Cardiovascular model in Appendix E

This notebook contains the code of the paper "Learning Physics between Digital Twins with Low-Fidelity Models and Physics-Informed Gaussian Processes". The models are fitted in rstan and the code is available in the folder "STAN/WK2".

Load packages

```
# uncomment to install
# install.packages("rstan")
# install.packages("ggplot2")
# install.packages("latex2exp")
library(rstan)
library(ggplot2)
library(latex2exp)

rstan_options(auto_write = TRUE)
options(mc.cores = 3) # allocate 3 cores (for each model we run 3 chains in parallel)
# Load flow data
d = readRDS("Data/Inflow_time.rds")
```

Reality and modelling choice

$$\mathcal{R}: \quad \frac{dP(t)}{dt} + \frac{P(t)}{R_2 C} = \frac{Q(t)}{C}\left(1 + \frac{R_1}{R_2}\right) + R_1 \frac{dQ(t)}{dt} \quad \text{(the misspesified model we use to fit the data)} \quad \text{[WK3]} \tag{1}$$

$$\eta: \quad Q(t) = \frac{1}{R}P(t) + C\frac{dP(t)}{dt} \quad \text{(the model we use to simulate data)} \quad \text{[WK2]} \tag{2}$$

Load posterior samples

First, we load the posterior samples from the fitted models obtained using the 'toy_paper_code.Rmd' or in the experiments folder, 'WK2_appendix.R'.

```
# choose some reasonable physical parameter values
library(ggplot2)
dat = readRDS("Data/post_wk2.rds") # posterior data for all models
flow_time=readRDS("Data/Inflow_time.rds")
time=flow_time$time
post_nwd = dat$df_ycd
attach(dat)
attach(data)
lP_pr = lI_pr = list()
t1=Sys.time()
R_val=c(1,1.15, 1.3); C_val = c(1.1,0.95,1.25)
RC=expand.grid(R_val,C_val) # create all possible combinations
```

```
Rtrue=RC[,1]; Ctrue=RC[,2]
nc=2
nP=12
nI=14
Ns=9
set.seed(123)
Zvec=sample(seq(0.02,0.1,by=0.01), Ns, replace = T)
nP_pred=50; nI_pred=50;
tP_pred = time[round(seq(1,length(time), length.out = nP_pred))]
tI_pred = time[round(seq(1,length(time), length.out = nI_pred))]
```

Model 1 (no-without delta in paper, Figure 6)

This is the misspecified model that does not account for model discrepancy (no-without delta in paper, Figure 6). For more details on prediction equations see Appendix C.2.

Stan code:

```
writeLines(readLines("STAN/WK2/WK2_nodelta_pred.stan"))
```

```
functions {
  vector mu_fn(real mu_wk2,
               real R,
               int nP,
               int nI){
                 vector[nP]  mP = rep_vector(mu_wk2,nP);
                 vector[nI]  mI = rep_vector((1/R)*mu_wk2,nI);
                 vector[nP + nI] mu;
                 mu= append_row(mP, mI);
                 return(mu);
  }

  matrix K_wk2(vector tP,
               vector tI,
               real rho,
               real alpha,
               real sigmaP,
               real sigmaI,
               real R,
               real C) {
    int nP = rows(tP);
    int nI = rows(tI);
    matrix[nP + nI, nP + nI] K;

    // KP
    for (i in 1:(nP-1)){
      K[i,i] = pow(alpha, 0.2e1);
      for (j in (i+1):nP){
        K[i,j] = exp(-pow(tP[i] - tP[j], 0.2e1) * pow(rho, -0.2e1));
        K[i,j] = pow(alpha, 0.2e1) * K[i,j];
        K[j,i] = K[i,j];
      }
      K[nP,nP] = pow(alpha, 0.2e1);
    }
```

```
  K[1:nP, 1:nP] = K[1:nP, 1:nP] + diag_matrix(rep_vector(pow(sigmaP, 0.2e1), nP));

  // KPI
  for (i in 1:nP){
    for (j in 1:nI){
      K[i, nP + j] = 0.1e1 / R * exp(-pow(tP[i] - tI[j], 0.2e1) * pow(rho, -0.2e1))
      + 0.2e1 * C * (tP[i] - tI[j])* pow(rho, -0.2e1)
      * exp(-pow(tP[i] - tI[j], 0.2e1) * pow(rho, -0.2e1));
      K[i, nP + j] = pow(alpha, 0.2e1) * K[i, nP + j];
    }
  }

  // KIP (KIP = KPI')
  //K[(nP + 1):(nP + nI), 1:nP] = K[1:nP, (nP + 1):(nP + nI)]';
  for (i in 1:nI){
    for (j in 1:nP){
     K[nP + i, j] = 0.1e1 / R * exp(-pow(tI[i] - tP[j], 0.2e1) * pow(rho, -0.2e1))
                    - 0.2e1 * C * (tI[i] - tP[j]) * pow(rho, -0.2e1)
                    * exp(-pow(tI[i] - tP[j], 0.2e1) * pow(rho, -0.2e1));
     K[nP + i, j] = pow(alpha, 0.2e1) * K[nP + i, j];
    }
  }
  // KI
  for (i in 1:(nI-1)){
   K[nP + i, nP +i] =
              pow(R, -0.2e1) * exp(-pow(tI[i] - tI[i], 0.2e1) * pow(rho, -0.2e1))
            + C * C * (0.2e1 * pow(rho, -0.2e1) * exp(-pow(tI[i] - tI[i], 0.2e1)
                                                      * pow(rho, -0.2e1)) - 0.4e1 * pow(tI[i] - tI[i], (
                       * pow(rho, -0.4e1) * exp(-pow(tI[i] - tI[i], 0.2e1) * pow(rho, -0.2e1)));
   K[nP + i, nP +i] = pow(alpha, 0.2e1) * K[nP + i, nP +i];
   for (j in (i+1):nI){
    K[nP + i, nP +j] =
                pow(R, -0.2e1) * exp(-pow(tI[i] - tI[j], 0.2e1) * pow(rho, -0.2e1))
                + C * C * (0.2e1 * pow(rho, -0.2e1) * exp(-pow(tI[i] - tI[j], 0.2e1)
                * pow(rho, -0.2e1)) - 0.4e1 * pow(tI[i] - tI[j], 0.2e1)
                * pow(rho, -0.4e1) * exp(-pow(tI[i] - tI[j], 0.2e1) * pow(rho, -0.2e1)));
    K[nP + i, nP +j] = pow(alpha, 0.2e1) * K[nP + i, nP +j];
    K[nP + j, nP +i] = K[nP + i, nP +j];
   }
   K[nP + nI, nP +nI] =
              pow(R, -0.2e1) * exp(-pow(tI[nI] - tI[nI], 0.2e1) * pow(rho, -0.2e1))
              + C * C * (0.2e1 * pow(rho, -0.2e1) * exp(-pow(tI[nI] - tI[nI], 0.2e1)
              * pow(rho, -0.2e1)) - 0.4e1 * pow(tI[nI] - tI[nI], 0.2e1)
              * pow(rho, -0.4e1) * exp(-pow(tI[nI] - tI[nI], 0.2e1) * pow(rho, -0.2e1)));
   K[nP + nI, nP +nI] = pow(alpha, 0.2e1) * K[nP + nI, nP +nI];
   }
   K[(nP + 1):(nP + nI), (nP + 1):(nP + nI)] = K[(nP + 1):(nP + nI), (nP + 1):(nP + nI)]
          + diag_matrix(rep_vector(pow(sigmaI, 0.2e1), nI));
   return cholesky_decompose(K);
}
matrix KP_pred(vector tP,
               vector tP_pred,
               real rho,
               real alpha
```

```
){
  int nP = rows(tP);
  int nP_pred = rows(tP_pred);
  matrix[nP_pred, nP] KP;

  for (i in 1:nP_pred){
    for (j in 1:nP){
      KP[i,j] = exp(-pow(tP_pred[i] - tP[j], 0.2e1) * pow(rho, -0.2e1));
      KP[i,j] = pow(alpha, 0.2e1) * KP[i,j];
    }
  }
  return KP;
}

matrix KPI_pred(vector tI,
                vector tP_pred,
                real rho,
                real alpha,
                real R,
                real C
){
  int nP = rows(tP_pred);
  vector[nP] tP = tP_pred;
  int nI = rows(tI);
  matrix[nP, nI] KPI;

  // KPI
  for (i in 1:nP){
    for (j in 1:nI){
      KPI[i, j] = 0.1e1 / R * exp(-pow(tP[i] - tI[j], 0.2e1) * pow(rho, -0.2e1))
      + 0.2e1 * C * (tP[i] - tI[j])* pow(rho, -0.2e1)
      * exp(-pow(tP[i] - tI[j], 0.2e1) * pow(rho, -0.2e1));
      KPI[i, j] = pow(alpha, 0.2e1) * KPI[i, j];
    }
  }
  return KPI;
}

matrix KIP_pred(vector tI_pred,
                vector tP,
                real rho,
                real alpha,
                real R,
                real C){
  int nI = rows(tI_pred);
  vector[nI] tI = tI_pred;
  int nP = rows(tP);
  matrix[nI, nP] KIP;

  for (i in 1:nI){
    for (j in 1:nP){
      KIP[i, j] = 0.1e1 / R * exp(-pow(tI[i] - tP[j], 0.2e1) * pow(rho, -0.2e1))
      - 0.2e1 * C * (tI[i] - tP[j]) * pow(rho, -0.2e1)
      * exp(-pow(tI[i] - tP[j], 0.2e1) * pow(rho, -0.2e1));
```

```
      KIP[i, j] = pow(alpha, 0.2e1) * KIP[i, j];
    }
  }
  return KIP;
}

matrix KI_pred(vector tI_pred,
               vector tI,
               real rho,
               real alpha,
               real R,
               real C){
  int nI = rows(tI);
  int nI_pred = rows(tI_pred);
  matrix[nI_pred, nI] KI;

  // KI
  for (i in 1:nI_pred){
    for (j in 1:nI){
      KI[i, j] =
        pow(R, -0.2e1) * exp(-pow(tI_pred[i] - tI[j], 0.2e1) * pow(rho, -0.2e1))
      + C * C * (0.2e1 * pow(rho, -0.2e1) * exp(-pow(tI_pred[i] - tI[j], 0.2e1)
                                                * pow(rho, -0.2e1)) - 0.4e1 * pow(tI_pred[i] - tI[j], 0
                 * pow(rho, -0.4e1) * exp(-pow(tI_pred[i] - tI[j], 0.2e1) * pow(rho, -0.2e1)));
      KI[i, j] = pow(alpha, 0.2e1) * KI[i, j];
    }
  }
  return KI;
}

vector Pgp_pred_rng(vector tP,
                    vector tI,
                    vector yP,
                    vector yI,
                    vector tP_pred,
                    real alpha,
                    real rho,
                    real R,
                    real C,
                    real sigmaP,
                    real sigmaI) {
  int nP = rows(tP);
  int nI = rows(tI);
  int N = nP + nI;
  int nP_pred = rows(tP_pred);
  vector[nP_pred] f2;
  {
    matrix[N, N] L_K;
    vector[N] K_div_y;
    matrix[nP_pred, N] qT_p;
    matrix[N, nP_pred] v_pred;
    vector[nP_pred] f2_mu;
    matrix[nP_pred, nP_pred] cov_f2;
    matrix[nP_pred, nP_pred] diag_delta;
```

```
    vector[N] y;
    y[1:nP] = yP[1:nP];
    y[(nP+1):N] = yI[1:nI];

    L_K =  K_wk2(tP, tI, rho, alpha, sigmaP, sigmaI, R, C);
    K_div_y = mdivide_left_tri_low(L_K, y);
    K_div_y = mdivide_right_tri_low(K_div_y', L_K)';

    qT_p[1:nP_pred, 1:nP] = KP_pred(tP, tP_pred, rho, alpha);
    qT_p[1:nP_pred, (nP+1):N] = KPI_pred(tI, tP_pred, rho, alpha, R, C);
    f2_mu = (qT_p * K_div_y);
    v_pred = mdivide_left_tri_low(L_K, qT_p');
    cov_f2 = KP_pred(tP_pred, tP_pred, rho, alpha)- v_pred' * v_pred;
    diag_delta = diag_matrix(rep_vector(1e-6, nP_pred));

    f2 = multi_normal_rng(f2_mu, cov_f2 + diag_delta);
  }
  return f2;
}

vector Igp_pred_rng(vector tP,
                    vector tI,
                    vector yP,
                    vector yI,
                    vector tI_pred,
                    real alpha,
                    real rho,
                    real R,
                    real C,
                    real sigmaP,
                    real sigmaI) {
  int nP = rows(tP);
  int nI = rows(tI);
  int N = nP + nI;
  int nI_pred = rows(tI_pred);
  vector[nI_pred] f2;
  {
    matrix[N, N] L_K;
    vector[N] K_div_y;
    matrix[nI_pred, N] qT_I;
    matrix[N, nI_pred] v_pred;
    vector[nI_pred] f2_mu;
    matrix[nI_pred, nI_pred] cov_f2;
    matrix[nI_pred, nI_pred] diag_delta;
    vector[N] y;

    y[1:nP] = yP[1:nP];
    y[(nP+1):N] = yI[1:nI];

    L_K = K_wk2(tP, tI, rho, alpha, sigmaP, sigmaI, R, C);
    K_div_y = mdivide_left_tri_low(L_K, y);
    K_div_y = mdivide_right_tri_low(K_div_y', L_K)';
    qT_I[1:nI_pred, 1:nP] = KIP_pred(tI_pred, tP, rho, alpha, R, C);
    qT_I[1:nI_pred, (nP+1):N] = KI_pred(tI_pred, tI, rho, alpha, R, C);
```

```
      f2_mu = (qT_I * K_div_y);
      v_pred = mdivide_left_tri_low(L_K, qT_I');
      cov_f2 = KI_pred(tI_pred, tI_pred, rho, alpha, R, C) - v_pred' * v_pred;
      diag_delta = diag_matrix(rep_vector(1e-6, nI_pred));

      f2 = multi_normal_rng(f2_mu, cov_f2 + diag_delta);
    }
    return f2;
  }
}
data {
  int<lower=1> nP;
  int<lower=1> nI;
  int<lower=1> nP_pred;
  int<lower=1> nI_pred;
  vector[nP] tP;
  vector[nI] tI;
  vector[nP_pred] tP_pred;
  vector[nI_pred] tI_pred;
  vector[nP] yP;
  vector[nI] yI;

  // posterior samples
  int N_samples;
  vector[N_samples] rho;
  vector[N_samples] alpha;
  vector[N_samples] sigmaP;
  vector[N_samples] sigmaI;
  vector[N_samples] R;
  vector[N_samples] C;

}
parameters {
}
model {
}
generated quantities {
  vector[nP_pred] f_P;
  matrix[N_samples, nP_pred] y_P;
  vector[nI_pred] f_I;
  matrix[N_samples, nI_pred] y_I;

  for(n in 1:N_samples) {
    f_P = Pgp_pred_rng(tP, tI, yP, yI, tP_pred[1:nP_pred], alpha[n], rho[n], R[n], C[n], sigmaP[n], sig
    for(np in 1:nP_pred) {
      y_P[n, np] = normal_rng(f_P[np], sigmaP[n]);
    }
  }

  for(n in 1:N_samples) {
    f_I = Igp_pred_rng(tP, tI, yP, yI, tI_pred[1:nI_pred], alpha[n], rho[n], R[n], C[n], sigmaP[n], sig
    for(np in 1:nI_pred) {
      y_I[n, np] = normal_rng(f_I[np], sigmaI[n]);
    }
```

```
  }
}
```

We predict for each individual $m = 1, \ldots, 9$

```
#-----------------------------------------------------------------
# no-without delta
#-----------------------------------------------------------------
post_nnd=l_df_nnd
N_samples = nrow(post_nnd[[1]])
# no-without delta model
for (i in 1:Ns){
  data_pred = list(nP=nc*nP, nI=nc*nI, tP=tP[i,], tI=tI[i,], yP=yP[i,], yI=yI[i,]
                   , tP_pred = tP_pred, tI_pred=tI_pred, nP_pred=nP_pred, nI_pred=nI_pred
                   , alpha=post_nnd[[i]]$alpha, rho=post_nnd[[i]]$rho
                   , sigmaP=post_nnd[[i]]$sigmaP, sigmaI=post_nnd[[i]]$sigmaI
                   , R=post_nnd[[i]]$R, C=post_nnd[[i]]$C, N_samples=N_samples
  )

  pred = stan(file = "STAN/WK2/WK2_nodelta_pred.stan",
              data = data_pred,
              chains = 1, iter = 1, seed=123,
              algorithm = "Fixed_param")
  ex_pred=extract(pred)
  Pvec = ex_pred$y_P
  Ivec = ex_pred$y_I
  P_post=matrix(NA, ncol = 4, nrow = nP_pred)
  # calculate mean and 95% CIs
  for(n in 1:nP_pred){
    ind=(1 +(n-1)*N_samples):(n*N_samples)
    v = Pvec[ind]
    P_post[n,] = c(mean(v), quantile(v, probs = 0.025), quantile(v, probs = 0.975),tP_pred[n])
  }
  I_post=matrix(NA, ncol = 4, nrow = nI_pred)
  for(n in 1:nI_pred){
    ind=(1 +(n-1)*N_samples):(n*N_samples)
    v = Ivec[ind]
    I_post[n,] = c(mean(v), quantile(v, probs = 0.025), quantile(v, probs = 0.975),tI_pred[n])
  }

  lP_pr[[i]]=P_post
  lI_pr[[i]]=I_post
}
```

```
SAMPLING FOR MODEL 'WK2_nodelta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.26926 seconds (Sampling)
Chain 1:                3.26926 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_nodelta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
```

```
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.31596 seconds (Sampling)
Chain 1:                3.31596 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_nodelta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.21595 seconds (Sampling)
Chain 1:                3.21595 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_nodelta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.22498 seconds (Sampling)
Chain 1:                3.22498 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_nodelta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.24587 seconds (Sampling)
Chain 1:                3.24587 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_nodelta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.28681 seconds (Sampling)
Chain 1:                3.28681 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_nodelta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.28492 seconds (Sampling)
Chain 1:                3.28492 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_nodelta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.44135 seconds (Sampling)
Chain 1:                3.44135 seconds (Total)
Chain 1:
```

```
SAMPLING FOR MODEL 'WK2_nodelta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.3439 seconds (Sampling)
Chain 1:                3.3439 seconds (Total)
Chain 1:
```

```
# prepare the data for plotting
ID = paste0(Zvec, " | ID = ", 1:Ns)
R1 = paste("expression(R[1] = )", Zvec)
P_nnd = data.frame(do.call(rbind, lP_pr))
P_nnd$ID = rep(ID, each = nP_pred)
P_nnd$t = rep(tP_pred, Ns)
colnames(P_nnd)[1:4] = c("mean", "lower", "upper", "time")
P_nnd$sharing = "no-without delta"

I_nnd = data.frame(do.call(rbind, lI_pr))
I_nnd$ID = rep(ID, each = nI_pred)
I_nnd$t = rep(tI_pred, Ns)
colnames(I_nnd)[1:4] = c("mean", "lower", "upper", "time")
I_nnd$sharing = "no-without delta"
```

Model 2 (no-with delta in paper, Figure 6)

Now we account for model discrepancy $\delta_m(x_m) \sim GP(0, K_\delta(x_m, x'_m))$, where we use the squared exponential kernel $K_\delta(x_m, x'_m) = \alpha_m^2 \exp\left(-\frac{(x_m - x'_m)^2}{2\rho_m^2}\right)$. More details on the prediction equations are given in the Appendix C.2.

Stan code:

```
writeLines(readLines("STAN/WK2/WK2_delta_pred.stan"))
```

```
functions {
  vector mu_fn(real mu_wk2,
               real R,
               int nP,
               int nI){
                 vector[nP]  mP = rep_vector(mu_wk2,nP);
                 vector[nI]  mI = rep_vector((1/R)*mu_wk2,nI);
                 vector[nP + nI] mu;
                 mu= append_row(mP, mI);
                 return(mu);
  }

  matrix K_wk2(vector tP,
               vector tI,
               real rho,
               real alpha,
               real sigmaP,
               real sigmaI,
               real rho_d,
               real alpha_d,
               real R,
```

```
           real C) {
 int nP = rows(tP);
 int nI = rows(tI);
 matrix[nP + nI, nP + nI] K;
 matrix[nP, nP] KB;

 // KP
 for (i in 1:(nP-1)){
   K[i,i] = pow(alpha, 0.2e1);
   for (j in (i+1):nP){
     K[i,j] = exp(-pow(tP[i] - tP[j], 0.2e1) * pow(rho, -0.2e1));
     K[i,j] = pow(alpha, 0.2e1) * K[i,j];
     K[j,i] = K[i,j];
   }
   K[nP,nP] = pow(alpha, 0.2e1);
 }
 K[1:nP, 1:nP] = K[1:nP, 1:nP] + diag_matrix(rep_vector(pow(sigmaP, 0.2e1), nP));

 // press_Bias
 for (i in 1:(nP-1)){
   KB[i,i] = pow(alpha_d, 0.2e1);
   for (j in (i+1):nP){
     KB[i,j] = exp(-pow(tP[i] - tP[j], 0.2e1) * pow(rho_d, -0.2e1));
     KB[i,j] = pow(alpha_d, 0.2e1) * KB[i,j];
     KB[j,i] = KB[i,j];
   }
   KB[nP,nP] = pow(alpha_d, 0.2e1);
 }
 K[1:nP, 1:nP] = K[1:nP, 1:nP] + KB[1:nP, 1:nP];

 // KPI
 for (i in 1:nP){
   for (j in 1:nI){
     K[i, nP + j] = 0.1e1 / R * exp(-pow(tP[i] - tI[j], 0.2e1) * pow(rho, -0.2e1))
     + 0.2e1 * C * (tP[i] - tI[j])* pow(rho, -0.2e1)
     * exp(-pow(tP[i] - tI[j], 0.2e1) * pow(rho, -0.2e1));
     K[i, nP + j] = pow(alpha, 0.2e1) * K[i, nP + j];
   }
 }

 // KIP (KIP = KPI')
 //K[(nP + 1):(nP + nI), 1:nP] = K[1:nP, (nP + 1):(nP + nI)]';
 for (i in 1:nI){
   for (j in 1:nP){
    K[nP + i, j] = 0.1e1 / R * exp(-pow(tI[i] - tP[j], 0.2e1) * pow(rho, -0.2e1))
                   - 0.2e1 * C * (tI[i] - tP[j]) * pow(rho, -0.2e1)
                   * exp(-pow(tI[i] - tP[j], 0.2e1) * pow(rho, -0.2e1));
    K[nP + i, j] = pow(alpha, 0.2e1) * K[nP + i, j];
  }
}
// KI
 for (i in 1:(nI-1)){
  K[nP + i, nP +i] =
             pow(R, -0.2e1) * exp(-pow(tI[i] - tI[i], 0.2e1) * pow(rho, -0.2e1))
```

```
            + C * C * (0.2e1 * pow(rho, -0.2e1) * exp(-pow(tI[i] - tI[i], 0.2e1)
                                                    * pow(rho, -0.2e1)) - 0.4e1 * pow(tI[i] - tI[i], 
                       * pow(rho, -0.4e1) * exp(-pow(tI[i] - tI[i], 0.2e1) * pow(rho, -0.2e1)));
   K[nP + i, nP +i] = pow(alpha, 0.2e1) * K[nP + i, nP +i];
   for (j in (i+1):nI){
    K[nP + i, nP +j] =
                pow(R, -0.2e1) * exp(-pow(tI[i] - tI[j], 0.2e1) * pow(rho, -0.2e1))
                + C * C * (0.2e1 * pow(rho, -0.2e1) * exp(-pow(tI[i] - tI[j], 0.2e1)
                * pow(rho, -0.2e1)) - 0.4e1 * pow(tI[i] - tI[j], 0.2e1)
                * pow(rho, -0.4e1) * exp(-pow(tI[i] - tI[j], 0.2e1) * pow(rho, -0.2e1)));
    K[nP + i, nP +j] = pow(alpha, 0.2e1) * K[nP + i, nP +j];
    K[nP + j, nP +i] = K[nP + i, nP +j];
   }
   K[nP + nI, nP +nI] =
              pow(R, -0.2e1) * exp(-pow(tI[nI] - tI[nI], 0.2e1) * pow(rho, -0.2e1))
              + C * C * (0.2e1 * pow(rho, -0.2e1) * exp(-pow(tI[nI] - tI[nI], 0.2e1)
              * pow(rho, -0.2e1)) - 0.4e1 * pow(tI[nI] - tI[nI], 0.2e1)
              * pow(rho, -0.4e1) * exp(-pow(tI[nI] - tI[nI], 0.2e1) * pow(rho, -0.2e1)));
   K[nP + nI, nP +nI] = pow(alpha, 0.2e1) * K[nP + nI, nP +nI];
   }
   K[(nP + 1):(nP + nI), (nP + 1):(nP + nI)] = K[(nP + 1):(nP + nI), (nP + 1):(nP + nI)]
          + diag_matrix(rep_vector(pow(sigmaI, 0.2e1), nI));
   return cholesky_decompose(K);
}
matrix KP_pred(vector tP,
               vector tP_pred,
               real rho,
               real alpha
){
  int nP = rows(tP);
  int nP_pred = rows(tP_pred);
  matrix[nP_pred, nP] KP;

  for (i in 1:nP_pred){
    for (j in 1:nP){
      KP[i,j] = exp(-pow(tP_pred[i] - tP[j], 0.2e1) * pow(rho, -0.2e1));
      KP[i,j] = pow(alpha, 0.2e1) * KP[i,j];
    }
  }
  return KP;
}


matrix KP_Bias(vector tP,
               vector tP_pred,
               real rho_d,
               real alpha_d
   ){
     int nP = rows(tP);
     int nP_pred = rows(tP_pred);
     matrix[nP_pred, nP] KB;
     for (i in 1:nP_pred){
       for (j in 1:nP){
       KB[i,j] = exp(-pow(tP_pred[i] - tP[j], 0.2e1) * pow(rho_d, -0.2e1));
```

```
       KB[i,j] = pow(alpha_d, 0.2e1) * KB[i,j];
       }
     }
     return KB;
}

matrix KPI_pred(vector tI,
                vector tP_pred,
                real rho,
                real alpha,
                real R,
                real C
){
  int nP = rows(tP_pred);
  vector[nP] tP = tP_pred;
  int nI = rows(tI);
  matrix[nP, nI] KPI;

  // KPI
  for (i in 1:nP){
    for (j in 1:nI){
      KPI[i, j] = 0.1e1 / R * exp(-pow(tP[i] - tI[j], 0.2e1) * pow(rho, -0.2e1))
      + 0.2e1 * C * (tP[i] - tI[j])* pow(rho, -0.2e1)
      * exp(-pow(tP[i] - tI[j], 0.2e1) * pow(rho, -0.2e1));
      KPI[i, j] = pow(alpha, 0.2e1) * KPI[i, j];
    }
  }
  return KPI;
}

matrix KIP_pred(vector tI_pred,
                vector tP,
                real rho,
                real alpha,
                real R,
                real C){
  int nI = rows(tI_pred);
  vector[nI] tI = tI_pred;
  int nP = rows(tP);
  matrix[nI, nP] KIP;

  for (i in 1:nI){
    for (j in 1:nP){
      KIP[i, j] = 0.1e1 / R * exp(-pow(tI[i] - tP[j], 0.2e1) * pow(rho, -0.2e1))
      - 0.2e1 * C * (tI[i] - tP[j]) * pow(rho, -0.2e1)
      * exp(-pow(tI[i] - tP[j], 0.2e1) * pow(rho, -0.2e1));
      KIP[i, j] = pow(alpha, 0.2e1) * KIP[i, j];
    }
  }
  return KIP;
}

matrix KI_pred(vector tI_pred,
               vector tI,
```

```
              real rho,
              real alpha,
              real R,
              real C){
  int nI = rows(tI);
  int nI_pred = rows(tI_pred);
  matrix[nI_pred, nI] KI;

  // KI
  for (i in 1:nI_pred){
    for (j in 1:nI){
      KI[i, j] =
        pow(R, -0.2e1) * exp(-pow(tI_pred[i] - tI[j], 0.2e1) * pow(rho, -0.2e1))
      + C * C * (0.2e1 * pow(rho, -0.2e1) * exp(-pow(tI_pred[i] - tI[j], 0.2e1)
                                              * pow(rho, -0.2e1)) - 0.4e1 * pow(tI_pred[i] - tI[j], 0
                 * pow(rho, -0.4e1) * exp(-pow(tI_pred[i] - tI[j], 0.2e1) * pow(rho, -0.2e1)));
      KI[i, j] = pow(alpha, 0.2e1) * KI[i, j];
    }
  }
  return KI;
}

vector Pgp_pred_rng(vector tP,
                    vector tI,
                    vector yP,
                    vector yI,
                    vector tP_pred,
                    real alpha,
                    real rho,
                    real rho_d,
                    real alpha_d,
                    real R,
                    real C,
                    real sigmaP,
                    real sigmaI) {
  int nP = rows(tP);
  int nI = rows(tI);
  int N = nP + nI;
  int nP_pred = rows(tP_pred);
  vector[nP_pred] f2;
  {
    matrix[N, N] L_K;
    vector[N] K_div_y;
    matrix[nP_pred, N] qT_p;
    matrix[N, nP_pred] v_pred;
    vector[nP_pred] f2_mu;
    matrix[nP_pred, nP_pred] cov_f2;
    matrix[nP_pred, nP_pred] diag_delta;
    vector[N] y;
    y[1:nP] = yP[1:nP];
    y[(nP+1):N] = yI[1:nI];

    L_K =  K_wk2(tP, tI, rho, alpha, sigmaP, sigmaI, rho_d, alpha_d, R, C);
    K_div_y = mdivide_left_tri_low(L_K, y);
```

```
    K_div_y = mdivide_right_tri_low(K_div_y', L_K)';

    qT_p[1:nP_pred, 1:nP] = KP_pred(tP, tP_pred, rho, alpha)+ KP_Bias(tP, tP_pred, rho_d, alpha_d);
    qT_p[1:nP_pred, (nP+1):N] = KPI_pred(tI, tP_pred, rho, alpha, R, C);
    f2_mu = (qT_p * K_div_y);
    v_pred = mdivide_left_tri_low(L_K, qT_p');
    cov_f2 = KP_pred(tP_pred, tP_pred, rho, alpha) + KP_Bias(tP_pred, tP_pred, rho_d, alpha_d) - v_pr
    diag_delta = diag_matrix(rep_vector(1e-6, nP_pred));

    f2 = multi_normal_rng(f2_mu, cov_f2 + diag_delta);
  }
  return f2;
}

vector Igp_pred_rng(vector tP,
                    vector tI,
                    vector yP,
                    vector yI,
                    vector tI_pred,
                    real alpha,
                    real rho,
                    real rho_d,
                    real alpha_d,
                    real R,
                    real C,
                    real sigmaP,
                    real sigmaI) {
  int nP = rows(tP);
  int nI = rows(tI);
  int N = nP + nI;
  int nI_pred = rows(tI_pred);
  vector[nI_pred] f2;
  {
    matrix[N, N] L_K;
    vector[N] K_div_y;
    matrix[nI_pred, N] qT_I;
    matrix[N, nI_pred] v_pred;
    vector[nI_pred] f2_mu;
    matrix[nI_pred, nI_pred] cov_f2;
    matrix[nI_pred, nI_pred] diag_delta;
    vector[N] y;

    y[1:nP] = yP[1:nP];
    y[(nP+1):N] = yI[1:nI];

    L_K = K_wk2(tP, tI, rho, alpha, sigmaP, sigmaI, rho_d, alpha_d, R, C);
    K_div_y = mdivide_left_tri_low(L_K, y);
    K_div_y = mdivide_right_tri_low(K_div_y', L_K)';
    qT_I[1:nI_pred, 1:nP] = KIP_pred(tI_pred, tP, rho, alpha, R, C);
    qT_I[1:nI_pred, (nP+1):N] = KI_pred(tI_pred, tI, rho, alpha, R, C);
    f2_mu = (qT_I * K_div_y);
    v_pred = mdivide_left_tri_low(L_K, qT_I');
    cov_f2 = KI_pred(tI_pred, tI_pred, rho, alpha, R, C) - v_pred' * v_pred;
    diag_delta = diag_matrix(rep_vector(1e-6, nI_pred));
```

```
      f2 = multi_normal_rng(f2_mu, cov_f2 + diag_delta);
    }
    return f2;
  }
}
data {
  int<lower=1> nP;
  int<lower=1> nI;
  int<lower=1> nP_pred;
  int<lower=1> nI_pred;
  vector[nP] tP;
  vector[nI] tI;
  vector[nP_pred] tP_pred;
  vector[nI_pred] tI_pred;
  vector[nP] yP;
  vector[nI] yI;

  // posterior samples
  int N_samples;
  vector[N_samples] rho;
  vector[N_samples] alpha;
  vector[N_samples] sigmaP;
  vector[N_samples] sigmaI;
  vector[N_samples] rho_d;
  vector[N_samples] alpha_d;
  vector[N_samples] R;
  vector[N_samples] C;

}
parameters {
}
model {
}
generated quantities {
  vector[nP_pred] f_P;
  matrix[N_samples, nP_pred] y_P;
  vector[nI_pred] f_I;
  matrix[N_samples, nI_pred] y_I;

  for(n in 1:N_samples) {
    f_P = Pgp_pred_rng(tP, tI, yP, yI, tP_pred[1:nP_pred], alpha[n], rho[n], rho_d[n], alpha_d[n], R[n]
    for(np in 1:nP_pred) {
      y_P[n, np] = normal_rng(f_P[np], sigmaP[n]);
    }
  }

  for(n in 1:N_samples) {
    f_I = Igp_pred_rng(tP, tI, yP, yI, tI_pred[1:nI_pred], alpha[n], rho[n], rho_d[n], alpha_d[n], R[n]
    for(np in 1:nI_pred) {
      y_I[n, np] = normal_rng(f_I[np], sigmaI[n]);
    }
  }
}
```

We predict for each individual $m = 1, \ldots, 9$

```
post_nwd=l_df_nwd
N_samples = nrow(post_nwd[[1]])
# no-with delta model
for (i in 1:Ns){
  data_pred = list(nP=nc*nP, nI=nc*nI, tP=tP[i,], tI=tI[i,], yP=yP[i,], yI=yI[i,]
                   , tP_pred = tP_pred, tI_pred=tI_pred, nP_pred=nP_pred, nI_pred=nI_pred
                   , alpha=post_nwd[[i]]$alpha, rho=post_nwd[[i]]$rho, alpha_d=post_nwd[[i]]$alpha_d
                   , rho_d=post_nwd[[i]]$rho_d, sigmaP=post_nwd[[i]]$sigmaP, sigmaI=post_nwd[[i]]$sigmaI
                   , R=post_nwd[[i]]$R, C=post_nwd[[i]]$C, N_samples=N_samples
  )

  pred = stan(file = "STAN/WK2/WK2_delta_pred.stan",
              data = data_pred,
              chains = 1, iter = 1, seed=123,
              algorithm = "Fixed_param")
  ex_pred=extract(pred)
  Pvec = ex_pred$y_P
  Ivec = ex_pred$y_I
  P_post=matrix(NA, ncol = 4, nrow = nP_pred)
  for(n in 1:nP_pred){
    ind=(1 +(n-1)*N_samples):(n*N_samples)
    v = Pvec[ind]
    P_post[n,] = c(mean(v), quantile(v, probs = 0.025), quantile(v, probs = 0.975),tP_pred[n])
  }
  I_post=matrix(NA, ncol = 4, nrow = nI_pred)
  for(n in 1:nI_pred){
    ind=(1 +(n-1)*N_samples):(n*N_samples)
    v = Ivec[ind]
    I_post[n,] = c(mean(v), quantile(v, probs = 0.025), quantile(v, probs = 0.975),tI_pred[n])
  }

  lP_pr[[i]]=P_post
  lI_pr[[i]]=I_post
}
```

```
SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.75802 seconds (Sampling)
Chain 1:                3.75802 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.65189 seconds (Sampling)
Chain 1:                3.65189 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
```

```
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.68313 seconds (Sampling)
Chain 1:                3.68313 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.60522 seconds (Sampling)
Chain 1:                3.60522 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.79806 seconds (Sampling)
Chain 1:                3.79806 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.80969 seconds (Sampling)
Chain 1:                3.80969 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.71955 seconds (Sampling)
Chain 1:                3.71955 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.56581 seconds (Sampling)
Chain 1:                3.56581 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.72944 seconds (Sampling)
Chain 1:                3.72944 seconds (Total)
Chain 1:
```

```
# prepare data for plotting
# ID = paste0(Zvec, " | ID = ", 1:Ns)
P_nwd = data.frame(do.call(rbind, lP_pr))
P_nwd$ID = rep(ID, each = nP_pred)
P_nwd$t = rep(tP_pred, Ns)
colnames(P_nwd)[1:4] = c("mean", "lower", "upper", "time")
P_nwd$sharing = "no-with delta"

I_nwd = data.frame(do.call(rbind, lI_pr))
I_nwd$ID = rep(ID, each = nI_pred)
I_nwd$t = rep(tI_pred, Ns)
colnames(I_nwd)[1:4] = c("mean", "lower", "upper", "time")
I_nwd$sharing = "no-with delta"
```

Model 3 (yes/common delta, Figure 6)

We allow individuals to share information about the physical parameters $u_m, m = 1, 2, \ldots, 10$ through a global level parameter as described in Section 3.2. The model assumes same discrepancy parameters for all individuals.

We predict using the same equations as before, but now with the posterior distributions obtained by the fit of Model 3 (yes/common delta).

```
post_ycd = list()

for(i in 1:Ns){
  v=c(paste0("alpha.",i), paste0("rho.",i), "alpha_d", "rho_d", "sigmaP", "sigmaI", paste0("R.",i), pas
  df_temp=df_ycd[, v]
  colnames(df_temp)=c("alpha", "rho", "alpha_d", "rho_d", "sigmaP", "sigmaI", "R", "C")
  post_ycd[[i]]=df_temp
}

# yes/common delta model
for (i in 1:Ns){
  data_pred = list(nP=nc*nP, nI=nc*nI, tP=tP[i,], tI=tI[i,], yP=yP[i,], yI=yI[i,]
                   , tP_pred = tP_pred, tI_pred=tI_pred, nP_pred=nP_pred, nI_pred=nI_pred
                   , alpha=post_ycd[[i]]$alpha, rho=post_ycd[[i]]$rho, alpha_d=post_ycd[[i]]$alpha_d
                   , rho_d=post_ycd[[i]]$rho_d, sigmaP=post_ycd[[i]]$sigmaP, sigmaI=post_ycd[[i]]$sigma
                   , R=post_ycd[[i]]$R, C=post_ycd[[i]]$C, N_samples=N_samples
  )

  pred = stan(file = "STAN/WK2/WK2_delta_pred.stan",
              data = data_pred,
              chains = 1, iter = 1, seed=123,
              algorithm = "Fixed_param")
  ex_pred=extract(pred)
  Pvec = ex_pred$y_P
  Ivec = ex_pred$y_I
  P_post=matrix(NA, ncol = 4, nrow = nP_pred)
  for(n in 1:nP_pred){
    ind=(1 +(n-1)*N_samples):(n*N_samples)
    v = Pvec[ind]
    P_post[n,] = c(mean(v), quantile(v, probs = 0.025), quantile(v, probs = 0.975),tP_pred[n])
```

```
  }
  I_post=matrix(NA, ncol = 4, nrow = nI_pred)
  for(n in 1:nI_pred){
    ind=(1 +(n-1)*N_samples):(n*N_samples)
    v = Ivec[ind]
    I_post[n,] = c(mean(v), quantile(v, probs = 0.025), quantile(v, probs = 0.975),tI_pred[n])
  }

  lP_pr[[i]]=P_post
  lI_pr[[i]]=I_post
}
```

```
SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.72052 seconds (Sampling)
Chain 1:                3.72052 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.81045 seconds (Sampling)
Chain 1:                3.81045 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.84672 seconds (Sampling)
Chain 1:                3.84672 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.56196 seconds (Sampling)
Chain 1:                3.56196 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.85698 seconds (Sampling)
Chain 1:                3.85698 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
```

```
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.78668 seconds (Sampling)
Chain 1:                3.78668 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.60449 seconds (Sampling)
Chain 1:                3.60449 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.7046 seconds (Sampling)
Chain 1:                3.7046 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.66531 seconds (Sampling)
Chain 1:                3.66531 seconds (Total)
Chain 1:
```

```r
# ID = paste0(Zvec, "| ID = ", 1:Ns)
P_ycd = data.frame(do.call(rbind, lP_pr))
P_ycd$ID = rep(ID, each = nP_pred)
P_ycd$t = rep(tP_pred, Ns)
colnames(P_ycd)[1:4] = c("mean", "lower", "upper", "time")
P_ycd$sharing = "yes/common delta"

I_ycd = data.frame(do.call(rbind, lI_pr))
I_ycd$ID = rep(ID, each = nI_pred)
I_ycd$t = rep(tI_pred, Ns)
colnames(I_ycd)[1:4] = c("mean", "lower", "upper", "time")
I_ycd$sharing = "yes/common delta"
```

Model 4 (yes/shared delta, Figure 6)

We allow individuals to share information about both the physical parameters $u_m, m = 1, 2, \ldots, 9$ and the discrepancy through a global level parameters for both as described in Section 3.1. The model assumes same discrepancy parameters for all individuals.

We predict using the same equations as before, but now with the posterior distributions obtained by the fit of Model 4 (yes/shared delta).

```r
#-----------------------------------------------------
# yes/ shared delta
#-----------------------------------------------------------------
post_ysd = list()

for(i in 1:Ns){
  v=c(paste0("alpha.",i), paste0("rho.",i), paste0("alpha_d.",i), paste0("rho_d.",i), "sigmaP", "sigmaI
  df_temp=df_ysd[, v]
  colnames(df_temp)=c("alpha", "rho", "alpha_d", "rho_d", "sigmaP", "sigmaI", "R", "C")
  post_ysd[[i]]=df_temp
}

# yes/common delta model
for (i in 1:Ns){
  data_pred = list(nP=nc*nP, nI=nc*nI, tP=tP[i,], tI=tI[i,], yP=yP[i,], yI=yI[i,]
                   , tP_pred = tP_pred, tI_pred=tI_pred, nP_pred=nP_pred, nI_pred=nI_pred
                   , alpha=post_ysd[[i]]$alpha, rho=post_ysd[[i]]$rho, alpha_d=post_ysd[[i]]$alpha_d
                   , rho_d=post_ysd[[i]]$rho_d, sigmaP=post_ysd[[i]]$sigmaP, sigmaI=post_ysd[[i]]$sigma
                   , R=post_ysd[[i]]$R, C=post_ysd[[i]]$C, N_samples=N_samples
  )

  pred = stan(file = "STAN/WK2/WK2_delta_pred.stan",
              data = data_pred,
              chains = 1, iter = 1, seed=123,
              algorithm = "Fixed_param")
  ex_pred=extract(pred)
  Pvec = ex_pred$y_P
  Ivec = ex_pred$y_I
  P_post=matrix(NA, ncol = 4, nrow = nP_pred)
  for(n in 1:nP_pred){
    ind=(1 +(n-1)*N_samples):(n*N_samples)
    v = Pvec[ind]
    P_post[n,] = c(mean(v), quantile(v, probs = 0.025), quantile(v, probs = 0.975),tP_pred[n])
  }
  I_post=matrix(NA, ncol = 4, nrow = nI_pred)
  for(n in 1:nI_pred){
    ind=(1 +(n-1)*N_samples):(n*N_samples)
    v = Ivec[ind]
    I_post[n,] = c(mean(v), quantile(v, probs = 0.025), quantile(v, probs = 0.975),tI_pred[n])
  }

  lP_pr[[i]]=P_post
  lI_pr[[i]]=I_post
}
```

```
SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.71178 seconds (Sampling)
Chain 1:                3.71178 seconds (Total)
Chain 1:
```

```
SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.79535 seconds (Sampling)
Chain 1:                3.79535 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.7747 seconds (Sampling)
Chain 1:                3.7747 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.59384 seconds (Sampling)
Chain 1:                3.59384 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.62783 seconds (Sampling)
Chain 1:                3.62783 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.59466 seconds (Sampling)
Chain 1:                3.59466 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.61148 seconds (Sampling)
Chain 1:                3.61148 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.58803 seconds (Sampling)
Chain 1:                3.58803 seconds (Total)
```

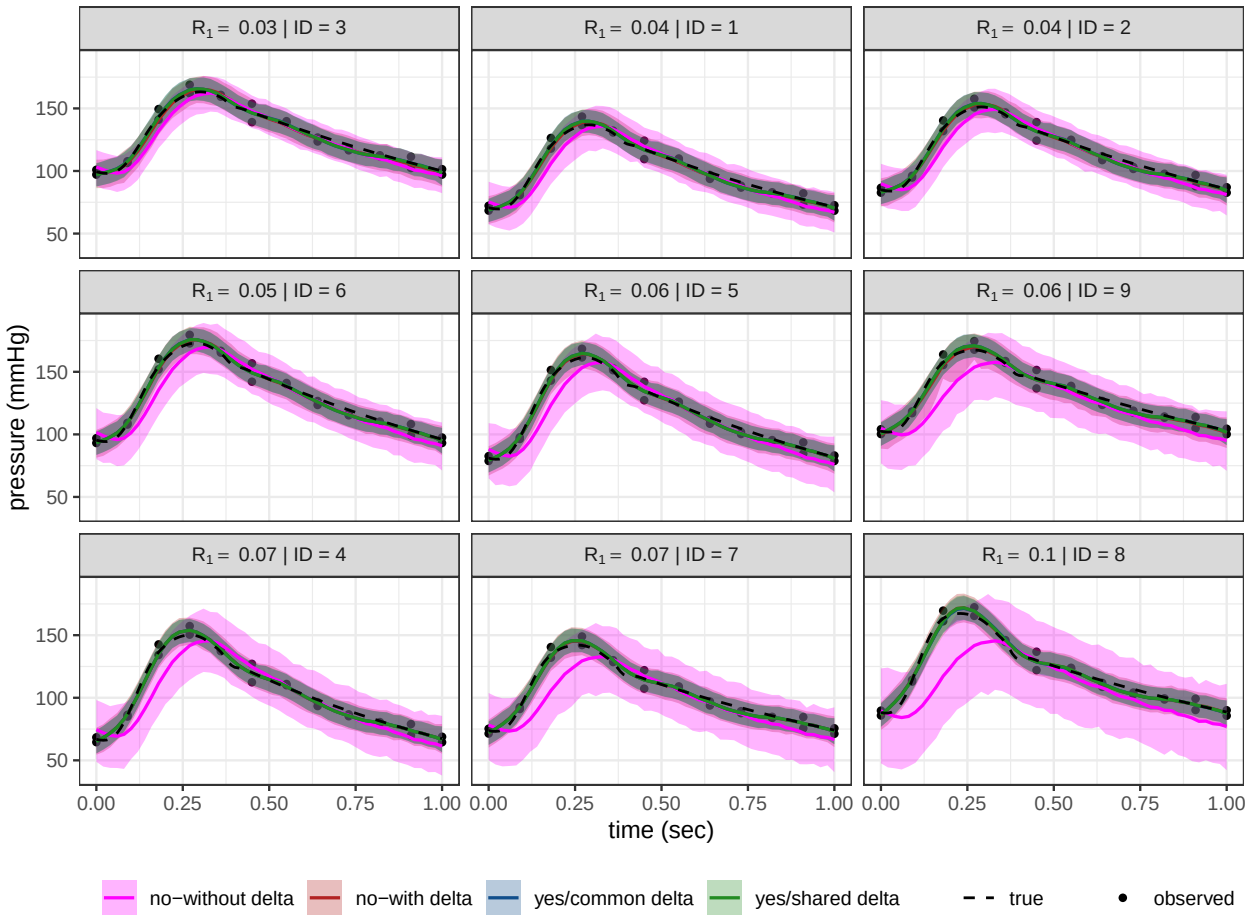
```
Chain 1:

SAMPLING FOR MODEL 'WK2_delta_pred' NOW (CHAIN 1).
Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0 seconds (Warm-up)
Chain 1:                3.6 seconds (Sampling)
Chain 1:                3.6 seconds (Total)
Chain 1:
```

```
t2=Sys.time()

# ID = paste0(Zvec, "| ID = ", 1:Ns)
P_ysd = data.frame(do.call(rbind, lP_pr))
P_ysd$ID = rep(ID, each = nP_pred)
P_ysd$t = rep(tP_pred, Ns)
colnames(P_ysd)[1:4] = c("mean", "lower", "upper", "time")
P_ysd$sharing = "yes/shared delta"

I_ysd = data.frame(do.call(rbind, lI_pr))
I_ysd$ID = rep(ID, each = nI_pred)
I_ysd$t = rep(tI_pred, Ns)
colnames(I_ysd)[1:4] = c("mean", "lower", "upper", "time")
I_ysd$sharing = "yes/shared delta"
```

Plot pressure predictions (Figure 5 in Appendix)

```
P_pred = rbind(P_nwd, P_ysd, P_ycd, P_nnd)
df_true = dat$P_true
P_true = data.frame(P=as.vector(dat$P_true), t = rep(time,Ns), ID = rep(ID, each = length(time)))

P_obs = data.frame(P=as.vector(t(yP)), t = as.vector(t(tP)), ID = rep(ID, each = nc*nP))

# New facet label names
appender = function(string) {
  TeX(paste("$R_1= $", string))
}

pl_WK_pred_P=ggplot()+
  geom_point(data=P_obs, aes(x=t, y=P, shape="observed"))+
  geom_ribbon(data = P_pred, aes(x=t,ymin=lower,ymax=upper, fill=sharing),alpha=0.3)+# alpha=0.38
  geom_line(data=P_pred, aes(x=t, y=mean, color= sharing), size=0.7)+
  geom_line(data=P_true, aes(x=t, y=P, linetype="true"), size=0.6)+
  facet_wrap(~ID, nrow = 3,labeller = as_labeller(appender, default = label_parsed))+
  theme_bw()+
  xlab("time (sec)")+ylab("pressure (mmHg)")+
  theme(legend.position = "bottom", legend.title = element_blank())+
  scale_color_manual(
    breaks=c('no-without delta','no-with delta', "yes/common delta", "yes/shared delta", "true"),
    values=c("magenta","firebrick","dodgerblue4", "forestgreen", "black")
  )+
  scale_linetype_manual(breaks=c('no-without delta','no-with delta', "yes/common delta", "yes/shared del
                        values=c(1,1,1, 1, 2)
  )+
```

```
  scale_fill_manual(
    breaks=c('no-without delta','no-with delta', "yes/common delta", "yes/shared delta"),
    values=c("magenta","firebrick","dodgerblue4", "forestgreen")
  )

pl_WK_pred_P
```




```
# ggsave("Figs/WK_pred_P.pdf", plot = pl_WK_pred_P, width = 20, height = 15, units = "cm")
```

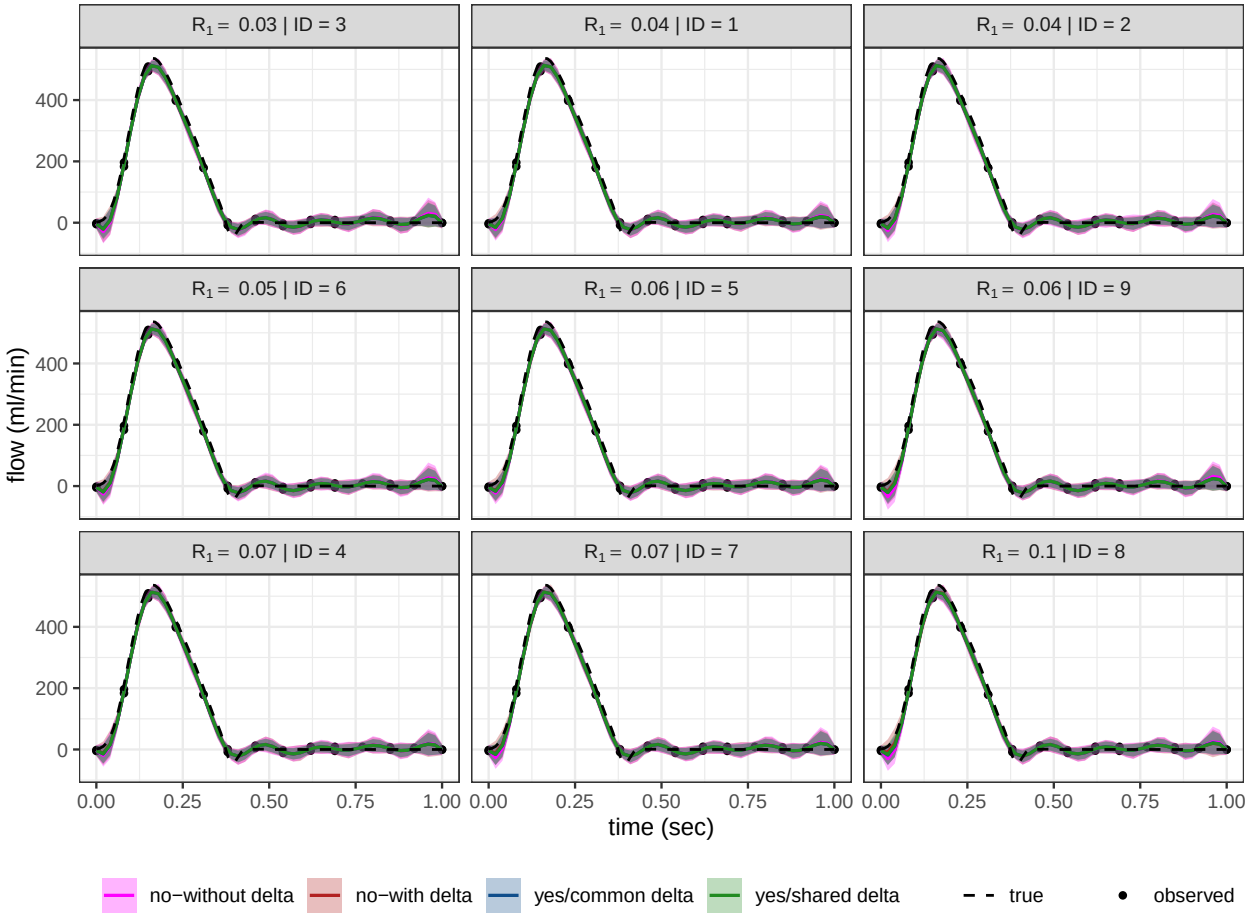
Plot flow predictions (Figure 6 in Appendix)

```
I_pred = rbind(I_nwd, I_ysd, I_ycd, I_nnd)
df_true_I = dat$I_true
I_true = data.frame(I=as.vector(flow_time$inflow), t = rep(time,Ns), ID = rep(ID, each = length(time)))

I_obs = data.frame(I=as.vector(t(yI)), t = as.vector(t(tI)), ID = rep(ID, each = nc*nI))
unique(I_pred$sharing)
```

```
[1] "no-with delta"    "yes/shared delta" "yes/common delta" "no-without delta"
```

```
pl_WK_pred_I=ggplot()+
  geom_point(data=I_obs, aes(x=t, y=I, shape="observed"))+
  geom_ribbon(data = I_pred, aes(x=t,ymin=lower,ymax=upper, fill=sharing),alpha=0.3)+# alpha=0.38
  geom_line(data=I_pred, aes(x=t, y=mean, color= sharing), size=0.7)+
```

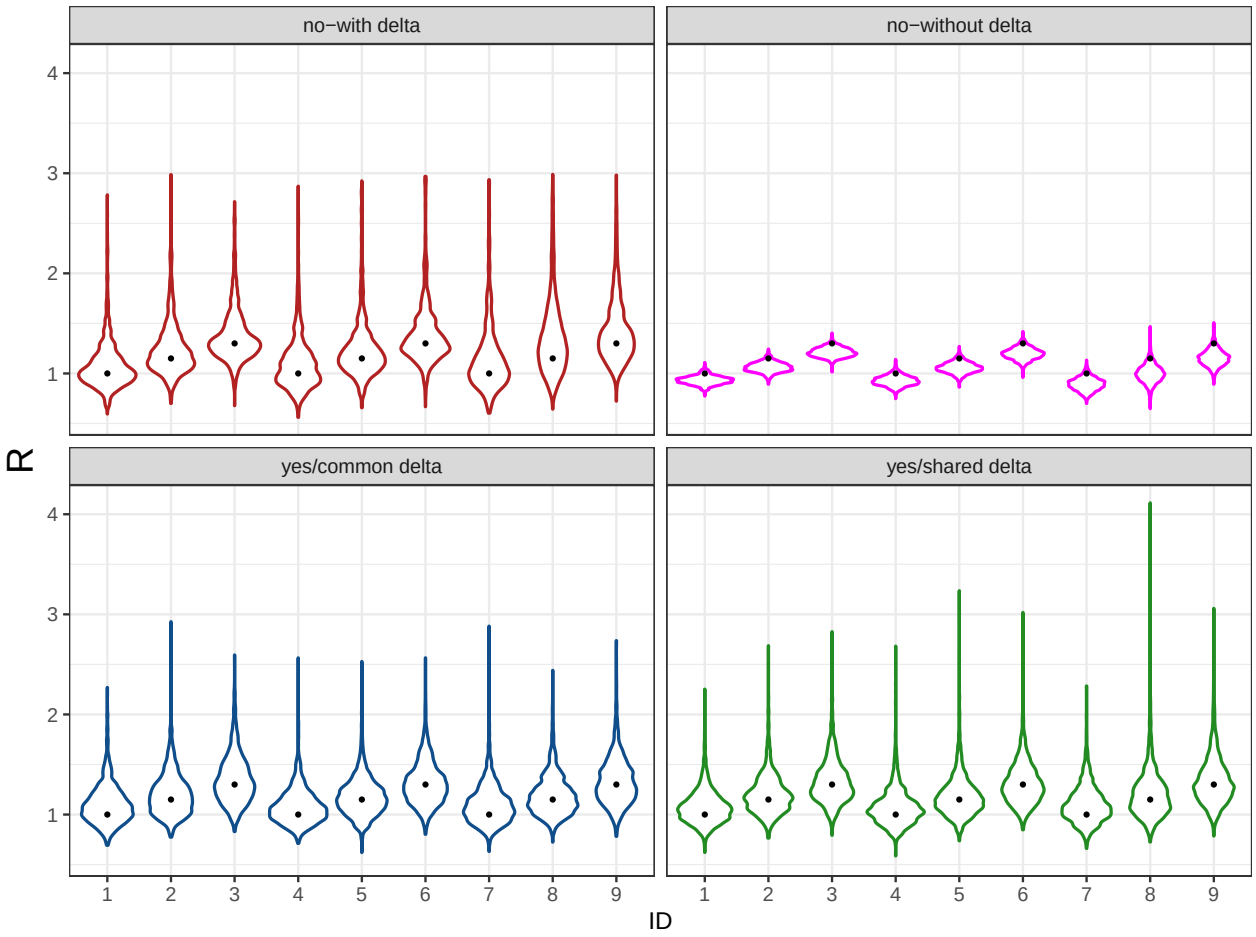
```
  geom_line(data=I_true, aes(x=t, y=I, linetype="true"), size=0.6)+
  facet_wrap(~ID, nrow = 3,labeller = as_labeller(appender, default = label_parsed))+
  theme_bw()+
  xlab("time (sec)")+ylab("flow (ml/min)")+
  theme(legend.position = "bottom", legend.title = element_blank())+
  scale_color_manual(
    breaks=c('no-without delta','no-with delta', "yes/common delta", "yes/shared delta", "true"),
    values=c("magenta","firebrick","dodgerblue4", "forestgreen", "black")
  )+
  scale_linetype_manual(breaks=c('no-without delta','no-with delta', "yes/common delta", "yes/shared del
                        values=c(1,1,1, 1, 2)
  )+
  scale_fill_manual(
    breaks=c('no-without delta','no-with delta', "yes/common delta", "yes/shared delta"),
    values=c("magenta","firebrick","dodgerblue4", "forestgreen")
  )

pl_WK_pred_I
```




```
# ggsave("Figs/WK_pred_I.pdf", plot = pl_WK_pred_I, width = 20, height = 15, units = "cm")
```

Plot posteriors of R for all individuals (Figure 3 in Appendix)

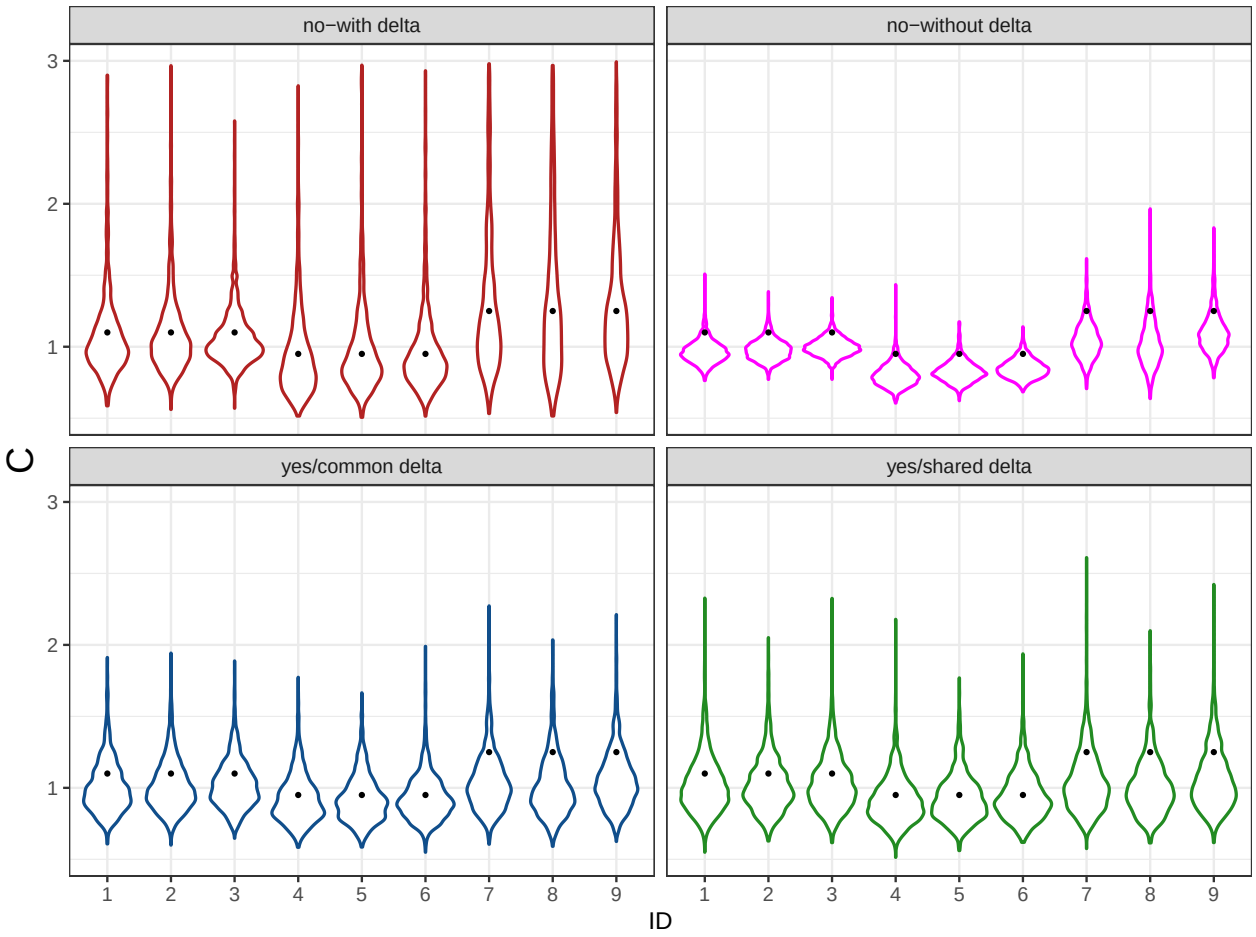
```
R_nnd = R_nwd = R_ycd = R_ysd = C_nnd = C_nwd = C_ycd = C_ysd = c()

for(i in 1:Ns){
  R_temp = post_nnd[[i]]$R
  R_nnd = c(R_nnd, R_temp)
  R_temp = post_nwd[[i]]$R
  R_nwd = c(R_nwd, R_temp)
  R_temp = post_ycd[[i]]$R
  R_ycd = c(R_ycd, R_temp)
  R_temp = post_ysd[[i]]$R
  R_ysd = c(R_ysd, R_temp)
  C_temp = post_nnd[[i]]$C
  C_nnd = c(C_nnd, C_temp)
  C_temp = post_nwd[[i]]$C
  C_nwd = c(C_nwd, C_temp)
  C_temp = post_ysd[[i]]$C
  C_ycd = c(C_ycd, C_temp)
  C_temp = post_ycd[[i]]$C
  C_ysd = c(C_ysd, C_temp)
}
id_post = rep(as.factor(1:Ns), each = N_samples)
shr = c('no-without delta','no-with delta', "yes/common delta", "yes/shared delta")
Rpost=data.frame(R = c(R_nnd, R_nwd, R_ycd, R_ysd), ID = rep(id_post,4), sharing = rep(rep(shr, each = I
Cpost=data.frame(C = c(C_nnd, C_nwd, C_ycd, C_ysd), ID = rep(id_post,4), sharing = rep(rep(shr, each = I

# R post
(pl_R=ggplot() +
    geom_violin(data = Rpost, aes(x = ID, y=R, color = sharing), size=0.7)+
    theme_bw()+
    theme(axis.title.y = element_text(size = rel(1.5)), legend.position = "none")+
    geom_point(data = data.frame(R = Rtrue, ID = as.factor(1:Ns)), aes(x = ID, y = R, shape = "true"),si
    facet_wrap(sharing~.)+
    xlab("ID")+
    scale_color_manual(
      breaks=c('no-without delta', 'no-with delta', "yes/common delta", "yes/shared delta"),
      values=c("magenta","firebrick","dodgerblue4", "forestgreen"))
)
```




```
# ggsave("Figs/post_WK_R.pdf", plot = pl_R, width = 18, height = 12, units = "cm")
```

Plot posteriors of C for all individuals (Figure 3 in Appendix)

```
# C post
(pl_C=ggplot() +
  geom_violin(data = Cpost, aes(x = ID, y=C, color = sharing),size=0.7)+
  theme_bw()+
  theme(axis.title.y = element_text(size = rel(1.5)), legend.position = "none")+
  geom_point(data = data.frame(C = Ctrue, ID = as.factor(1:Ns)), aes(x = ID, y = C, shape = "true"),siz
  facet_wrap(sharing~.)+
  xlab("ID")+
  scale_color_manual(
    breaks=c('no-without delta', 'no-with delta', "yes/common delta", "yes/shared delta"),
    values=c("magenta","firebrick","dodgerblue4", "forestgreen")))
```




```
# ggsave("Figs/post_WK_C.pdf", plot = pl_C, width = 18, height = 12, units = "cm")
```

Session information

```
sessionInfo()
```

```
R version 4.0.3 (2020-10-10)
Platform: x86_64-apple-darwin17.0 (64-bit)
Running under: macOS Big Sur 10.16

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] latex2exp_0.9.0      rstan_2.21.3         ggplot2_3.3.5
[4] StanHeaders_2.21.0-7
```

```
loaded via a namespace (and not attached):
 [1] tidyselect_1.1.1   xfun_0.29          purrr_0.3.4        colorspace_2.0-2
 [5] vctrs_0.3.8        generics_0.1.2     htmltools_0.5.2    stats4_4.0.3
 [9] loo_2.4.1          yaml_2.2.2         utf8_1.2.2         rlang_1.0.0
[13] pkgbuild_1.3.1     pillar_1.7.0       glue_1.6.1         withr_2.4.3
[17] DBI_1.1.2          matrixStats_0.61.0 lifecycle_1.0.1    stringr_1.4.0
[21] munsell_0.5.0      gtable_0.3.0       codetools_0.2-18   evaluate_0.14
[25] labeling_0.4.2     inline_0.3.19      knitr_1.37         callr_3.7.0
[29] fastmap_1.1.0      ps_1.6.0           parallel_4.0.3     fansi_1.0.2
[33] Rcpp_1.0.8         scales_1.1.1       RcppParallel_5.1.5 farver_2.1.0
[37] gridExtra_2.3      digest_0.6.29      stringi_1.7.6      processx_3.5.2
[41] dplyr_1.0.7        grid_4.0.3         cli_3.1.1          tools_4.0.3
[45] magrittr_2.0.2     tibble_3.1.6       crayon_1.4.2       pkgconfig_2.0.3
[49] ellipsis_0.3.2     prettyunits_1.1.1  assertthat_0.2.1   rmarkdown_2.11
[53] rstudioapi_0.13    R6_2.5.1           compiler_4.0.3
```