Ejemplos

Lizbeth Naranjo Albarrán y Luz Judith Rodriguez Esparza

Paper: *Modelos ocultos de Markov:*

una aplicación de estimación Bayesiana para series de tiempo financieras

Authors: Lizbeth Naranjo Albarrán & Luz Judith Rodríguez Esparza

Journal: Mixba'al

Year: 2023

https://github.com/lizbethna/HMMBayes.git

Este archivo muestra las instrucciones para correr los códigos de R y Stan.

Markov switching GARCH

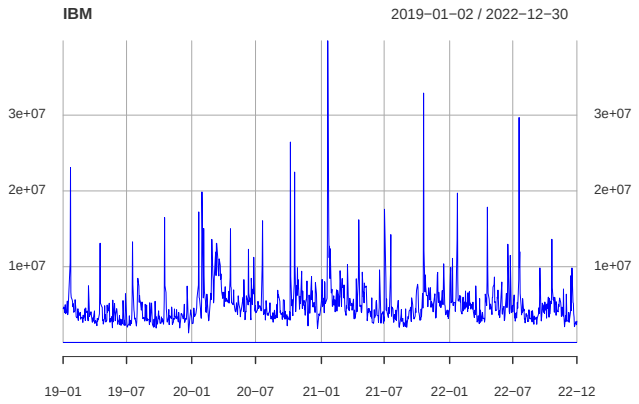
```
library(ggplot2)
library(rstan) # RStan
library(quantmod) # Quantitative Financial Modelling Framework
```

```
plot_statepath <- function(zstar) {
  K <- length(unique(as.vector(zstar)))
  x <- index(zstar)
  t <- 1:dim(zstar)[1]
  opar <- par(no.readonly = TRUE)
  zcol <- (1:K)[zstar]

  layout(matrix(c(1, 2), nrow = 2, ncol = 1), heights = c(0.95, 0.05))
  plot(x = x, y = zstar,
    xlab = bquote(t), ylab = bquote(hat(z)[t]),
    main = bquote("Secuencia mas probable de estados ocultos"),
    ylim = c(1, K), type = 'l', col = 'gray')

  points(x=x, y=zstar,
           pch = 21, bg = zcol, col = zcol, cex = 0.7)

  par(mai = c(0, 0, 0, 0))
  plot.new()
  legend(x = "center",
         legend = c('Trayectoria mas probable', paste('Estado', 1:K)),
         pch = c(NA, rep(21, K)),
         lwd = c(2, rep(NA, K)),
         col = c('lightgray', 1:K),
         pt.bg = c('lightgray', 1:K),
```

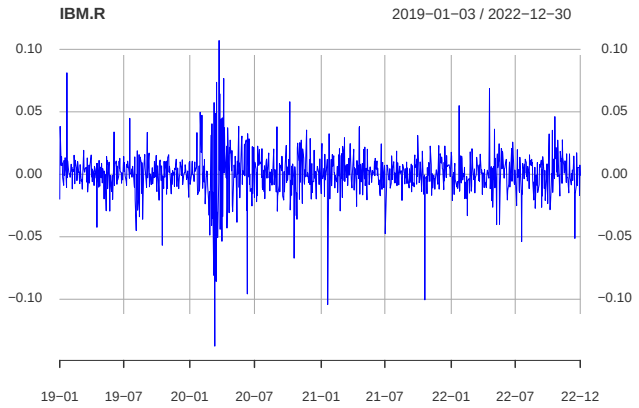
```
          bty = 'n', cex = 0.7,
          horiz = TRUE)
  par(opar)
}
```

Datos

```
IBM <- getSymbols("IBM",src='yahoo',
           from = "2019-01-01", to = "2022-12-31", auto.assign = FALSE)    # Obtener los datos
IBM.R <- na.omit(ROC(Ad(IBM)));    # Obtener los retornos

plot(IBM, format.labels="%y-%m", col="blue", lwd=0.5)
```




```
plot(IBM.R, format.labels="%y-%m", col="blue", lwd=0.5)
```



Código Stan

```
# Markov-switching GARCH
msgarch_fit <- function(y) {
  rstan_options(auto_write = TRUE)
  options(mc.cores = parallel::detectCores())
```

```
  stan.model = 'hmm_garch.stan'

  y <- as.vector(coredata(y));
  stan.data = list(
    T = length(y),
    y = y
  )

  stan(file = stan.model,
       data = stan.data, verbose = T,
       iter = 1000, warmup = 500,
       thin = 1, chains = 1,
       cores = 1, seed = 900)
}
# Fit GARCH
fit <- msgarch_fit(IBM.R)
```

```
TRANSLATING MODEL 'hmm_garch' FROM Stan CODE TO C++ CODE NOW.
successful in parsing the Stan model 'hmm_garch'.
OS: x86_64, darwin17.0; rstan: 2.21.3; Rcpp: 1.0.7; 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"/L
In file included from <built-in>:1:
In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/StanHeaders/includ
In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/include/
In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/include/
/Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/include/Eigen/src/Core/util/Ma
namespace Eigen {
^
/Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/include/Eigen/src/Core/util/Ma
namespace Eigen {
               ^
               ;
In file included from <built-in>:1:
In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/StanHeaders/includ
In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/include/
/Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/include/Eigen/Core:96:10: fata
#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//libStanServices.
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 model792915fa0784_hmm_garch_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", "model792915fa0784_hmm_garch");
32 :     reader.add_event(156, 154, "end", "model792915fa0784_hmm_garch");
33 :     return reader;
34 : }
35 :
36 : class model792915fa0784_hmm_garch
37 :   : public stan::model::model_base_crtp<model792915fa0784_hmm_garch> {
38 : private:
39 :         int T;
40 :         std::vector<double> y;
41 : public:
42 :     model792915fa0784_hmm_garch(rstan::io::rlist_ref_var_context& context__,
43 :         std::ostream* pstream__ = 0)
44 :         : model_base_crtp(0) {
45 :         ctor_body(context__, 0, pstream__);
46 :     }
47 :
48 :     model792915fa0784_hmm_garch(stan::io::var_context& context__,
49 :         unsigned int random_seed__,
50 :         std::ostream* pstream__ = 0)
51 :         : model_base_crtp(0) {
52 :         ctor_body(context__, random_seed__, pstream__);
53 :     }
54 :
55 :     void ctor_body(stan::io::var_context& context__,
56 :                    unsigned int random_seed__,
57 :                    std::ostream* pstream__) {
58 :         typedef double local_scalar_t__;
59 :
60 :         boost::ecuyer1988 base_rng__ =
61 :           stan::services::util::create_rng(random_seed__, 0);
```

```
 62 :         (void) base_rng__;  // suppress unused var warning
 63 :
 64 :         current_statement_begin__ = -1;
 65 :
 66 :         static const char* function__ = "model792915fa0784_hmm_garch_namespace::model792915fa078
 67 :         (void) function__;  // dummy to suppress unused var warning
 68 :         size_t pos__;
 69 :         (void) pos__;  // dummy to suppress unused var warning
 70 :         std::vector<int> vals_i__;
 71 :         std::vector<double> vals_r__;
 72 :         local_scalar_t__ DUMMY_VAR__(std::numeric_limits<double>::quiet_NaN());
 73 :         (void) DUMMY_VAR__;  // suppress unused var warning
 74 :
 75 :         try {
 76 :             // initialize data block variables from context__
 77 :             current_statement_begin__ = 6;
 78 :             context__.validate_dims("data initialization", "T", "int", context__.to_vec());
 79 :             T = int(0);
 80 :             vals_i__ = context__.vals_i("T");
 81 :             pos__ = 0;
 82 :             T = vals_i__[pos__++];
 83 :             check_greater_or_equal(function__, "T", T, 0);
 84 :
 85 :             current_statement_begin__ = 7;
 86 :             validate_non_negative_index("y", "T", T);
 87 :             context__.validate_dims("data initialization", "y", "double", context__.to_vec(T));
 88 :             y = std::vector<double>(T, double(0));
 89 :             vals_r__ = context__.vals_r("y");
 90 :             pos__ = 0;
 91 :             size_t y_k_0_max__ = T;
 92 :             for (size_t k_0__ = 0; k_0__ < y_k_0_max__; ++k_0__) {
 93 :                 y[k_0__] = vals_r__[pos__++];
 94 :             }
 95 :
 96 :
 97 :             // initialize transformed data variables
 98 :             // execute transformed data statements
 99 :
100 :             // validate transformed data
101 :
102 :             // validate, set parameter ranges
103 :             num_params_r__ = 0U;
104 :             param_ranges_i__.clear();
105 :             current_statement_begin__ = 12;
106 :             validate_non_negative_index("alpha0", "2", 2);
107 :             num_params_r__ += 2;
108 :             current_statement_begin__ = 14;
109 :             validate_non_negative_index("alpha1", "2", 2);
110 :             num_params_r__ += (1 * 2);
111 :             current_statement_begin__ = 15;
112 :             num_params_r__ += 1;
113 :             current_statement_begin__ = 16;
114 :             num_params_r__ += 1;
115 :             current_statement_begin__ = 20;
```

```
116 :             validate_non_negative_index("p_remain", "2", 2);
117 :             num_params_r__ += (1 * 2);
118 :         } catch (const std::exception& e) {
119 :             stan::lang::rethrow_located(e, current_statement_begin__, prog_reader__());
120 :             // Next line prevents compiler griping about no return
121 :             throw std::runtime_error("*** IF YOU SEE THIS, PLEASE REPORT A BUG ***");
122 :         }
123 :     }
124 :
125 :     ~model792915fa0784_hmm_garch() { }
126 :
127 :
128 :     void transform_inits(const stan::io::var_context& context__,
129 :                          std::vector<int>& params_i__,
130 :                          std::vector<double>& params_r__,
131 :                          std::ostream* pstream__) const {
132 :         typedef double local_scalar_t__;
133 :         stan::io::writer<double> writer__(params_r__, params_i__);
134 :         size_t pos__;
135 :         (void) pos__; // dummy call to supress warning
136 :         std::vector<double> vals_r__;
137 :         std::vector<int> vals_i__;
138 :
139 :         current_statement_begin__ = 12;
140 :         if (!(context__.contains_r("alpha0")))
141 :             stan::lang::rethrow_located(std::runtime_error(std::string("Variable alpha0 missing"
142 :         vals_r__ = context__.vals_r("alpha0");
143 :         pos__ = 0U;
144 :         validate_non_negative_index("alpha0", "2", 2);
145 :         context__.validate_dims("parameter initialization", "alpha0", "vector_d", context__.to_v
146 :         Eigen::Matrix<double, Eigen::Dynamic, 1> alpha0(2);
147 :         size_t alpha0_j_1_max__ = 2;
148 :         for (size_t j_1__ = 0; j_1__ < alpha0_j_1_max__; ++j_1__) {
149 :             alpha0(j_1__) = vals_r__[pos__++];
150 :         }
151 :         try {
152 :             writer__.positive_ordered_unconstrain(alpha0);
153 :         } catch (const std::exception& e) {
154 :             stan::lang::rethrow_located(std::runtime_error(std::string("Error transforming varia
155 :         }
156 :
157 :         current_statement_begin__ = 14;
158 :         if (!(context__.contains_r("alpha1")))
159 :             stan::lang::rethrow_located(std::runtime_error(std::string("Variable alpha1 missing"
160 :         vals_r__ = context__.vals_r("alpha1");
161 :         pos__ = 0U;
162 :         validate_non_negative_index("alpha1", "2", 2);
163 :         context__.validate_dims("parameter initialization", "alpha1", "double", context__.to_vec
164 :         std::vector<double> alpha1(2, double(0));
165 :         size_t alpha1_k_0_max__ = 2;
166 :         for (size_t k_0__ = 0; k_0__ < alpha1_k_0_max__; ++k_0__) {
167 :             alpha1[k_0__] = vals_r__[pos__++];
168 :         }
169 :         size_t alpha1_i_0_max__ = 2;
```

```
170 :         for (size_t i_0__ = 0; i_0__ < alpha1_i_0_max__; ++i_0__) {
171 :             try {
172 :                 writer__.scalar_lub_unconstrain(0, 1, alpha1[i_0__]);
173 :             } catch (const std::exception& e) {
174 :                 stan::lang::rethrow_located(std::runtime_error(std::string("Error transforming va
175 :             }
176 :         }
177 :
178 :         current_statement_begin__ = 15;
179 :         if (!(context__.contains_r("beta1_1")))
180 :             stan::lang::rethrow_located(std::runtime_error(std::string("Variable beta1_1 missing
181 :         vals_r__ = context__.vals_r("beta1_1");
182 :         pos__ = 0U;
183 :         context__.validate_dims("parameter initialization", "beta1_1", "double", context__.to_ve
184 :         double beta1_1(0);
185 :         beta1_1 = vals_r__[pos__++];
186 :         try {
187 :             writer__.scalar_lub_unconstrain(0, (1 - get_base1(alpha1, 1, "alpha1", 1)), beta1_1)
188 :         } catch (const std::exception& e) {
189 :             stan::lang::rethrow_located(std::runtime_error(std::string("Error transforming varia
190 :         }
191 :
192 :         current_statement_begin__ = 16;
193 :         if (!(context__.contains_r("beta1_2")))
194 :             stan::lang::rethrow_located(std::runtime_error(std::string("Variable beta1_2 missing
195 :         vals_r__ = context__.vals_r("beta1_2");
196 :         pos__ = 0U;
197 :         context__.validate_dims("parameter initialization", "beta1_2", "double", context__.to_ve
198 :         double beta1_2(0);
199 :         beta1_2 = vals_r__[pos__++];
200 :         try {
201 :             writer__.scalar_lub_unconstrain(0, (1 - get_base1(alpha1, 2, "alpha1", 1)), beta1_2)
202 :         } catch (const std::exception& e) {
203 :             stan::lang::rethrow_located(std::runtime_error(std::string("Error transforming varia
204 :         }
205 :
206 :         current_statement_begin__ = 20;
207 :         if (!(context__.contains_r("p_remain")))
208 :             stan::lang::rethrow_located(std::runtime_error(std::string("Variable p_remain missing
209 :         vals_r__ = context__.vals_r("p_remain");
210 :         pos__ = 0U;
211 :         validate_non_negative_index("p_remain", "2", 2);
212 :         context__.validate_dims("parameter initialization", "p_remain", "double", context__.to_v
213 :         std::vector<double> p_remain(2, double(0));
214 :         size_t p_remain_k_0_max__ = 2;
215 :         for (size_t k_0__ = 0; k_0__ < p_remain_k_0_max__; ++k_0__) {
216 :             p_remain[k_0__] = vals_r__[pos__++];
217 :         }
218 :         size_t p_remain_i_0_max__ = 2;
219 :         for (size_t i_0__ = 0; i_0__ < p_remain_i_0_max__; ++i_0__) {
220 :             try {
221 :                 writer__.scalar_lub_unconstrain(0, 1, p_remain[i_0__]);
222 :             } catch (const std::exception& e) {
223 :                 stan::lang::rethrow_located(std::runtime_error(std::string("Error transforming va
```

```
224 :             }
225 :         }
226 :
227 :         params_r__ = writer__.data_r();
228 :         params_i__ = writer__.data_i();
229 :     }
230 :
231 :     void transform_inits(const stan::io::var_context& context,
232 :                          Eigen::Matrix<double, Eigen::Dynamic, 1>& params_r,
233 :                          std::ostream* pstream__) const {
234 :       std::vector<double> params_r_vec;
235 :       std::vector<int> params_i_vec;
236 :       transform_inits(context, params_i_vec, params_r_vec, pstream__);
237 :       params_r.resize(params_r_vec.size());
238 :       for (int i = 0; i < params_r.size(); ++i)
239 :         params_r(i) = params_r_vec[i];
240 :     }
241 :
242 :
243 :     template <bool propto__, bool jacobian__, typename T__>
244 :     T__ log_prob(std::vector<T__>& params_r__,
245 :                  std::vector<int>& params_i__,
246 :                  std::ostream* pstream__ = 0) const {
247 :
248 :         typedef T__ local_scalar_t__;
249 :
250 :         local_scalar_t__ DUMMY_VAR__(std::numeric_limits<double>::quiet_NaN());
251 :         (void) DUMMY_VAR__;  // dummy to suppress unused var warning
252 :
253 :         T__ lp__(0.0);
254 :         stan::math::accumulator<T__> lp_accum__;
255 :         try {
256 :             stan::io::reader<local_scalar_t__> in__(params_r__, params_i__);
257 :
258 :             // model parameters
259 :             current_statement_begin__ = 12;
260 :             Eigen::Matrix<local_scalar_t__, Eigen::Dynamic, 1> alpha0;
261 :             (void) alpha0;  // dummy to suppress unused var warning
262 :             if (jacobian__)
263 :                 alpha0 = in__.positive_ordered_constrain(2, lp__);
264 :             else
265 :                 alpha0 = in__.positive_ordered_constrain(2);
266 :
267 :             current_statement_begin__ = 14;
268 :             std::vector<local_scalar_t__> alpha1;
269 :             size_t alpha1_d_0_max__ = 2;
270 :             alpha1.reserve(alpha1_d_0_max__);
271 :             for (size_t d_0__ = 0; d_0__ < alpha1_d_0_max__; ++d_0__) {
272 :                 if (jacobian__)
273 :                     alpha1.push_back(in__.scalar_lub_constrain(0, 1, lp__));
274 :                 else
275 :                     alpha1.push_back(in__.scalar_lub_constrain(0, 1));
276 :             }
277 :
```

```
278 :             current_statement_begin__ = 15;
279 :             local_scalar_t__ beta1_1;
280 :             (void) beta1_1;  // dummy to suppress unused var warning
281 :             if (jacobian__)
282 :                 beta1_1 = in__.scalar_lub_constrain(0, (1 - get_base1(alpha1, 1, "alpha1", 1)), l
283 :             else
284 :                 beta1_1 = in__.scalar_lub_constrain(0, (1 - get_base1(alpha1, 1, "alpha1", 1)));
285 :
286 :             current_statement_begin__ = 16;
287 :             local_scalar_t__ beta1_2;
288 :             (void) beta1_2;  // dummy to suppress unused var warning
289 :             if (jacobian__)
290 :                 beta1_2 = in__.scalar_lub_constrain(0, (1 - get_base1(alpha1, 2, "alpha1", 1)), l
291 :             else
292 :                 beta1_2 = in__.scalar_lub_constrain(0, (1 - get_base1(alpha1, 2, "alpha1", 1)));
293 :
294 :             current_statement_begin__ = 20;
295 :             std::vector<local_scalar_t__> p_remain;
296 :             size_t p_remain_d_0_max__ = 2;
297 :             p_remain.reserve(p_remain_d_0_max__);
298 :             for (size_t d_0__ = 0; d_0__ < p_remain_d_0_max__; ++d_0__) {
299 :                 if (jacobian__)
300 :                     p_remain.push_back(in__.scalar_lub_constrain(0, 1, lp__));
301 :                 else
302 :                     p_remain.push_back(in__.scalar_lub_constrain(0, 1));
303 :             }
304 :
305 :             // transformed parameters
306 :             current_statement_begin__ = 25;
307 :             validate_non_negative_index("beta1", "2", 2);
308 :             std::vector<local_scalar_t__> beta1(2, local_scalar_t__(0));
309 :             stan::math::initialize(beta1, DUMMY_VAR__);
310 :             stan::math::fill(beta1, DUMMY_VAR__);
311 :
312 :             current_statement_begin__ = 28;
313 :             validate_non_negative_index("sigma_t", "2", 2);
314 :             validate_non_negative_index("sigma_t", "T", T);
315 :             std::vector<Eigen::Matrix<local_scalar_t__, Eigen::Dynamic, 1> > sigma_t(T, Eigen::Ma
316 :             stan::math::initialize(sigma_t, DUMMY_VAR__);
317 :             stan::math::fill(sigma_t, DUMMY_VAR__);
318 :
319 :             current_statement_begin__ = 31;
320 :             validate_non_negative_index("log_alpha", "2", 2);
321 :             validate_non_negative_index("log_alpha", "T", T);
322 :             std::vector<Eigen::Matrix<local_scalar_t__, Eigen::Dynamic, 1> > log_alpha(T, Eigen:
323 :             stan::math::initialize(log_alpha, DUMMY_VAR__);
324 :             stan::math::fill(log_alpha, DUMMY_VAR__);
325 :
326 :             current_statement_begin__ = 34;
327 :             validate_non_negative_index("P", "2", 2);
328 :             validate_non_negative_index("P", "2", 2);
329 :             Eigen::Matrix<local_scalar_t__, Eigen::Dynamic, Eigen::Dynamic> P(2, 2);
330 :             stan::math::initialize(P, DUMMY_VAR__);
331 :             stan::math::fill(P, DUMMY_VAR__);
```

```
332 :
333 :             // transformed parameters block statements
334 :             current_statement_begin__ = 35;
335 :             stan::model::assign(P,
336 :                         stan::model::cons_list(stan::model::index_uni(1), stan::model::cons_list
337 :                         get_base1(p_remain, 1, "p_remain", 1),
338 :                         "assigning variable P");
339 :             current_statement_begin__ = 36;
340 :             stan::model::assign(P,
341 :                         stan::model::cons_list(stan::model::index_uni(1), stan::model::cons_list
342 :                         (1 - get_base1(p_remain, 1, "p_remain", 1)),
343 :                         "assigning variable P");
344 :             current_statement_begin__ = 37;
345 :             stan::model::assign(P,
346 :                         stan::model::cons_list(stan::model::index_uni(2), stan::model::cons_list
347 :                         (1 - get_base1(p_remain, 2, "p_remain", 1)),
348 :                         "assigning variable P");
349 :             current_statement_begin__ = 38;
350 :             stan::model::assign(P,
351 :                         stan::model::cons_list(stan::model::index_uni(2), stan::model::cons_list
352 :                         get_base1(p_remain, 2, "p_remain", 1),
353 :                         "assigning variable P");
354 :             current_statement_begin__ = 44;
355 :             stan::model::assign(beta1,
356 :                         stan::model::cons_list(stan::model::index_uni(1), stan::model::nil_index
357 :                         beta1_1,
358 :                         "assigning variable beta1");
359 :             current_statement_begin__ = 45;
360 :             stan::model::assign(beta1,
361 :                         stan::model::cons_list(stan::model::index_uni(2), stan::model::nil_index
362 :                         beta1_2,
363 :                         "assigning variable beta1");
364 :             current_statement_begin__ = 48;
365 :             stan::model::assign(sigma_t,
366 :                         stan::model::cons_list(stan::model::index_uni(1), stan::model::cons_list
367 :                         (get_base1(alpha0, 1, "alpha0", 1) / ((1 - get_base1(alpha1, 1, "alpha1"
368 :                         "assigning variable sigma_t");
369 :             current_statement_begin__ = 49;
370 :             stan::model::assign(sigma_t,
371 :                         stan::model::cons_list(stan::model::index_uni(1), stan::model::cons_list
372 :                         (get_base1(alpha0, 2, "alpha0", 1) / ((1 - get_base1(alpha1, 2, "alpha1"
373 :                         "assigning variable sigma_t");
374 :             current_statement_begin__ = 52;
375 :             for (int t = 2; t <= T; ++t) {
376 :
377 :                 current_statement_begin__ = 53;
378 :                 for (int i = 1; i <= 2; ++i) {
379 :
380 :                     current_statement_begin__ = 54;
381 :                     stan::model::assign(sigma_t,
382 :                                 stan::model::cons_list(stan::model::index_uni(t), stan::model::co
383 :                                 stan::math::sqrt(((get_base1(alpha0, i, "alpha0", 1) + (get_base
384 :                                 "assigning variable sigma_t");
385 :                 }
```

```
386 :             }
387 :             {
388 :             current_statement_begin__ = 68;
389 :             validate_non_negative_index("accumulator", "2", 2);
390 :             std::vector<local_scalar_t__  > accumulator(2, local_scalar_t__(DUMMY_VAR__));
391 :             stan::math::initialize(accumulator, DUMMY_VAR__);
392 :             stan::math::fill(accumulator, DUMMY_VAR__);
393 :
394 :
395 :             current_statement_begin__ = 72;
396 :             stan::model::assign(log_alpha,
397 :                         stan::model::cons_list(stan::model::index_uni(1), stan::model::cons_list
398 :                         (stan::math::log(0.5) + normal_log(get_base1(y, 1, "y", 1), 0, get_base1
399 :                         "assigning variable log_alpha");
400 :             current_statement_begin__ = 73;
401 :             stan::model::assign(log_alpha,
402 :                         stan::model::cons_list(stan::model::index_uni(1), stan::model::cons_list
403 :                         (stan::math::log(0.5) + normal_log(get_base1(y, 1, "y", 1), 0, get_base1
404 :                         "assigning variable log_alpha");
405 :             current_statement_begin__ = 75;
406 :             for (int t = 2; t <= T; ++t) {
407 :
408 :                 current_statement_begin__ = 76;
409 :                 for (int j = 1; j <= 2; ++j) {
410 :
411 :                     current_statement_begin__ = 77;
412 :                     for (int i = 1; i <= 2; ++i) {
413 :
414 :                         current_statement_begin__ = 78;
415 :                         stan::model::assign(accumulator,
416 :                                     stan::model::cons_list(stan::model::index_uni(i), stan::model
417 :                                     ((get_base1(get_base1(log_alpha, (t - 1), "log_alpha", 1), i
418 :                                     "assigning variable accumulator");
419 :                     }
420 :                     current_statement_begin__ = 83;
421 :                     stan::model::assign(log_alpha,
422 :                                 stan::model::cons_list(stan::model::index_uni(t), stan::model::co
423 :                                 log_sum_exp(accumulator),
424 :                                 "assigning variable log_alpha");
425 :                 }
426 :             }
427 :             }
428 :
429 :             // validate transformed parameters
430 :             const char* function__ = "validate transformed params";
431 :             (void) function__;  // dummy to suppress unused var warning
432 :
433 :             current_statement_begin__ = 25;
434 :             size_t beta1_k_0_max__ = 2;
435 :             for (size_t k_0__ = 0; k_0__ < beta1_k_0_max__; ++k_0__) {
436 :                 if (stan::math::is_uninitialized(beta1[k_0__])) {
437 :                     std::stringstream msg__;
438 :                     msg__ << "Undefined transformed parameter: beta1" << "[" << k_0__ << "]";
439 :                     stan::lang::rethrow_located(std::runtime_error(std::string("Error initializi
```

```
440 :                 }
441 :             }
442 :             size_t beta1_i_0_max__ = 2;
443 :             for (size_t i_0__ = 0; i_0__ < beta1_i_0_max__; ++i_0__) {
444 :                 check_greater_or_equal(function__, "beta1[i_0__]", beta1[i_0__], 0);
445 :             }
446 :
447 :             current_statement_begin__ = 28;
448 :             size_t sigma_t_k_0_max__ = T;
449 :             size_t sigma_t_j_1_max__ = 2;
450 :             for (size_t k_0__ = 0; k_0__ < sigma_t_k_0_max__; ++k_0__) {
451 :                 for (size_t j_1__ = 0; j_1__ < sigma_t_j_1_max__; ++j_1__) {
452 :                     if (stan::math::is_uninitialized(sigma_t[k_0__](j_1__))) {
453 :                         std::stringstream msg__;
454 :                         msg__ << "Undefined transformed parameter: sigma_t" << "[" << k_0__ << "
455 :                         stan::lang::rethrow_located(std::runtime_error(std::string("Error initial
456 :                     }
457 :                 }
458 :             }
459 :             current_statement_begin__ = 31;
460 :             size_t log_alpha_k_0_max__ = T;
461 :             size_t log_alpha_j_1_max__ = 2;
462 :             for (size_t k_0__ = 0; k_0__ < log_alpha_k_0_max__; ++k_0__) {
463 :                 for (size_t j_1__ = 0; j_1__ < log_alpha_j_1_max__; ++j_1__) {
464 :                     if (stan::math::is_uninitialized(log_alpha[k_0__](j_1__))) {
465 :                         std::stringstream msg__;
466 :                         msg__ << "Undefined transformed parameter: log_alpha" << "[" << k_0__ <<
467 :                         stan::lang::rethrow_located(std::runtime_error(std::string("Error initial
468 :                     }
469 :                 }
470 :             }
471 :             current_statement_begin__ = 34;
472 :             size_t P_j_1_max__ = 2;
473 :             size_t P_j_2_max__ = 2;
474 :             for (size_t j_1__ = 0; j_1__ < P_j_1_max__; ++j_1__) {
475 :                 for (size_t j_2__ = 0; j_2__ < P_j_2_max__; ++j_2__) {
476 :                     if (stan::math::is_uninitialized(P(j_1__, j_2__))) {
477 :                         std::stringstream msg__;
478 :                         msg__ << "Undefined transformed parameter: P" << "(" << j_1__ << ", " <<
479 :                         stan::lang::rethrow_located(std::runtime_error(std::string("Error initial
480 :                     }
481 :                 }
482 :             }
483 :
484 :             // model body
485 :
486 :             current_statement_begin__ = 93;
487 :             lp_accum__.add(normal_log<propto__>(alpha0, 0, 0.5));
488 :             current_statement_begin__ = 94;
489 :             lp_accum__.add(normal_log<propto__>(alpha1, 0, 1));
490 :             current_statement_begin__ = 95;
491 :             lp_accum__.add(normal_log<propto__>(beta1, 1, 1));
492 :             current_statement_begin__ = 98;
493 :             lp_accum__.add(beta_log<propto__>(p_remain, 3, 1));
```

```
494 :             current_statement_begin__ = 101;
495 :             lp_accum__.add(log_sum_exp(get_base1(log_alpha, T, "log_alpha", 1)));
496 :
497 :         } catch (const std::exception& e) {
498 :             stan::lang::rethrow_located(e, current_statement_begin__, prog_reader__());
499 :             // Next line prevents compiler griping about no return
500 :             throw std::runtime_error("*** IF YOU SEE THIS, PLEASE REPORT A BUG ***");
501 :         }
502 :
503 :         lp_accum__.add(lp__);
504 :         return lp_accum__.sum();
505 :
506 :     } // log_prob()
507 :
508 :     template <bool propto, bool jacobian, typename T_>
509 :     T_ log_prob(Eigen::Matrix<T_,Eigen::Dynamic,1>& params_r,
510 :                 std::ostream* pstream = 0) const {
511 :       std::vector<T_> vec_params_r;
512 :       vec_params_r.reserve(params_r.size());
513 :       for (int i = 0; i < params_r.size(); ++i)
514 :         vec_params_r.push_back(params_r(i));
515 :       std::vector<int> vec_params_i;
516 :       return log_prob<propto,jacobian,T_>(vec_params_r, vec_params_i, pstream);
517 :     }
518 :
519 :
520 :     void get_param_names(std::vector<std::string>& names__) const {
521 :         names__.resize(0);
522 :         names__.push_back("alpha0");
523 :         names__.push_back("alpha1");
524 :         names__.push_back("beta1_1");
525 :         names__.push_back("beta1_2");
526 :         names__.push_back("p_remain");
527 :         names__.push_back("beta1");
528 :         names__.push_back("sigma_t");
529 :         names__.push_back("log_alpha");
530 :         names__.push_back("P");
531 :         names__.push_back("alpha");
532 :         names__.push_back("zstar");
533 :         names__.push_back("logp_zstar");
534 :     }
535 :
536 :
537 :     void get_dims(std::vector<std::vector<size_t> >& dimss__) const {
538 :         dimss__.resize(0);
539 :         std::vector<size_t> dims__;
540 :         dims__.resize(0);
541 :         dims__.push_back(2);
542 :         dimss__.push_back(dims__);
543 :         dims__.resize(0);
544 :         dims__.push_back(2);
545 :         dimss__.push_back(dims__);
546 :         dims__.resize(0);
547 :         dimss__.push_back(dims__);
```

```
548 :         dims__.resize(0);
549 :         dimss__.push_back(dims__);
550 :         dims__.resize(0);
551 :         dims__.push_back(2);
552 :         dimss__.push_back(dims__);
553 :         dims__.resize(0);
554 :         dims__.push_back(2);
555 :         dimss__.push_back(dims__);
556 :         dims__.resize(0);
557 :         dims__.push_back(T);
558 :         dims__.push_back(2);
559 :         dimss__.push_back(dims__);
560 :         dims__.resize(0);
561 :         dims__.push_back(T);
562 :         dims__.push_back(2);
563 :         dimss__.push_back(dims__);
564 :         dims__.resize(0);
565 :         dims__.push_back(2);
566 :         dims__.push_back(2);
567 :         dimss__.push_back(dims__);
568 :         dims__.resize(0);
569 :         dims__.push_back(T);
570 :         dims__.push_back(2);
571 :         dimss__.push_back(dims__);
572 :         dims__.resize(0);
573 :         dims__.push_back(T);
574 :         dimss__.push_back(dims__);
575 :         dims__.resize(0);
576 :         dimss__.push_back(dims__);
577 :     }
578 :
579 :     template <typename RNG>
580 :     void write_array(RNG& base_rng__,
581 :                      std::vector<double>& params_r__,
582 :                      std::vector<int>& params_i__,
583 :                      std::vector<double>& vars__,
584 :                      bool include_tparams__ = true,
585 :                      bool include_gqs__ = true,
586 :                      std::ostream* pstream__ = 0) const {
587 :         typedef double local_scalar_t__;
588 :
589 :         vars__.resize(0);
590 :         stan::io::reader<local_scalar_t__> in__(params_r__, params_i__);
591 :         static const char* function__ = "model792915fa0784_hmm_garch_namespace::write_array";
592 :         (void) function__;  // dummy to suppress unused var warning
593 :
594 :         // read-transform, write parameters
595 :         Eigen::Matrix<double, Eigen::Dynamic, 1> alpha0 = in__.positive_ordered_constrain(2);
596 :         size_t alpha0_j_1_max__ = 2;
597 :         for (size_t j_1__ = 0; j_1__ < alpha0_j_1_max__; ++j_1__) {
598 :             vars__.push_back(alpha0(j_1__));
599 :         }
600 :
601 :         std::vector<double> alpha1;
```

```
602 :         size_t alpha1_d_0_max__ = 2;
603 :         alpha1.reserve(alpha1_d_0_max__);
604 :         for (size_t d_0__ = 0; d_0__ < alpha1_d_0_max__; ++d_0__) {
605 :             alpha1.push_back(in__.scalar_lub_constrain(0, 1));
606 :         }
607 :         size_t alpha1_k_0_max__ = 2;
608 :         for (size_t k_0__ = 0; k_0__ < alpha1_k_0_max__; ++k_0__) {
609 :             vars__.push_back(alpha1[k_0__]);
610 :         }
611 :
612 :         double beta1_1 = in__.scalar_lub_constrain(0, (1 - get_base1(alpha1, 1, "alpha1", 1)));
613 :         vars__.push_back(beta1_1);
614 :
615 :         double beta1_2 = in__.scalar_lub_constrain(0, (1 - get_base1(alpha1, 2, "alpha1", 1)));
616 :         vars__.push_back(beta1_2);
617 :
618 :         std::vector<double> p_remain;
619 :         size_t p_remain_d_0_max__ = 2;
620 :         p_remain.reserve(p_remain_d_0_max__);
621 :         for (size_t d_0__ = 0; d_0__ < p_remain_d_0_max__; ++d_0__) {
622 :             p_remain.push_back(in__.scalar_lub_constrain(0, 1));
623 :         }
624 :         size_t p_remain_k_0_max__ = 2;
625 :         for (size_t k_0__ = 0; k_0__ < p_remain_k_0_max__; ++k_0__) {
626 :             vars__.push_back(p_remain[k_0__]);
627 :         }
628 :
629 :         double lp__ = 0.0;
630 :         (void) lp__;  // dummy to suppress unused var warning
631 :         stan::math::accumulator<double> lp_accum__;
632 :
633 :         local_scalar_t__ DUMMY_VAR__(std::numeric_limits<double>::quiet_NaN());
634 :         (void) DUMMY_VAR__;  // suppress unused var warning
635 :
636 :         if (!include_tparams__ && !include_gqs__) return;
637 :
638 :         try {
639 :             // declare and define transformed parameters
640 :             current_statement_begin__ = 25;
641 :             validate_non_negative_index("beta1", "2", 2);
642 :             std::vector<double> beta1(2, double(0));
643 :             stan::math::initialize(beta1, DUMMY_VAR__);
644 :             stan::math::fill(beta1, DUMMY_VAR__);
645 :
646 :             current_statement_begin__ = 28;
647 :             validate_non_negative_index("sigma_t", "2", 2);
648 :             validate_non_negative_index("sigma_t", "T", T);
649 :             std::vector<Eigen::Matrix<double, Eigen::Dynamic, 1> > sigma_t(T, Eigen::Matrix<doubl
650 :             stan::math::initialize(sigma_t, DUMMY_VAR__);
651 :             stan::math::fill(sigma_t, DUMMY_VAR__);
652 :
653 :             current_statement_begin__ = 31;
654 :             validate_non_negative_index("log_alpha", "2", 2);
655 :             validate_non_negative_index("log_alpha", "T", T);
```

```
656 :             std::vector<Eigen::Matrix<double, Eigen::Dynamic, 1> > log_alpha(T, Eigen::Matrix<dou
657 :             stan::math::initialize(log_alpha, DUMMY_VAR__);
658 :             stan::math::fill(log_alpha, DUMMY_VAR__);
659 :
660 :             current_statement_begin__ = 34;
661 :             validate_non_negative_index("P", "2", 2);
662 :             validate_non_negative_index("P", "2", 2);
663 :             Eigen::Matrix<double, Eigen::Dynamic, Eigen::Dynamic> P(2, 2);
664 :             stan::math::initialize(P, DUMMY_VAR__);
665 :             stan::math::fill(P, DUMMY_VAR__);
666 :
667 :             // do transformed parameters statements
668 :             current_statement_begin__ = 35;
669 :             stan::model::assign(P,
670 :                         stan::model::cons_list(stan::model::index_uni(1), stan::model::cons_list
671 :                         get_base1(p_remain, 1, "p_remain", 1),
672 :                         "assigning variable P");
673 :             current_statement_begin__ = 36;
674 :             stan::model::assign(P,
675 :                         stan::model::cons_list(stan::model::index_uni(1), stan::model::cons_list
676 :                         (1 - get_base1(p_remain, 1, "p_remain", 1)),
677 :                         "assigning variable P");
678 :             current_statement_begin__ = 37;
679 :             stan::model::assign(P,
680 :                         stan::model::cons_list(stan::model::index_uni(2), stan::model::cons_list
681 :                         (1 - get_base1(p_remain, 2, "p_remain", 1)),
682 :                         "assigning variable P");
683 :             current_statement_begin__ = 38;
684 :             stan::model::assign(P,
685 :                         stan::model::cons_list(stan::model::index_uni(2), stan::model::cons_list
686 :                         get_base1(p_remain, 2, "p_remain", 1),
687 :                         "assigning variable P");
688 :             current_statement_begin__ = 44;
689 :             stan::model::assign(beta1,
690 :                         stan::model::cons_list(stan::model::index_uni(1), stan::model::nil_index
691 :                         beta1_1,
692 :                         "assigning variable beta1");
693 :             current_statement_begin__ = 45;
694 :             stan::model::assign(beta1,
695 :                         stan::model::cons_list(stan::model::index_uni(2), stan::model::nil_index
696 :                         beta1_2,
697 :                         "assigning variable beta1");
698 :             current_statement_begin__ = 48;
699 :             stan::model::assign(sigma_t,
700 :                         stan::model::cons_list(stan::model::index_uni(1), stan::model::cons_list
701 :                         (get_base1(alpha0, 1, "alpha0", 1) / ((1 - get_base1(alpha1, 1, "alpha1"
702 :                         "assigning variable sigma_t");
703 :             current_statement_begin__ = 49;
704 :             stan::model::assign(sigma_t,
705 :                         stan::model::cons_list(stan::model::index_uni(1), stan::model::cons_list
706 :                         (get_base1(alpha0, 2, "alpha0", 1) / ((1 - get_base1(alpha1, 2, "alpha1"
707 :                         "assigning variable sigma_t");
708 :             current_statement_begin__ = 52;
709 :             for (int t = 2; t <= T; ++t) {
```

```
710 :
711 :                 current_statement_begin__ = 53;
712 :                 for (int i = 1; i <= 2; ++i) {
713 :
714 :                     current_statement_begin__ = 54;
715 :                     stan::model::assign(sigma_t,
716 :                                 stan::model::cons_list(stan::model::index_uni(t), stan::model::c
717 :                                 stan::math::sqrt(((get_base1(alpha0, i, "alpha0", 1) + (get_base
718 :                                 "assigning variable sigma_t");
719 :                 }
720 :             }
721 :             {
722 :             current_statement_begin__ = 68;
723 :             validate_non_negative_index("accumulator", "2", 2);
724 :             std::vector<local_scalar_t__  > accumulator(2, local_scalar_t__(DUMMY_VAR__));
725 :             stan::math::initialize(accumulator, DUMMY_VAR__);
726 :             stan::math::fill(accumulator, DUMMY_VAR__);
727 :
728 :
729 :             current_statement_begin__ = 72;
730 :             stan::model::assign(log_alpha,
731 :                         stan::model::cons_list(stan::model::index_uni(1), stan::model::cons_list
732 :                         (stan::math::log(0.5) + normal_log(get_base1(y, 1, "y", 1), 0, get_base1
733 :                         "assigning variable log_alpha");
734 :             current_statement_begin__ = 73;
735 :             stan::model::assign(log_alpha,
736 :                         stan::model::cons_list(stan::model::index_uni(1), stan::model::cons_list
737 :                         (stan::math::log(0.5) + normal_log(get_base1(y, 1, "y", 1), 0, get_base1
738 :                         "assigning variable log_alpha");
739 :             current_statement_begin__ = 75;
740 :             for (int t = 2; t <= T; ++t) {
741 :
742 :                 current_statement_begin__ = 76;
743 :                 for (int j = 1; j <= 2; ++j) {
744 :
745 :                     current_statement_begin__ = 77;
746 :                     for (int i = 1; i <= 2; ++i) {
747 :
748 :                         current_statement_begin__ = 78;
749 :                         stan::model::assign(accumulator,
750 :                                     stan::model::cons_list(stan::model::index_uni(i), stan::model
751 :                                     ((get_base1(get_base1(log_alpha, (t - 1), "log_alpha", 1), i
752 :                                     "assigning variable accumulator");
753 :                     }
754 :                     current_statement_begin__ = 83;
755 :                     stan::model::assign(log_alpha,
756 :                                 stan::model::cons_list(stan::model::index_uni(t), stan::model::c
757 :                                 log_sum_exp(accumulator),
758 :                                 "assigning variable log_alpha");
759 :                 }
760 :             }
761 :             }
762 :
763 :             if (!include_gqs__ && !include_tparams__) return;
```

```
764 :             // validate transformed parameters
765 :             const char* function__ = "validate transformed params";
766 :             (void) function__;  // dummy to suppress unused var warning
767 :
768 :             current_statement_begin__ = 25;
769 :             size_t beta1_i_0_max__ = 2;
770 :             for (size_t i_0__ = 0; i_0__ < beta1_i_0_max__; ++i_0__) {
771 :                 check_greater_or_equal(function__, "beta1[i_0__]", beta1[i_0__], 0);
772 :             }
773 :
774 :             // write transformed parameters
775 :             if (include_tparams__) {
776 :                 size_t beta1_k_0_max__ = 2;
777 :                 for (size_t k_0__ = 0; k_0__ < beta1_k_0_max__; ++k_0__) {
778 :                     vars__.push_back(beta1[k_0__]);
779 :                 }
780 :                 size_t sigma_t_j_1_max__ = 2;
781 :                 size_t sigma_t_k_0_max__ = T;
782 :                 for (size_t j_1__ = 0; j_1__ < sigma_t_j_1_max__; ++j_1__) {
783 :                     for (size_t k_0__ = 0; k_0__ < sigma_t_k_0_max__; ++k_0__) {
784 :                         vars__.push_back(sigma_t[k_0__](j_1__));
785 :                     }
786 :                 }
787 :                 size_t log_alpha_j_1_max__ = 2;
788 :                 size_t log_alpha_k_0_max__ = T;
789 :                 for (size_t j_1__ = 0; j_1__ < log_alpha_j_1_max__; ++j_1__) {
790 :                     for (size_t k_0__ = 0; k_0__ < log_alpha_k_0_max__; ++k_0__) {
791 :                         vars__.push_back(log_alpha[k_0__](j_1__));
792 :                     }
793 :                 }
794 :                 size_t P_j_2_max__ = 2;
795 :                 size_t P_j_1_max__ = 2;
796 :                 for (size_t j_2__ = 0; j_2__ < P_j_2_max__; ++j_2__) {
797 :                     for (size_t j_1__ = 0; j_1__ < P_j_1_max__; ++j_1__) {
798 :                         vars__.push_back(P(j_1__, j_2__));
799 :                     }
800 :                 }
801 :             }
802 :             if (!include_gqs__) return;
803 :             // declare and define generated quantities
804 :             current_statement_begin__ = 105;
805 :             validate_non_negative_index("alpha", "2", 2);
806 :             validate_non_negative_index("alpha", "T", T);
807 :             std::vector<Eigen::Matrix<double, Eigen::Dynamic, 1> > alpha(T, Eigen::Matrix<double
808 :             stan::math::initialize(alpha, DUMMY_VAR__);
809 :             stan::math::fill(alpha, DUMMY_VAR__);
810 :
811 :             current_statement_begin__ = 107;
812 :             validate_non_negative_index("zstar", "T", T);
813 :             std::vector<int> zstar(T, int(0));
814 :             stan::math::fill(zstar, std::numeric_limits<int>::min());
815 :
816 :             current_statement_begin__ = 108;
817 :             double logp_zstar;
```

```
818 :             (void) logp_zstar;  // dummy to suppress unused var warning
819 :             stan::math::initialize(logp_zstar, DUMMY_VAR__);
820 :             stan::math::fill(logp_zstar, DUMMY_VAR__);
821 :
822 :             // generated quantities statements
823 :             current_statement_begin__ = 111;
824 :             for (int t = 1; t <= T; ++t) {
825 :
826 :                 current_statement_begin__ = 112;
827 :                 stan::model::assign(alpha,
828 :                             stan::model::cons_list(stan::model::index_uni(t), stan::model::nil_i
829 :                             softmax(get_base1(log_alpha, t, "log_alpha", 1)),
830 :                             "assigning variable alpha");
831 :             }
832 :             {
833 :             current_statement_begin__ = 118;
834 :             validate_non_negative_index("bpointer", "T", T);
835 :             validate_non_negative_index("bpointer", "2", 2);
836 :             std::vector<std::vector<int  >  > bpointer(T, std::vector<int>(2, int(0)));
837 :             stan::math::fill(bpointer, std::numeric_limits<int>::min());
838 :
839 :             current_statement_begin__ = 119;
840 :             validate_non_negative_index("delta", "T", T);
841 :             validate_non_negative_index("delta", "2", 2);
842 :             std::vector<std::vector<local_scalar_t__  >  > delta(T, std::vector<local_scalar_t__
843 :             stan::math::initialize(delta, DUMMY_VAR__);
844 :             stan::math::fill(delta, DUMMY_VAR__);
845 :
846 :
847 :             current_statement_begin__ = 122;
848 :             for (int j = 1; j <= 2; ++j) {
849 :                 current_statement_begin__ = 123;
850 :                 stan::model::assign(delta,
851 :                             stan::model::cons_list(stan::model::index_uni(1), stan::model::cons_
852 :                             normal_log(get_base1(y, 1, "y", 1), 0, get_base1(get_base1(sigma_t,
853 :                             "assigning variable delta");
854 :             }
855 :             current_statement_begin__ = 124;
856 :             stan::model::assign(delta,
857 :                         stan::model::cons_list(stan::model::index_uni(1), stan::model::cons_list
858 :                         normal_log(get_base1(y, 1, "y", 1), 0, get_base1(get_base1(sigma_t, 1, "
859 :                         "assigning variable delta");
860 :             current_statement_begin__ = 126;
861 :             for (int t = 2; t <= T; ++t) {
862 :
863 :                 current_statement_begin__ = 127;
864 :                 for (int j = 1; j <= 2; ++j) {
865 :
866 :                     current_statement_begin__ = 128;
867 :                     stan::model::assign(delta,
868 :                                 stan::model::cons_list(stan::model::index_uni(t), stan::model::c
869 :                                 stan::math::negative_infinity(),
870 :                                 "assigning variable delta");
871 :                     current_statement_begin__ = 129;
```

```
872 :                     for (int i = 1; i <= 2; ++i) {
873 :                         {
874 :                         current_statement_begin__ = 130;
875 :                         local_scalar_t__ logp(DUMMY_VAR__);
876 :                         (void) logp;  // dummy to suppress unused var warning
877 :                         stan::math::initialize(logp, DUMMY_VAR__);
878 :                         stan::math::fill(logp, DUMMY_VAR__);
879 :
880 :
881 :                         current_statement_begin__ = 131;
882 :                         stan::math::assign(logp, ((get_base1(get_base1(delta, (t - 1), "delta",
883 :                         current_statement_begin__ = 132;
884 :                         if (as_bool(logical_gt(logp, get_base1(get_base1(delta, t, "delta", 1),
885 :
886 :                             current_statement_begin__ = 133;
887 :                             stan::model::assign(bpointer,
888 :                                         stan::model::cons_list(stan::model::index_uni(t), stan::
889 :                                         i,
890 :                                         "assigning variable bpointer");
891 :                             current_statement_begin__ = 134;
892 :                             stan::model::assign(delta,
893 :                                         stan::model::cons_list(stan::model::index_uni(t), stan::
894 :                                         logp,
895 :                                         "assigning variable delta");
896 :                         }
897 :                         }
898 :                     }
899 :                 }
900 :             }
901 :             current_statement_begin__ = 140;
902 :             stan::math::assign(logp_zstar, max(get_base1(delta, T, "delta", 1)));
903 :             current_statement_begin__ = 142;
904 :             for (int j = 1; j <= 2; ++j) {
905 :                 current_statement_begin__ = 143;
906 :                 if (as_bool(logical_eq(get_base1(get_base1(delta, T, "delta", 1), j, "delta", 2)
907 :                     current_statement_begin__ = 144;
908 :                     stan::model::assign(zstar,
909 :                                 stan::model::cons_list(stan::model::index_uni(T), stan::model::ni
910 :                                 j,
911 :                                 "assigning variable zstar");
912 :                 }
913 :             }
914 :             current_statement_begin__ = 146;
915 :             for (int t = 1; t <= (T - 1); ++t) {
916 :
917 :                 current_statement_begin__ = 147;
918 :                 stan::model::assign(zstar,
919 :                             stan::model::cons_list(stan::model::index_uni((T - t)), stan::model:
920 :                             get_base1(get_base1(bpointer, ((T - t) + 1), "bpointer", 1), get_base
921 :                             "assigning variable zstar");
922 :             }
923 :             }
924 :
925 :             // validate, write generated quantities
```

```
926 :             current_statement_begin__ = 105;
927 :             size_t alpha_j_1_max__ = 2;
928 :             size_t alpha_k_0_max__ = T;
929 :             for (size_t j_1__ = 0; j_1__ < alpha_j_1_max__; ++j_1__) {
930 :                 for (size_t k_0__ = 0; k_0__ < alpha_k_0_max__; ++k_0__) {
931 :                     vars__.push_back(alpha[k_0__](j_1__));
932 :                 }
933 :             }
934 :
935 :             current_statement_begin__ = 107;
936 :             size_t zstar_i_0_max__ = T;
937 :             for (size_t i_0__ = 0; i_0__ < zstar_i_0_max__; ++i_0__) {
938 :                 check_greater_or_equal(function__, "zstar[i_0__]", zstar[i_0__], 1);
939 :                 check_less_or_equal(function__, "zstar[i_0__]", zstar[i_0__], 2);
940 :             }
941 :
942 :             size_t zstar_k_0_max__ = T;
943 :             for (size_t k_0__ = 0; k_0__ < zstar_k_0_max__; ++k_0__) {
944 :                 vars__.push_back(zstar[k_0__]);
945 :             }
946 :
947 :             current_statement_begin__ = 108;
948 :             vars__.push_back(logp_zstar);
949 :
950 :         } catch (const std::exception& e) {
951 :             stan::lang::rethrow_located(e, current_statement_begin__, prog_reader__());
952 :             // Next line prevents compiler griping about no return
953 :             throw std::runtime_error("*** IF YOU SEE THIS, PLEASE REPORT A BUG ***");
954 :         }
955 :     }
956 :
957 :     template <typename RNG>
958 :     void write_array(RNG& base_rng,
959 :                      Eigen::Matrix<double,Eigen::Dynamic,1>& params_r,
960 :                      Eigen::Matrix<double,Eigen::Dynamic,1>& vars,
961 :                      bool include_tparams = true,
962 :                      bool include_gqs = true,
963 :                      std::ostream* pstream = 0) const {
964 :       std::vector<double> params_r_vec(params_r.size());
965 :       for (int i = 0; i < params_r.size(); ++i)
966 :         params_r_vec[i] = params_r(i);
967 :       std::vector<double> vars_vec;
968 :       std::vector<int> params_i_vec;
969 :       write_array(base_rng, params_r_vec, params_i_vec, vars_vec, include_tparams, include_gqs, p
970 :       vars.resize(vars_vec.size());
971 :       for (int i = 0; i < vars.size(); ++i)
972 :         vars(i) = vars_vec[i];
973 :     }
974 :
975 :     std::string model_name() const {
976 :         return "model792915fa0784_hmm_garch";
977 :     }
978 :
979 :
```

```
 980 :     void constrained_param_names(std::vector<std::string>& param_names__,
 981 :                                  bool include_tparams__ = true,
 982 :                                  bool include_gqs__ = true) const {
 983 :         std::stringstream param_name_stream__;
 984 :         size_t alpha0_j_1_max__ = 2;
 985 :         for (size_t j_1__ = 0; j_1__ < alpha0_j_1_max__; ++j_1__) {
 986 :             param_name_stream__.str(std::string());
 987 :             param_name_stream__ << "alpha0" << '.' << j_1__ + 1;
 988 :             param_names__.push_back(param_name_stream__.str());
 989 :         }
 990 :         size_t alpha1_k_0_max__ = 2;
 991 :         for (size_t k_0__ = 0; k_0__ < alpha1_k_0_max__; ++k_0__) {
 992 :             param_name_stream__.str(std::string());
 993 :             param_name_stream__ << "alpha1" << '.' << k_0__ + 1;
 994 :             param_names__.push_back(param_name_stream__.str());
 995 :         }
 996 :         param_name_stream__.str(std::string());
 997 :         param_name_stream__ << "beta1_1";
 998 :         param_names__.push_back(param_name_stream__.str());
 999 :         param_name_stream__.str(std::string());
1000 :         param_name_stream__ << "beta1_2";
1001 :         param_names__.push_back(param_name_stream__.str());
1002 :         size_t p_remain_k_0_max__ = 2;
1003 :         for (size_t k_0__ = 0; k_0__ < p_remain_k_0_max__; ++k_0__) {
1004 :             param_name_stream__.str(std::string());
1005 :             param_name_stream__ << "p_remain" << '.' << k_0__ + 1;
1006 :             param_names__.push_back(param_name_stream__.str());
1007 :         }
1008 :
1009 :         if (!include_gqs__ && !include_tparams__) return;
1010 :
1011 :         if (include_tparams__) {
1012 :             size_t beta1_k_0_max__ = 2;
1013 :             for (size_t k_0__ = 0; k_0__ < beta1_k_0_max__; ++k_0__) {
1014 :                 param_name_stream__.str(std::string());
1015 :                 param_name_stream__ << "beta1" << '.' << k_0__ + 1;
1016 :                 param_names__.push_back(param_name_stream__.str());
1017 :             }
1018 :             size_t sigma_t_j_1_max__ = 2;
1019 :             size_t sigma_t_k_0_max__ = T;
1020 :             for (size_t j_1__ = 0; j_1__ < sigma_t_j_1_max__; ++j_1__) {
1021 :                 for (size_t k_0__ = 0; k_0__ < sigma_t_k_0_max__; ++k_0__) {
1022 :                     param_name_stream__.str(std::string());
1023 :                     param_name_stream__ << "sigma_t" << '.' << k_0__ + 1 << '.' << j_1__ + 1;
1024 :                     param_names__.push_back(param_name_stream__.str());
1025 :                 }
1026 :             }
1027 :             size_t log_alpha_j_1_max__ = 2;
1028 :             size_t log_alpha_k_0_max__ = T;
1029 :             for (size_t j_1__ = 0; j_1__ < log_alpha_j_1_max__; ++j_1__) {
1030 :                 for (size_t k_0__ = 0; k_0__ < log_alpha_k_0_max__; ++k_0__) {
1031 :                     param_name_stream__.str(std::string());
1032 :                     param_name_stream__ << "log_alpha" << '.' << k_0__ + 1 << '.' << j_1__ + 1;
1033 :                     param_names__.push_back(param_name_stream__.str());
```

```
1034 :                 }
1035 :             }
1036 :             size_t P_j_2_max__ = 2;
1037 :             size_t P_j_1_max__ = 2;
1038 :             for (size_t j_2__ = 0; j_2__ < P_j_2_max__; ++j_2__) {
1039 :                 for (size_t j_1__ = 0; j_1__ < P_j_1_max__; ++j_1__) {
1040 :                     param_name_stream__.str(std::string());
1041 :                     param_name_stream__ << "P" << '.' << j_1__ + 1 << '.' << j_2__ + 1;
1042 :                     param_names__.push_back(param_name_stream__.str());
1043 :                 }
1044 :             }
1045 :         }
1046 :
1047 :         if (!include_gqs__) return;
1048 :         size_t alpha_j_1_max__ = 2;
1049 :         size_t alpha_k_0_max__ = T;
1050 :         for (size_t j_1__ = 0; j_1__ < alpha_j_1_max__; ++j_1__) {
1051 :             for (size_t k_0__ = 0; k_0__ < alpha_k_0_max__; ++k_0__) {
1052 :                 param_name_stream__.str(std::string());
1053 :                 param_name_stream__ << "alpha" << '.' << k_0__ + 1 << '.' << j_1__ + 1;
1054 :                 param_names__.push_back(param_name_stream__.str());
1055 :             }
1056 :         }
1057 :         size_t zstar_k_0_max__ = T;
1058 :         for (size_t k_0__ = 0; k_0__ < zstar_k_0_max__; ++k_0__) {
1059 :             param_name_stream__.str(std::string());
1060 :             param_name_stream__ << "zstar" << '.' << k_0__ + 1;
1061 :             param_names__.push_back(param_name_stream__.str());
1062 :         }
1063 :         param_name_stream__.str(std::string());
1064 :         param_name_stream__ << "logp_zstar";
1065 :         param_names__.push_back(param_name_stream__.str());
1066 :     }
1067 :
1068 :
1069 :     void unconstrained_param_names(std::vector<std::string>& param_names__,
1070 :                                    bool include_tparams__ = true,
1071 :                                    bool include_gqs__ = true) const {
1072 :         std::stringstream param_name_stream__;
1073 :         size_t alpha0_j_1_max__ = 2;
1074 :         for (size_t j_1__ = 0; j_1__ < alpha0_j_1_max__; ++j_1__) {
1075 :             param_name_stream__.str(std::string());
1076 :             param_name_stream__ << "alpha0" << '.' << j_1__ + 1;
1077 :             param_names__.push_back(param_name_stream__.str());
1078 :         }
1079 :         size_t alpha1_k_0_max__ = 2;
1080 :         for (size_t k_0__ = 0; k_0__ < alpha1_k_0_max__; ++k_0__) {
1081 :             param_name_stream__.str(std::string());
1082 :             param_name_stream__ << "alpha1" << '.' << k_0__ + 1;
1083 :             param_names__.push_back(param_name_stream__.str());
1084 :         }
1085 :         param_name_stream__.str(std::string());
1086 :         param_name_stream__ << "beta1_1";
1087 :         param_names__.push_back(param_name_stream__.str());
```

```
1088 :         param_name_stream__.str(std::string());
1089 :         param_name_stream__ << "beta1_2";
1090 :         param_names__.push_back(param_name_stream__.str());
1091 :         size_t p_remain_k_0_max__ = 2;
1092 :         for (size_t k_0__ = 0; k_0__ < p_remain_k_0_max__; ++k_0__) {
1093 :             param_name_stream__.str(std::string());
1094 :             param_name_stream__ << "p_remain" << '.' << k_0__ + 1;
1095 :             param_names__.push_back(param_name_stream__.str());
1096 :         }
1097 :
1098 :         if (!include_gqs__ && !include_tparams__) return;
1099 :
1100 :         if (include_tparams__) {
1101 :             size_t beta1_k_0_max__ = 2;
1102 :             for (size_t k_0__ = 0; k_0__ < beta1_k_0_max__; ++k_0__) {
1103 :                 param_name_stream__.str(std::string());
1104 :                 param_name_stream__ << "beta1" << '.' << k_0__ + 1;
1105 :                 param_names__.push_back(param_name_stream__.str());
1106 :             }
1107 :             size_t sigma_t_j_1_max__ = 2;
1108 :             size_t sigma_t_k_0_max__ = T;
1109 :             for (size_t j_1__ = 0; j_1__ < sigma_t_j_1_max__; ++j_1__) {
1110 :                 for (size_t k_0__ = 0; k_0__ < sigma_t_k_0_max__; ++k_0__) {
1111 :                     param_name_stream__.str(std::string());
1112 :                     param_name_stream__ << "sigma_t" << '.' << k_0__ + 1 << '.' << j_1__ + 1;
1113 :                     param_names__.push_back(param_name_stream__.str());
1114 :                 }
1115 :             }
1116 :             size_t log_alpha_j_1_max__ = 2;
1117 :             size_t log_alpha_k_0_max__ = T;
1118 :             for (size_t j_1__ = 0; j_1__ < log_alpha_j_1_max__; ++j_1__) {
1119 :                 for (size_t k_0__ = 0; k_0__ < log_alpha_k_0_max__; ++k_0__) {
1120 :                     param_name_stream__.str(std::string());
1121 :                     param_name_stream__ << "log_alpha" << '.' << k_0__ + 1 << '.' << j_1__ + 1;
1122 :                     param_names__.push_back(param_name_stream__.str());
1123 :                 }
1124 :             }
1125 :             size_t P_j_2_max__ = 2;
1126 :             size_t P_j_1_max__ = 2;
1127 :             for (size_t j_2__ = 0; j_2__ < P_j_2_max__; ++j_2__) {
1128 :                 for (size_t j_1__ = 0; j_1__ < P_j_1_max__; ++j_1__) {
1129 :                     param_name_stream__.str(std::string());
1130 :                     param_name_stream__ << "P" << '.' << j_1__ + 1 << '.' << j_2__ + 1;
1131 :                     param_names__.push_back(param_name_stream__.str());
1132 :                 }
1133 :             }
1134 :         }
1135 :
1136 :         if (!include_gqs__) return;
1137 :         size_t alpha_j_1_max__ = 2;
1138 :         size_t alpha_k_0_max__ = T;
1139 :         for (size_t j_1__ = 0; j_1__ < alpha_j_1_max__; ++j_1__) {
1140 :             for (size_t k_0__ = 0; k_0__ < alpha_k_0_max__; ++k_0__) {
1141 :                 param_name_stream__.str(std::string());
```

```
1142 :                 param_name_stream__ << "alpha" << '.' << k_0__ + 1 << '.' << j_1__ + 1;
1143 :                 param_names__.push_back(param_name_stream__.str());
1144 :             }
1145 :         }
1146 :         size_t zstar_k_0_max__ = T;
1147 :         for (size_t k_0__ = 0; k_0__ < zstar_k_0_max__; ++k_0__) {
1148 :             param_name_stream__.str(std::string());
1149 :             param_name_stream__ << "zstar" << '.' << k_0__ + 1;
1150 :             param_names__.push_back(param_name_stream__.str());
1151 :         }
1152 :         param_name_stream__.str(std::string());
1153 :         param_name_stream__ << "logp_zstar";
1154 :         param_names__.push_back(param_name_stream__.str());
1155 :     }
1156 :
1157 : }; // model
1158 :
1159 : }  // namespace
1160 :
1161 : typedef model792915fa0784_hmm_garch_namespace::model792915fa0784_hmm_garch stan_model;
1162 :
1163 : #ifndef USING_R
1164 :
1165 : stan::model::model_base& new_model(
1166 :         stan::io::var_context& data_context,
1167 :         unsigned int seed,
1168 :         std::ostream* msg_stream) {
1169 :   stan_model* m = new stan_model(data_context, seed, msg_stream);
1170 :   return *m;
1171 : }
1172 :
1173 : #endif
1174 :
1175 :
1176 :
1177 : #include <rstan_next/stan_fit.hpp>
1178 :
1179 : struct stan_model_holder {
1180 :     stan_model_holder(rstan::io::rlist_ref_var_context rcontext,
1181 :                       unsigned int random_seed)
1182 :     : rcontext_(rcontext), random_seed_(random_seed)
1183 :      {
1184 :      }
1185 :
1186 :    //stan::math::ChainableStack ad_stack;
1187 :    rstan::io::rlist_ref_var_context rcontext_;
1188 :    unsigned int random_seed_;
1189 : };
1190 :
1191 : Rcpp::XPtr<stan::model::model_base> model_ptr(stan_model_holder* smh) {
1192 :   Rcpp::XPtr<stan::model::model_base> model_instance(new stan_model(smh->rcontext_, smh->random_s
1193 :   return model_instance;
1194 : }
1195 :
```

```
1196 : Rcpp::XPtr<rstan::stan_fit_base> fit_ptr(stan_model_holder* smh) {
1197 :   return Rcpp::XPtr<rstan::stan_fit_base>(new rstan::stan_fit(model_ptr(smh), smh->random_seed_)
1198 : }
1199 :
1200 : std::string model_name(stan_model_holder* smh) {
1201 :   return model_ptr(smh).get()->model_name();
1202 : }
1203 :
1204 : RCPP_MODULE(stan_fit4model792915fa0784_hmm_garch_mod){
1205 :   Rcpp::class_<stan_model_holder>("stan_fit4model792915fa0784_hmm_garch")
1206 :   .constructor<rstan::io::rlist_ref_var_context, unsigned int>()
1207 :   .method("model_ptr", &model_ptr)
1208 :   .method("fit_ptr", &fit_ptr)
1209 :   .method("model_name", &model_name)
1210 :   ;
1211 : }
1212 :
1213 :
1214 : // declarations
1215 : extern "C" {
1216 : SEXP file7929e7a6d42( ) ;
1217 : }
1218 :
1219 : // definition
1220 : SEXP file7929e7a6d42() {
1221 :  return Rcpp::wrap("hmm_garch");
1222 : }

CHECKING DATA AND PREPROCESSING FOR MODEL 'hmm_garch' NOW.

COMPILING MODEL 'hmm_garch' NOW.

STARTING SAMPLER FOR MODEL 'hmm_garch' NOW.

SAMPLING FOR MODEL 'hmm_garch' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 0.001647 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 16.47 seconds.
Chain 1: Adjust your expectations accordingly!
Chain 1:
Chain 1:
Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
Chain 1:
```

```
Chain 1:  Elapsed Time: 12.1201 seconds (Warm-up)
Chain 1:                10.1016 seconds (Sampling)
Chain 1:                22.2217 seconds (Total)
Chain 1:
```

Resultados

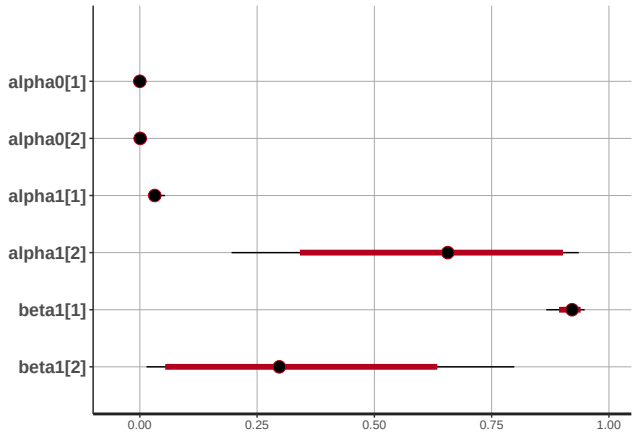
```
round(summary(fit, pars=c("alpha0","alpha1","beta1","P"))$summary,3)
```

```
           mean se_mean    sd  2.5%   25%   50%   75% 97.5%   n_eff  Rhat
alpha0[1] 0.000   0.000 0.000 0.000 0.000 0.000 0.000 0.000 190.382 1.004
alpha0[2] 0.001   0.000 0.000 0.000 0.001 0.001 0.001 0.002 206.553 1.002
alpha1[1] 0.033   0.000 0.009 0.019 0.027 0.032 0.037 0.053 349.877 0.999
alpha1[2] 0.638   0.015 0.212 0.196 0.477 0.656 0.827 0.936 209.112 0.999
beta1[1]  0.918   0.001 0.021 0.866 0.908 0.922 0.932 0.948 205.122 1.002
beta1[2]  0.329   0.016 0.223 0.014 0.134 0.297 0.495 0.798 202.070 1.000
P[1,1]    0.937   0.001 0.016 0.903 0.927 0.940 0.949 0.963 388.598 1.002
P[1,2]    0.063   0.001 0.016 0.037 0.051 0.060 0.073 0.097 388.598 1.002
P[2,1]    0.543   0.004 0.093 0.363 0.486 0.535 0.604 0.740 479.927 1.004
P[2,2]    0.457   0.004 0.093 0.260 0.396 0.465 0.514 0.637 479.927 1.004
```

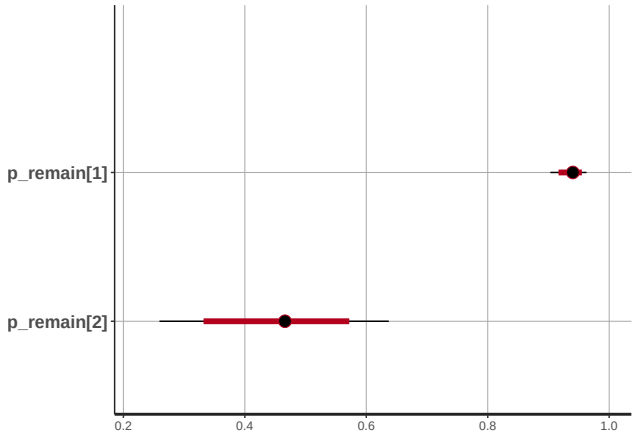
```
round(summary(fit, pars=c("alpha0","alpha1","beta1","P"))$c_summary,3)
```

```
, , chains = chain:1

           stats
parameter    mean    sd  2.5%   25%   50%   75% 97.5%
  alpha0[1] 0.000 0.000 0.000 0.000 0.000 0.000 0.000
  alpha0[2] 0.001 0.000 0.000 0.001 0.001 0.001 0.002
  alpha1[1] 0.033 0.009 0.019 0.027 0.032 0.037 0.053
  alpha1[2] 0.638 0.212 0.196 0.477 0.656 0.827 0.936
  beta1[1]  0.918 0.021 0.866 0.908 0.922 0.932 0.948
  beta1[2]  0.329 0.223 0.014 0.134 0.297 0.495 0.798
  P[1,1]    0.937 0.016 0.903 0.927 0.940 0.949 0.963
  P[1,2]    0.063 0.016 0.037 0.051 0.060 0.073 0.097
  P[2,1]    0.543 0.093 0.363 0.486 0.535 0.604 0.740
  P[2,2]    0.457 0.093 0.260 0.396 0.465 0.514 0.637
```

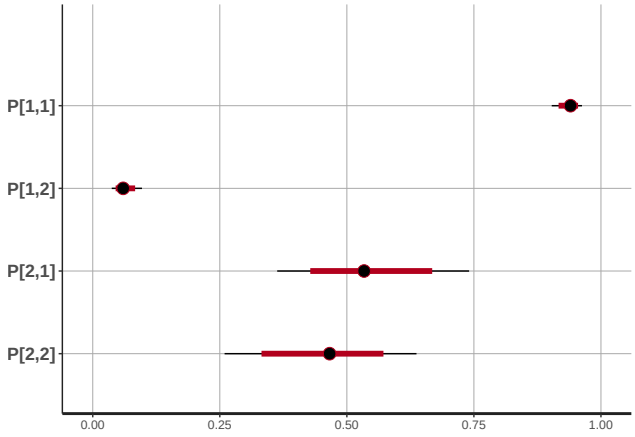
```
plot(fit,pars=c("alpha0","alpha1","beta1"))
```



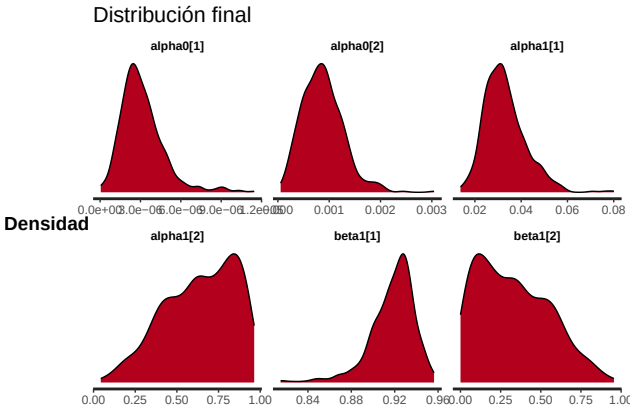

```
plot(fit,pars="p_remain")
```



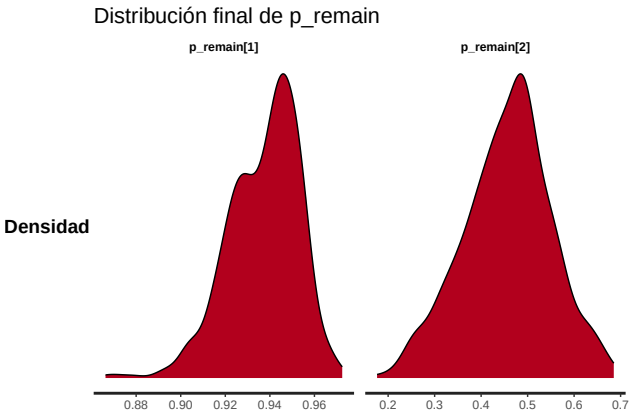

```
plot(fit,pars="P")
```



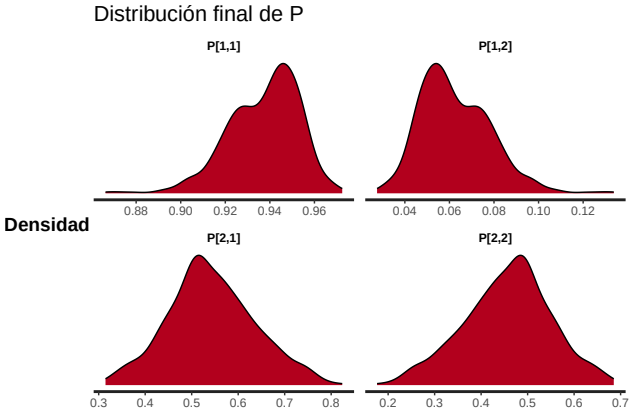

```
stan_dens(fit,pars=c("alpha0","alpha1","beta1"), point_est = "mean", show_density = TRUE) +
  ggtitle(expression("Distribución final",alpha[0],alpha[1],beta[1])) + ylab("Densidad") +
  theme(axis.title.x=element_text(size=14), axis.title.y=element_text(size=14),
        plot.title = element_text(size=16))
```



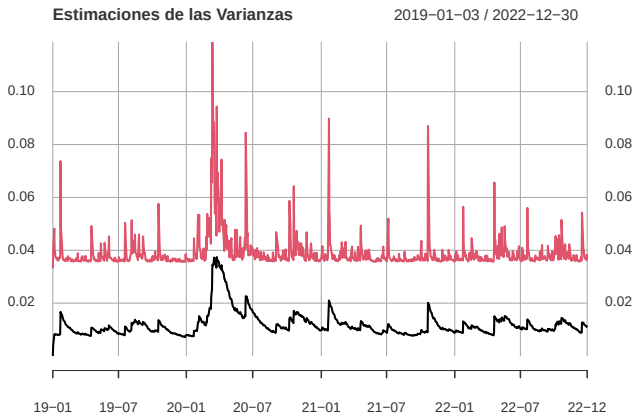

```
stan_dens(fit,pars="p_remain", point_est = "mean", show_density = TRUE) +
  ggtitle("Distribución final de p_remain") + ylab("Densidad") +
  theme(axis.title.x=element_text(size=14), axis.title.y=element_text(size=14),
        plot.title = element_text(size=16))
```



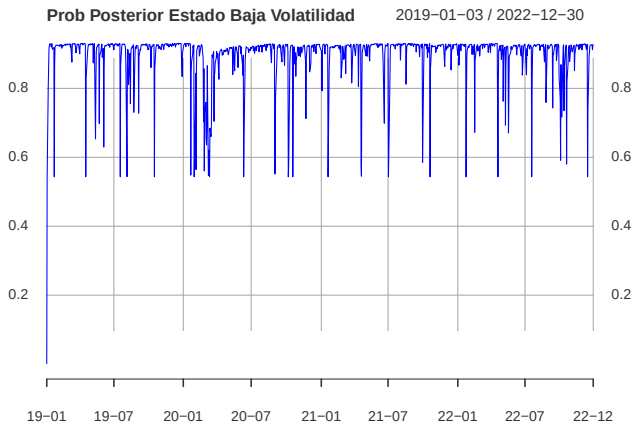

```
stan_dens(fit,pars="P", point_est = "mean", show_density = TRUE) +
  ggtitle("Distribución final de P") + ylab("Densidad") +
  theme(axis.title.x=element_text(size=14), axis.title.y=element_text(size=14),
        plot.title = element_text(size=16))
```



```
garch_posterior_means <- xts(apply(extract(fit, "sigma_t")[[1]], 2:3, mean),
                             index(IBM.R))
colnames(garch_posterior_means) <- c("Low-Vol State", "High-Vol State")

plot(
  garch_posterior_means,
  main = "Estimaciones de las Varianzas",
  format.labels = "%y-%m"
)
```

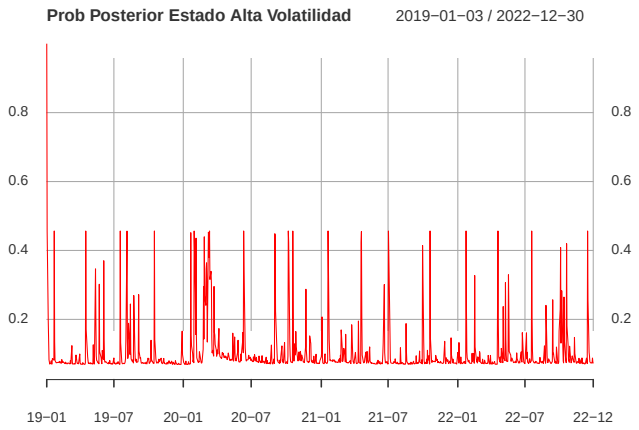



```
garch_posterior_prob1 <- xts(apply(extract(fit, "alpha")[[1]], 2:3, mean)[,1],
                             index(IBM.R))

plot(
  garch_posterior_prob1,
  main = "Prob Posterior Estado Baja Volatilidad",
  format.labels = "%y-%m",
  col="blue",lwd=1
)
```




```
garch_posterior_prob2 <- xts(apply(extract(fit, "alpha")[[1]], 2:3, mean)[,2],
                             index(IBM.R))
plot(
  garch_posterior_prob2,
  main = "Prob Posterior Estado Alta Volatilidad",
```

```
  format.labels = "%y-%m",
  col="red",lwd=1
)
```




```
zstar <- xts(apply(extract(fit, "zstar")[[1]], 2, median),
                                  index(IBM.R))
plot_statepath(zstar)
```