BDA - Assignment 6

Anonymous

Contents

Generalized linar model: Bioassay with sTAN **1**

```
# To install aaltobda, see the General information in the assignment.
remotes::install_github("avehtari/BDA_course_Aalto", subdir = "rpackage", upgrade = "never")
```

```
## Skipping install of 'aaltobda' from a github remote, the SHA1 (38f34d35) has not changed since last i
##   Use `force = TRUE` to force installation
```

```
library(aaltobda)
data("bioassay")
```

Generalized linar model: Bioassay with sTAN

1 Below is my Stan code as long as the chain iterations

```
library(rstan)
```

```
## Loading required package: StanHeaders
```

```
## Loading required package: ggplot2
```

```
## rstan (Version 2.21.3, GitRev: 2e1f913d3ca3)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

```
data <- list(I = length(bioassay$n),
             x = bioassay$x,
             n = 5,
             y = bioassay$y)

fit <- stan(file = 'bioassay.stan', data = data, verbose = TRUE)
```

```
##
## TRANSLATING MODEL 'bioassay' FROM Stan CODE TO C++ CODE NOW.
## successful in parsing the Stan model 'bioassay'.
## OS: x86_64, darwin17.0; rstan: 2.21.3; Rcpp: 1.0.7; 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 model3c34ac716e8_bioassay_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", "model3c34ac716e8_bioassay");
##   32 :     reader.add_event(27, 25, "end", "model3c34ac716e8_bioassay");
##   33 :     return reader;
##   34 : }
##   35 :
##   36 : class model3c34ac716e8_bioassay
##   37 :   : public stan::model::model_base_crtp<model3c34ac716e8_bioassay> {
##   38 : private:
##   39 :         int I;
##   40 :         int n;
##   41 :         std::vector<double> x;
##   42 :         std::vector<int> y;
##   43 : public:
##   44 :     model3c34ac716e8_bioassay(rstan::io::rlist_ref_var_context& context__,
##   45 :         std::ostream* pstream__ = 0)
##   46 :         : model_base_crtp(0) {
##   47 :         ctor_body(context__, 0, pstream__);
##   48 :     }
##   49 :
##   50 :     model3c34ac716e8_bioassay(stan::io::var_context& context__,
##   51 :         unsigned int random_seed__,
##   52 :         std::ostream* pstream__ = 0)
##   53 :         : model_base_crtp(0) {
##   54 :         ctor_body(context__, random_seed__, pstream__);
```

```
##   55 :     }
##   56 :
##   57 :     void ctor_body(stan::io::var_context& context__,
##   58 :                    unsigned int random_seed__,
##   59 :                    std::ostream* pstream__) {
##   60 :         typedef double local_scalar_t__;
##   61 :
##   62 :         boost::ecuyer1988 base_rng__ =
##   63 :           stan::services::util::create_rng(random_seed__, 0);
##   64 :         (void) base_rng__;  // suppress unused var warning
##   65 :
##   66 :         current_statement_begin__ = -1;
##   67 :
##   68 :         static const char* function__ = "model3c34ac716e8_bioassay_namespace::model3c34ac716e8
##   69 :         (void) function__;  // dummy to suppress unused var warning
##   70 :         size_t pos__;
##   71 :         (void) pos__;  // dummy to suppress unused var warning
##   72 :         std::vector<int> vals_i__;
##   73 :         std::vector<double> vals_r__;
##   74 :         local_scalar_t__ DUMMY_VAR__(std::numeric_limits<double>::quiet_NaN());
##   75 :         (void) DUMMY_VAR__;  // suppress unused var warning
##   76 :
##   77 :         try {
##   78 :             // initialize data block variables from context__
##   79 :             current_statement_begin__ = 4;
##   80 :             context__.validate_dims("data initialization", "I", "int", context__.to_vec());
##   81 :             I = int(0);
##   82 :             vals_i__ = context__.vals_i("I");
##   83 :             pos__ = 0;
##   84 :             I = vals_i__[pos__++];
##   85 :             check_greater_or_equal(function__, "I", I, 0);
##   86 :
##   87 :             current_statement_begin__ = 5;
##   88 :             context__.validate_dims("data initialization", "n", "int", context__.to_vec());
##   89 :             n = int(0);
##   90 :             vals_i__ = context__.vals_i("n");
##   91 :             pos__ = 0;
##   92 :             n = vals_i__[pos__++];
##   93 :             check_greater_or_equal(function__, "n", n, 0);
##   94 :
##   95 :             current_statement_begin__ = 6;
##   96 :             validate_non_negative_index("x", "I", I);
##   97 :             context__.validate_dims("data initialization", "x", "double", context__.to_vec(I)
##   98 :             x = std::vector<double>(I, double(0));
##   99 :             vals_r__ = context__.vals_r("x");
##  100 :             pos__ = 0;
##  101 :             size_t x_k_0_max__ = I;
##  102 :             for (size_t k_0__ = 0; k_0__ < x_k_0_max__; ++k_0__) {
##  103 :                 x[k_0__] = vals_r__[pos__++];
##  104 :             }
##  105 :
##  106 :             current_statement_begin__ = 7;
##  107 :             validate_non_negative_index("y", "I", I);
##  108 :             context__.validate_dims("data initialization", "y", "int", context__.to_vec(I));
```

```
##  109 :             y = std::vector<int>(I, int(0));
##  110 :             vals_i__ = context__.vals_i("y");
##  111 :             pos__ = 0;
##  112 :             size_t y_k_0_max__ = I;
##  113 :             for (size_t k_0__ = 0; k_0__ < y_k_0_max__; ++k_0__) {
##  114 :                 y[k_0__] = vals_i__[pos__++];
##  115 :             }
##  116 :             size_t y_i_0_max__ = I;
##  117 :             for (size_t i_0__ = 0; i_0__ < y_i_0_max__; ++i_0__) {
##  118 :                 check_greater_or_equal(function__, "y[i_0__]", y[i_0__], 0);
##  119 :                 check_less_or_equal(function__, "y[i_0__]", y[i_0__], n);
##  120 :             }
##  121 :
##  122 :
##  123 :             // initialize transformed data variables
##  124 :             // execute transformed data statements
##  125 :
##  126 :             // validate transformed data
##  127 :
##  128 :             // validate, set parameter ranges
##  129 :             num_params_r__ = 0U;
##  130 :             param_ranges_i__.clear();
##  131 :             current_statement_begin__ = 10;
##  132 :             validate_non_negative_index("theta", "2", 2);
##  133 :             num_params_r__ += 2;
##  134 :         } catch (const std::exception& e) {
##  135 :             stan::lang::rethrow_located(e, current_statement_begin__, prog_reader__());
##  136 :             // Next line prevents compiler griping about no return
##  137 :             throw std::runtime_error("*** IF YOU SEE THIS, PLEASE REPORT A BUG ***");
##  138 :         }
##  139 :     }
##  140 :
##  141 :     ~model3c34ac716e8_bioassay() { }
##  142 :
##  143 :
##  144 :     void transform_inits(const stan::io::var_context& context__,
##  145 :                          std::vector<int>& params_i__,
##  146 :                          std::vector<double>& params_r__,
##  147 :                          std::ostream* pstream__) const {
##  148 :         typedef double local_scalar_t__;
##  149 :         stan::io::writer<double> writer__(params_r__, params_i__);
##  150 :         size_t pos__;
##  151 :         (void) pos__; // dummy call to supress warning
##  152 :         std::vector<double> vals_r__;
##  153 :         std::vector<int> vals_i__;
##  154 :
##  155 :         current_statement_begin__ = 10;
##  156 :         if (!(context__.contains_r("theta")))
##  157 :             stan::lang::rethrow_located(std::runtime_error(std::string("Variable theta missing
##  158 :         vals_r__ = context__.vals_r("theta");
##  159 :         pos__ = 0U;
##  160 :         validate_non_negative_index("theta", "2", 2);
##  161 :         context__.validate_dims("parameter initialization", "theta", "vector_d", context__.to_
##  162 :         Eigen::Matrix<double, Eigen::Dynamic, 1> theta(2);
```

```
##  163 :         size_t theta_j_1_max__ = 2;
##  164 :         for (size_t j_1__ = 0; j_1__ < theta_j_1_max__; ++j_1__) {
##  165 :             theta(j_1__) = vals_r__[pos__++];
##  166 :         }
##  167 :         try {
##  168 :             writer__.vector_unconstrain(theta);
##  169 :         } catch (const std::exception& e) {
##  170 :             stan::lang::rethrow_located(std::runtime_error(std::string("Error transforming va
##  171 :         }
##  172 :
##  173 :         params_r__ = writer__.data_r();
##  174 :         params_i__ = writer__.data_i();
##  175 :     }
##  176 :
##  177 :     void transform_inits(const stan::io::var_context& context,
##  178 :                          Eigen::Matrix<double, Eigen::Dynamic, 1>& params_r,
##  179 :                          std::ostream* pstream__) const {
##  180 :       std::vector<double> params_r_vec;
##  181 :       std::vector<int> params_i_vec;
##  182 :       transform_inits(context, params_i_vec, params_r_vec, pstream__);
##  183 :       params_r.resize(params_r_vec.size());
##  184 :       for (int i = 0; i < params_r.size(); ++i)
##  185 :         params_r(i) = params_r_vec[i];
##  186 :     }
##  187 :
##  188 :
##  189 :     template <bool propto__, bool jacobian__, typename T__>
##  190 :     T__ log_prob(std::vector<T__>& params_r__,
##  191 :                  std::vector<int>& params_i__,
##  192 :                  std::ostream* pstream__ = 0) const {
##  193 :
##  194 :         typedef T__ local_scalar_t__;
##  195 :
##  196 :         local_scalar_t__ DUMMY_VAR__(std::numeric_limits<double>::quiet_NaN());
##  197 :         (void) DUMMY_VAR__;  // dummy to suppress unused var warning
##  198 :
##  199 :         T__ lp__(0.0);
##  200 :         stan::math::accumulator<T__> lp_accum__;
##  201 :         try {
##  202 :             stan::io::reader<local_scalar_t__> in__(params_r__, params_i__);
##  203 :
##  204 :             // model parameters
##  205 :             current_statement_begin__ = 10;
##  206 :             Eigen::Matrix<local_scalar_t__, Eigen::Dynamic, 1> theta;
##  207 :             (void) theta;  // dummy to suppress unused var warning
##  208 :             if (jacobian__)
##  209 :                 theta = in__.vector_constrain(2, lp__);
##  210 :             else
##  211 :                 theta = in__.vector_constrain(2);
##  212 :
##  213 :             // model body
##  214 :             {
##  215 :             current_statement_begin__ = 13;
##  216 :             validate_non_negative_index("mu", "2", 2);
```

```
##  217 :             Eigen::Matrix<local_scalar_t__, 1, Eigen::Dynamic> mu(2);
##  218 :             stan::math::initialize(mu, DUMMY_VAR__);
##  219 :             stan::math::fill(mu, DUMMY_VAR__);
##  220 :             stan::math::assign(mu,stan::math::to_row_vector(stan::math::array_builder<double >
##  221 :
##  222 :             current_statement_begin__ = 14;
##  223 :             validate_non_negative_index("sigma", "2", 2);
##  224 :             validate_non_negative_index("sigma", "2", 2);
##  225 :             Eigen::Matrix<local_scalar_t__, Eigen::Dynamic, Eigen::Dynamic> sigma(2, 2);
##  226 :             stan::math::initialize(sigma, DUMMY_VAR__);
##  227 :             stan::math::fill(sigma, DUMMY_VAR__);
##  228 :             stan::math::assign(sigma,stan::math::to_matrix(stan::math::array_builder<Eigen::Ma
##  229 :
##  230 :             current_statement_begin__ = 15;
##  231 :             validate_non_negative_index("logLike", "I", I);
##  232 :             Eigen::Matrix<local_scalar_t__, Eigen::Dynamic, 1> logLike(I);
##  233 :             stan::math::initialize(logLike, DUMMY_VAR__);
##  234 :             stan::math::fill(logLike, DUMMY_VAR__);
##  235 :
##  236 :
##  237 :             current_statement_begin__ = 18;
##  238 :             lp_accum__.add(multi_normal_log(theta, mu, sigma));
##  239 :             current_statement_begin__ = 21;
##  240 :             for (int i = 1; i <= I; ++i) {
##  241 :
##  242 :                 current_statement_begin__ = 22;
##  243 :                 stan::model::assign(logLike,
##  244 :                             stan::model::cons_list(stan::model::index_uni(i), stan::model::nil
##  245 :                             binomial_logit_log(get_base1(y, i, "y", 1), n, (get_base1(theta, 1
##  246 :                             "assigning variable logLike");
##  247 :             }
##  248 :             current_statement_begin__ = 24;
##  249 :             lp_accum__.add(sum(logLike));
##  250 :             }
##  251 :
##  252 :         } catch (const std::exception& e) {
##  253 :             stan::lang::rethrow_located(e, current_statement_begin__, prog_reader__());
##  254 :             // Next line prevents compiler griping about no return
##  255 :             throw std::runtime_error("*** IF YOU SEE THIS, PLEASE REPORT A BUG ***");
##  256 :         }
##  257 :
##  258 :         lp_accum__.add(lp__);
##  259 :         return lp_accum__.sum();
##  260 :
##  261 :     } // log_prob()
##  262 :
##  263 :     template <bool propto, bool jacobian, typename T_>
##  264 :     T_ log_prob(Eigen::Matrix<T_,Eigen::Dynamic,1>& params_r,
##  265 :                 std::ostream* pstream = 0) const {
##  266 :       std::vector<T_> vec_params_r;
##  267 :       vec_params_r.reserve(params_r.size());
##  268 :       for (int i = 0; i < params_r.size(); ++i)
##  269 :         vec_params_r.push_back(params_r(i));
##  270 :       std::vector<int> vec_params_i;
```

```
##  271 :       return log_prob<propto,jacobian,T_>(vec_params_r, vec_params_i, pstream);
##  272 :     }
##  273 :
##  274 :
##  275 :     void get_param_names(std::vector<std::string>& names__) const {
##  276 :         names__.resize(0);
##  277 :         names__.push_back("theta");
##  278 :     }
##  279 :
##  280 :
##  281 :     void get_dims(std::vector<std::vector<size_t> >& dimss__) const {
##  282 :         dimss__.resize(0);
##  283 :         std::vector<size_t> dims__;
##  284 :         dims__.resize(0);
##  285 :         dims__.push_back(2);
##  286 :         dimss__.push_back(dims__);
##  287 :     }
##  288 :
##  289 :     template <typename RNG>
##  290 :     void write_array(RNG& base_rng__,
##  291 :                      std::vector<double>& params_r__,
##  292 :                      std::vector<int>& params_i__,
##  293 :                      std::vector<double>& vars__,
##  294 :                      bool include_tparams__ = true,
##  295 :                      bool include_gqs__ = true,
##  296 :                      std::ostream* pstream__ = 0) const {
##  297 :         typedef double local_scalar_t__;
##  298 :
##  299 :         vars__.resize(0);
##  300 :         stan::io::reader<local_scalar_t__> in__(params_r__, params_i__);
##  301 :         static const char* function__ = "model3c34ac716e8_bioassay_namespace::write_array";
##  302 :         (void) function__;  // dummy to suppress unused var warning
##  303 :
##  304 :         // read-transform, write parameters
##  305 :         Eigen::Matrix<double, Eigen::Dynamic, 1> theta = in__.vector_constrain(2);
##  306 :         size_t theta_j_1_max__ = 2;
##  307 :         for (size_t j_1__ = 0; j_1__ < theta_j_1_max__; ++j_1__) {
##  308 :             vars__.push_back(theta(j_1__));
##  309 :         }
##  310 :
##  311 :         double lp__ = 0.0;
##  312 :         (void) lp__;  // dummy to suppress unused var warning
##  313 :         stan::math::accumulator<double> lp_accum__;
##  314 :
##  315 :         local_scalar_t__ DUMMY_VAR__(std::numeric_limits<double>::quiet_NaN());
##  316 :         (void) DUMMY_VAR__;  // suppress unused var warning
##  317 :
##  318 :         if (!include_tparams__ && !include_gqs__) return;
##  319 :
##  320 :         try {
##  321 :             if (!include_gqs__ && !include_tparams__) return;
##  322 :             if (!include_gqs__) return;
##  323 :         } catch (const std::exception& e) {
##  324 :             stan::lang::rethrow_located(e, current_statement_begin__, prog_reader__());
```

```
##  325 :             // Next line prevents compiler griping about no return
##  326 :             throw std::runtime_error("*** IF YOU SEE THIS, PLEASE REPORT A BUG ***");
##  327 :         }
##  328 :     }
##  329 :
##  330 :     template <typename RNG>
##  331 :     void write_array(RNG& base_rng,
##  332 :                      Eigen::Matrix<double,Eigen::Dynamic,1>& params_r,
##  333 :                      Eigen::Matrix<double,Eigen::Dynamic,1>& vars,
##  334 :                      bool include_tparams = true,
##  335 :                      bool include_gqs = true,
##  336 :                      std::ostream* pstream = 0) const {
##  337 :       std::vector<double> params_r_vec(params_r.size());
##  338 :       for (int i = 0; i < params_r.size(); ++i)
##  339 :         params_r_vec[i] = params_r(i);
##  340 :       std::vector<double> vars_vec;
##  341 :       std::vector<int> params_i_vec;
##  342 :       write_array(base_rng, params_r_vec, params_i_vec, vars_vec, include_tparams, include_gqs
##  343 :       vars.resize(vars_vec.size());
##  344 :       for (int i = 0; i < vars.size(); ++i)
##  345 :         vars(i) = vars_vec[i];
##  346 :     }
##  347 :
##  348 :     std::string model_name() const {
##  349 :         return "model3c34ac716e8_bioassay";
##  350 :     }
##  351 :
##  352 :
##  353 :     void constrained_param_names(std::vector<std::string>& param_names__,
##  354 :                                  bool include_tparams__ = true,
##  355 :                                  bool include_gqs__ = true) const {
##  356 :         std::stringstream param_name_stream__;
##  357 :         size_t theta_j_1_max__ = 2;
##  358 :         for (size_t j_1__ = 0; j_1__ < theta_j_1_max__; ++j_1__) {
##  359 :             param_name_stream__.str(std::string());
##  360 :             param_name_stream__ << "theta" << '.' << j_1__ + 1;
##  361 :             param_names__.push_back(param_name_stream__.str());
##  362 :         }
##  363 :
##  364 :         if (!include_gqs__ && !include_tparams__) return;
##  365 :
##  366 :         if (include_tparams__) {
##  367 :         }
##  368 :
##  369 :         if (!include_gqs__) return;
##  370 :     }
##  371 :
##  372 :
##  373 :     void unconstrained_param_names(std::vector<std::string>& param_names__,
##  374 :                                    bool include_tparams__ = true,
##  375 :                                    bool include_gqs__ = true) const {
##  376 :         std::stringstream param_name_stream__;
##  377 :         size_t theta_j_1_max__ = 2;
##  378 :         for (size_t j_1__ = 0; j_1__ < theta_j_1_max__; ++j_1__) {
```

```
##  379 :             param_name_stream__.str(std::string());
##  380 :             param_name_stream__ << "theta" << '.' << j_1__ + 1;
##  381 :             param_names__.push_back(param_name_stream__.str());
##  382 :         }
##  383 :
##  384 :         if (!include_gqs__ && !include_tparams__) return;
##  385 :
##  386 :         if (include_tparams__) {
##  387 :         }
##  388 :
##  389 :         if (!include_gqs__) return;
##  390 :     }
##  391 :
##  392 : }; // model
##  393 :
##  394 : }  // namespace
##  395 :
##  396 : typedef model3c34ac716e8_bioassay_namespace::model3c34ac716e8_bioassay stan_model;
##  397 :
##  398 : #ifndef USING_R
##  399 :
##  400 : stan::model::model_base& new_model(
##  401 :         stan::io::var_context& data_context,
##  402 :         unsigned int seed,
##  403 :         std::ostream* msg_stream) {
##  404 :   stan_model* m = new stan_model(data_context, seed, msg_stream);
##  405 :   return *m;
##  406 : }
##  407 :
##  408 : #endif
##  409 :
##  410 :
##  411 :
##  412 : #include <rstan_next/stan_fit.hpp>
##  413 :
##  414 : struct stan_model_holder {
##  415 :     stan_model_holder(rstan::io::rlist_ref_var_context rcontext,
##  416 :                       unsigned int random_seed)
##  417 :     : rcontext_(rcontext), random_seed_(random_seed)
##  418 :      {
##  419 :      }
##  420 :
##  421 :    //stan::math::ChainableStack ad_stack;
##  422 :    rstan::io::rlist_ref_var_context rcontext_;
##  423 :    unsigned int random_seed_;
##  424 : };
##  425 :
##  426 : Rcpp::XPtr<stan::model::model_base> model_ptr(stan_model_holder* smh) {
##  427 :   Rcpp::XPtr<stan::model::model_base> model_instance(new stan_model(smh->rcontext_, smh->rando
##  428 :   return model_instance;
##  429 : }
##  430 :
##  431 : Rcpp::XPtr<rstan::stan_fit_base> fit_ptr(stan_model_holder* smh) {
##  432 :   return Rcpp::XPtr<rstan::stan_fit_base>(new rstan::stan_fit(model_ptr(smh), smh->random_seed
```

```
##  433 : }
##  434 :
##  435 : std::string model_name(stan_model_holder* smh) {
##  436 :   return model_ptr(smh).get()->model_name();
##  437 : }
##  438 :
##  439 : RCPP_MODULE(stan_fit4model3c34ac716e8_bioassay_mod){
##  440 :   Rcpp::class_<stan_model_holder>("stan_fit4model3c34ac716e8_bioassay")
##  441 :   .constructor<rstan::io::rlist_ref_var_context, unsigned int>()
##  442 :   .method("model_ptr", &model_ptr)
##  443 :   .method("fit_ptr", &fit_ptr)
##  444 :   .method("model_name", &model_name)
##  445 :   ;
##  446 : }
##  447 :
##  448 :
##  449 : // declarations
##  450 : extern "C" {
##  451 : SEXP file3c36cbc980c( ) ;
##  452 : }
##  453 :
##  454 : // definition
##  455 : SEXP file3c36cbc980c() {
##  456 :  return Rcpp::wrap("bioassay");
##  457 : }
##
## CHECKING DATA AND PREPROCESSING FOR MODEL 'bioassay' NOW.
##
## COMPILING MODEL 'bioassay' NOW.
##
## STARTING SAMPLER FOR MODEL 'bioassay' NOW.
##
## SAMPLING FOR MODEL 'bioassay' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 4.4e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.44 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.039333 seconds (Warm-up)
## Chain 1:                0.034619 seconds (Sampling)
```

```
## Chain 1:                0.073952 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'bioassay' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.3e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.13 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.037131 seconds (Warm-up)
## Chain 2:                0.036182 seconds (Sampling)
## Chain 2:                0.073313 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'bioassay' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 1.3e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.13 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.034711 seconds (Warm-up)
## Chain 3:                0.031247 seconds (Sampling)
## Chain 3:                0.065958 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'bioassay' NOW (CHAIN 4).
```

```
## Chain 4:
## Chain 4: Gradient evaluation took 1.1e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.11 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.035617 seconds (Warm-up)
## Chain 4:                0.027492 seconds (Sampling)
## Chain 4:                0.063109 seconds (Total)
## Chain 4:
```

```
cat("\n")
```

```
cat("\n")
```

```
writeLines(readLines("bioassay.stan"))
```

```
## // Note:
## // Script runs byoassay model in BDA v3 from Section 3.7
## data { // Data is from Table 3.1
##     int<lower=0> I;       // Number of groups
##     int<lower=0> n;    // Number of animals
##     real x[I];            // Doses
##     int<lower=0,upper=n> y[I];    // Number of deaths
## }
## parameters {
##     vector[2] theta;  // [alpha, beta]
## }
## model {
##     row_vector[2] mu = [0,10];               // Location param
##     matrix[2,2] sigma =[[4, 12],[12, 100]];  // Shape param
##     vector[I] logLike;  // log-likelihood contribution per group
##
##     // The prior distribution
##     target += multi_normal_lpdf(theta | mu, sigma);
##
##     // The log-likelihood term
##     for(i in 1:I) {
##         logLike[i] = binomial_logit_lpmf(y[i] | n, theta[1] + theta[2]*x[i]);
##     }
##     target += sum(logLike);
## }
```

2. Below is my R-hat diagnostics as well as my explanaiton

$\hat{R}$ is a measured of convergence across the different chains (e.g., separate simulation experiments characterized by different starting points). It is a ratio of weighted total variance, variance between the chains and variance within the chains, and simply the variance between the chains. As the variance between the chains becomes smaller, $\hat{R}$ approaches zero, because the numerator and denominator are becoming the same. Large variance between the chains indicates that simulation experiments have not approached a common density, so whenever $\hat{R}$ is large, it indicates that we should tweak something in the model like the number of simulations per experiment. Although several reasons can explain the lack of convergence, simple examples can be fixed by just increasing the number of simulations.

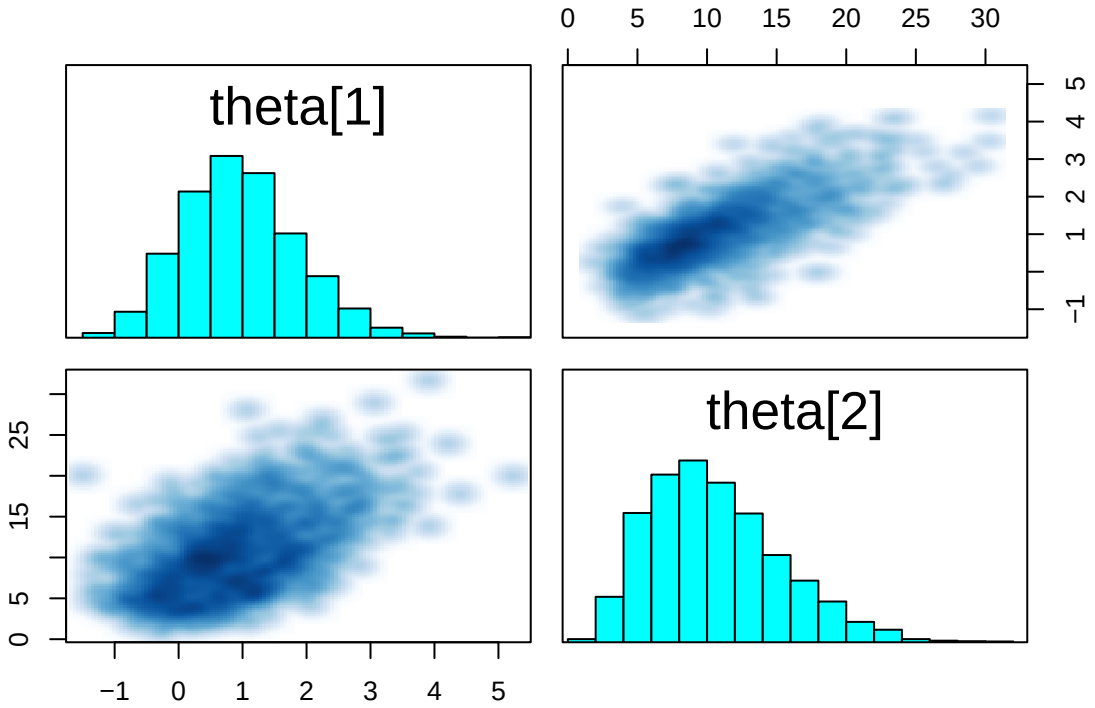
```
print(fit) # Fit diagnostics
```

```
## Inference for Stan model: bioassay.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##           mean se_mean   sd   2.5%   25%   50%   75% 97.5% n_eff Rhat
## theta[1]  0.96    0.03 0.90  -0.65  0.33  0.90  1.52  2.86  1252 1.00
## theta[2] 10.57    0.13 4.67   3.46  7.09  9.92 13.48 20.99  1252 1.01
## lp__     -7.83    0.03 1.01 -10.47 -8.21 -7.51 -7.10 -6.84  1471 1.00
##
## Samples were drawn using NUTS(diag_e) at Wed Mar 23 13:21:25 2022.
## 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).
```

3) Below I show the scatterplot between the estimate coefficient distributions

As my previous measure indicates, $\theta[1] = \alpha = 1.0$ since $MCSE = 0.03$, uninformative values after the second decimal point, and $\theta[2] = \beta = 10.0$ since $MCSE = 0.16$, uninformative values after the first decimal point. These are similar to the values reported in BDA3 - Section 3.7 at 0.8 and 7.7, respectively.

```
pairs(fit, pars = c("theta"))
```



4) Course feedback:

- I used mostly Windows (work computer), but since my work computer does not knit pdfs for some reason, I complemented my work with my mac.
- R
- RStan
- I had so many installations problems with rstan, but mostly they came because of not knowing how to handle the makevars files, and that my computer had some admin permission limitation.
- I am experienced in several programming languages other than R, but I stick to R because it has really cool documentation. Rstan, in my opinion, falls short. I wish there were more documentation of functions to know for instance what are the inputs and the outputs as well as the classes of the inputs and outputs and their dimensions.