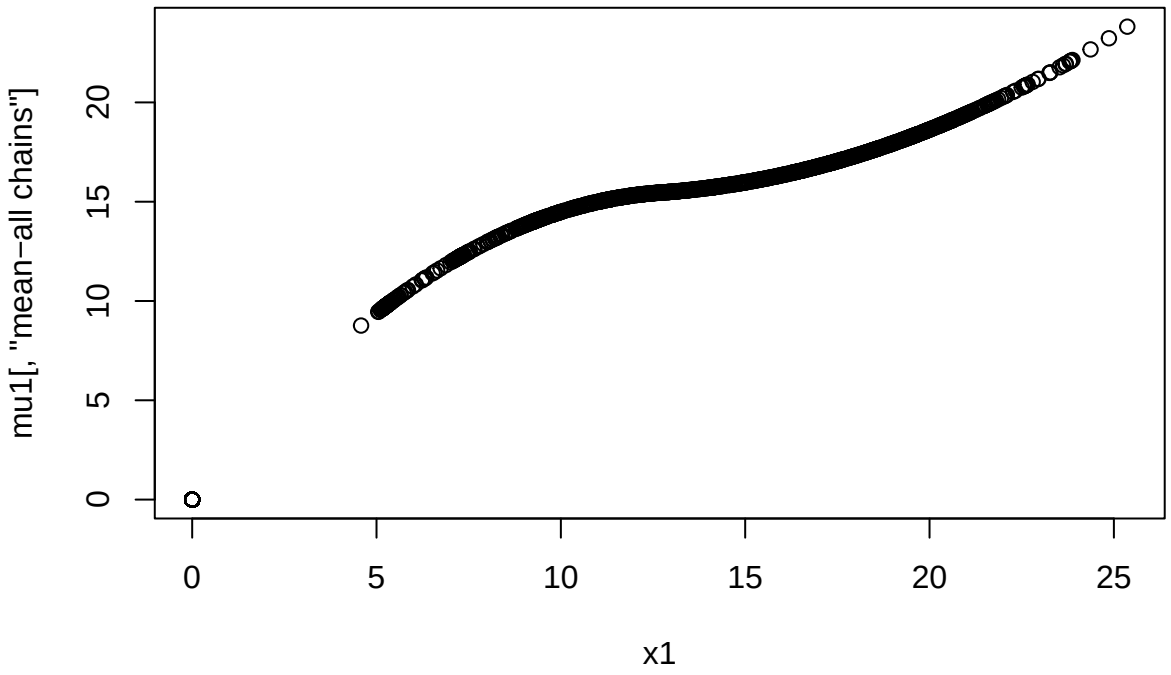

kappa[1]
kappa[2]
kappa[3]
invsig2
sig2
sigma
lambda
rho
0
500
1000
b1[1]
b1[2]
b1[3]
2.0
2.4
2.8
0.0
0.1
0.2
0.3
0.9
1.2
1.5
1.8



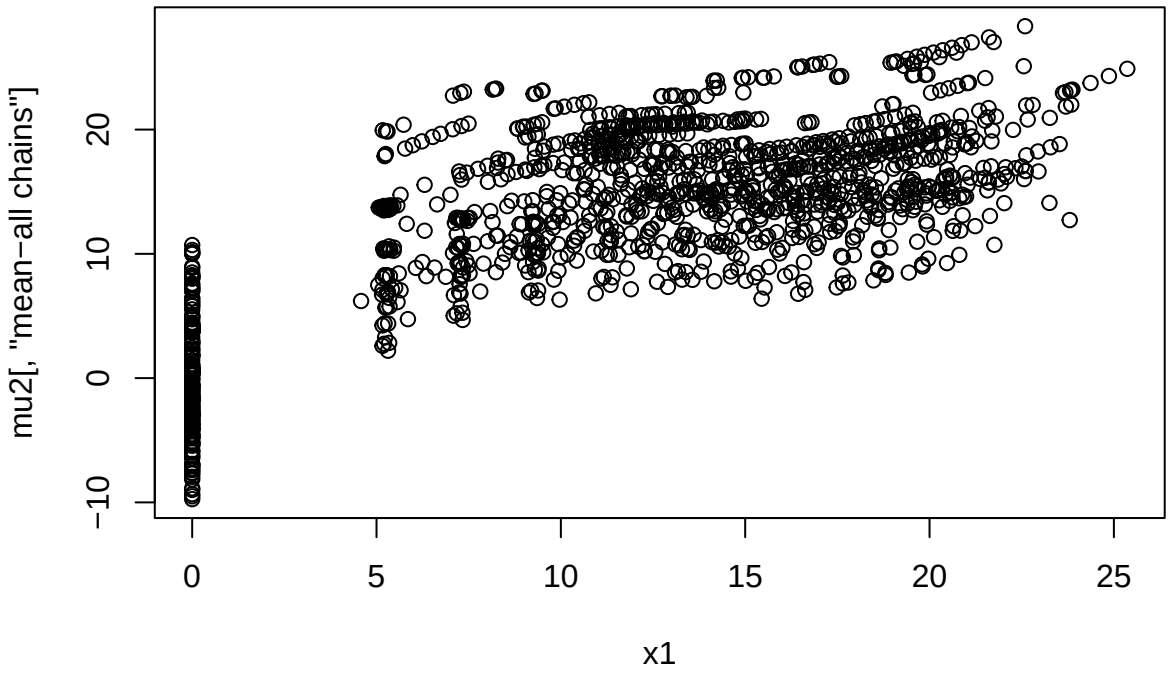

```
pairs(fit.lme.add.incr.reintrcpt, pars = c("b1"), las = 1)
```




```
mu1 = get_posterior_mean(fit.lme.add.incr.reintrcpt,"mu1")
plot(x1,mu1[,"mean-all chains"])
```




```
mu2 = get_posterior_mean(fit.lme.add.incr.reintrcpt,"mu2")
plot(x1,mu2[,"mean-all chains"])
```

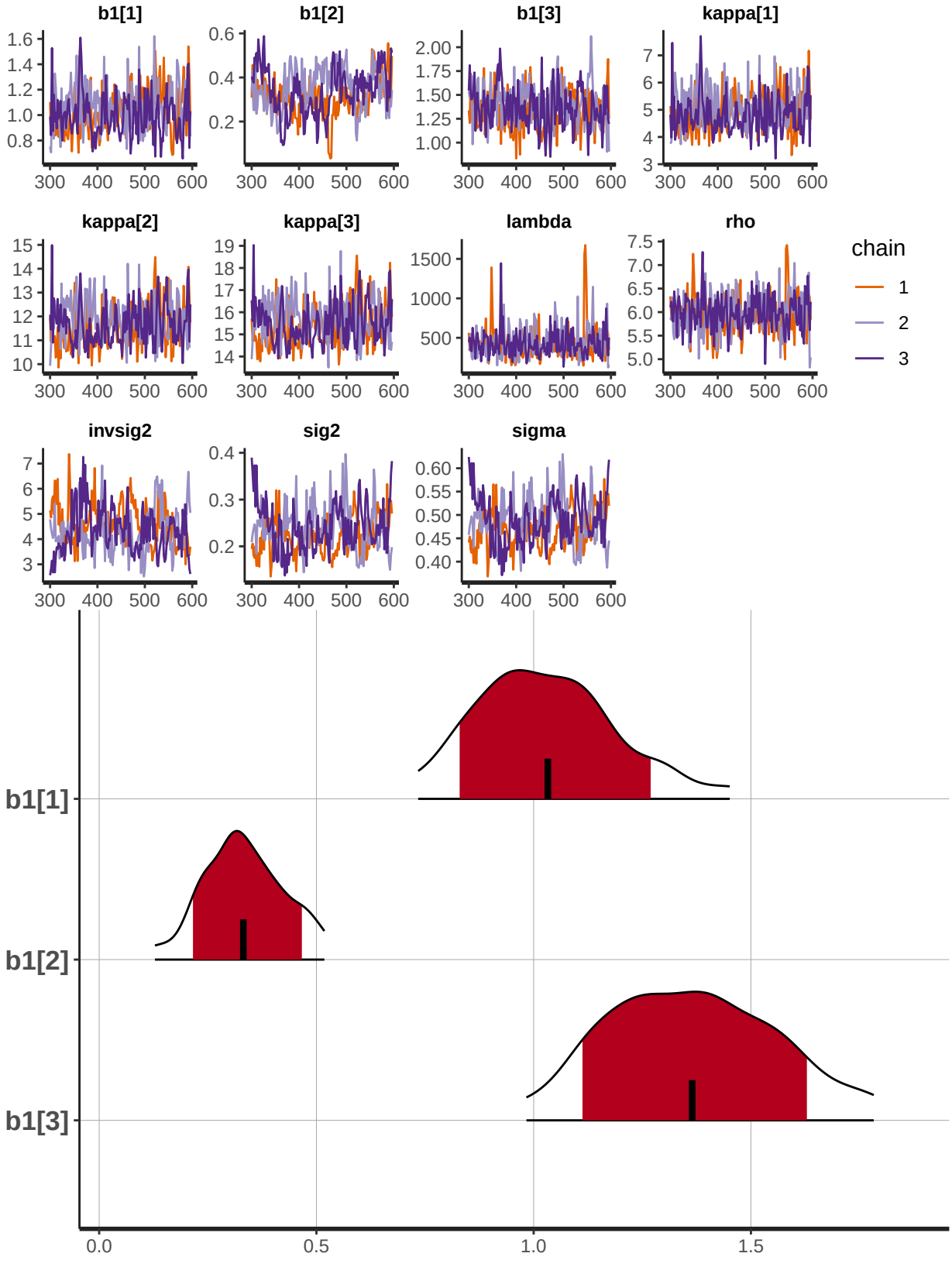


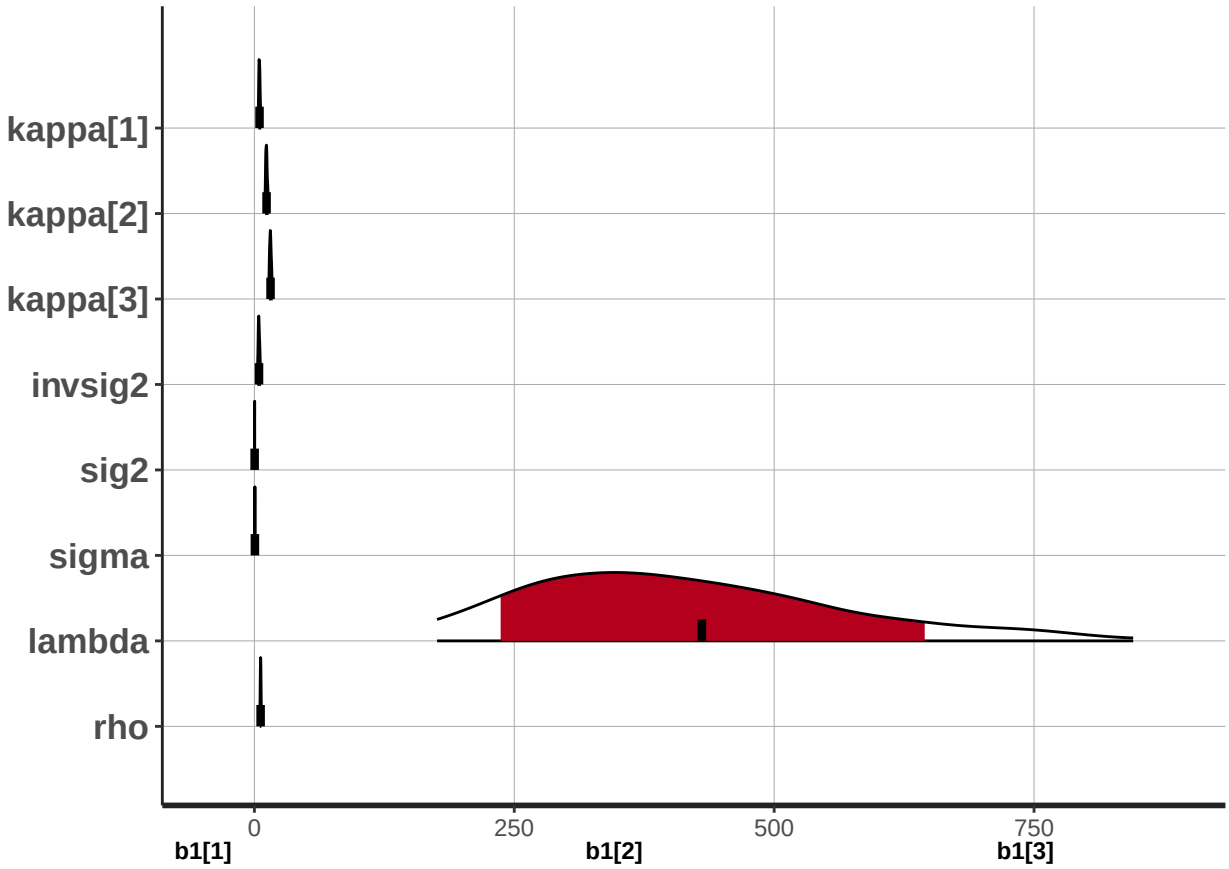

```
fit.lme.add.incr.reslope <- stan("jagam_10_aneur_ordinal_lme_add_incr_reslope.stan",
            data=datos.lme.add.incr,
            chains=3,warmup=300,iter=600,thin=2,cores=4,
            init= inits.lme.add.incr)
```

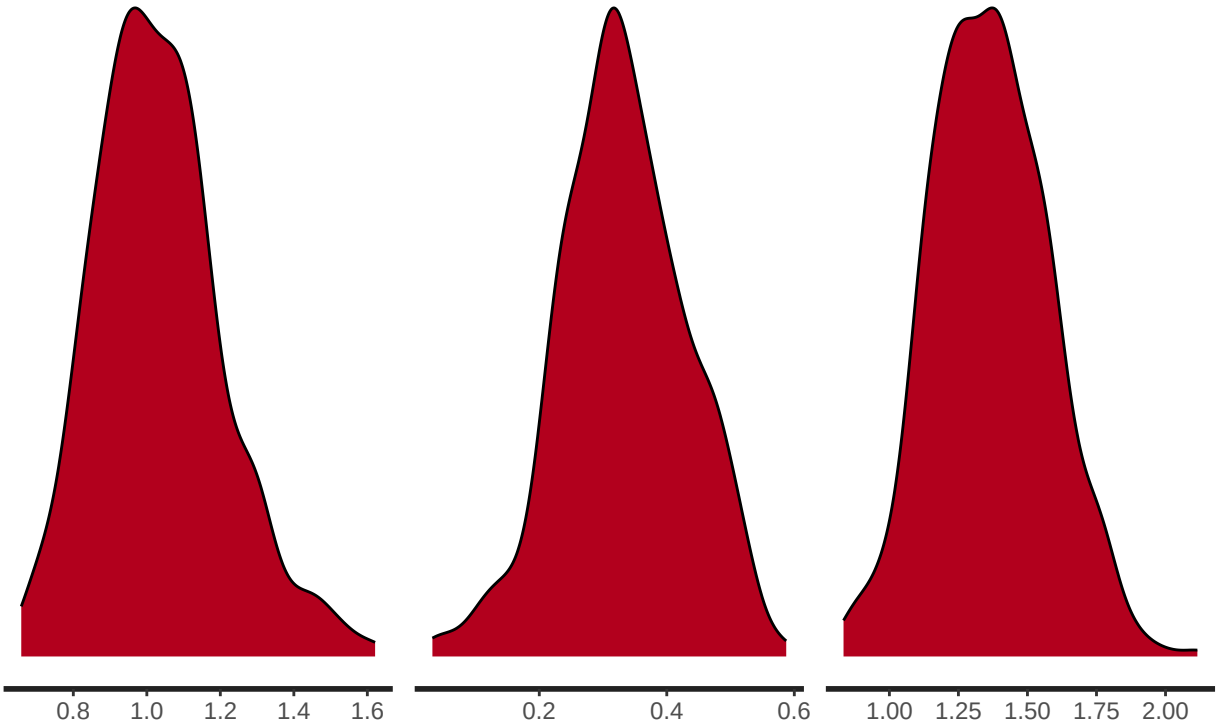
```
print(fit.lme.add.incr.reslope, pars=param.lme.add)
```

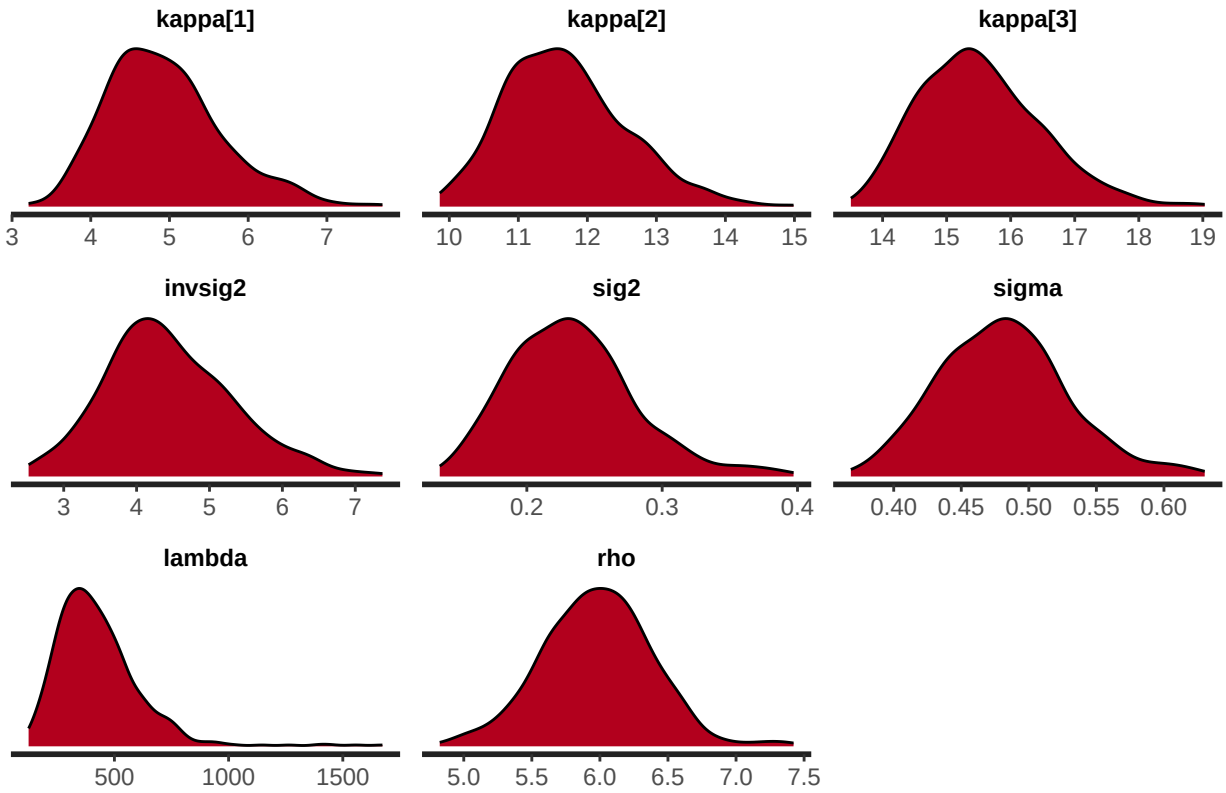
```
## Inference for Stan model: jagam_10_aneur_ordinal_lme_add_incr_reslope.
## 3 chains, each with iter=600; warmup=300; thin=2;
## post-warmup draws per chain=150, total post-warmup draws=450.
##
##            mean se_mean     sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
## b1[1]      1.04    0.01   0.17   0.73   0.91   1.02   1.14   1.45   158 1.04
## b1[2]      0.33    0.01   0.10   0.13   0.27   0.33   0.40   0.52    44 1.03
## b1[3]      1.36    0.02   0.21   0.97   1.21   1.36   1.51   1.78   174 1.01
## kappa[1]   4.94    0.05   0.73   3.78   4.41   4.86   5.35   6.57   227 1.02
## kappa[2]  11.70    0.07   0.87  10.27  11.04  11.60  12.24  13.64   143 1.03
## kappa[3]  15.54    0.09   0.94  13.97  14.82  15.42  16.10  17.62   117 1.04
## lambda   430.11   11.16 196.46 174.72 294.89 393.73 514.80 868.38   310 1.00
## rho        5.98    0.02   0.41   5.16   5.69   5.98   6.24   6.77   349 1.00
## invsig2    4.44    0.10   0.87   2.93   3.84   4.33   5.01   6.34    75 1.03
## sig2       0.23    0.01   0.05   0.16   0.20   0.23   0.26   0.34    73 1.03
## sigma      0.48    0.01   0.05   0.40   0.45   0.48   0.51   0.58    73 1.03
##
## Samples were drawn using NUTS(diag_e) at Sun Aug 13 16:12:45 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
stan_trace(fit.lme.add.incr.reslope, pars=param.lme.add)
stan_plot(fit.lme.add.incr.reslope, pars=c("b1"), point_est = "mean", show_density = TRUE)
stan_plot(fit.lme.add.incr.reslope, pars=c("kappa", "invsig2","sig2","sigma",  "lambda","rho"), point_e
stan_dens(fit.lme.add.incr.reslope, pars=c("b1"))
stan_dens(fit.lme.add.incr.reslope, pars=c("kappa", "invsig2","sig2","sigma", "lambda","rho"))
```

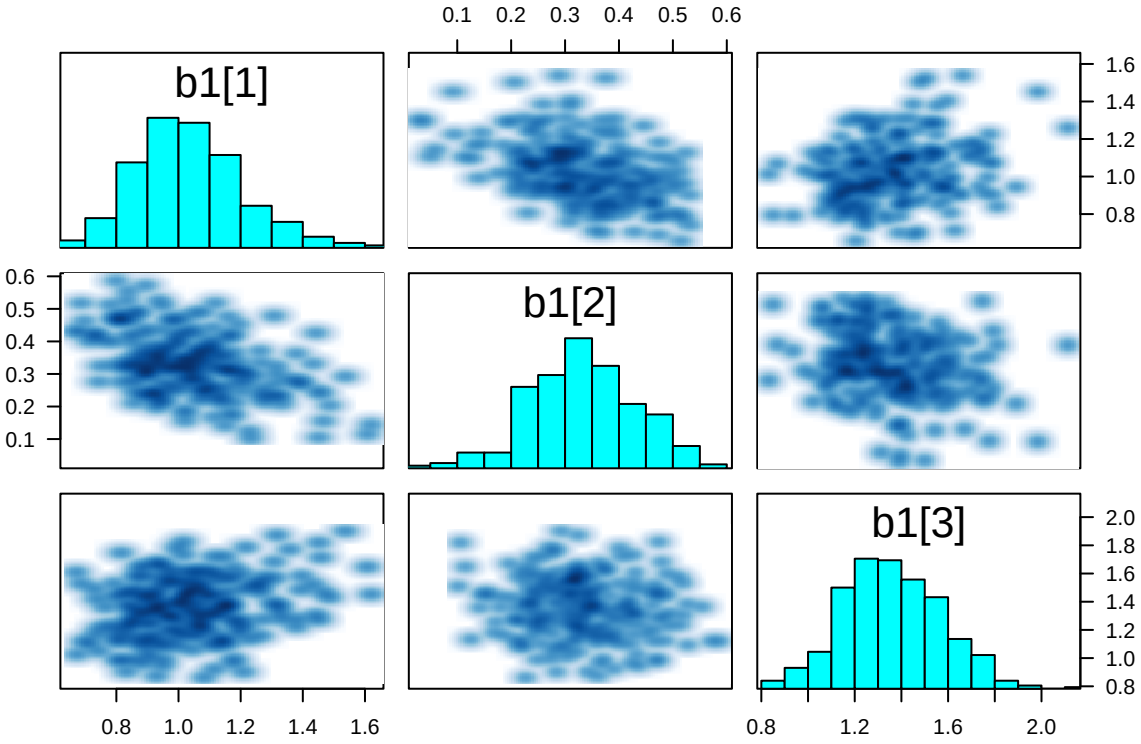

b1[1]
b1[2]
b1[3]
kappa[1]
kappa[2]
kappa[3]
lambda
rho
invsig2
sig2
sigma
chain
1
2
3
b1[1]
b1[2]
b1[3]
0.0
0.5
1.0
1.5


kappa[1]
kappa[2]
kappa[3]
invsig2
sig2
sigma
lambda
rho
0
250
500
750
b1[1]
b1[2]
b1[3]

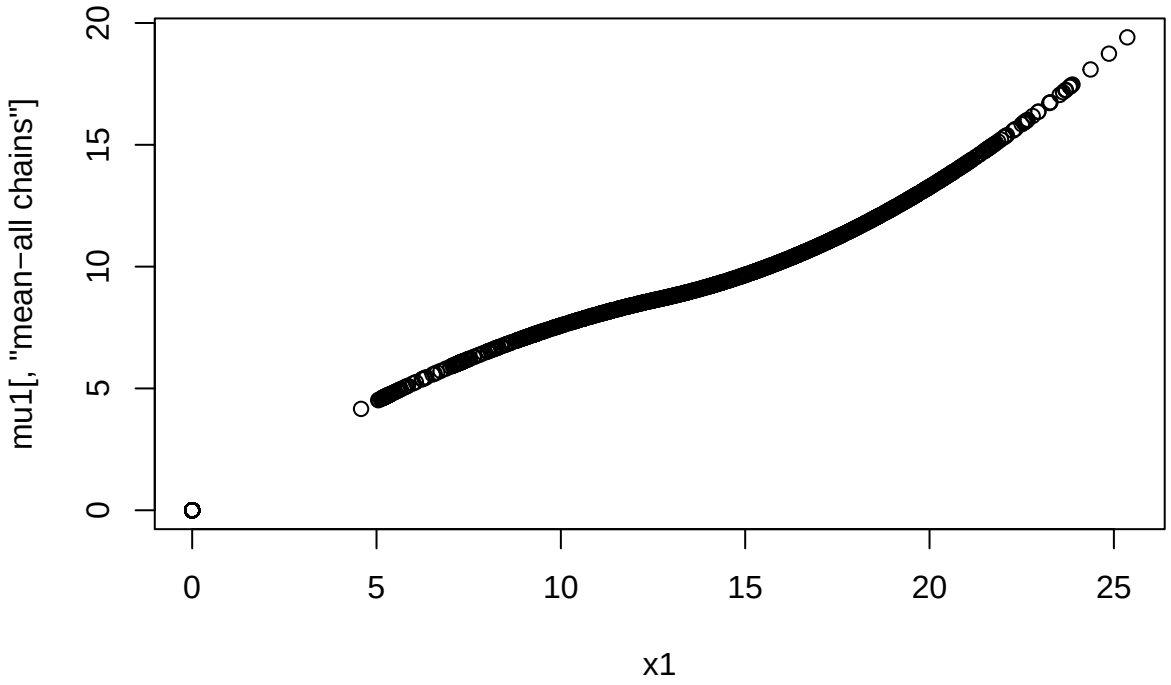

0.8
1.0
1.2
1.4
1.6
0.2
0.4
0.6
1.00
1.25
1.50
1.75
2.00



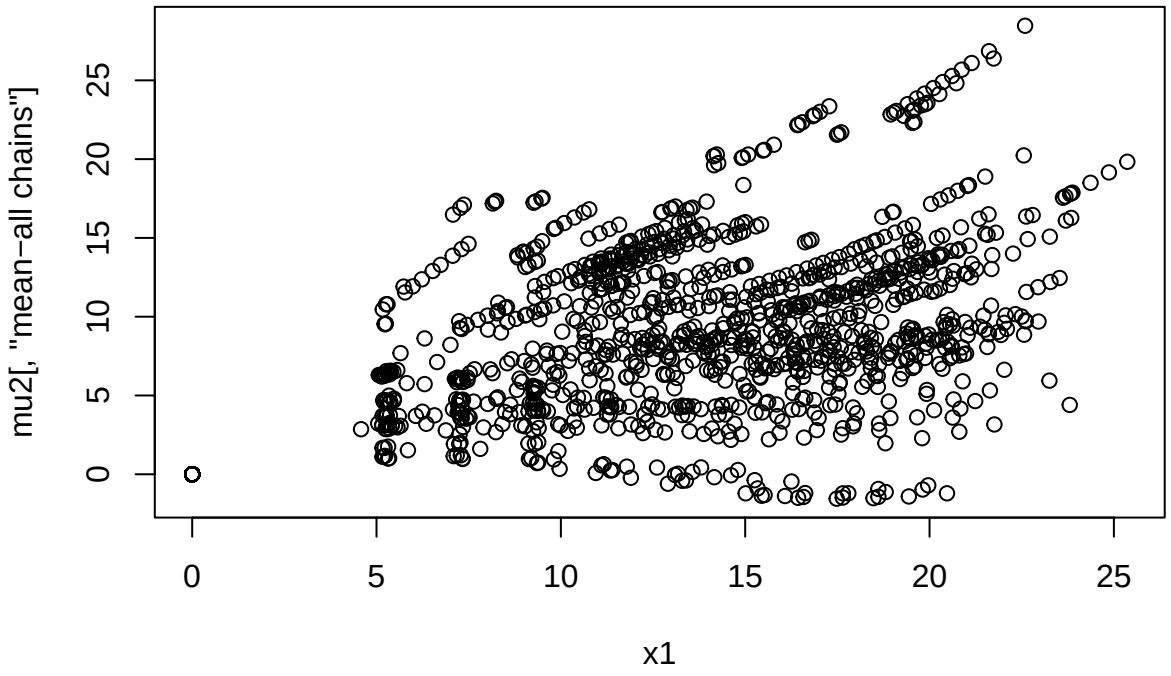

```
pairs(fit.lme.add.incr.reslope, pars = c("b1"), las = 1)
```




```
mu1 = get_posterior_mean(fit.lme.add.incr.reslope,"mu1")
plot(x1,mu1[,"mean-all chains"])
```




```
mu2 = get_posterior_mean(fit.lme.add.incr.reslope,"mu2")
plot(x1,mu2[,"mean-all chains"])
```



Comparar

```
### http://ritsokiguess.site/docs/2019/06/25/going-to-the-loo-using-stan-for-model-comparison/
library(loo)
```

```
## This is loo version 2.4.1

## - Online documentation and vignettes at mc-stan.org/loo

## - As of v2.0.0 loo defaults to 1 core but we recommend using as many as possible. Use the 'cores' ar

##
## Attaching package: 'loo'

## The following object is masked from 'package:rstan':
##
##     loo
```

```
loo1_sample = fit.lin.non
loo2_sample = fit.lme.non.reintrcpt
loo3_sample = fit.lme.non.reslope
loo4_sample = fit.lin.incr
loo5_sample = fit.lme.incr.reintrcpt
loo6_sample = fit.lme.incr.reslope
loo7_sample = fit.add.non
loo8_sample = fit.lme.add.non.reintrcpt
loo9_sample = fit.lme.add.non.reslope
loo10_sample = fit.add.incr
loo11_sample = fit.lme.add.incr.reintrcpt
loo12_sample = fit.lme.add.incr.reslope

### we have to extract those log-likelihood terms that we so carefully had Stan calculate for us:
log_lik_1 =extract_log_lik(loo1_sample, merge_chains = F)
log_lik_2 =extract_log_lik(loo2_sample, merge_chains = F)
log_lik_3 =extract_log_lik(loo3_sample, merge_chains = F)
log_lik_4 =extract_log_lik(loo4_sample, merge_chains = F)
log_lik_5 =extract_log_lik(loo5_sample, merge_chains = F)
log_lik_6 =extract_log_lik(loo6_sample, merge_chains = F)
log_lik_7 =extract_log_lik(loo7_sample, merge_chains = F)
log_lik_8 =extract_log_lik(loo8_sample, merge_chains = F)
log_lik_9 =extract_log_lik(loo9_sample, merge_chains = F)
log_lik_10 =extract_log_lik(loo10_sample, merge_chains = F)
log_lik_11 =extract_log_lik(loo11_sample, merge_chains = F)
log_lik_12 =extract_log_lik(loo12_sample, merge_chains = F)

r_eff_1 =relative_eff(log_lik_1)
r_eff_2 =relative_eff(log_lik_2)
r_eff_3 =relative_eff(log_lik_3)
r_eff_4 =relative_eff(log_lik_4)
r_eff_5 =relative_eff(log_lik_5)
r_eff_6 =relative_eff(log_lik_6)
r_eff_7 =relative_eff(log_lik_7)
r_eff_8 =relative_eff(log_lik_8)
r_eff_9 =relative_eff(log_lik_9)
r_eff_10 =relative_eff(log_lik_10)
r_eff_11 =relative_eff(log_lik_11)
r_eff_12 =relative_eff(log_lik_12)
```

```
###  look at the results for each model, first the one with mu estimated:
(loo_1 <- loo(log_lik_1, r_eff=r_eff_1))
```

```
##
## Computed from 450 by 1387 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo  -1497.3 26.1
## p_loo         3.8  0.1
## looic      2994.7 52.2
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

```
(loo_2 <- loo(log_lik_2, r_eff=r_eff_2))
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
##
## Computed from 450 by 1387 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -878.3 32.0
## p_loo       240.6 14.3
## looic      1756.7 64.0
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     1136  81.9%   37
##  (0.5, 0.7]   (ok)        152  11.0%   21
##    (0.7, 1]   (bad)        78   5.6%   5
##    (1, Inf)   (very bad)   21   1.5%   2
## See help('pareto-k-diagnostic') for details.
```

```
(loo_3 <- loo(log_lik_3, r_eff=r_eff_3))
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
##
## Computed from 450 by 1387 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -747.7 31.8
## p_loo       199.7 13.9
## looic      1495.4 63.6
## ------
## Monte Carlo SE of elpd_loo is NA.
```

```
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     1213  87.5%   45
##  (0.5, 0.7]   (ok)        114   8.2%   31
##    (0.7, 1]   (bad)        49   3.5%   6
##    (1, Inf)   (very bad)   11   0.8%   5
## See help('pareto-k-diagnostic') for details.
```

```
(loo_4 <- loo(log_lik_4, r_eff=r_eff_4))
```

```
##
## Computed from 450 by 1387 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo  -1497.4 26.1
## p_loo         3.9  0.1
## looic      2994.9 52.1
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

```
(loo_5 <- loo(log_lik_5, r_eff=r_eff_5))
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
##
## Computed from 450 by 1387 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -872.8 31.6
## p_loo       233.1 13.6
## looic      1745.5 63.2
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     1145  82.6%   70
##  (0.5, 0.7]   (ok)        147  10.6%   26
##    (0.7, 1]   (bad)        70   5.0%   10
##    (1, Inf)   (very bad)   25   1.8%   3
## See help('pareto-k-diagnostic') for details.
```

```
(loo_6 <- loo(log_lik_6, r_eff=r_eff_6))
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
##
```

```
## Computed from 450 by 1387 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -746.9 31.3
## p_loo       197.8 13.6
## looic      1493.8 62.6
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     1181  85.1%   58
##  (0.5, 0.7]   (ok)        142  10.2%   42
##    (0.7, 1]   (bad)        55   4.0%   7
##    (1, Inf)   (very bad)    9   0.6%   4
## See help('pareto-k-diagnostic') for details.
```

```
(loo_7 <- loo(log_lik_7, r_eff=r_eff_7))
```

```
##
## Computed from 450 by 1387 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo  -1421.4 25.3
## p_loo         5.7  0.2
## looic      2842.8 50.6
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

```
(loo_8 <- loo(log_lik_8, r_eff=r_eff_8))
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
##
## Computed from 450 by 1387 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -788.3 28.3
## p_loo       195.7 10.5
## looic      1576.6 56.7
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     1064  76.7%   36
##  (0.5, 0.7]   (ok)        203  14.6%   35
##    (0.7, 1]   (bad)       100   7.2%   17
##    (1, Inf)   (very bad)   20   1.4%   3
## See help('pareto-k-diagnostic') for details.
```

```
(loo_9 <- loo(log_lik_9, r_eff=r_eff_9))
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
##
## Computed from 450 by 1387 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -745.6 31.1
## p_loo       197.0 13.7
## looic      1491.3 62.1
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     1185  85.4%   65
##  (0.5, 0.7]   (ok)        135   9.7%   42
##    (0.7, 1]   (bad)        58   4.2%   8
##    (1, Inf)   (very bad)    9   0.6%   5
## See help('pareto-k-diagnostic') for details.
```

```
(loo_10 <- loo(log_lik_10, r_eff=r_eff_10))
```

```
##
## Computed from 450 by 1387 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo  -1424.7 25.4
## p_loo         4.7  0.2
## looic      2849.4 50.8
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

```
(loo_11 <- loo(log_lik_11, r_eff=r_eff_11))
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
##
## Computed from 450 by 1387 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -789.6 28.8
## p_loo       199.8 10.6
## looic      1579.2 57.7
## ------
## Monte Carlo SE of elpd_loo is NA.
##
```

```
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     1065  76.8%   51
##  (0.5, 0.7]   (ok)        186  13.4%   26
##    (0.7, 1]   (bad)       121   8.7%   8
##    (1, Inf)   (very bad)   15   1.1%   3
## See help('pareto-k-diagnostic') for details.
```

```
(loo_12 <- loo(log_lik_12, r_eff=r_eff_12))
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
##
## Computed from 450 by 1387 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -746.0 30.9
## p_loo       196.2 13.5
## looic      1491.9 61.7
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     1201  86.6%   62
##  (0.5, 0.7]   (ok)        129   9.3%   18
##    (0.7, 1]   (bad)        50   3.6%   8
##    (1, Inf)   (very bad)    7   0.5%   6
## See help('pareto-k-diagnostic') for details.
```

```
#compare(loo_1, loo_2)

### The second model fits better than the first one, since its looic is smaller.
```