G. Stan code

To unambiguously define the models, we provide the Stan code in the following.

Listing 1. "Stan code for the Chauchy example"

```
data{
  int<lower=0> N;
  real<lower=0> gamma;
  vector[N] y;
}
parameters{
  real xi;
}
model{
  y ~ cauchy(xi, gamma);
  xi ~ normal(0, 1);
}
```

Listing 2. "Stan code for the toy linear regression example"

```
data {
  int<lower=0> N;
  int<lower=1> P;
  vector[N] y;
  matrix[N,P] x;
}

parameters {
  vector[P] w;
  real b;
  real<lower=0> sigma;
}

model {
  y ~ normal(x * w + b, sigma);
  b ~ normal(0,10);
  w ~ normal(0, 10);
  sigma ~ lognormal(0.5,1);
}
```

Listing 3. "Stan code for the Diamond Example"

```
// The code has been taken from https://github.com/stan-dev/posteriordb
// generated with brms 2.10.0
functions {
}
data {
  int<lower=1> N;  // number of observations
  vector[N] Y;  // response variable
  int<lower=1> K;  // number of population-level effects
  matrix[N, K] X;  // population-level design matrix
  int prior_only;  // should the likelihood be ignored?
}
transformed data {
  int Kc = K - 1;
  matrix[N, Kc] Xc;  // centered version of X without an intercept
  vector[Kc] means_X;  // column means of X before centering
  for (i in 2:K) {
```

```
    means_X[i - 1] = mean(X[, i]);
    Xc[, i - 1] = X[, i] - means_X[i - 1];
  }
}
parameters {
  vector[Kc] b;  // population-level effects
  // temporary intercept for centered predictors
  real Intercept;
  real<lower=0> sigma;  // residual SD
}
transformed parameters {
}
model {
  // priors including all constants
  target += normal_lpdf(b | 0, 1);
  target += student_t_lpdf(Intercept | 3, 8, 10);
  target += student_t_lpdf(sigma | 3, 0, 10)
    - 1 * student_t_lccdf(0 | 3, 0, 10);
  // likelihood including all constants
  if (!prior_only) {
    target += normal_id_glm_lpdf(Y | Xc, Intercept, b, sigma);
  }
}
generated quantities {
  // actual population-level intercept
  real b_Intercept = Intercept - dot_product(means_X, b);
}
```

Listing 4. "Stan code for 8 Schools in the NCP parameterization"

```
//eight_schools_ncp.stan
data {
  int<lower=0> J;
  real y[J];
  real<lower=0> sigma[J];
}

parameters {
  real mu;
  real<lower=0> tau;
  real theta_tilde[J];
}

transformed parameters {
  real theta[J];
  for (j in 1:J)
    theta[j] = mu + tau * theta_tilde[j]; //theta[j] ~ N(mu, tau*theta_tilde[j])
}

model {
  mu ~ normal(0, 5);
  tau ~ cauchy(0, 5);
  theta_tilde ~ normal(0, 1);
  y ~ normal(theta, sigma);
}
```

Listing 5. "Stan code for the 8 schools example in the CP parametrization"

```
//eight_schools_cp.stan
data {
  int<lower=0> J;
  real y[J];
  real<lower=0> sigma[J];
}

parameters {
  real mu;
  real<lower=0> tau;
  real theta[J];
}

model {
  //Priors for p(mu, tau, theta)
  mu ~ normal(0, 5);
  tau ~ cauchy(0, 5);
  theta ~ normal(mu, tau);
  //Likelihood
  y ~ normal(theta, sigma);
}
```

Listing 6. "Stan code for the NN based non-linear regression example"

```
functions {
    vector calculate_mu(matrix X, matrix bias_first_m,
        real bias_output, matrix w_first, vector w_output, int num_layers) {
                int N = rows(X);
                int num_nodes = rows(w_first);
                matrix[N, num_nodes] layer_values[num_layers - 2];
                vector[N] mu;
        layer_values[1] = inv_logit(bias_first_m + X * w_first');
                mu = bias_output + layer_values[num_layers - 2] * w_output;
      return mu;
    }
  }
  data {
    int<lower=0> N;                                   // num data
    int<lower=0> d;                                   // dim x
    int<lower=0> num_nodes;                       // num hidden unites
    int<lower=1> num_middle_layers;           // num hidden layer
    matrix[N,d] X;                                    // X
    real y[N];                                        // y
        int<lower=0> Nt;                              // num predicive data
        matrix[Nt,d] Xt;                              // X predicive
        real<lower=0> sigma;                      // const sigma
  }
  transformed data {
    int num_layers;
    num_layers = num_middle_layers + 2;
  }
  parameters {
    vector[num_nodes] bias_first;
    real bias_output;
    matrix[num_nodes, d] w_first;
    vector[num_nodes] w_output;
        // hyperparameters
```

```
  real<lower=0> bias_first_h;
  real<lower=0> w_first_h;
  real<lower=0> w_output_h;
}
transformed parameters {
  matrix[N, num_nodes] bias_first_m = rep_matrix(bias_first', N);
}
model{
  vector[N] mu;
  mu = calculate_mu(X, bias_first_m, bias_output, w_first, w_output, num_layers);
  y ~ normal(mu,sigma);
  //priors
  bias_first_h ~ normal(0, 1);
  bias_first ~ normal(0, 1);
  bias_output ~ normal(0, 1);
  w_first_h ~ normal(0, 1);
  to_vector(w_first) ~ normal(0, 1);
  w_output_h ~ normal(0, 1);
  w_output ~ normal(0, 1);
}
generated quantities{
  vector[Nt] predictions;
      matrix[Nt, num_nodes] bias_first_mg = rep_matrix(bias_first', Nt);
      vector[Nt] mu;
      mu = calculate_mu(Xt, bias_first_mg, bias_output,w_first, w_output, num_layers);
      for(i in 1:Nt){
              predictions[i] = normal_rng(mu[i],sigma);
      }
 }
```