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
```

```r
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
```

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

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

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

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

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

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

```r
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
```

```r
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]
    }
}
```

```r
(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
```

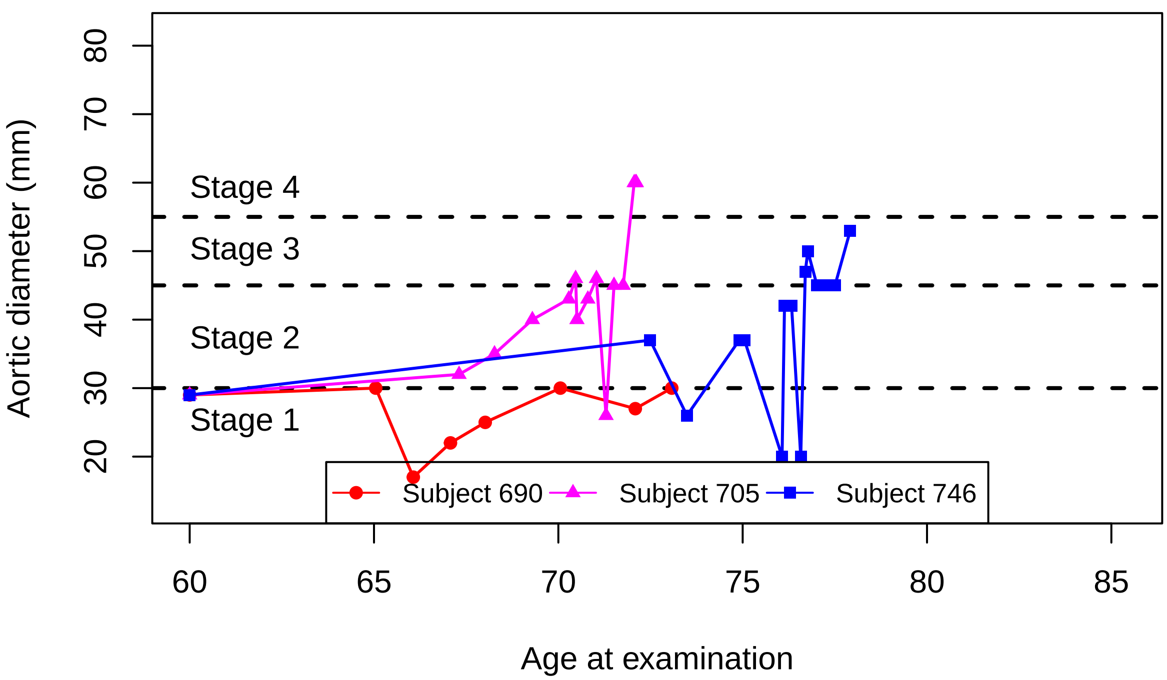
```
(Ki[11:18])
```

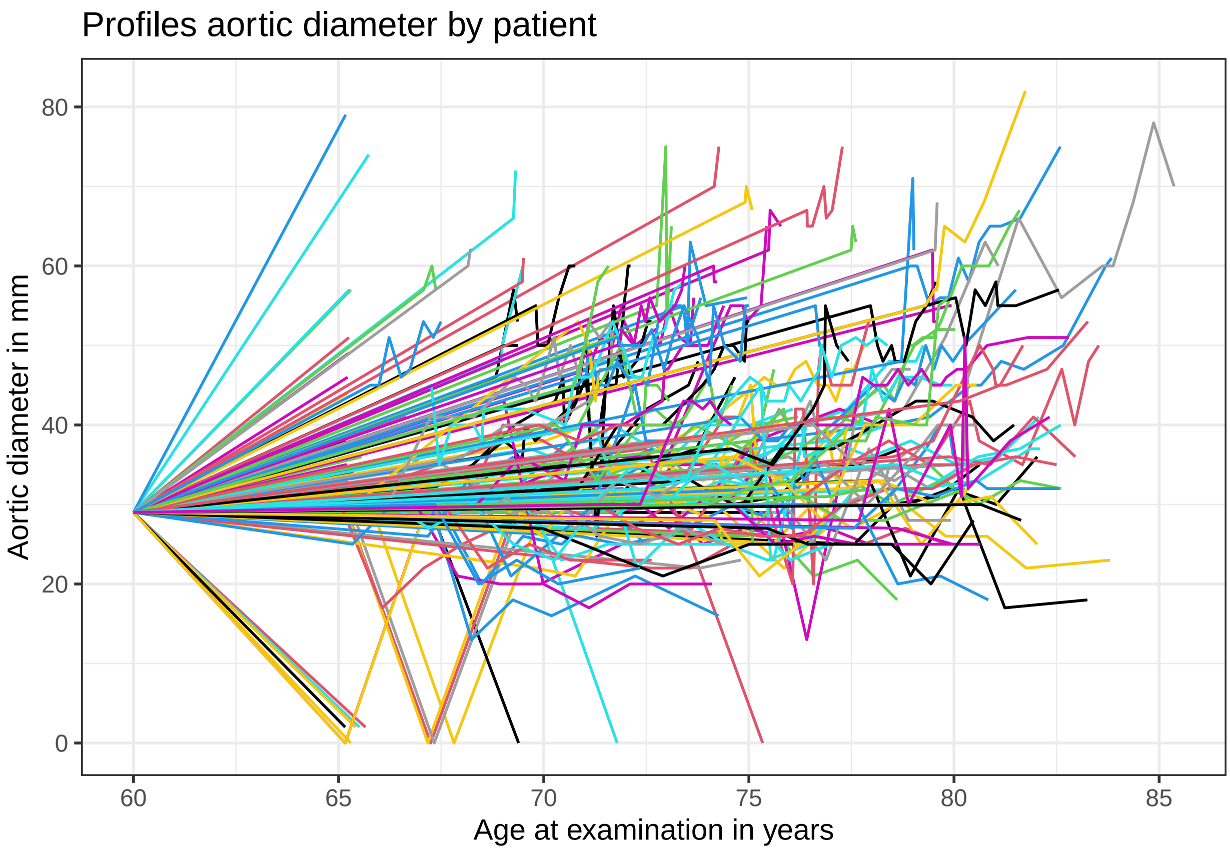
```
##
## 11 12 13 14 15 16 17 18
##  7  7  7  2  7  7  4  3
```

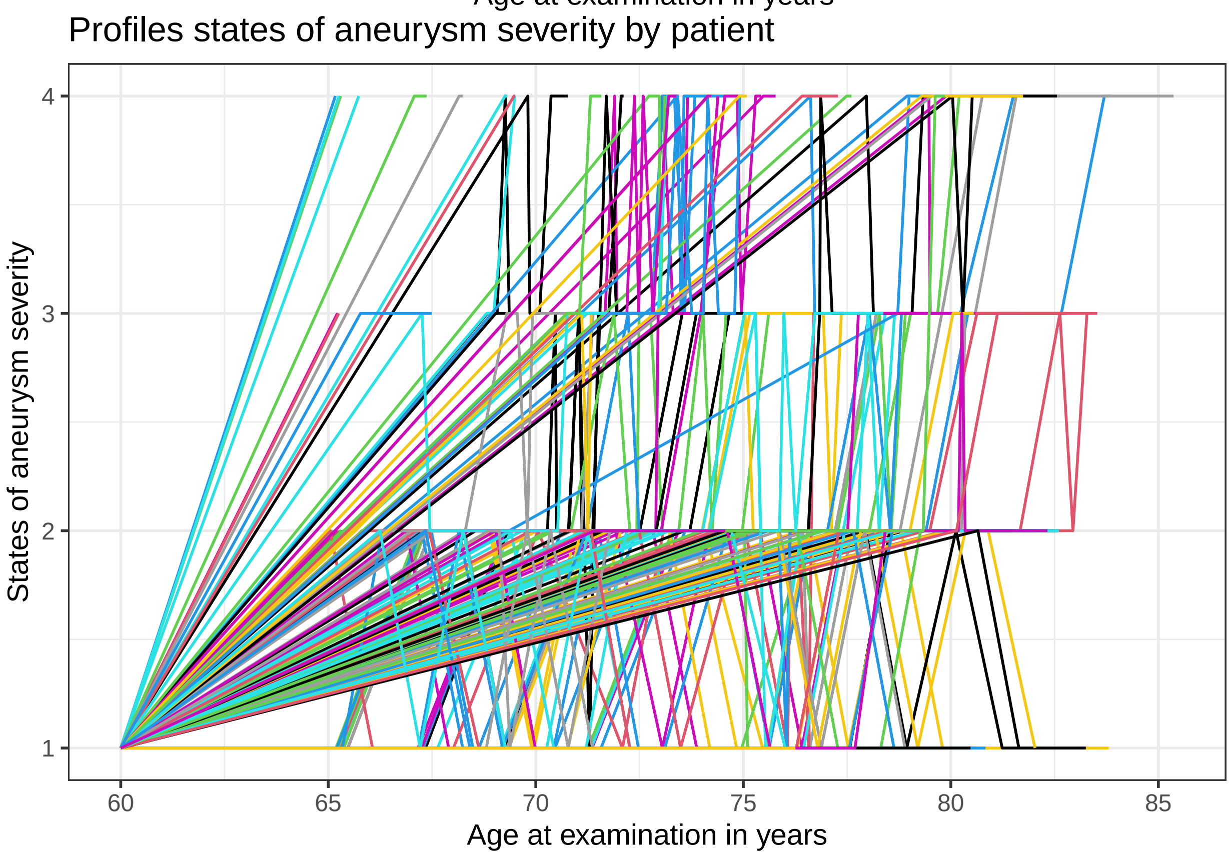
```
### Considering only data having more than one screen (state>1)
idx2 = c()
for(i in 1:N){
  if( sum(Y_state[i,1:Ki[i]])>Ki[i]){
    idx2 = c(idx2,i)
  }
}
Y2_diam = Y_diam[idx2,]
Y2_state = Y_state[idx2,]
X2_age = X_age[idx2,]
N2 = length(idx2)
Ki2 = Ki[idx2]

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

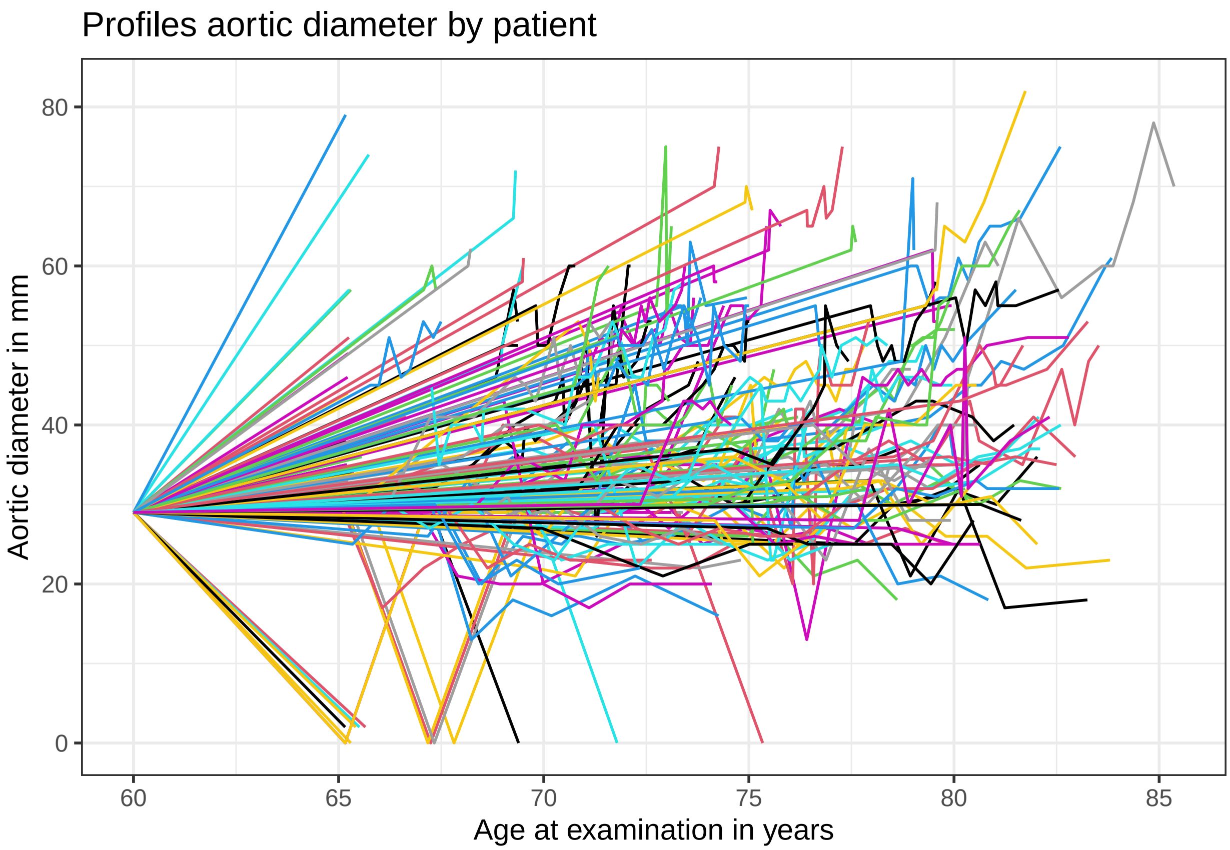
```
idx3 = c()
for(i in 1:N){
  if( min(Y_diam[i,1:Ki[i]])!=max(Y_diam[i,1:Ki[i]])){
    idx3 = c(idx3,i)
  }
}
Y3_diam = Y_diam[idx3,]
Y3_state = Y_state[idx3,]
X3_age = X_age[idx3,]
N3 = length(idx3)
Ki3 = Ki[idx3]

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

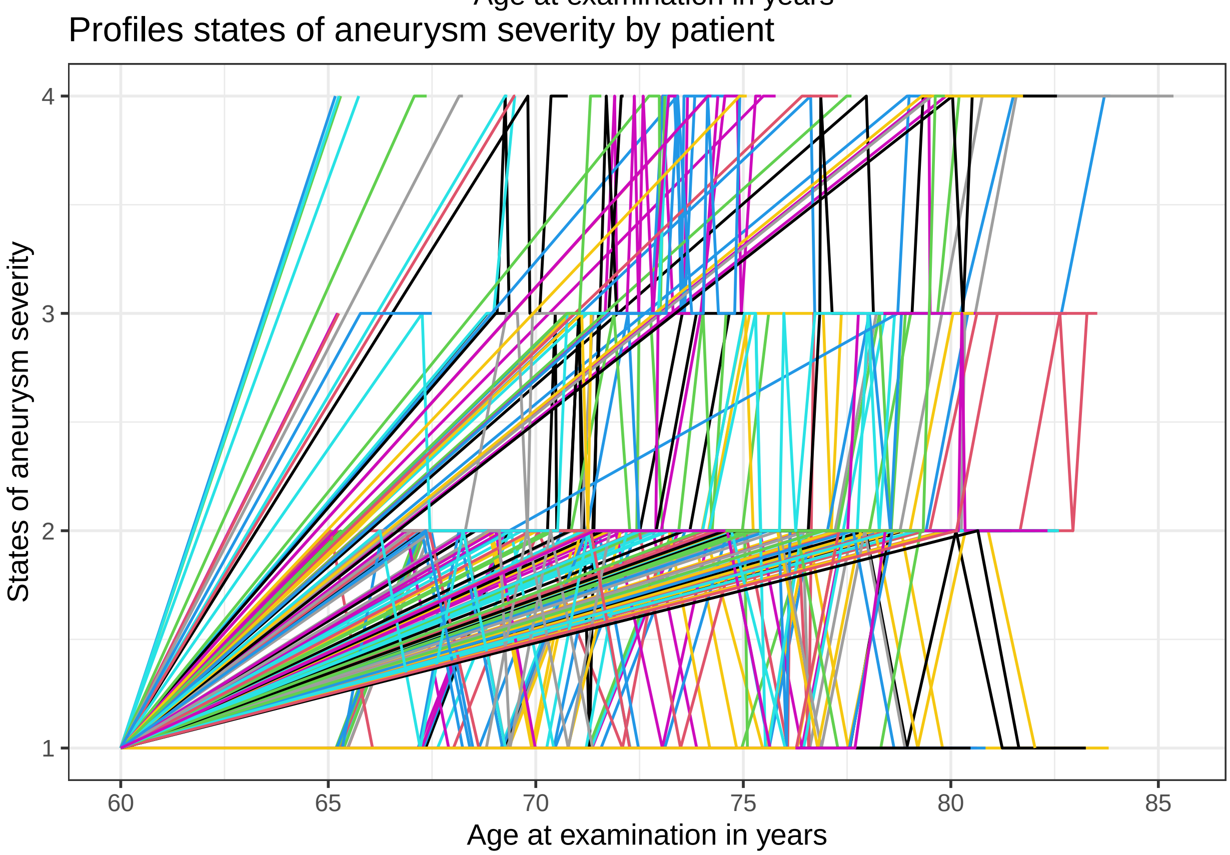
```
##
##  67  80 119
##   6   6   6
```




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))
```

```
## [1] 1387
```

```
(N = n_distinct(id))
```

```
## [1] 229
```

```
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:

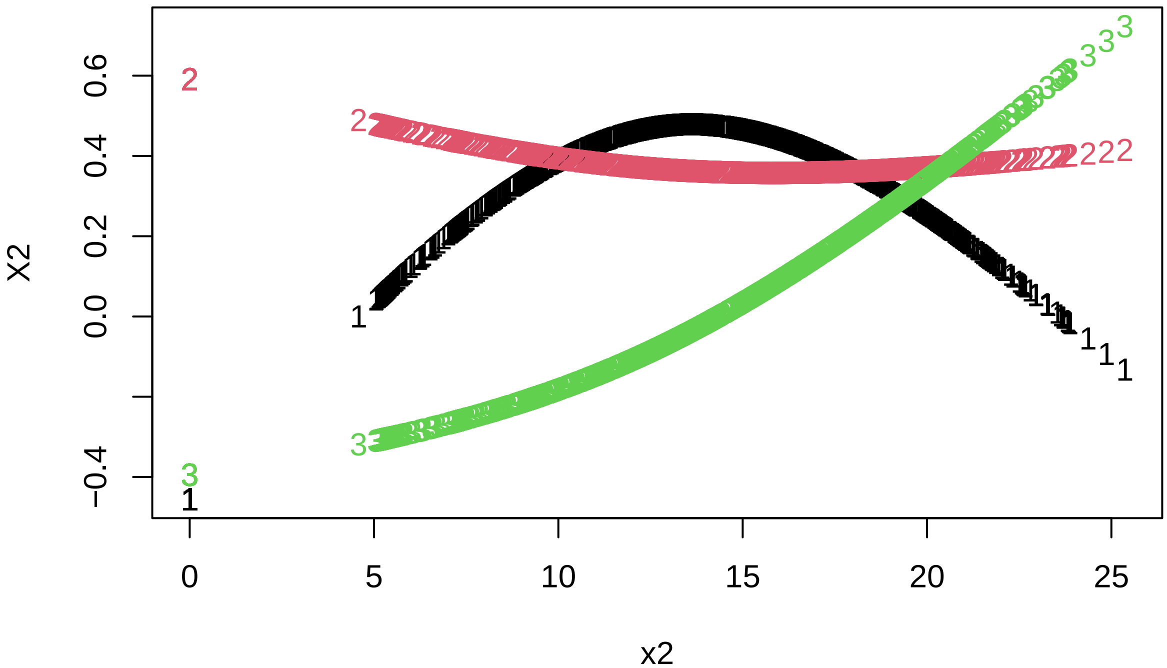
$$
\begin{aligned}
f(x) &= f_1(x_1) \\
&= \sum_{j=1}^{h_1} \beta_{1j} I_{1j}(x)
\end{aligned}
$$

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.

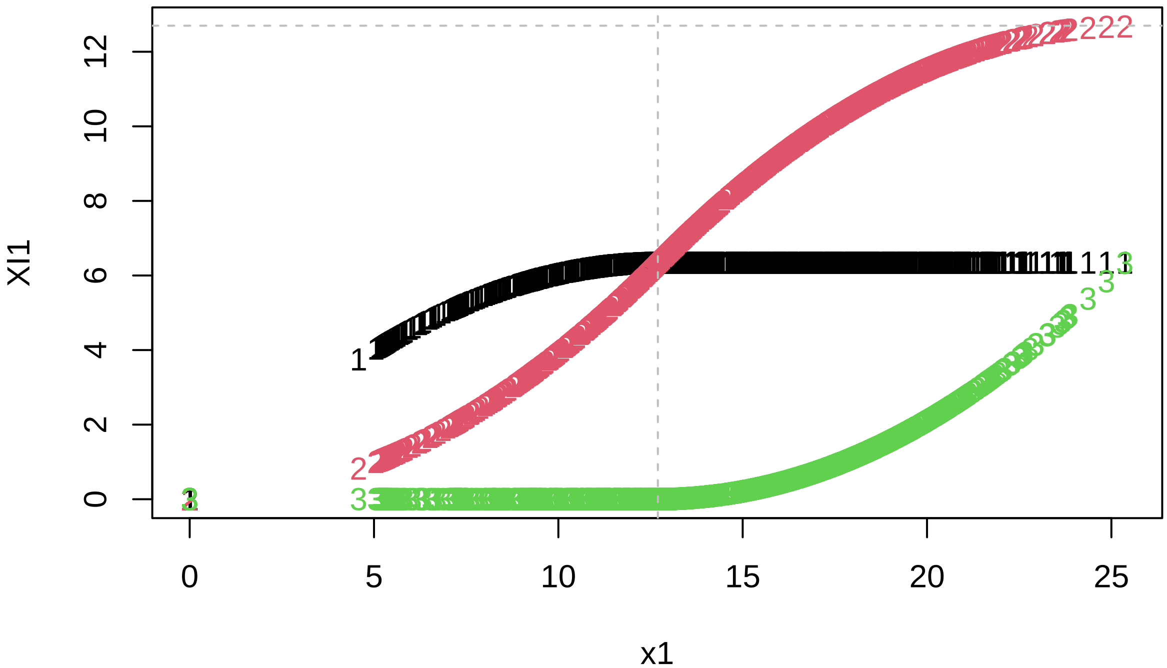
```
# Generate a basis matrix for Natural Cubic Splines
X2 <- ns(x = x2, knots = knots2, intercept = TRUE)
###X2 = (X2-mean(X2))/sd(X2)
matplot(x2, X2)
```



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

$$
\begin{aligned}
f_1(x_1) & \quad \sum_{j=1}^{h_1} \beta_{1j} I_{1j}(x_1) \\
I_{1j}(x_1) & = \int_{x_0}^{x_1} B_{1j}(u) d_u
\end{aligned}
$$

```
### 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:

5.2 LME: Lineal fit without constraints:

```
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_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
##  >> 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 model72a3494fd723_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", "model72a3494fd723_24b938afe5b29efe8d963b6430c621be");
##   32 :     reader.add_event(57, 55, "end", "model72a3494fd723_24b938afe5b29efe8d963b6430c621be");
##   33 :     return reader;
##   34 : }
##   35 :
##   36 : class model72a3494fd723_24b938afe5b29efe8d963b6430c621be
##   37 :   : public stan::model::model_base_crtp<model72a3494fd723_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 :     model72a3494fd723_24b938afe5b29efe8d963b6430c621be(rstan::io::rlist_ref_var_context& conte
##   47 :         std::ostream* pstream__ = 0)
##   48 :         : model_base_crtp(0) {
##   49 :         ctor_body(context__, 0, pstream__);
##   50 :     }
##   51 :
##   52 :     model72a3494fd723_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__ = "model72a3494fd723_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");
```

```
##             pos__ = 0;
##             N = vals_i__[pos__++];
##             check_greater_or_equal(function__, "N", N, 0);
##
##             current_statement_begin__ = 9;
##             context__.validate_dims("data initialization", "n", "int", context__.to_vec());
##             n = int(0);
##             vals_i__ = context__.vals_i("n");
##             pos__ = 0;
##             n = vals_i__[pos__++];
##             check_greater_or_equal(function__, "n", n, 0);
##
##             current_statement_begin__ = 10;
##             validate_non_negative_index("Ni", "(N + 1)", (N + 1));
##             context__.validate_dims("data initialization", "Ni", "int", context__.to_vec((N +
##             Ni = std::vector<int>((N + 1), int(0));
##             vals_i__ = context__.vals_i("Ni");
##             pos__ = 0;
##             size_t Ni_k_0_max__ = (N + 1);
##             for (size_t k_0__ = 0; k_0__ < Ni_k_0_max__; ++k_0__) {
##                 Ni[k_0__] = vals_i__[pos__++];
##             }
##             size_t Ni_i_0_max__ = (N + 1);
##             for (size_t i_0__ = 0; i_0__ < Ni_i_0_max__; ++i_0__) {
##                 check_greater_or_equal(function__, "Ni[i_0__]", Ni[i_0__], 0);
##             }
##
##             current_statement_begin__ = 11;
##             validate_non_negative_index("y", "n", n);
##             context__.validate_dims("data initialization", "y", "int", context__.to_vec(n));
##             y = std::vector<int>(n, int(0));
##             vals_i__ = context__.vals_i("y");
##             pos__ = 0;
##             size_t y_k_0_max__ = n;
##             for (size_t k_0__ = 0; k_0__ < y_k_0_max__; ++k_0__) {
##                 y[k_0__] = vals_i__[pos__++];
##             }
##             size_t y_i_0_max__ = n;
##             for (size_t i_0__ = 0; i_0__ < y_i_0_max__; ++i_0__) {
##                 check_greater_or_equal(function__, "y[i_0__]", y[i_0__], 1);
##                 check_less_or_equal(function__, "y[i_0__]", y[i_0__], 4);
##             }
##
##             current_statement_begin__ = 12;
##             validate_non_negative_index("x1", "n", n);
##             context__.validate_dims("data initialization", "x1", "double", context__.to_vec(n
##             x1 = std::vector<double>(n, double(0));
##             vals_r__ = context__.vals_r("x1");
##             pos__ = 0;
##             size_t x1_k_0_max__ = n;
##             for (size_t k_0__ = 0; k_0__ < x1_k_0_max__; ++k_0__) {
##                 x1[k_0__] = vals_r__[pos__++];
##             }
##
```

```
## 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 :     ~model72a3494fd723_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__;
```

```
##
##             current_statement_begin__ = 17;
##             if (!(context__.contains_r("kappa")))
##                 stan::lang::rethrow_located(std::runtime_error(std::string("Variable kappa missing
##             vals_r__ = context__.vals_r("kappa");
##             pos__ = 0U;
##             validate_non_negative_index("kappa", "3", 3);
##             context__.validate_dims("parameter initialization", "kappa", "vector_d", context__.to_
##             Eigen::Matrix<double, Eigen::Dynamic, 1> kappa(3);
##             size_t kappa_j_1_max__ = 3;
##             for (size_t j_1__ = 0; j_1__ < kappa_j_1_max__; ++j_1__) {
##                 kappa(j_1__) = vals_r__[pos__++];
##             }
##             try {
##                 writer__.ordered_unconstrain(kappa);
##             } catch (const std::exception& e) {
##                 stan::lang::rethrow_located(std::runtime_error(std::string("Error transforming va
##             }
##
##             current_statement_begin__ = 18;
##             if (!(context__.contains_r("b1")))
##                 stan::lang::rethrow_located(std::runtime_error(std::string("Variable b1 missing")
##             vals_r__ = context__.vals_r("b1");
##             pos__ = 0U;
##             context__.validate_dims("parameter initialization", "b1", "double", context__.to_vec(
##             double b1(0);
##             b1 = vals_r__[pos__++];
##             try {
##                 writer__.scalar_unconstrain(b1);
##             } catch (const std::exception& e) {
##                 stan::lang::rethrow_located(std::runtime_error(std::string("Error transforming va
##             }
##
##             current_statement_begin__ = 19;
##             if (!(context__.contains_r("bre1")))
##                 stan::lang::rethrow_located(std::runtime_error(std::string("Variable bre1 missing
##             vals_r__ = context__.vals_r("bre1");
##             pos__ = 0U;
##             validate_non_negative_index("bre1", "N", N);
##             context__.validate_dims("parameter initialization", "bre1", "double", context__.to_ve
##             std::vector<double> bre1(N, double(0));
##             size_t bre1_k_0_max__ = N;
##             for (size_t k_0__ = 0; k_0__ < bre1_k_0_max__; ++k_0__) {
##                 bre1[k_0__] = vals_r__[pos__++];
##             }
##             size_t bre1_i_0_max__ = N;
##             for (size_t i_0__ = 0; i_0__ < bre1_i_0_max__; ++i_0__) {
##                 try {
##                     writer__.scalar_unconstrain(bre1[i_0__]);
##                 } catch (const std::exception& e) {
##                     stan::lang::rethrow_located(std::runtime_error(std::string("Error transforming
##                 }
##             }
##
```

```
## 247 :         current_statement_begin__ = 20;
## 248 :         if (!(context__.contains_r("invsig2")))
## 249 :             stan::lang::rethrow_located(std::runtime_error(std::string("Variable invsig2 miss
## 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 :
```

```
##             current_statement_begin__ = 18;
##             local_scalar_t__ b1;
##             (void) b1;  // dummy to suppress unused var warning
##             if (jacobian__)
##                 b1 = in__.scalar_constrain(lp__);
##             else
##                 b1 = in__.scalar_constrain();
##
##             current_statement_begin__ = 19;
##             std::vector<local_scalar_t__> bre1;
##             size_t bre1_d_0_max__ = N;
##             bre1.reserve(bre1_d_0_max__);
##             for (size_t d_0__ = 0; d_0__ < bre1_d_0_max__; ++d_0__) {
##                 if (jacobian__)
##                     bre1.push_back(in__.scalar_constrain(lp__));
##                 else
##                     bre1.push_back(in__.scalar_constrain());
##             }
##
##             current_statement_begin__ = 20;
##             local_scalar_t__ invsig2;
##             (void) invsig2;  // dummy to suppress unused var warning
##             if (jacobian__)
##                 invsig2 = in__.scalar_lb_constrain(0, lp__);
##             else
##                 invsig2 = in__.scalar_lb_constrain(0);
##
##             // transformed parameters
##             current_statement_begin__ = 24;
##             validate_non_negative_index("mu", "n", n);
##             Eigen::Matrix<local_scalar_t__, Eigen::Dynamic, 1> mu(n);
##             stan::math::initialize(mu, DUMMY_VAR__);
##             stan::math::fill(mu, DUMMY_VAR__);
##
##             current_statement_begin__ = 25;
##             local_scalar_t__ sig2;
##             (void) sig2;  // dummy to suppress unused var warning
##             stan::math::initialize(sig2, DUMMY_VAR__);
##             stan::math::fill(sig2, DUMMY_VAR__);
##
##             current_statement_begin__ = 26;
##             local_scalar_t__ sigma;
##             (void) sigma;  // dummy to suppress unused var warning
##             stan::math::initialize(sigma, DUMMY_VAR__);
##             stan::math::fill(sigma, DUMMY_VAR__);
##
##             // transformed parameters block statements
##             current_statement_begin__ = 27;
##             for (int i = 1; i <= N; ++i) {
##
##                 current_statement_begin__ = 28;
##                 for (int t = get_base1(Ni, i, "Ni", 1); t <= (get_base1(Ni, (i + 1), "Ni", 1)
##
##                     current_statement_begin__ = 29;
```

```
##                          stan::model::assign(mu,
##                                      stan::model::cons_list(stan::model::index_uni(t), stan::model
##                                      ((get_base1(x1, t, "x1", 1) * b1) + (get_base1(x1, t, "x1", 1
##                                      "assigning variable mu");
##                      }
##                  }
##                  current_statement_begin__ = 32;
##                  stan::math::assign(sig2, (1 / invsig2));
##                  current_statement_begin__ = 33;
##                  stan::math::assign(sigma, pow(sig2, 0.5));
##
##                  // validate transformed parameters
##                  const char* function__ = "validate transformed params";
##                  (void) function__;  // dummy to suppress unused var warning
##
##                  current_statement_begin__ = 24;
##                  size_t mu_j_1_max__ = n;
##                  for (size_t j_1__ = 0; j_1__ < mu_j_1_max__; ++j_1__) {
##                      if (stan::math::is_uninitialized(mu(j_1__))) {
##                          std::stringstream msg__;
##                          msg__ << "Undefined transformed parameter: mu" << "(" << j_1__ << ")";
##                          stan::lang::rethrow_located(std::runtime_error(std::string("Error initiali
##                      }
##                  }
##                  current_statement_begin__ = 25;
##                  if (stan::math::is_uninitialized(sig2)) {
##                      std::stringstream msg__;
##                      msg__ << "Undefined transformed parameter: sig2";
##                      stan::lang::rethrow_located(std::runtime_error(std::string("Error initializing
##                  }
##                  check_greater_or_equal(function__, "sig2", sig2, 0);
##
##                  current_statement_begin__ = 26;
##                  if (stan::math::is_uninitialized(sigma)) {
##                      std::stringstream msg__;
##                      msg__ << "Undefined transformed parameter: sigma";
##                      stan::lang::rethrow_located(std::runtime_error(std::string("Error initializing
##                  }
##                  check_greater_or_equal(function__, "sigma", sigma, 0);
##
##
##                  // model body
##
##                  current_statement_begin__ = 37;
##                  lp_accum__.add(gamma_log<propto__>(invsig2, .05, .005));
##                  current_statement_begin__ = 38;
##                  lp_accum__.add(normal_log<propto__>(b1, 0, 9.9e+06));
##                  current_statement_begin__ = 39;
##                  for (int i = 1; i <= N; ++i) {
##
##                      current_statement_begin__ = 40;
##                      lp_accum__.add(normal_log<propto__>(get_base1(bre1, i, "bre1", 1), 0, sigma))
##                      current_statement_begin__ = 41;
##                      for (int t = get_base1(Ni, i, "Ni", 1); t <= (get_base1(Ni, (i + 1), "Ni", 1)
```

```
##
##                     current_statement_begin__ = 42;
##                     lp_accum__.add(ordered_logistic_log<propto__>(get_base1(y, t, "y", 1), ge
##                 }
##             }
##
##         } catch (const std::exception& e) {
##             stan::lang::rethrow_located(e, current_statement_begin__, prog_reader__());
##             // Next line prevents compiler griping about no return
##             throw std::runtime_error("*** IF YOU SEE THIS, PLEASE REPORT A BUG ***");
##         }
##
##         lp_accum__.add(lp__);
##         return lp_accum__.sum();
##
##     } // log_prob()
##
##     template <bool propto, bool jacobian, typename T_>
##     T_ log_prob(Eigen::Matrix<T_,Eigen::Dynamic,1>& params_r,
##                std::ostream* pstream = 0) const {
##       std::vector<T_> vec_params_r;
##       vec_params_r.reserve(params_r.size());
##       for (int i = 0; i < params_r.size(); ++i)
##         vec_params_r.push_back(params_r(i));
##       std::vector<int> vec_params_i;
##       return log_prob<propto,jacobian,T_>(vec_params_r, vec_params_i, pstream);
##     }
##
##
##     void get_param_names(std::vector<std::string>& names__) const {
##         names__.resize(0);
##         names__.push_back("kappa");
##         names__.push_back("b1");
##         names__.push_back("bre1");
##         names__.push_back("invsig2");
##         names__.push_back("mu");
##         names__.push_back("sig2");
##         names__.push_back("sigma");
##         names__.push_back("log_lik");
##     }
##
##
##     void get_dims(std::vector<std::vector<size_t> >& dimss__) const {
##         dimss__.resize(0);
##         std::vector<size_t> dims__;
##         dims__.resize(0);
##         dims__.push_back(3);
##         dimss__.push_back(dims__);
##         dims__.resize(0);
##         dimss__.push_back(dims__);
##         dims__.resize(0);
##         dims__.push_back(N);
##         dimss__.push_back(dims__);
##         dims__.resize(0);
```

```
        dimss__.push_back(dims__);
        dims__.resize(0);
        dims__.push_back(n);
        dimss__.push_back(dims__);
        dims__.resize(0);
        dimss__.push_back(dims__);
        dims__.resize(0);
        dimss__.push_back(dims__);
        dims__.resize(0);
        dims__.push_back(n);
        dimss__.push_back(dims__);
    }

    template <typename RNG>
    void write_array(RNG& base_rng__,
                     std::vector<double>& params_r__,
                     std::vector<int>& params_i__,
                     std::vector<double>& vars__,
                     bool include_tparams__ = true,
                     bool include_gqs__ = true,
                     std::ostream* pstream__ = 0) const {
        typedef double local_scalar_t__;

        vars__.resize(0);
        stan::io::reader<local_scalar_t__> in__(params_r__, params_i__);
        static const char* function__ = "model72a3494fd723_24b938afe5b29efe8d963b6430c621be_na
        (void) function__;  // dummy to suppress unused var warning

        // read-transform, write parameters
        Eigen::Matrix<double, Eigen::Dynamic, 1> kappa = in__.ordered_constrain(3);
        size_t kappa_j_1_max__ = 3;
        for (size_t j_1__ = 0; j_1__ < kappa_j_1_max__; ++j_1__) {
            vars__.push_back(kappa(j_1__));
        }

        double b1 = in__.scalar_constrain();
        vars__.push_back(b1);

        std::vector<double> bre1;
        size_t bre1_d_0_max__ = N;
        bre1.reserve(bre1_d_0_max__);
        for (size_t d_0__ = 0; d_0__ < bre1_d_0_max__; ++d_0__) {
            bre1.push_back(in__.scalar_constrain());
        }
        size_t bre1_k_0_max__ = N;
        for (size_t k_0__ = 0; k_0__ < bre1_k_0_max__; ++k_0__) {
            vars__.push_back(bre1[k_0__]);
        }

        double invsig2 = in__.scalar_lb_constrain(0);
        vars__.push_back(invsig2);

        double lp__ = 0.0;
        (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_gq
## 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 "model72a3494fd723_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());
```

```
##                 param_name_stream__ << "sigma";
##                 param_names__.push_back(param_name_stream__.str());
##             }
## 
##             if (!include_gqs__) return;
##             size_t log_lik_j_1_max__ = n;
##             for (size_t j_1__ = 0; j_1__ < log_lik_j_1_max__; ++j_1__) {
##                 param_name_stream__.str(std::string());
##                 param_name_stream__ << "log_lik" << '.' << j_1__ + 1;
##                 param_names__.push_back(param_name_stream__.str());
##             }
##         }
## 
## 
##         void unconstrained_param_names(std::vector<std::string>& param_names__,
##                                        bool include_tparams__ = true,
##                                        bool include_gqs__ = true) const {
##             std::stringstream param_name_stream__;
##             size_t kappa_j_1_max__ = 3;
##             for (size_t j_1__ = 0; j_1__ < kappa_j_1_max__; ++j_1__) {
##                 param_name_stream__.str(std::string());
##                 param_name_stream__ << "kappa" << '.' << j_1__ + 1;
##                 param_names__.push_back(param_name_stream__.str());
##             }
##             param_name_stream__.str(std::string());
##             param_name_stream__ << "b1";
##             param_names__.push_back(param_name_stream__.str());
##             size_t bre1_k_0_max__ = N;
##             for (size_t k_0__ = 0; k_0__ < bre1_k_0_max__; ++k_0__) {
##                 param_name_stream__.str(std::string());
##                 param_name_stream__ << "bre1" << '.' << k_0__ + 1;
##                 param_names__.push_back(param_name_stream__.str());
##             }
##             param_name_stream__.str(std::string());
##             param_name_stream__ << "invsig2";
##             param_names__.push_back(param_name_stream__.str());
## 
##             if (!include_gqs__ && !include_tparams__) return;
## 
##             if (include_tparams__) {
##                 size_t mu_j_1_max__ = n;
##                 for (size_t j_1__ = 0; j_1__ < mu_j_1_max__; ++j_1__) {
##                     param_name_stream__.str(std::string());
##                     param_name_stream__ << "mu" << '.' << j_1__ + 1;
##                     param_names__.push_back(param_name_stream__.str());
##                 }
##                 param_name_stream__.str(std::string());
##                 param_name_stream__ << "sig2";
##                 param_names__.push_back(param_name_stream__.str());
##                 param_name_stream__.str(std::string());
##                 param_name_stream__ << "sigma";
##                 param_names__.push_back(param_name_stream__.str());
##             }
## 
```

```
##  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 model72a3494fd723_24b938afe5b29efe8d963b6430c621be_namespace::model72a3494fd723_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->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();
```

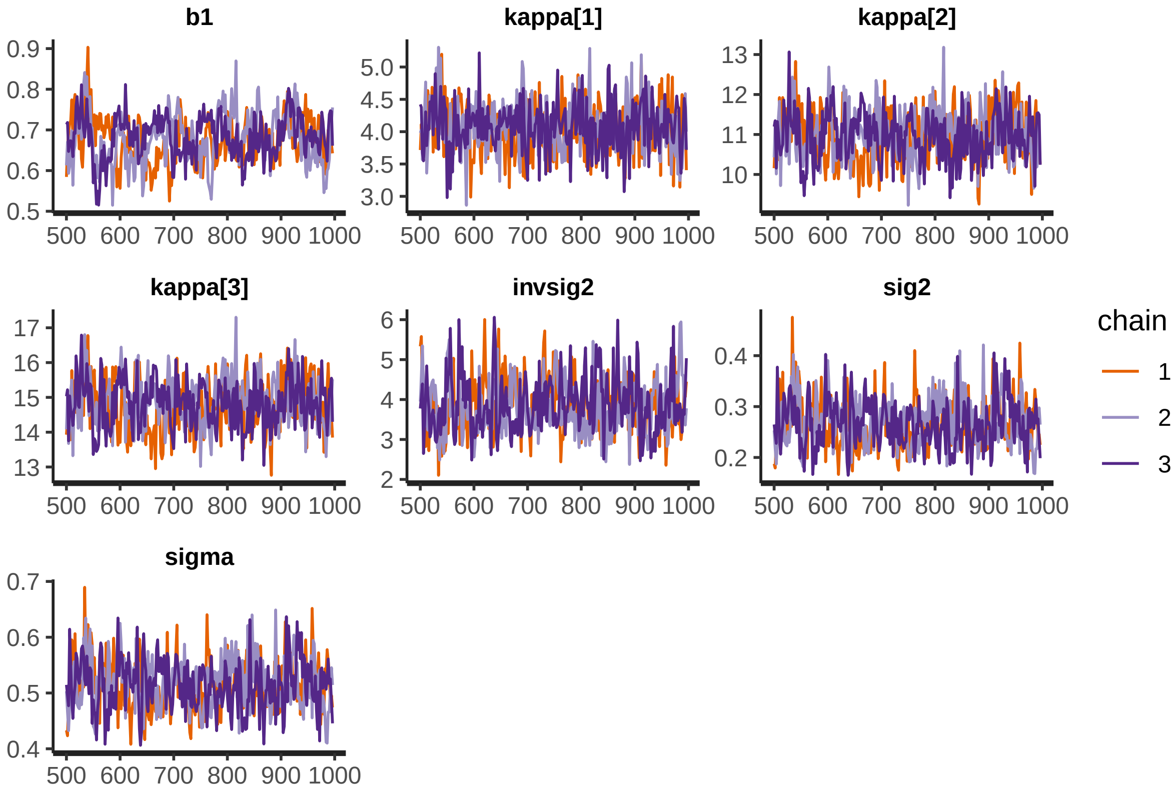
```
## 787 : }
## 788 :
## 789 : RCPP_MODULE(stan_fit4model72a3494fd723_24b938afe5b29efe8d963b6430c621be_mod){
## 790 :   Rcpp::class_<stan_model_holder>("stan_fit4model72a3494fd723_24b938afe5b29efe8d963b6430c621be
## 791 :   .constructor<rstan::io::rlist_ref_var_context, unsigned int>()
## 792 :   .method("model_ptr", &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 file72a332734574( ) ;
## 802 : }
## 803 :
## 804 : // definition
## 805 : SEXP file72a332734574() {
## 806 :  return Rcpp::wrap("24b938afe5b29efe8d963b6430c621be");
## 807 : }
```

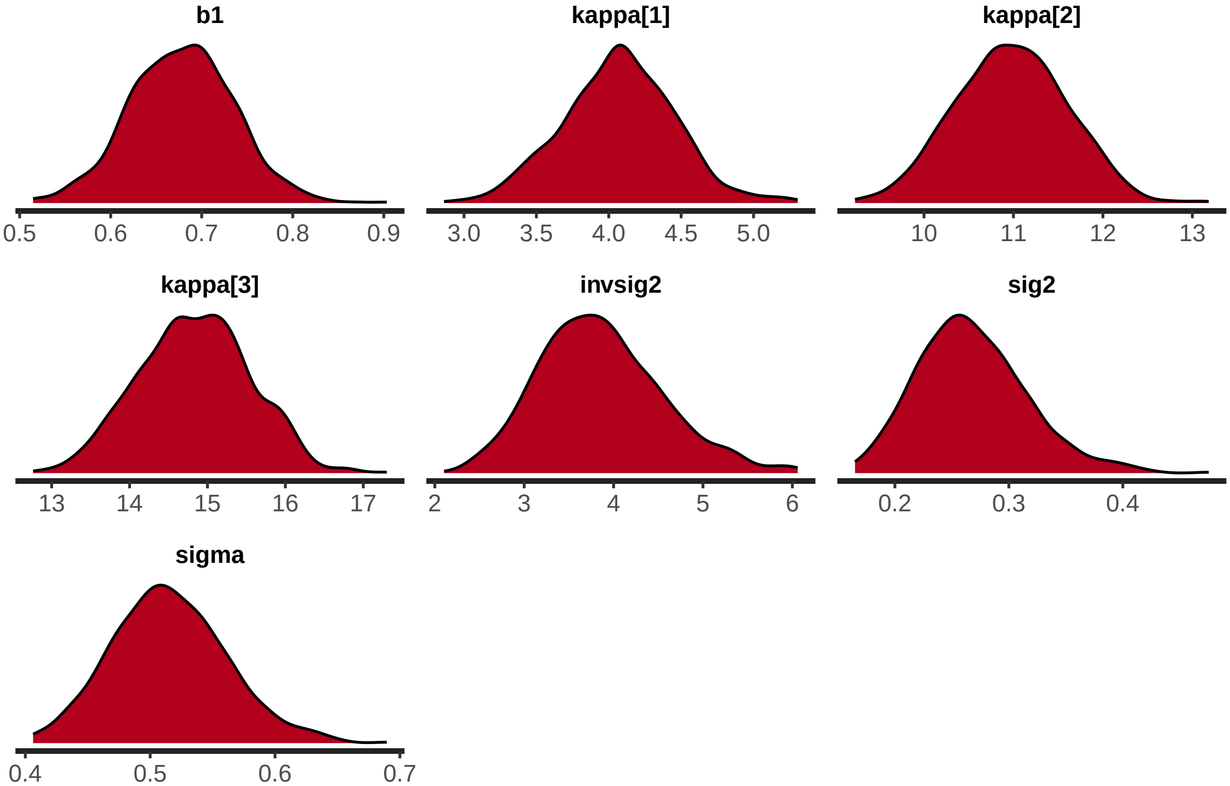
```
fit.lme.non.reslope <- sampling(mod,
                                data=datos.lme,
                                chains=3,warmup=500,iter=1000,thin=2,cores=4 )
```

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

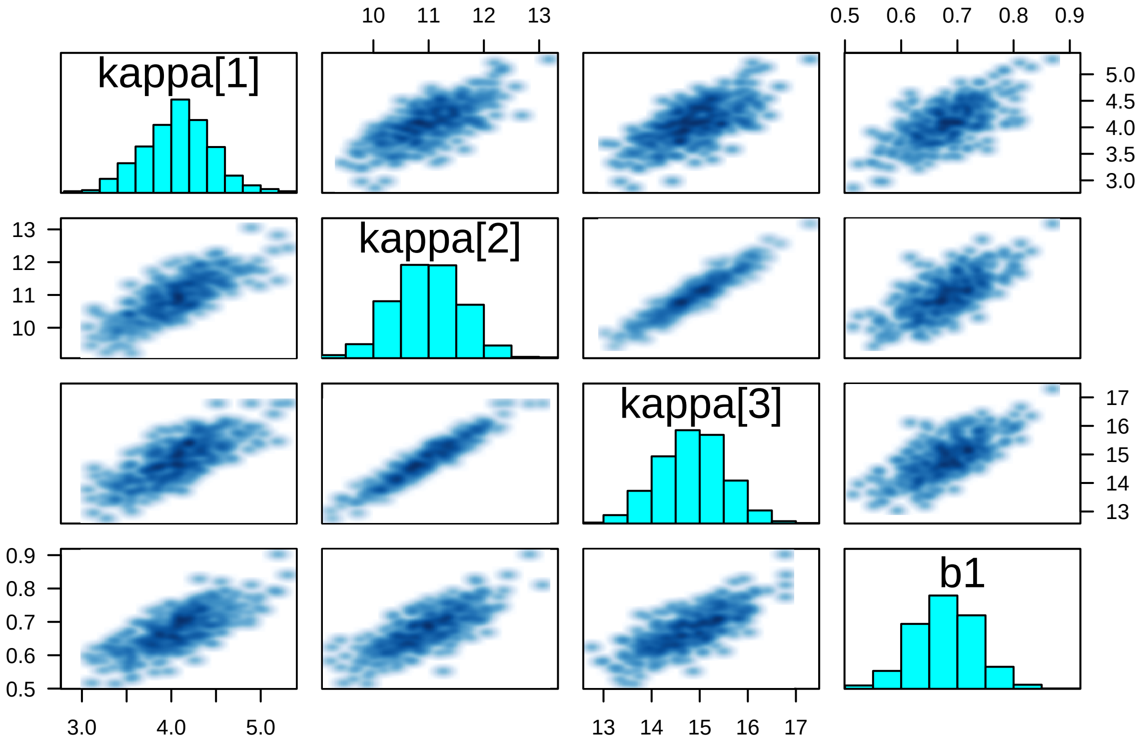
```
## Inference for Stan model: 24b938afe5b29efe8d963b6430c621be.
## 3 chains, each with iter=1000; warmup=500; thin=2;
## post-warmup draws per chain=250, total post-warmup draws=750.
##
##           mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## b1        0.68    0.01 0.06  0.56  0.64  0.68  0.72  0.79   108 1.01
## kappa[1]  4.07    0.02 0.39  3.33  3.82  4.08  4.33  4.86   448 1.00
## kappa[2] 10.98    0.04 0.62  9.77 10.54 10.99 11.41 12.18   316 1.00
## kappa[3] 14.84    0.04 0.71 13.46 14.35 14.86 15.32 16.13   293 1.00
## invsig2   3.85    0.04 0.69  2.65  3.36  3.79  4.29  5.35   307 1.00
## sig2      0.27    0.00 0.05  0.19  0.23  0.26  0.30  0.38   311 1.00
## sigma     0.52    0.00 0.05  0.43  0.48  0.51  0.55  0.61   308 1.00
##
## Samples were drawn using NUTS(diag_e) at Tue Jan  9 22:18:53 2024.
## 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)
```

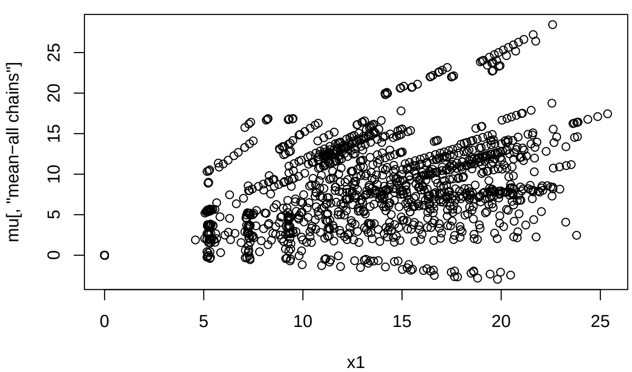





```
pairs(fit.lme.non.reslope, pars = c("kappa","b1"), las = 1)
```




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



6. Lineal creciente

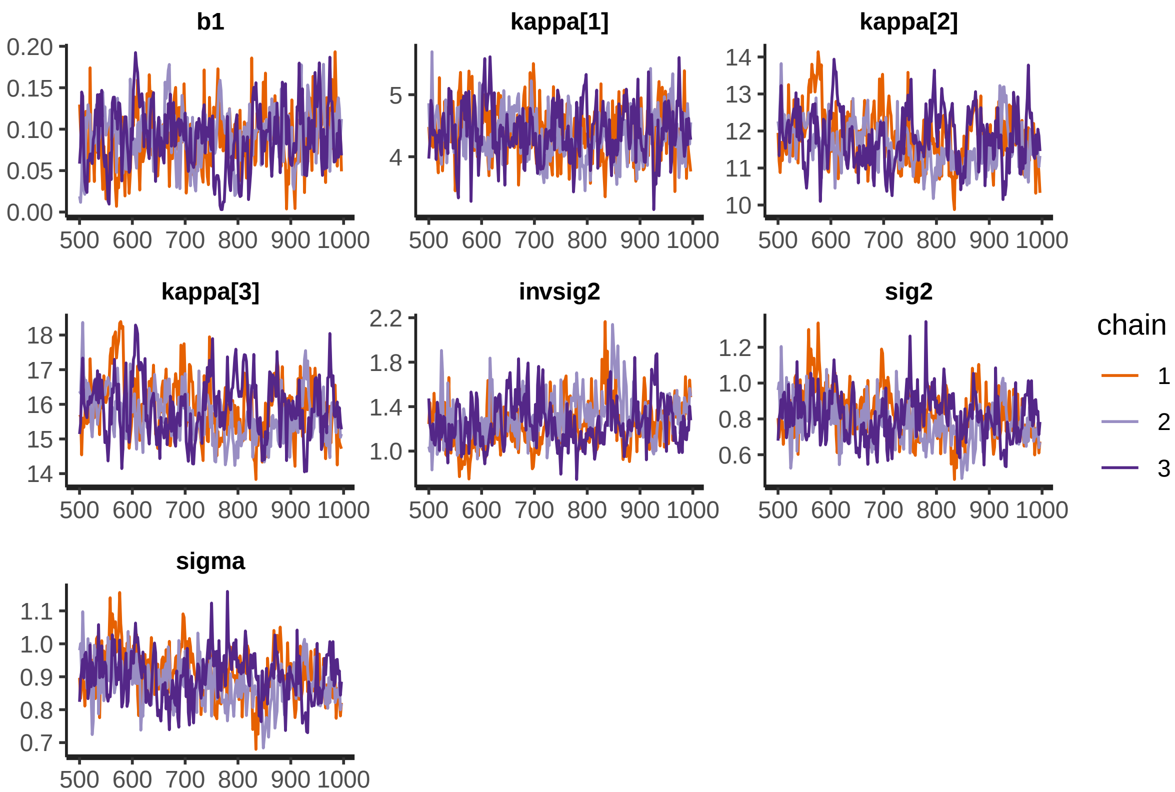
6.1. Lineal creciente

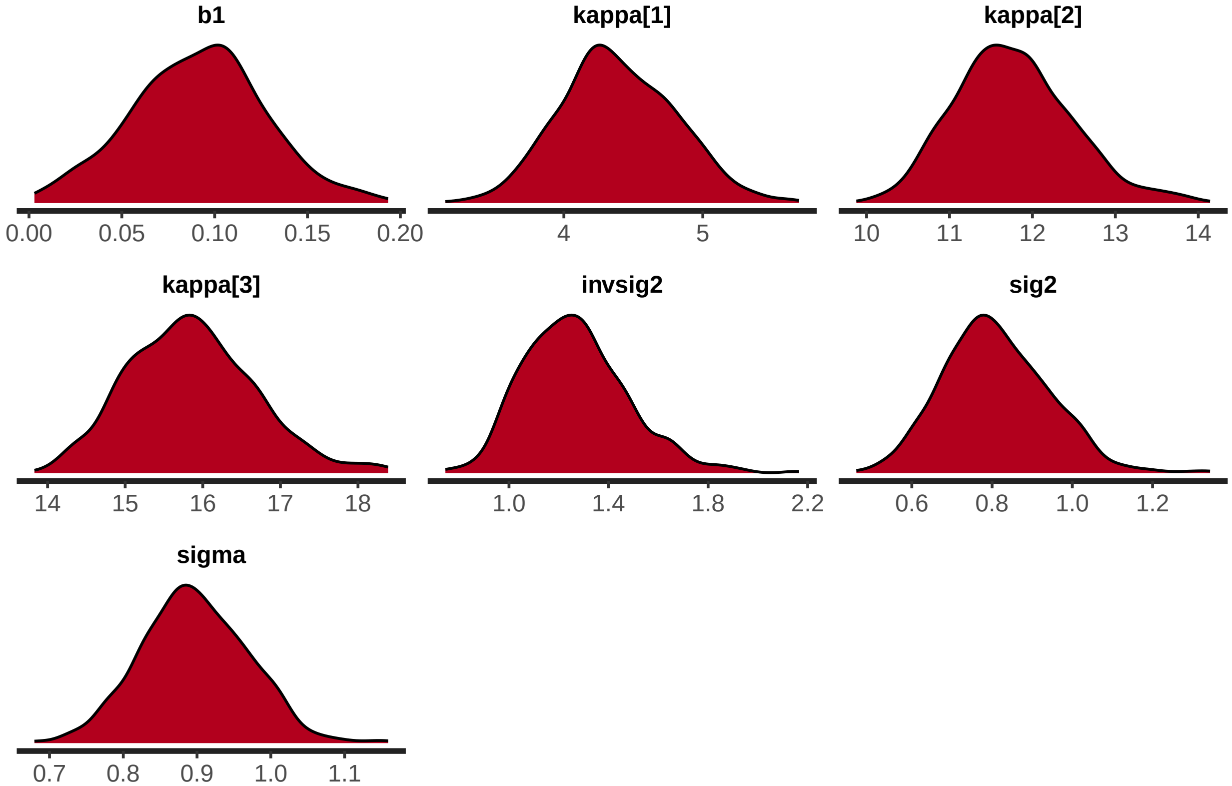
6.2. LME: Lineal creciente

```
fit.lme.incr.reslope <- stan("jagam_10_aneur_ordinal_lme_incr_reslope.stan",
            data=datos.lme,
            chains=3,warmup=500,iter=1000,thin=2,cores=4 )

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

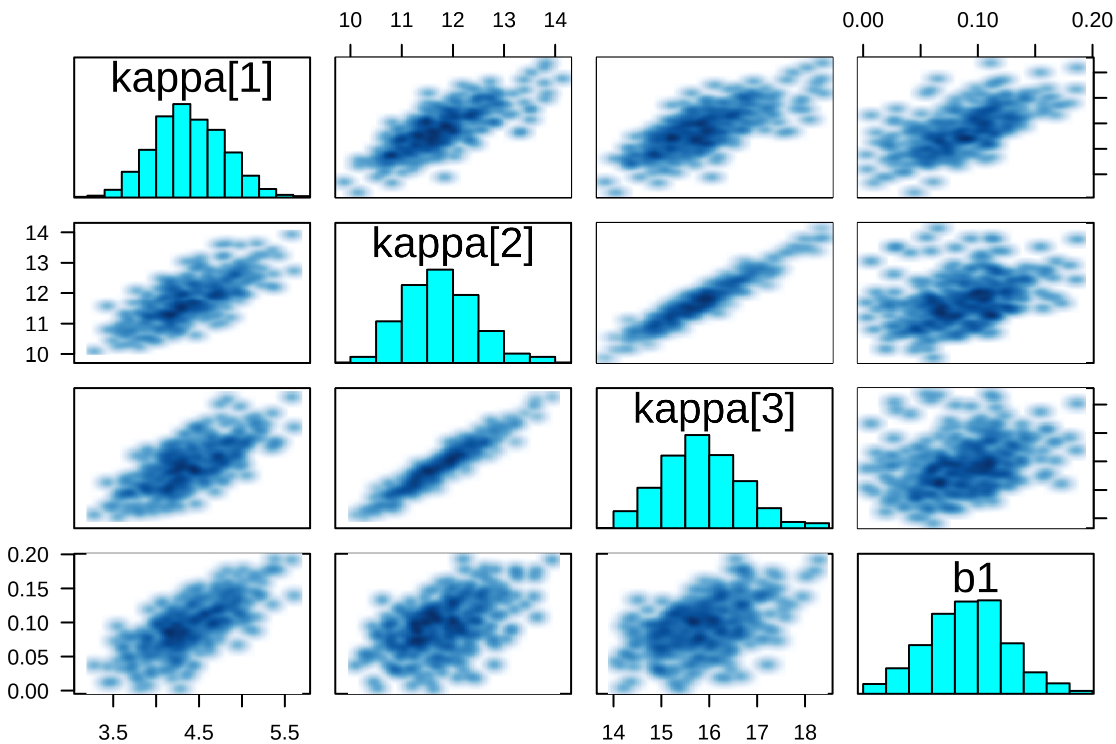
```
## Inference for Stan model: jagam_10_aneur_ordinal_lme_incr_reslope.
## 3 chains, each with iter=1000; warmup=500; thin=2;
## post-warmup draws per chain=250, total post-warmup draws=750.
##
##           mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## b1        0.09    0.00 0.04  0.02  0.07  0.09  0.11  0.16   228 1.01
## kappa[1]  4.38    0.02 0.42  3.63  4.11  4.35  4.68  5.20   333 1.00
## kappa[2] 11.76    0.07 0.71 10.53 11.27 11.70 12.21 13.35    90 1.05
## kappa[3] 15.85    0.09 0.82 14.37 15.24 15.82 16.34 17.60    85 1.05
## invsig2   1.27    0.02 0.21  0.93  1.12  1.25  1.40  1.72    94 1.05
## sig2      0.81    0.01 0.13  0.58  0.72  0.80  0.89  1.08    87 1.05
## sigma     0.90    0.01 0.07  0.76  0.85  0.89  0.94  1.04    88 1.05
##
## Samples were drawn using NUTS(diag_e) at Tue Jan  9 22:21:06 2024.
## 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)
```

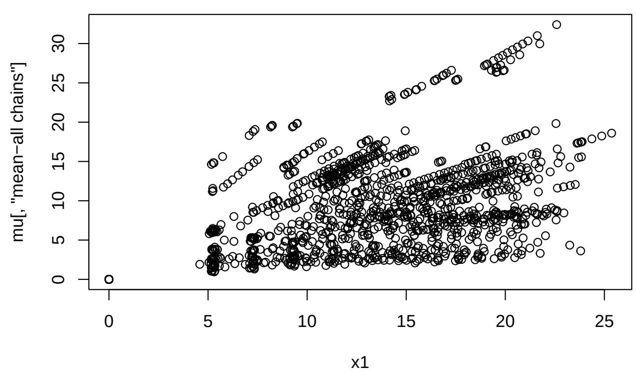





```
pairs(fit.lme.incr.reslope, pars = c("kappa","b1"), las = 1)
```




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



7. Spline NO restricciones

7.1 For a spline-based fit without constraints:

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

```
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")
```

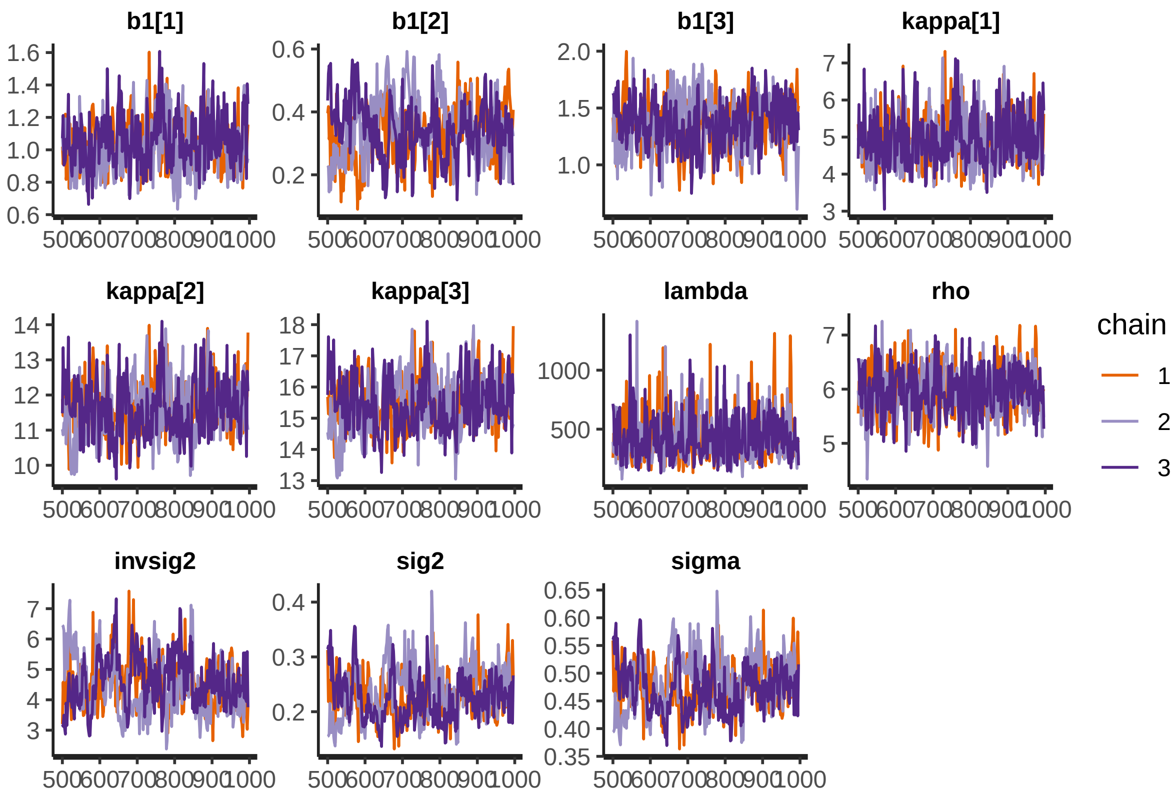
```
fit.lme.add.non.reslope <- stan("jagam_10_aneur_ordinal_lme_add_non_reslope.stan",
            data=datos.lme.add.non,
            chains=3,warmup=500,iter=1000,thin=2,cores=4,
            init= inits.lme.add.non)
```

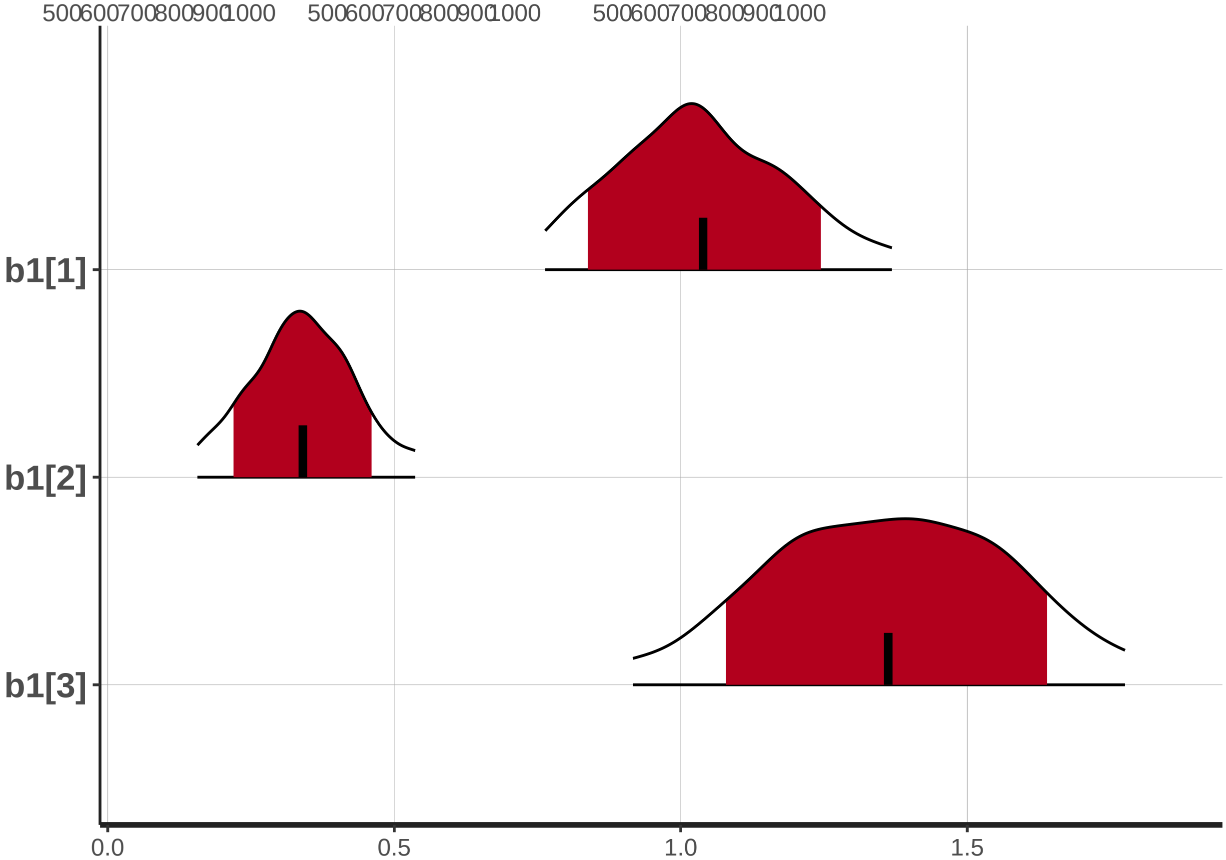
```
print(fit.lme.add.non.reslope, pars=param.lme.add)
```

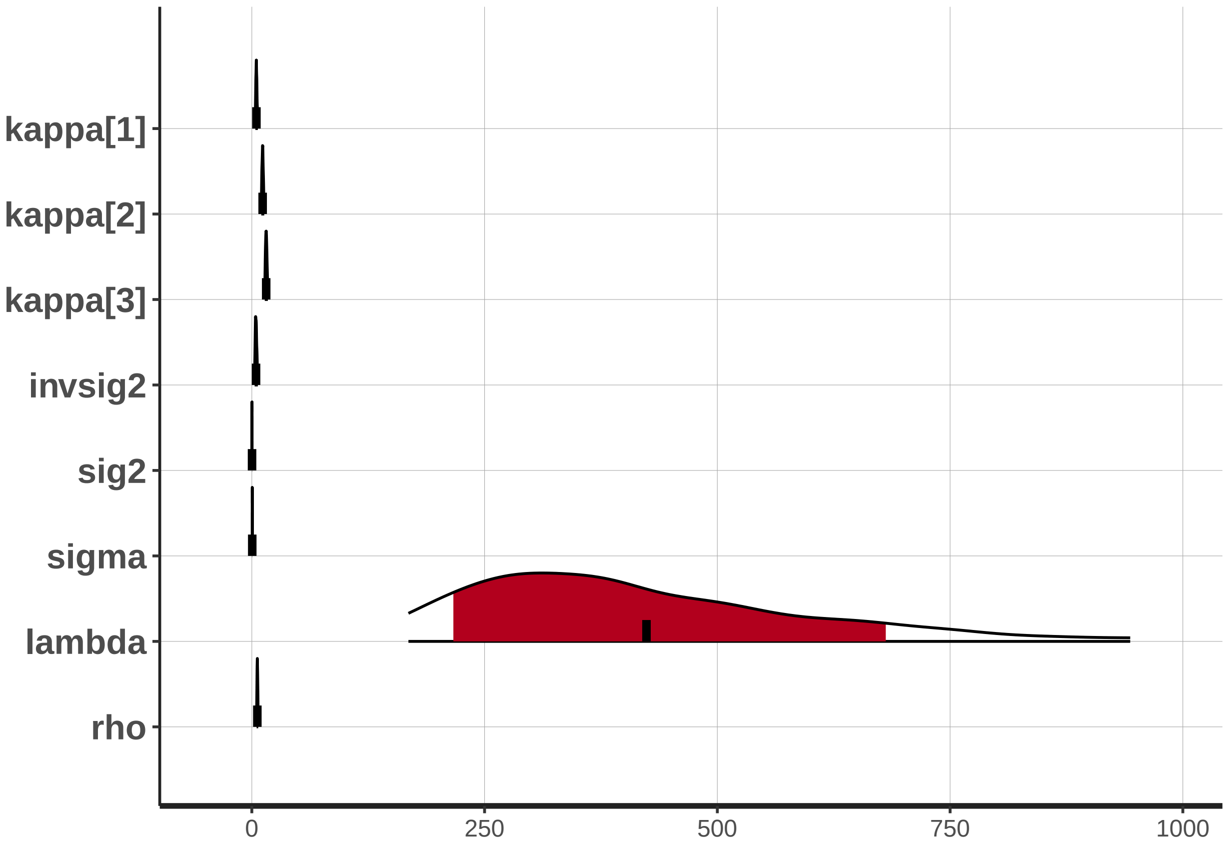
```
## Inference for Stan model: jagam_10_aneur_ordinal_lme_add_non_reslope.
## 3 chains, each with iter=1000; warmup=500; thin=2;
## post-warmup draws per chain=250, total post-warmup draws=750.
##
##             mean se_mean     sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
## b1[1]       1.04    0.01   0.16   0.76   0.93   1.03   1.15   1.37   272 1.02
## b1[2]       0.34    0.01   0.09   0.16   0.28   0.34   0.40   0.54    53 1.03
## b1[3]       1.36    0.02   0.22   0.92   1.20   1.37   1.52   1.78   188 1.00
## kappa[1]    4.94    0.03   0.67   3.80   4.44   4.88   5.38   6.49   460 1.00
## kappa[2]   11.63    0.05   0.79  10.15  11.07  11.60  12.14  13.35   251 1.00
## kappa[3]   15.45    0.06   0.86  13.89  14.84  15.42  16.04  17.20   185 1.00
## lambda    423.53    7.84 199.66 168.26 281.27 379.10 519.27 943.24   649 1.00
## rho         5.95    0.02   0.45   5.13   5.64   5.94   6.25   6.85   616 1.00
## invsig2     4.53    0.08   0.87   3.00   3.90   4.46   5.05   6.45   113 1.01
## sig2        0.23    0.00   0.04   0.16   0.20   0.22   0.26   0.33   122 1.01
## sigma       0.48    0.00   0.05   0.39   0.45   0.47   0.51   0.58   118 1.01
##
## Samples were drawn using NUTS(diag_e) at Tue Jan  9 22:24:27 2024.
## 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).
```

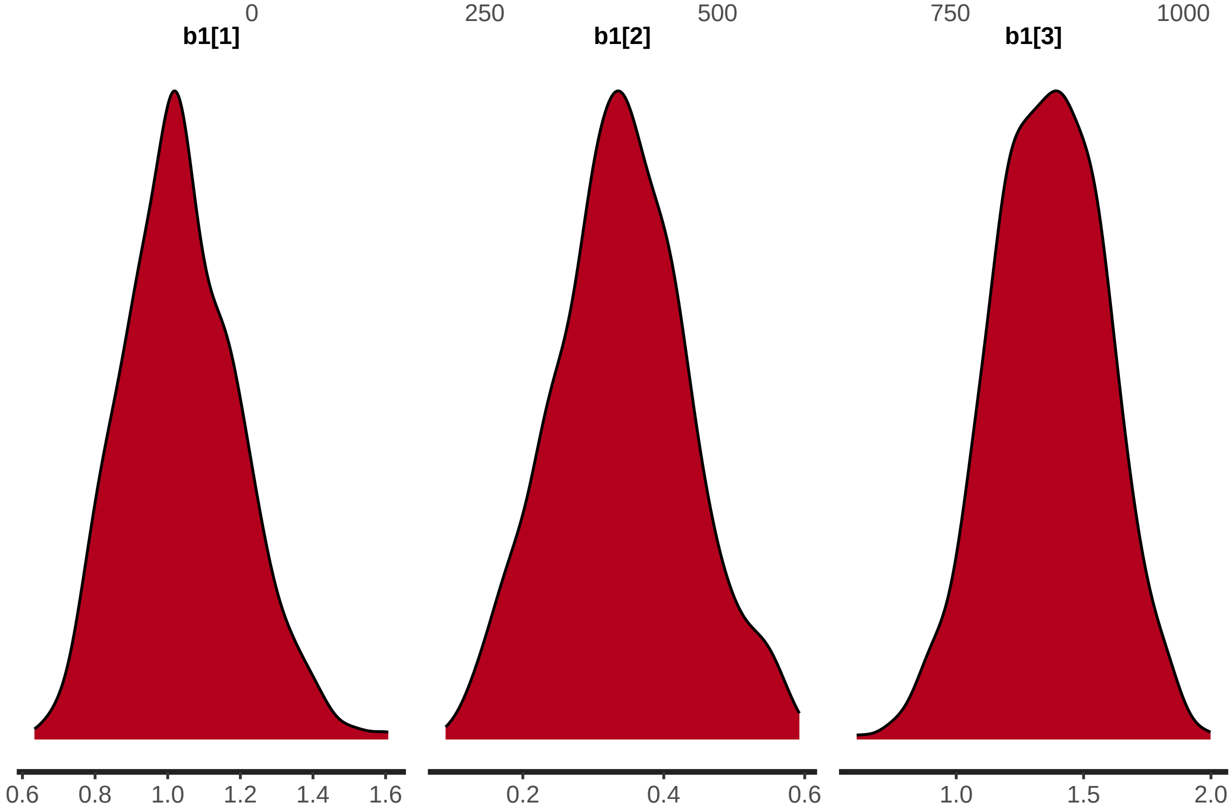
```
stan_trace(fit.lme.add.non.reslope, pars=param.lme.add)
stan_plot(fit.lme.add.non.reslope, pars=c("b1"), point_est = "mean", show_density = TRUE)
stan_plot(fit.lme.add.non.reslope, pars=c("kappa", "invsig2","sig2","sigma",  "lambda","rho"), point_es
```

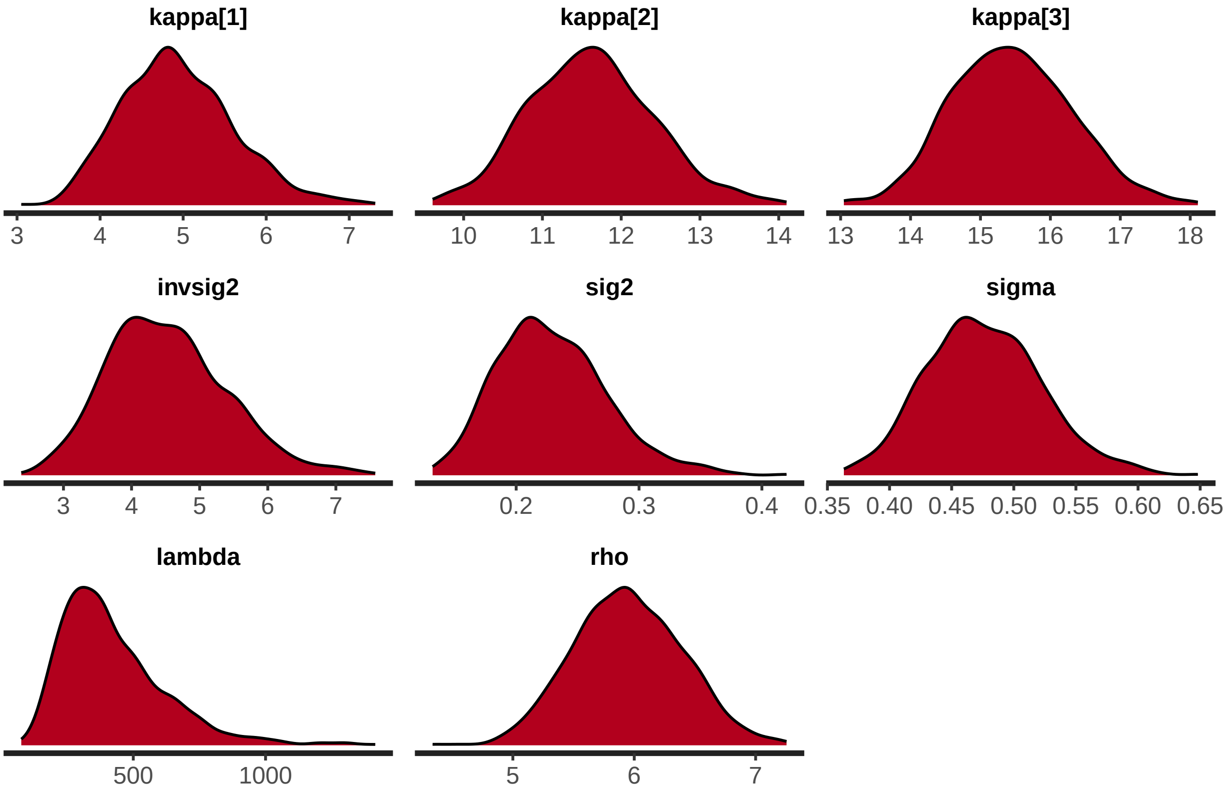
```
stan_dens(fit.lme.add.non.reslope, pars=c("b1"))
stan_dens(fit.lme.add.non.reslope, pars=c("kappa", "invsig2","sig2","sigma", "lambda","rho"))
```



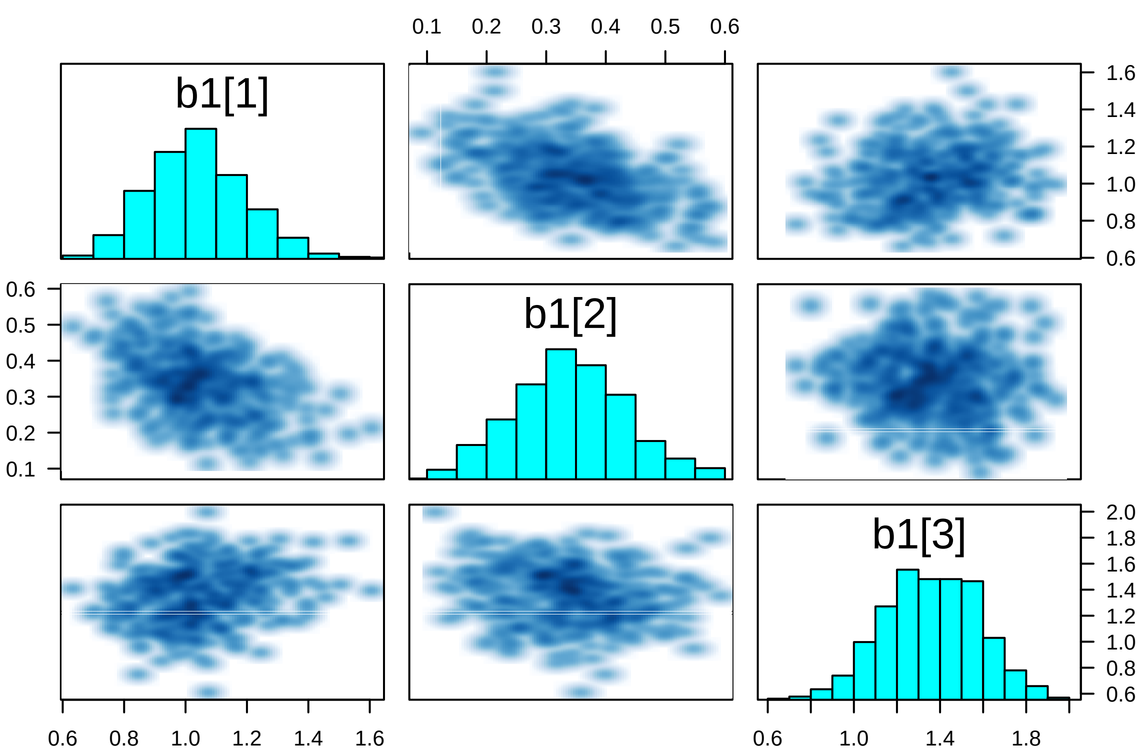



kappa[1]
kappa[2]
kappa[3]
invsig2
sig2
sigma
lambda
rho
0
250
500
750
1000

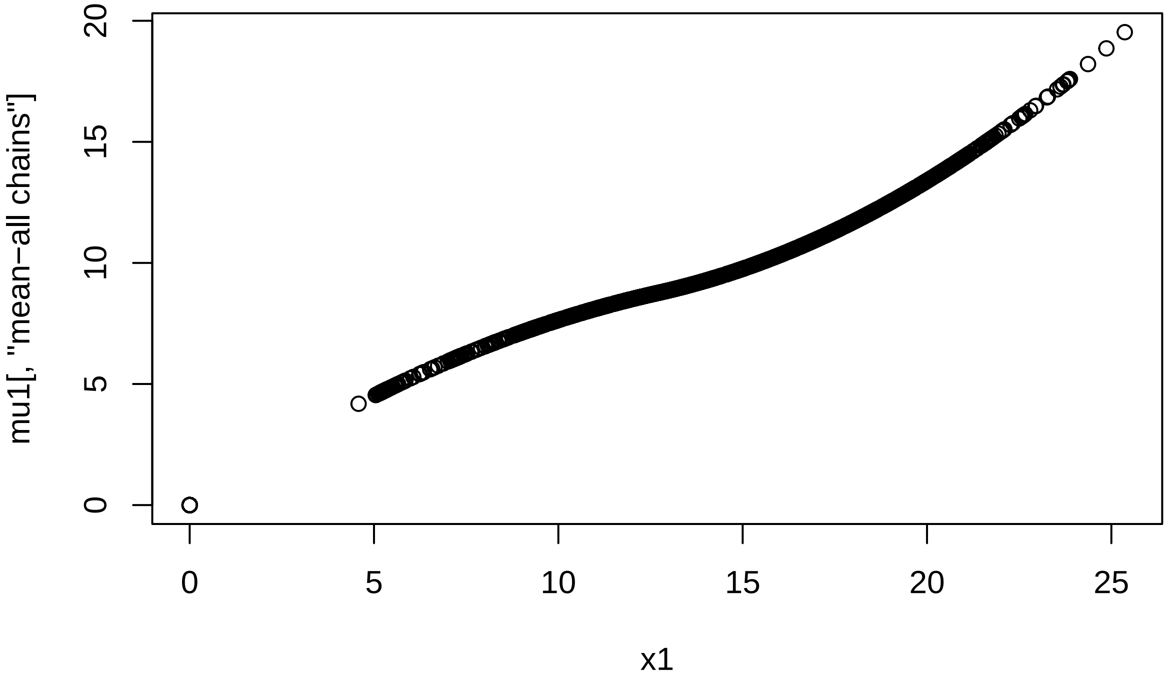

b1[1]
b1[2]
b1[3]
0.6
0.8
1.0
1.2
1.4
1.6
0.2
0.4
0.6
1.0
1.5
2.0



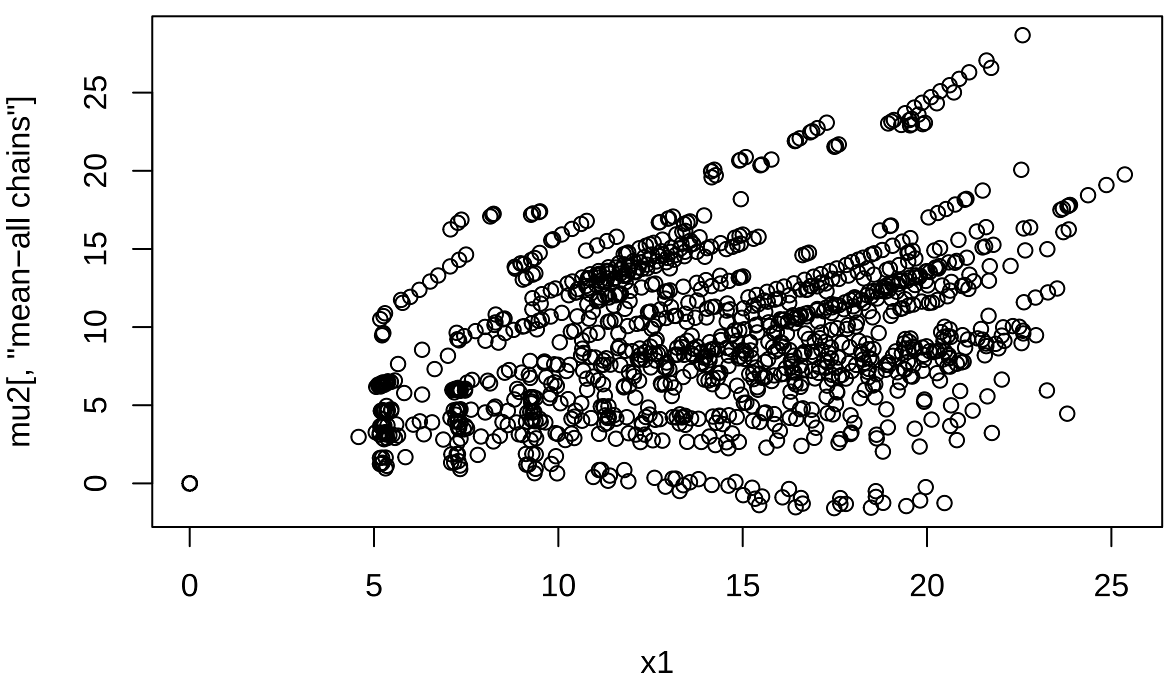

```
pairs(fit.lme.add.non.reslope, pars = c("b1"), las = 1)
```




```
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"])
```



8. Spline con restricciones creciente

8.1. LIN: Spline con restricciones creciente

8.2. LME: Spline con restricciones creciente

```r
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)
  ) }
```

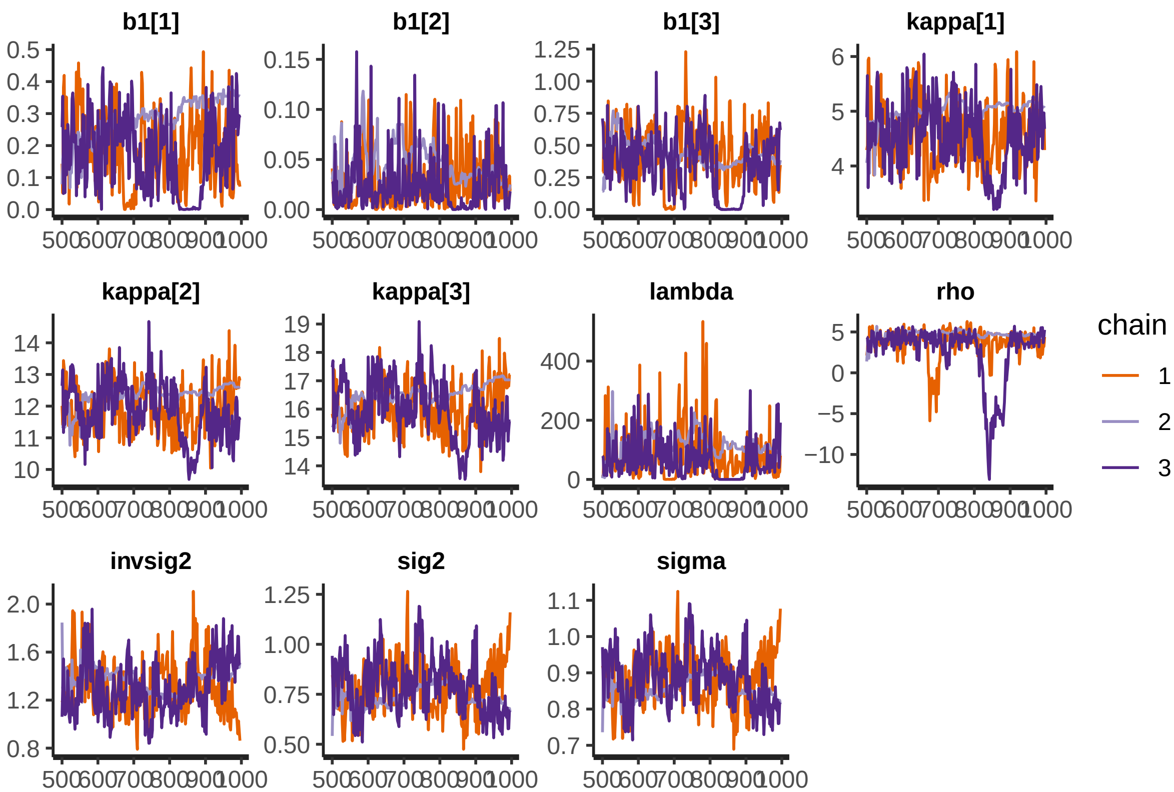
```r
fit.lme.add.incr.reslope <- stan("jagam_10_aneur_ordinal_lme_add_incr_reslope.stan",
            data=datos.lme.add.incr,
            chains=3,warmup=500,iter=1000,thin=2,cores=4,
            init= inits.lme.add.incr)
```

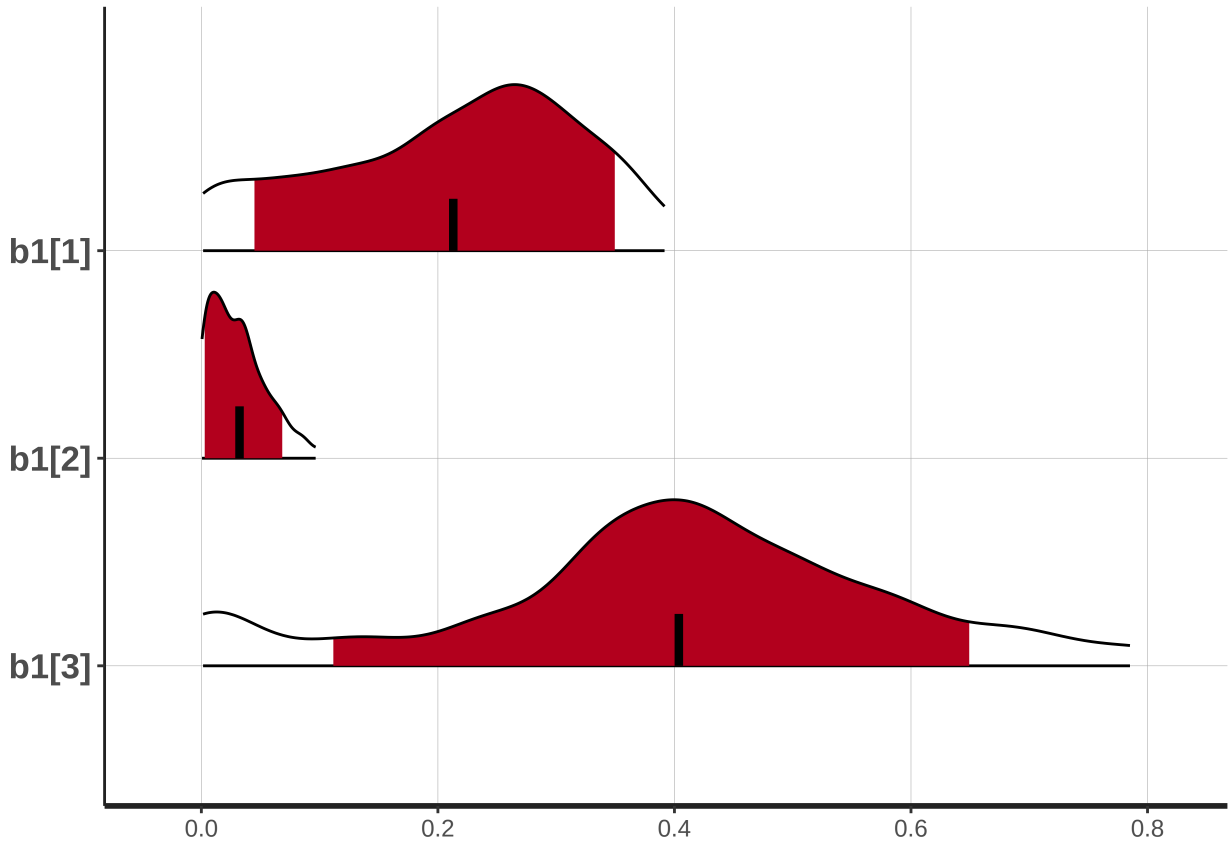
```r
print(fit.lme.add.incr.reslope, pars=param.lme.add, digits=5)
```

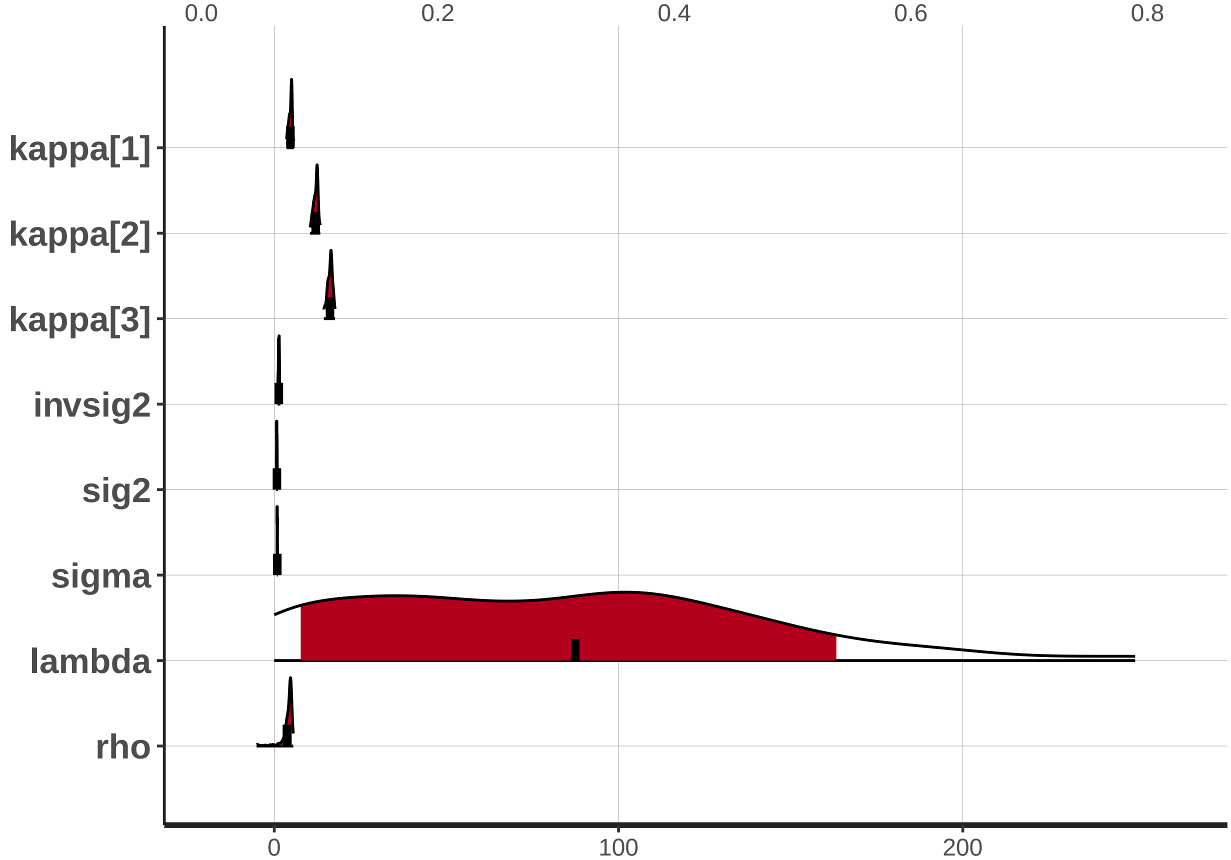
```
## Inference for Stan model: jagam_10_aneur_ordinal_lme_add_incr_reslope.
## 3 chains, each with iter=1000; warmup=500; thin=2;
## post-warmup draws per chain=250, total post-warmup draws=750.
##
##                mean  se_mean       sd     2.5%      25%      50%       75%
## b1[1]       0.21296  0.03513  0.10981  0.00149  0.13295  0.22987   0.29594
## b1[2]       0.03222  0.00612  0.02617  0.00055  0.01193  0.02719   0.04651
## b1[3]       0.40387  0.02309  0.19527  0.00144  0.32206  0.40966   0.51382
## kappa[1]    4.69186  0.15740  0.53892  3.56210  4.31711  4.82584   5.07672
## kappa[2]   12.02928  0.13312  0.74129 10.39872 11.52228 12.19801  12.52401
## kappa[3]   16.18608  0.13648  0.83109 14.38611 15.64551 16.31484  16.70719
## lambda     87.52883 13.40195 67.82209  0.00581 36.36246 81.91899 122.62249
## rho         3.66481  0.49576  2.44553 -5.14774  3.59354  4.40573   4.80911
## invsig2     1.32590  0.02457  0.19208  0.95433  1.21032  1.32884   1.43905
## sig2        0.77052  0.01594  0.11571  0.56438  0.69490  0.75254   0.82623
## sigma       0.87540  0.00872  0.06476  0.75125  0.83361  0.86749   0.90897
##                97.5% n_eff    Rhat
## b1[1]        0.39158    10 1.19270
## b1[2]        0.09650    18 1.09680
## b1[3]        0.79169    72 1.04948
## kappa[1]     5.68835    12 1.14479
## kappa[2]    13.36169    31 1.11874
## kappa[3]    17.66058    37 1.11430
## lambda     249.85117    26 1.07265
## rho          5.52085    24 1.14950
## invsig2      1.77185    61 1.02250
## sig2         1.04785    53 1.03856
## sigma        1.02365    55 1.03468
```

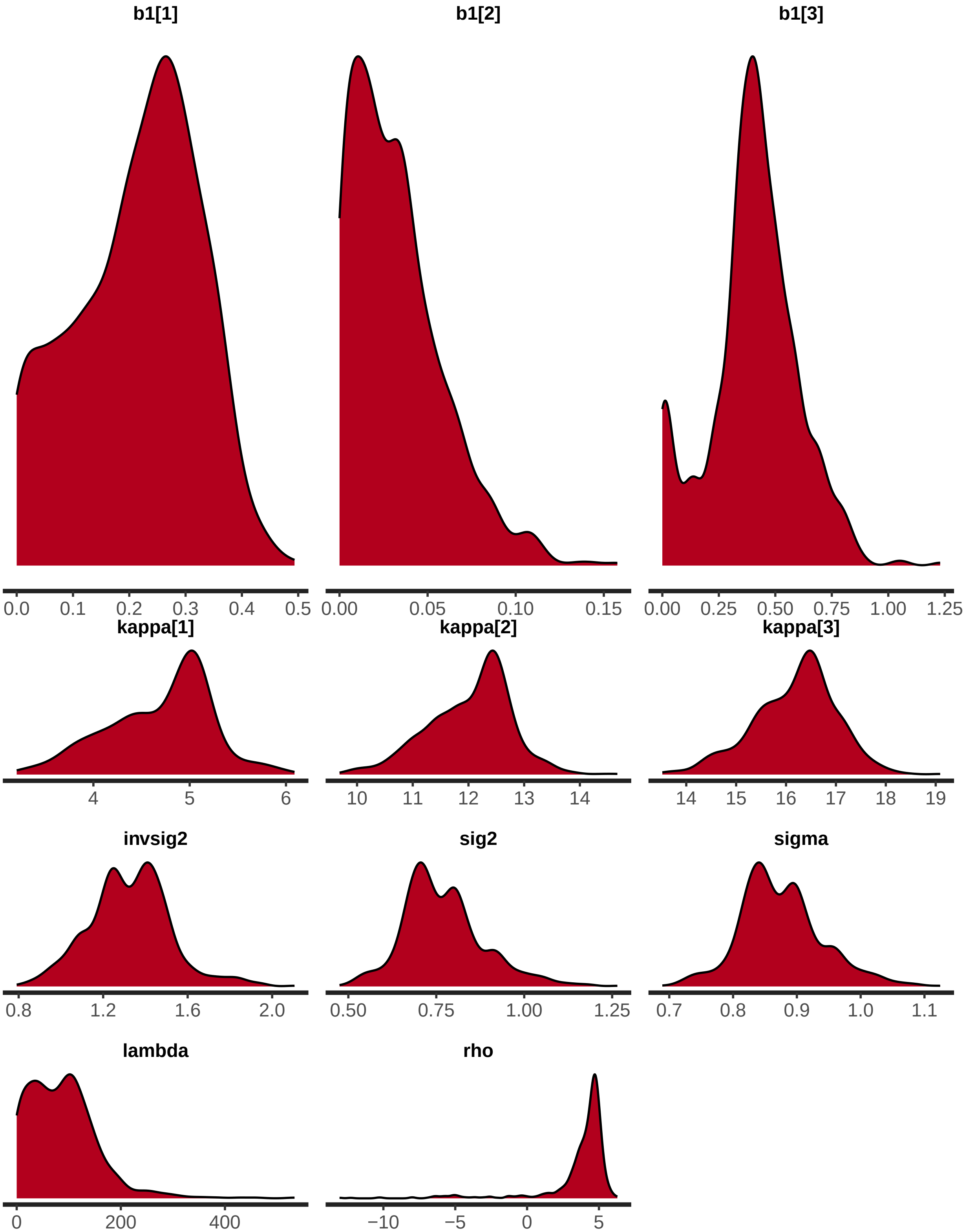
```
##
## Samples were drawn using NUTS(diag_e) at Tue Jan  9 22:28:04 2024.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
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"))
```

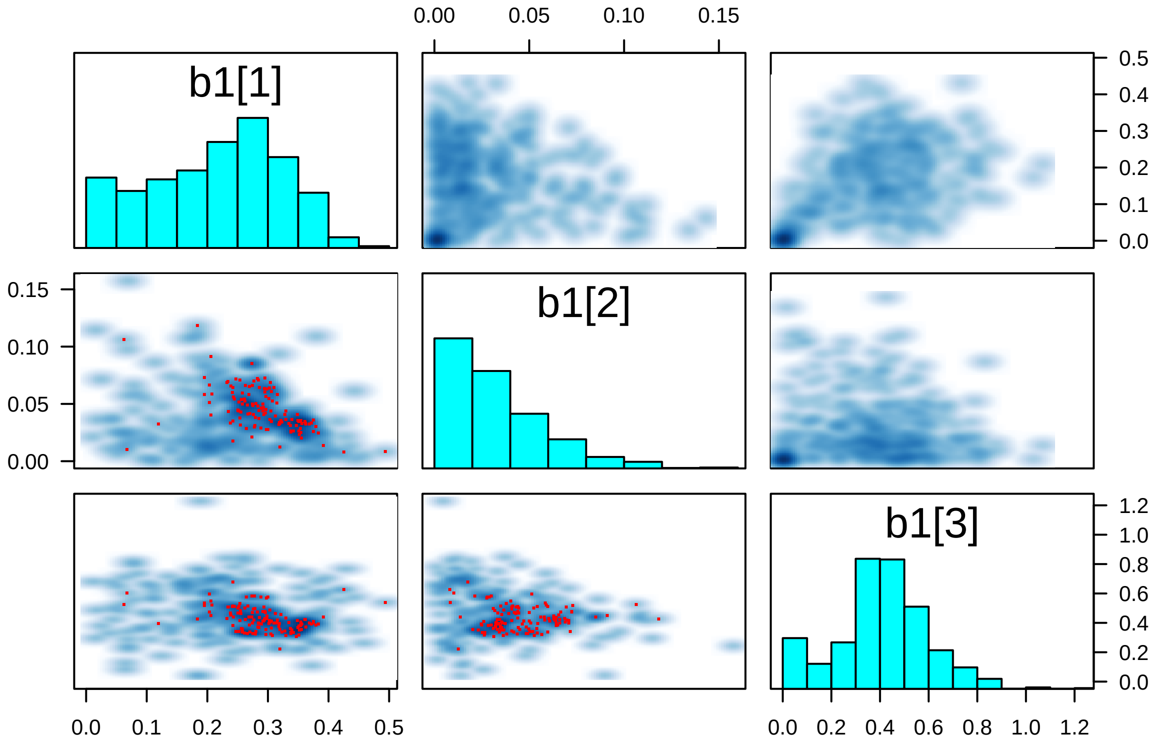



b1[1]
b1[2]
b1[3]
0.0
0.2
0.4
0.6
0.8

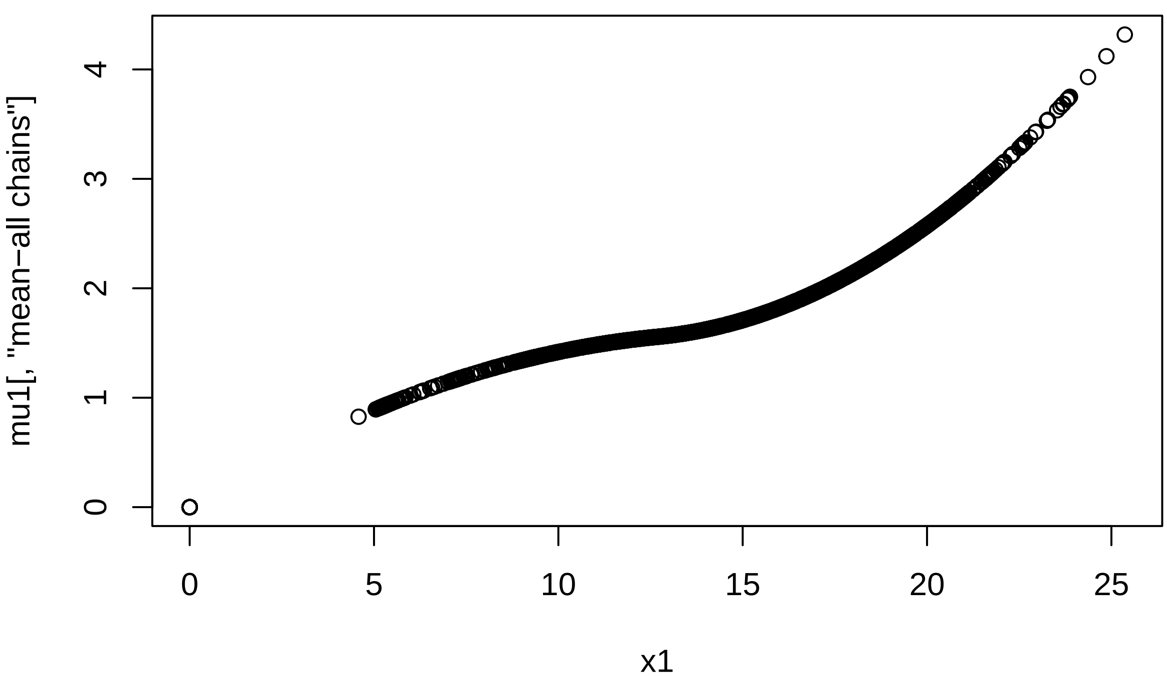

kappa[1]
kappa[2]
kappa[3]
invsig2
sig2
sigma
lambda
rho
0
100
200



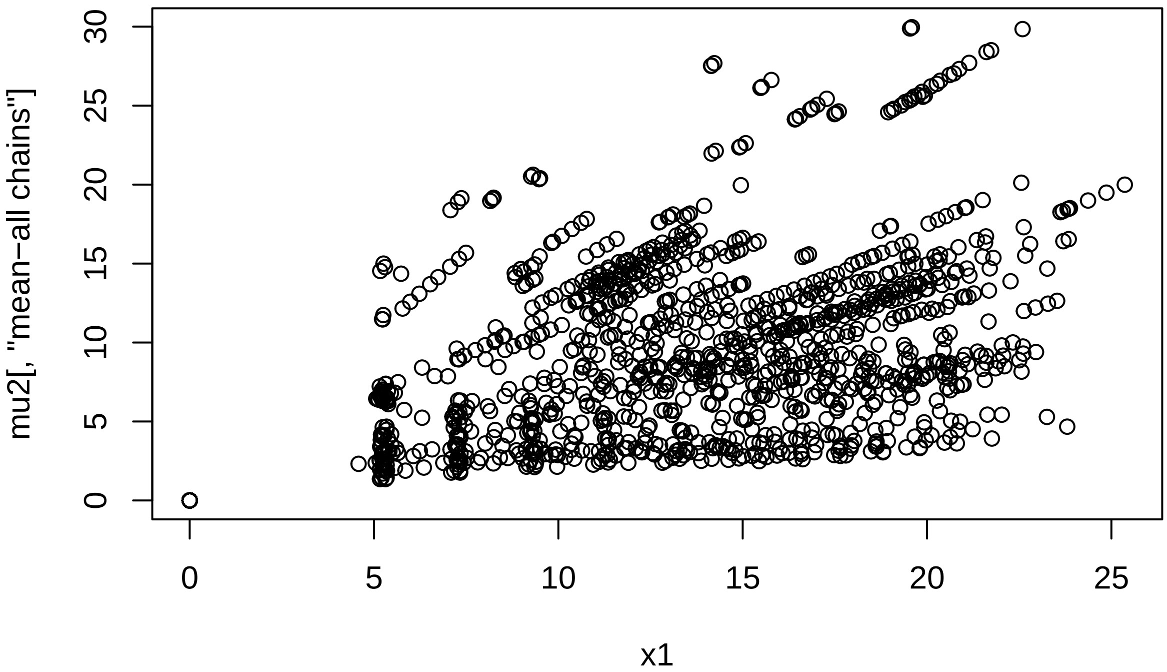

```
pairs(fit.lme.add.incr.reslope, pars = c("b1"), las = 1)
```




```
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' arg

##
## Attaching package: 'loo'

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

```
loo3_sample = fit.lme.non.reslope
loo6_sample = fit.lme.incr.reslope
loo9_sample = fit.lme.add.non.reslope
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_3 =extract_log_lik(loo3_sample, merge_chains = F)
log_lik_6 =extract_log_lik(loo6_sample, merge_chains = F)
log_lik_9 =extract_log_lik(loo9_sample, merge_chains = F)
```

```r
log_lik_12 =extract_log_lik(loo12_sample, merge_chains = F)

r_eff_3 =relative_eff(log_lik_3)
r_eff_6 =relative_eff(log_lik_6)
r_eff_9 =relative_eff(log_lik_9)
r_eff_12 =relative_eff(log_lik_12)
```

```r
###  look at the results for each model, first the one with mu estimated:
(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 750 by 1387 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -752.1 31.6
## p_loo       202.4 14.1
## looic      1504.3 63.2
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     1215  87.6%   112
##  (0.5, 0.7]   (ok)        102   7.4%   39
##    (0.7, 1]   (bad)        64   4.6%   9
##    (1, Inf)   (very bad)    6   0.4%   4
## See help('pareto-k-diagnostic') for details.
```

```r
(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 750 by 1387 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -721.5 29.4
## p_loo       183.3 11.4
## looic      1443.1 58.8
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     1236  89.1%   74
##  (0.5, 0.7]   (ok)        109   7.9%   30
##    (0.7, 1]   (bad)        32   2.3%   11
##    (1, Inf)   (very bad)   10   0.7%   2
## 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 750 by 1387 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -747.8 30.5
## p_loo       197.0 13.0
## looic      1495.5 61.0
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     1223  88.2%   85
##  (0.5, 0.7]   (ok)        105   7.6%   24
##    (0.7, 1]   (bad)        54   3.9%   11
##    (1, Inf)   (very bad)    5   0.4%   4
## 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 750 by 1387 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -710.3 28.9
## p_loo       171.3 10.9
## looic      1420.6 57.9
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     1183  85.3%   1
##  (0.5, 0.7]   (ok)        124   8.9%   0
##    (0.7, 1]   (bad)        71   5.1%   0
##    (1, Inf)   (very bad)    9   0.6%   0
## 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.
```

```
###  look at the results for each model, first the one with mu estimated:
(waic_3 <- waic(log_lik_3, r_eff=r_eff_3))
```

```
## Warning:
## 117 (8.4%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

```
##
## Computed from 750 by 1387 log-likelihood matrix
##
##           Estimate   SE
## elpd_waic   -739.5 31.7
## p_waic       189.7 14.4
## waic        1479.0 63.4
##
## 117 (8.4%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

```
(waic_6 <- waic(log_lik_6, r_eff=r_eff_6))
```

```
## Warning:
## 118 (8.5%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

```
##
## Computed from 750 by 1387 log-likelihood matrix
##
##           Estimate   SE
## elpd_waic   -707.8 28.7
## p_waic       169.5  9.9
## waic        1415.6 57.3
##
## 118 (8.5%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

```
(waic_9 <- waic(log_lik_9, r_eff=r_eff_9))
```

```
## Warning:
## 107 (7.7%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

```
##
## Computed from 750 by 1387 log-likelihood matrix
##
##           Estimate   SE
## elpd_waic   -735.6 30.5
## p_waic       184.8 13.1
## waic        1471.2 61.0
##
## 107 (7.7%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

```
(waic_12 <- waic(log_lik_12, r_eff=r_eff_12))
```

```
## Warning:
## 102 (7.4%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

```
##
## Computed from 750 by 1387 log-likelihood matrix
##
##           Estimate   SE
## elpd_waic   -699.1 28.4
## p_waic       160.2  9.8
```

```
## waic        1398.3 56.8
##
## 102 (7.4%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

```
#compare(waic_1, waic_2)
### The second model fits better than the first one, since its looic is smaller.
```