Cardiovascular model in Section 5

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("SAVE")
library(rstan)
library(ggplot2)

rstan_options(auto_write = TRUE)
options(mc.cores = 3) # allocate 3 cores (for each model we run 3 chains in parallel)

# Numerical simulator of the WK3 model
source("WK_numerical_simulators/WK2and3_sim_fn.R")
# Load flow data
d = readRDS("Data/Inflow_time.rds")
```

Reality and modelling choice

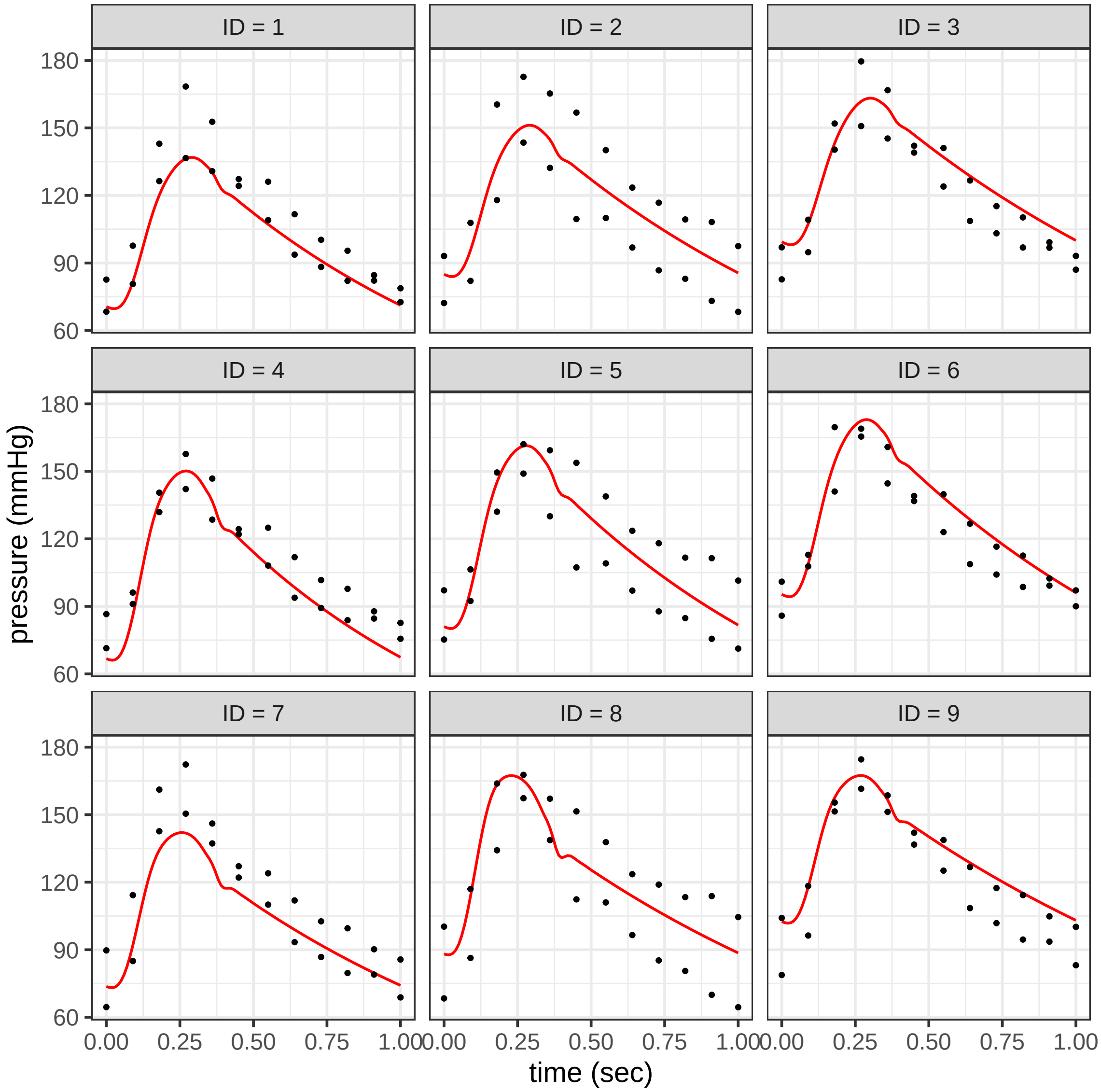
$$\mathcal{R}: \quad \frac{dP(t)}{dt} + \frac{P(t)}{R_2 C} = \frac{Q(t)}{C}\left(1 + \frac{R_1}{R_2}\right) + R_1\frac{dQ(t)}{dt} \quad \text{(the misspesified model we use to fit the data)} \quad \text{[WK3]} \tag{1}$$

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

```
# choose some reasonable physical parameter values
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]
Ns=length(Rtrue) # number of individuals
flow = d$inflow*0.95 # flow data
time = d$time # corresponding time
nP=12 # number of pressure data
nI=14 # number of inflow data
nc=2  # number of cardiac cycles
nflow = length(flow)
post_nnd=post_nwd=l_df_nnd=l_df_nwd=list()

set.seed(123)
Zvec=sample(seq(0.02,0.1,by=0.01), length(Rtrue), replace = T)
```

```r
yP=tP=matrix(NA,nrow = Ns, ncol = nP*nc)
yI=tI=matrix(NA,nrow = Ns, ncol = nI*nc)
P_true = matrix(NA, nrow = nflow, ncol = Ns)
t1=Sys.time()
for(i in 1:Ns){
  # 1. simulate WK3 data (R=R_2, Z=R_1)
  Psim=WK3_simulate(flow = flow, time = time, R = Rtrue[i], C = Ctrue[i], Z=Zvec[i]) # simulate WK3 dat
  P_true[,i] = Psim
  # 2. choose pressure and inflow indices
  indP = round(seq(1, nflow, length.out = nP)); indI = round(seq(1, nflow, length.out = nI))
  yP_real = Psim[indP]; yI_real = flow[indI] # noise free fimulated pressure and flow
  # 3. Add noise
  set.seed(123)
  Pnoise = rnorm(nP*nc, 0, 4) # sample pressure noise from N(0, 4^2)
  Inoise =rnorm(nI*nc, 0, 10) # sample flow noise from N(0,10^2)
  yP_real = rep(yP_real,nc) # create 2 replicates (2 cardiac cycles/heart beats)
  yI_real = rep(yI_real,nc) # create 2 replicates (2 cardiac cycles/heart beats)
  # 4. store individual data in the population matrices
  yP[i,]= yP_real + Pnoise # add noise
  yI[i,]= yI_real + Inoise # add noise
  tP[i,] = time[indP] # corresponding time (synchronized for the two cycles)
  tI[i,] = time[indI] # corresponding time (synchronized for the two cycles)
}
id=1:Ns
data_population = list(nP=nc*nP, nI=nc*nI, tP=tP, tI=tI, yP=yP, yI=yI,id=id, Ns=Ns)
```

```r
ID = paste0("ID = ", 1:Ns)
df_Ptrue = data.frame(pressure = as.vector(P_true), time = rep(time,Ns), ID = rep(ID, each=nflow))
df_Pobs = data.frame(pressure = as.vector(t(yP)), time=as.vector(t(tP)), ID = rep(ID, each=nP))

ggplot()+
  geom_line(data=df_Ptrue, aes(x=time,y=pressure), color="red")+
  geom_point(data=df_Pobs, aes(x=time,y=pressure), color="black", size=0.5)+
  facet_wrap(ID~., nrow = 3)+theme_bw()+xlab("time (sec)")+ ylab("pressure (mmHg)")
```



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 the physics-informed Gaussian process prior see Appendix E.1.

Stan code:

```
writeLines(readLines('STAN/WK2/WK2_nodelta.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);
```

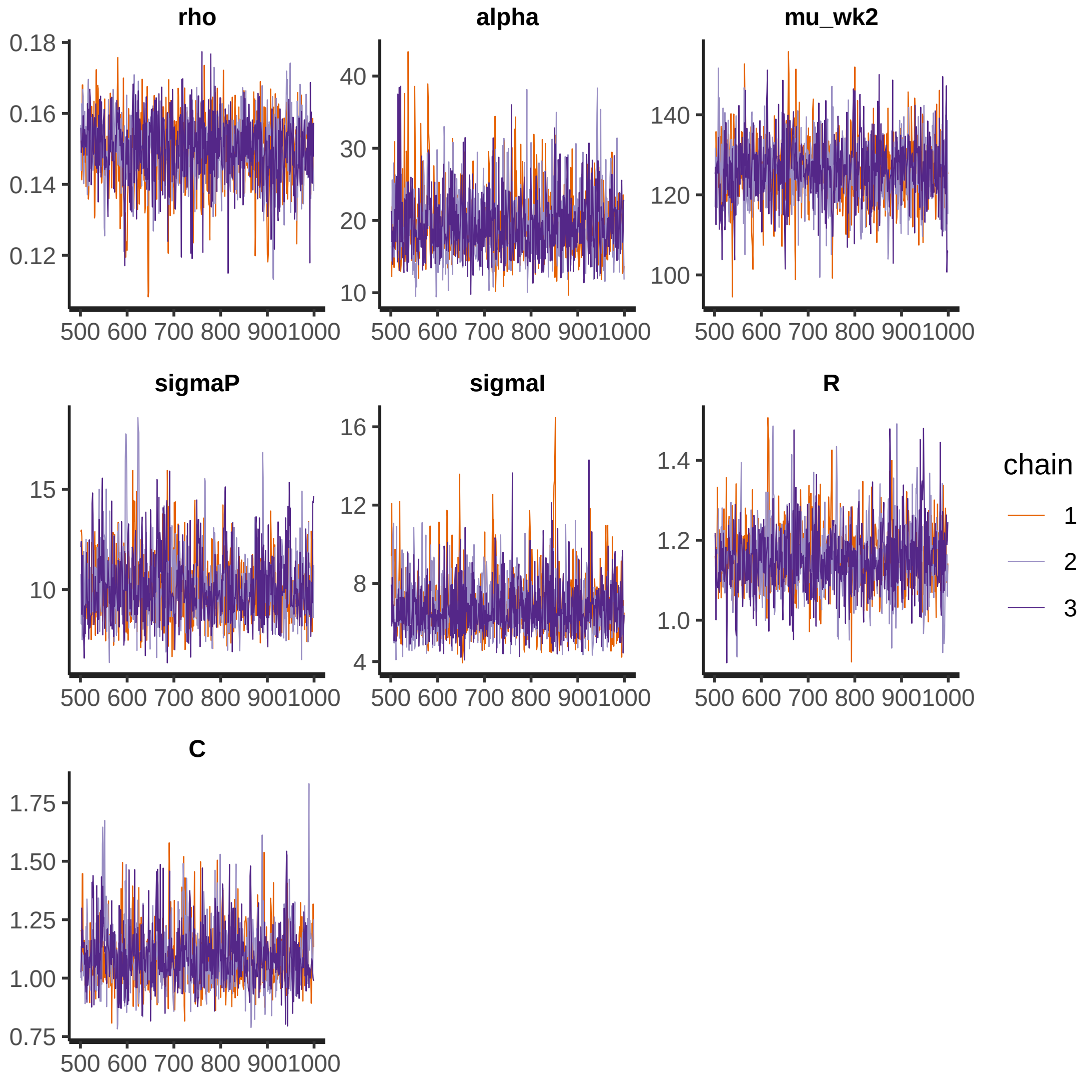
```
              return(mu);
  }
  // Physics informed prior kernel of the WK2 model
  matrix K_wk2(vector tP,
               vector tI,
               real rho,
               real alpha,
               real sigmaP,
               real sigmaI,
               real R,
               real C) {
    int 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
                         * 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[n
                         * 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);
  }
}

data {
  int<lower=1> nP;
  int<lower=1> nI;
  vector[nP] tP;
  vector[nI] tI;
  vector[nP] yP;
  vector[nI] yI;
}

transformed data {
  vector[nP + nI] y = append_row(yP, yI);
}

parameters {
  // hyper-parameters
  real<lower=0.05> rho;
  real<lower=5> alpha;
  real<lower=0,upper=400> mu_wk2;
  real<lower=0,upper=20> sigmaP;
  real<lower=0,upper=20> sigmaI;
  // physical parameters
  real<lower=0.5, upper=3> R;
  real<lower=0.5, upper=3> C;
}

model {
  // Chol. of PI kernel
  matrix[nP + nI, nP + nI] L_K = K_wk2(tP, tI, rho, alpha, sigmaP, sigmaI, R, C);
  // mean vector
  vector[nP + nI] mu = mu_fn(mu_wk2, R, nP, nI);
  // priors
```

```
  rho ~ normal(0,1.0/3);
  alpha ~ normal(0,20);
  mu_wk2 ~ normal(mean(yP), 20);

  y ~ multi_normal_cholesky(mu, L_K);
}
```

We fit the model to each individual data set and we plot the trace for the last individual

```
#--------------------------------------------------
### Model 1 (no-without delta in paper, Figure 6)
# WK2 PI prior / no delta (magenta model)
for(i in 1:Ns){
  data_ind = list(nP=nc*nP, nI=nc*nI, tP=tP[i,], tI=tI[i,], yP=yP[i,], yI=yI[i,])
  fit_nnd= stan(file= 'STAN/WK2/WK2_nodelta.stan',
                   data=data_ind,
                   chains=3,
                   iter=1000,
                   seed=123
  )
  post_nnd[[i]]=extract(fit_nnd)
  l_df_nnd[[i]]=data.frame(extract(fit_nnd))
}
stan_trace(fit_nnd, size=0.2)
```



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 about the model are given in the Appendix E.1.

Stan code:

```
writeLines(readLines('STAN/WK2/WK2_delta.stan'))
```

```
functions {
  // Physics informed prior mean of the WK2 model
  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);
```

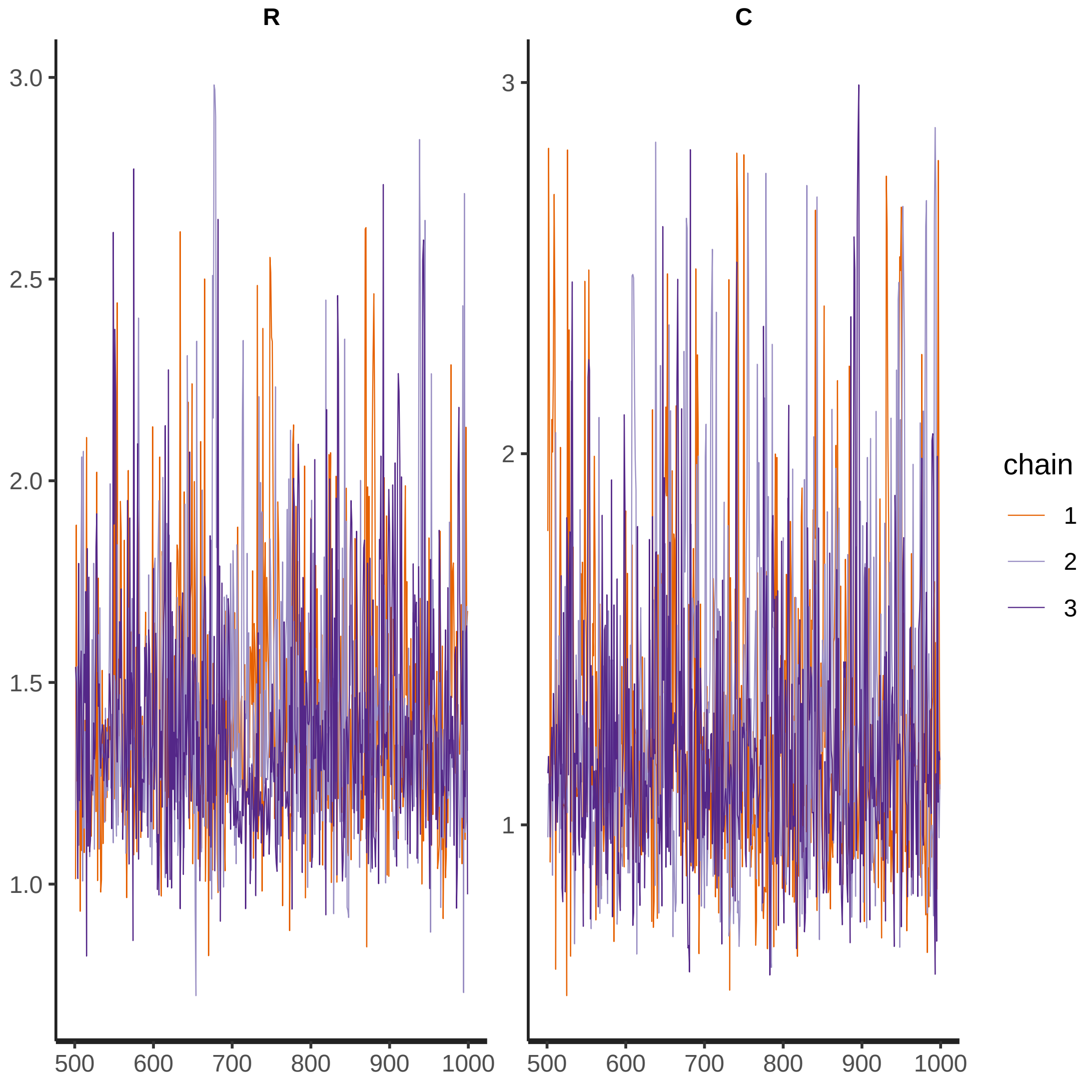
```
                return(mu);
  }
  // Physics informed prior kernel of the WK2 model
  matrix K_wk2(vector tP,
               vector tI,
               real rho,
               real alpha,
               real sigmaP,
               real sigmaI,
               real rho_d,
               real alpha_d,
               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));
    // K_delta (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[
                           * 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[n
                         * 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);
  }
}

data {
  int<lower=1> nP;
  int<lower=1> nI;
  vector[nP] tP;
  vector[nI] tI;
  vector[nP] yP;
  vector[nI] yI;
}

transformed data {
  vector[nP + nI] y = append_row(yP, yI);
}

parameters {
  // hyper-parameters
  real<lower=0.05> rho;
  real<lower=0> alpha;
```

```
  real<lower=0> rho_d;
  real<lower=0> alpha_d;
  real<lower=0,upper=400> mu_wk2;
  real<lower=0,upper=20> sigmaP;
  real<lower=0,upper=20> sigmaI;
  // physical parameters
  real<lower=0.5, upper=3> R;
  real<lower=0.5, upper=3> C;
}

model {
  // Chol. of PI kernel
  matrix[nP + nI, nP + nI] L_K = K_wk2(tP, tI, rho, alpha, sigmaP, sigmaI, rho_d, alpha_d, R, C);
  // mean vector
  vector[nP + nI] mu = mu_fn(mu_wk2, R, nP, nI);
  // priors
  rho ~ normal(0,1.0/3);
  alpha ~ normal(0,20);
  rho_d ~ normal(0,1.0/3);
  alpha_d ~ normal(0,20);
  mu_wk2 ~ normal(mean(yP), 20);

  y ~ multi_normal_cholesky(mu, L_K);
}
```

We fit this model to each individual data set separately

```r
# WK2 PI prior / with pressure delta (red model)
for(i in 1:Ns){
  data_ind = list(nP=nc*nP, nI=nc*nI, tP=tP[i,], tI=tI[i,], yP=yP[i,], yI=yI[i,])
  fit_nwd= stan(file= 'STAN/WK2/WK2_delta.stan', #'STAN_WK/WK_ind_delta.stan',
                          data=data_ind,
                          chains=3,
                          iter=1000,
                          seed=123
  )
  post_nwd[[i]]=extract(fit_nwd)
  l_df_nwd[[i]]=data.frame(extract(fit_nwd))
}
stan_trace(fit_nwd, size=0.2, pars = c("R", "C"))
```



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.

Stan code:

```
writeLines(readLines('STAN/WK2/WK2_common_delta.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);
```

```
  }
  // Physics informed prior kernel of the WK2 model
  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));

// K_delta
// 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[
                        * 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[n
                      * 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);
  }
}
data {
  int<lower=1> nP;
  int<lower=1> nI;
  int<lower=1> Ns;
  int<lower=1,upper=Ns> id[Ns];
  vector[nP] tP[Ns];
  vector[nI] tI[Ns];
  matrix[Ns, nP] yP;
  matrix[Ns, nI] yI;
}

transformed data {
  int<lower=1> N_tot=nP+nI;
  matrix[Ns,N_tot] y;
  y=append_col(yP, yI);
}
parameters {
```

```
  real<lower=0> R_tilde[Ns];
  real<lower=0> C_tilde[Ns];

  real<lower=0> mu_wk2_tilde[Ns];
  real rho_tilde[Ns];   //non-centered sd of rho
  real alpha_tilde[Ns]; //non-centered sd of alpha
  // delta
  real rho_d_tilde;   //non-centered sd of rho (delta)
  real alpha_d_tilde; //non-centered sd of marginal sd (delta)
  // noise sds
  real<lower=0,upper=10> sigmaP;
  real<lower=0,upper=20> sigmaI;
  // global parameters
  real<lower=0> rho_m;   // median of individual prior on rho
  real<lower=0> rho_s;   //sd of individual prior on rho
  real<lower=0> alpha_m; // median of individual prior on alpha
  real<lower=0> alpha_s; //sd of individual prior on alpha
  // delta
  real<lower=0.05,upper=1> rho_d;  // length scale for delta
  real<lower=5, upper=40> alpha_d; // marginal standard deviation
  // global parameters
  real<lower=0.5,upper=2> mu_R;
  real<lower=1,upper=2> tau_R;
  real<lower=0.5,upper=2> mu_C;
  real<lower=1,upper=2> tau_C;
  real<lower=60,upper=100> mu_muWK2;
  real<lower=20> tau_muWK2;
}
transformed parameters {
  real<lower=0> mu_wk2[Ns]; // physical parameters
  real<lower=0> rho[Ns];    // length scale
  real<lower=0> alpha[Ns]; //marginal standard deviation
  real<lower=0> R[Ns];
  real<lower=0> C[Ns];
  // Non-centered parameterization of individual parameters
  for (s in 1:Ns) {
    rho[s] = exp(log(rho_m) + rho_s * rho_tilde[s]);
    alpha[s] = exp(log(alpha_m) + alpha_s * alpha_tilde[s]);
    R[s] = mu_R + tau_R * R_tilde[s];
    C[s] = mu_C + tau_C * C_tilde[s];
    mu_wk2[s] = mu_muWK2 + tau_muWK2 * mu_wk2_tilde[s];
  }
}
model {
  matrix[nP + nI, nP + nI] L_K[Ns];
  vector[nP + nI] mu[Ns];
  for (s in 1:Ns) {
    L_K[s] = K_wk2(tP[s], tI[s], rho[s], alpha[s], sigmaP, sigmaI, rho_d, alpha_d, R[s], C[s]);
    mu[s] = mu_fn(mu_wk2[s], R[s], nP, nI);
  }
  // Global level
  rho_m ~ inv_gamma(2, 2);
  alpha_m ~ normal(0,20);
  rho_s ~ normal(0, 0.5);
```

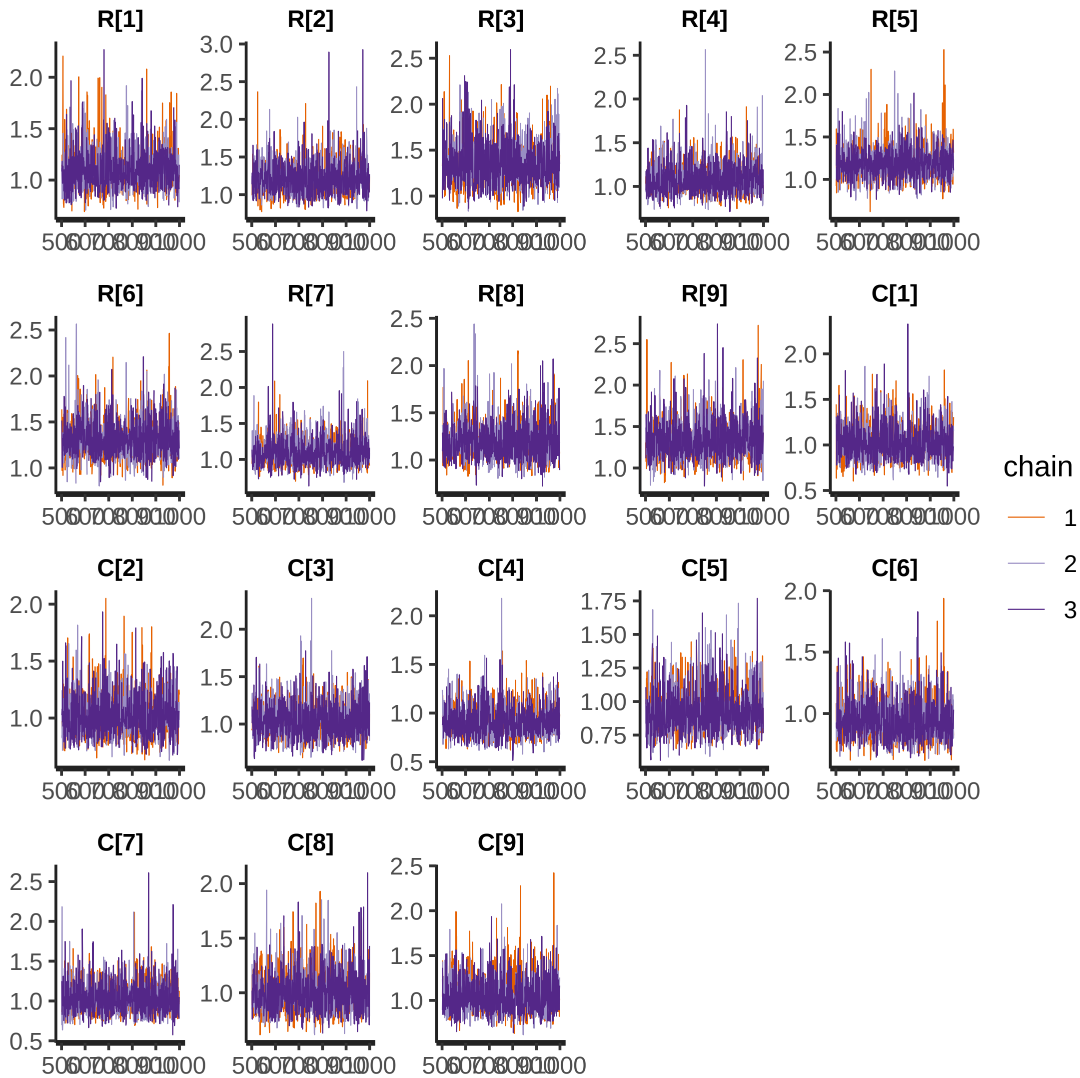
```
  alpha_s ~ normal(0, 10);
  // delta
  rho_d~ inv_gamma(2, 0.5);
  alpha_d ~ normal(0,20);

  // non centered parameterization
  rho_tilde ~ normal(0, 1);
  alpha_tilde ~ normal(0, 1);
  // delta
  rho_d_tilde ~ normal(0, 1);
  alpha_d_tilde ~ normal(0, 1);
  R_tilde ~ normal(0, 1);
  C_tilde ~ normal(0, 1);
  mu_wk2_tilde ~ normal(0, 1);

  // likelihood
  for (i in 1:Ns){
    y[i] ~ multi_normal_cholesky(mu[id[i]], L_K[id[i]]);
  }
}
```

```r
#--------------------------------------------------
# shared R,C common delta (blue model)
fit_yes_common_delta = stan(file='STAN/WK2/WK2_common_delta.stan',
                            data=data_population,
                            chains=3,
                            iter=1000,
                            seed=123
)
names(fit_yes_common_delta)
```

```
 [1] "R_tilde[1]"      "R_tilde[2]"      "R_tilde[3]"      "R_tilde[4]"
 [5] "R_tilde[5]"      "R_tilde[6]"      "R_tilde[7]"      "R_tilde[8]"
 [9] "R_tilde[9]"      "C_tilde[1]"      "C_tilde[2]"      "C_tilde[3]"
[13] "C_tilde[4]"      "C_tilde[5]"      "C_tilde[6]"      "C_tilde[7]"
[17] "C_tilde[8]"      "C_tilde[9]"      "mu_wk2_tilde[1]" "mu_wk2_tilde[2]"
[21] "mu_wk2_tilde[3]" "mu_wk2_tilde[4]" "mu_wk2_tilde[5]" "mu_wk2_tilde[6]"
[25] "mu_wk2_tilde[7]" "mu_wk2_tilde[8]" "mu_wk2_tilde[9]" "rho_tilde[1]"
[29] "rho_tilde[2]"    "rho_tilde[3]"    "rho_tilde[4]"    "rho_tilde[5]"
[33] "rho_tilde[6]"    "rho_tilde[7]"    "rho_tilde[8]"    "rho_tilde[9]"
[37] "alpha_tilde[1]"  "alpha_tilde[2]"  "alpha_tilde[3]"  "alpha_tilde[4]"
[41] "alpha_tilde[5]"  "alpha_tilde[6]"  "alpha_tilde[7]"  "alpha_tilde[8]"
[45] "alpha_tilde[9]"  "rho_d_tilde"     "alpha_d_tilde"   "sigmaP"
[49] "sigmaI"          "rho_m"           "rho_s"           "alpha_m"
[53] "alpha_s"         "rho_d"           "alpha_d"         "mu_R"
[57] "tau_R"           "mu_C"            "tau_C"           "mu_muWK2"
[61] "tau_muWK2"       "mu_wk2[1]"       "mu_wk2[2]"       "mu_wk2[3]"
[65] "mu_wk2[4]"       "mu_wk2[5]"       "mu_wk2[6]"       "mu_wk2[7]"
[69] "mu_wk2[8]"       "mu_wk2[9]"       "rho[1]"          "rho[2]"
[73] "rho[3]"          "rho[4]"          "rho[5]"          "rho[6]"
[77] "rho[7]"          "rho[8]"          "rho[9]"          "alpha[1]"
[81] "alpha[2]"        "alpha[3]"        "alpha[4]"        "alpha[5]"
[85] "alpha[6]"        "alpha[7]"        "alpha[8]"        "alpha[9]"
[89] "R[1]"            "R[2]"            "R[3]"            "R[4]"
[93] "R[5]"            "R[6]"            "R[7]"            "R[8]"
```

```
 [97] "R[9]"           "C[1]"           "C[2]"           "C[3]"
[101] "C[4]"           "C[5]"           "C[6]"           "C[7]"
[105] "C[8]"           "C[9]"           "lp__"
```

```
stan_trace(fit_yes_common_delta, pars = c("R","C"), size=0.2)
```




```
ex_ycd=extract(fit_yes_common_delta)
df_ycd =data.frame(ex_ycd)
```

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.

Stan code:

```
writeLines(readLines('STAN/WK2/WK2_common_delta.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);
  }
  // Physics informed prior kernel of the WK2 model
  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));

// K_delta
// 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[
                      * 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[n
                    * 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);
  }
}
data {
  int<lower=1> nP;
  int<lower=1> nI;
  int<lower=1> Ns;
  int<lower=1,upper=Ns> id[Ns];
  vector[nP] tP[Ns];
  vector[nI] tI[Ns];
  matrix[Ns, nP] yP;
  matrix[Ns, nI] yI;
```

```
}

transformed data {
  int<lower=1> N_tot=nP+nI;
  matrix[Ns,N_tot] y;
  y=append_col(yP, yI);
}
parameters {
  real<lower=0> R_tilde[Ns];
  real<lower=0> C_tilde[Ns];

  real<lower=0> mu_wk2_tilde[Ns];
  real rho_tilde[Ns];   //non-centered sd of rho
  real alpha_tilde[Ns]; //non-centered sd of alpha
  // delta
  real rho_d_tilde;   //non-centered sd of rho (delta)
  real alpha_d_tilde; //non-centered sd of marginal sd (delta)
  // noise sds
  real<lower=0,upper=10> sigmaP;
  real<lower=0,upper=20> sigmaI;
  // global parameters
  real<lower=0> rho_m;   // median of individual prior on rho
  real<lower=0> rho_s;   //sd of individual prior on rho
  real<lower=0> alpha_m; // median of individual prior on alpha
  real<lower=0> alpha_s; //sd of individual prior on alpha
  // delta
  real<lower=0.05,upper=1> rho_d;  // length scale for delta
  real<lower=5, upper=40> alpha_d; // marginal standard deviation
  // global parameters
  real<lower=0.5,upper=2> mu_R;
  real<lower=1,upper=2> tau_R;
  real<lower=0.5,upper=2> mu_C;
  real<lower=1,upper=2> tau_C;
  real<lower=60,upper=100> mu_muWK2;
  real<lower=20> tau_muWK2;
}
transformed parameters {
  real<lower=0> mu_wk2[Ns]; // physical parameters
  real<lower=0> rho[Ns];    // length scale
  real<lower=0> alpha[Ns]; //marginal standard deviation
  real<lower=0> R[Ns];
  real<lower=0> C[Ns];
  // Non-centered parameterization of individual parameters
  for (s in 1:Ns) {
    rho[s] = exp(log(rho_m) + rho_s * rho_tilde[s]);
    alpha[s] = exp(log(alpha_m) + alpha_s * alpha_tilde[s]);
    R[s] = mu_R + tau_R * R_tilde[s];
    C[s] = mu_C + tau_C * C_tilde[s];
    mu_wk2[s] = mu_muWK2 + tau_muWK2 * mu_wk2_tilde[s];
  }
}
model {
  matrix[nP + nI, nP + nI] L_K[Ns];
  vector[nP + nI] mu[Ns];
```

```
  for (s in 1:Ns) {
    L_K[s] = K_wk2(tP[s], tI[s], rho[s], alpha[s], sigmaP, sigmaI, rho_d, alpha_d, R[s], C[s]);
    mu[s] = mu_fn(mu_wk2[s], R[s], nP, nI);
  }
  // Global level
  rho_m ~ inv_gamma(2, 2);
  alpha_m ~ normal(0,20);
  rho_s ~ normal(0, 0.5);
  alpha_s ~ normal(0, 10);
  // delta
  rho_d~ inv_gamma(2, 0.5);
  alpha_d ~ normal(0,20);

  // non centered parameterization
  rho_tilde ~ normal(0, 1);
  alpha_tilde ~ normal(0, 1);
  // delta
  rho_d_tilde ~ normal(0, 1);
  alpha_d_tilde ~ normal(0, 1);
  R_tilde ~ normal(0, 1);
  C_tilde ~ normal(0, 1);
  mu_wk2_tilde ~ normal(0, 1);

  // likelihood
  for (i in 1:Ns){
    y[i] ~ multi_normal_cholesky(mu[id[i]], L_K[id[i]]);
  }
}
```

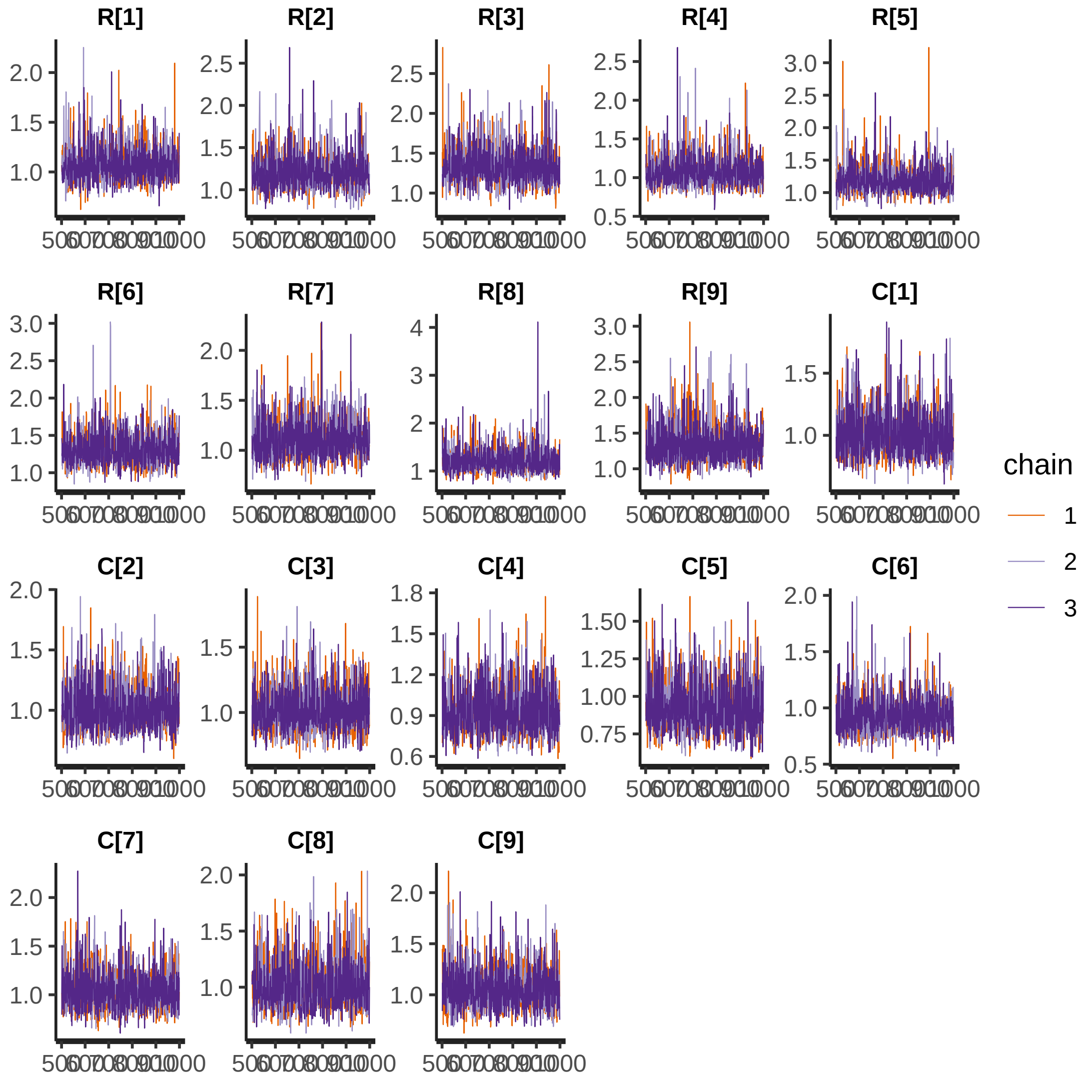
Model fit:

```r
#--------------------------------------------------
# shared R,C shared delta (green model)
fit_ysd = stan(file='STAN/WK2/WK2_shared_delta.stan',
                            data=data_population,
                            chains=3,
                            iter=1000,
                            seed=123
)
names(fit_ysd)
```

```
  [1] "R_tilde[1]"       "R_tilde[2]"       "R_tilde[3]"
  [4] "R_tilde[4]"       "R_tilde[5]"       "R_tilde[6]"
  [7] "R_tilde[7]"       "R_tilde[8]"       "R_tilde[9]"
 [10] "C_tilde[1]"       "C_tilde[2]"       "C_tilde[3]"
 [13] "C_tilde[4]"       "C_tilde[5]"       "C_tilde[6]"
 [16] "C_tilde[7]"       "C_tilde[8]"       "C_tilde[9]"
 [19] "mu_wk2_tilde[1]"  "mu_wk2_tilde[2]"  "mu_wk2_tilde[3]"
 [22] "mu_wk2_tilde[4]"  "mu_wk2_tilde[5]"  "mu_wk2_tilde[6]"
 [25] "mu_wk2_tilde[7]"  "mu_wk2_tilde[8]"  "mu_wk2_tilde[9]"
 [28] "rho_tilde[1]"     "rho_tilde[2]"     "rho_tilde[3]"
 [31] "rho_tilde[4]"     "rho_tilde[5]"     "rho_tilde[6]"
 [34] "rho_tilde[7]"     "rho_tilde[8]"     "rho_tilde[9]"
 [37] "alpha_tilde[1]"   "alpha_tilde[2]"   "alpha_tilde[3]"
 [40] "alpha_tilde[4]"   "alpha_tilde[5]"   "alpha_tilde[6]"
```

```
 [43] "alpha_tilde[7]"   "alpha_tilde[8]"   "alpha_tilde[9]"
 [46] "rho_d_tilde[1]"   "rho_d_tilde[2]"   "rho_d_tilde[3]"
 [49] "rho_d_tilde[4]"   "rho_d_tilde[5]"   "rho_d_tilde[6]"
 [52] "rho_d_tilde[7]"   "rho_d_tilde[8]"   "rho_d_tilde[9]"
 [55] "alpha_d_tilde[1]" "alpha_d_tilde[2]" "alpha_d_tilde[3]"
 [58] "alpha_d_tilde[4]" "alpha_d_tilde[5]" "alpha_d_tilde[6]"
 [61] "alpha_d_tilde[7]" "alpha_d_tilde[8]" "alpha_d_tilde[9]"
 [64] "sigmaP"           "sigmaI"           "rho_m"
 [67] "rho_s"            "alpha_m"          "alpha_s"
 [70] "rho_d_m"          "rho_d_s"          "alpha_d_m"
 [73] "alpha_d_s"        "mu_R"             "tau_R"
 [76] "mu_C"             "tau_C"            "mu_muWK2"
 [79] "tau_muWK2"        "mu_wk2[1]"        "mu_wk2[2]"
 [82] "mu_wk2[3]"        "mu_wk2[4]"        "mu_wk2[5]"
 [85] "mu_wk2[6]"        "mu_wk2[7]"        "mu_wk2[8]"
 [88] "mu_wk2[9]"        "rho[1]"           "rho[2]"
 [91] "rho[3]"           "rho[4]"           "rho[5]"
 [94] "rho[6]"           "rho[7]"           "rho[8]"
 [97] "rho[9]"           "alpha[1]"         "alpha[2]"
[100] "alpha[3]"         "alpha[4]"         "alpha[5]"
[103] "alpha[6]"         "alpha[7]"         "alpha[8]"
[106] "alpha[9]"         "rho_d[1]"         "rho_d[2]"
[109] "rho_d[3]"         "rho_d[4]"         "rho_d[5]"
[112] "rho_d[6]"         "rho_d[7]"         "rho_d[8]"
[115] "rho_d[9]"         "alpha_d[1]"       "alpha_d[2]"
[118] "alpha_d[3]"       "alpha_d[4]"       "alpha_d[5]"
[121] "alpha_d[6]"       "alpha_d[7]"       "alpha_d[8]"
[124] "alpha_d[9]"       "R[1]"             "R[2]"
[127] "R[3]"             "R[4]"             "R[5]"
[130] "R[6]"             "R[7]"             "R[8]"
[133] "R[9]"             "C[1]"             "C[2]"
[136] "C[3]"             "C[4]"             "C[5]"
[139] "C[6]"             "C[7]"             "C[8]"
[142] "C[9]"             "lp__"
```

```r
stan_trace(fit_ysd, pars = c("R", "C"), size=0.2)
```



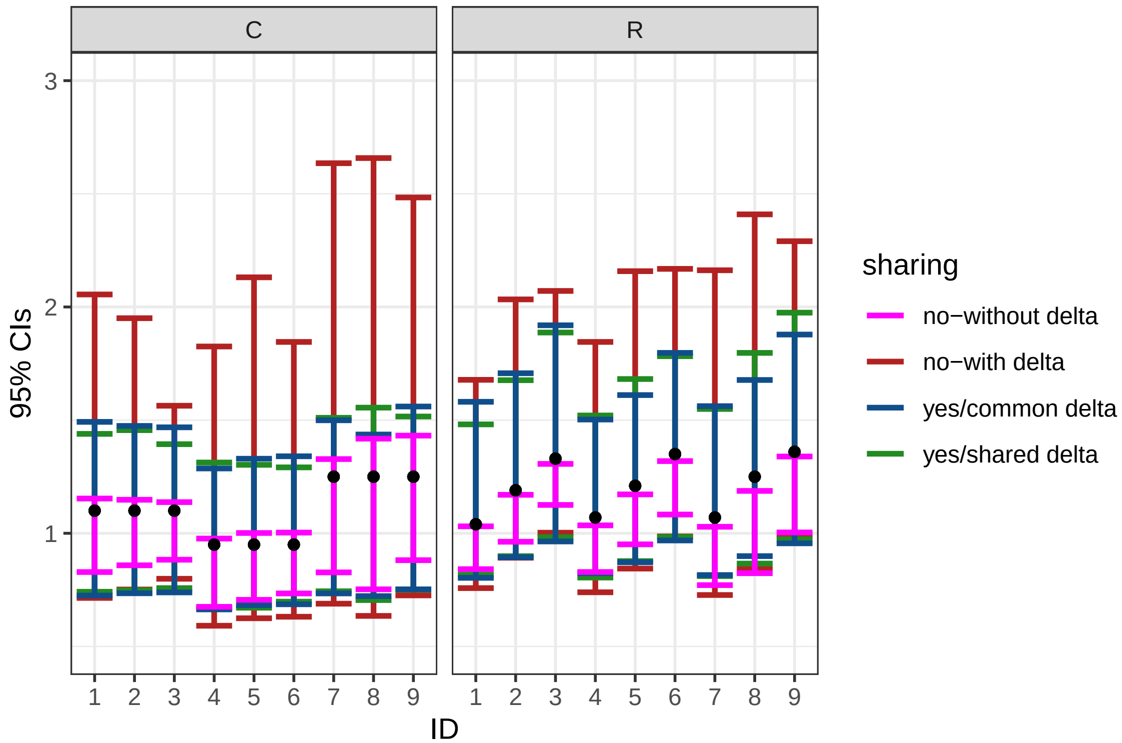

```
ex_ysd=extract(fit_ysd)
df_ysd=data.frame(ex_ysd)
```

Plot 95% CIs for all methods (Figure 6 in the paper)

	low	upper	par	id	true	sharing
1	0.7578565	1.6781373	R	1	1.04	no-with delta
2	0.8921701	2.0335097	R	2	1.19	no-with delta
3	1.0026595	2.0708657	R	3	1.33	no-with delta
4	0.7394544	1.8455380	R	4	1.07	no-with delta
5	0.8441593	2.1581726	R	5	1.21	no-with delta
6	0.9863287	2.1683877	R	6	1.35	no-with delta
7	0.7272905	2.1623086	R	7	1.07	no-with delta
8	0.8426894	2.4092519	R	8	1.25	no-with delta
9	0.9921635	2.2905770	R	9	1.36	no-with delta
10	0.7144321	2.0552788	C	1	1.10	no-with delta
11	0.7516362	1.9504450	C	2	1.10	no-with delta
12	0.7990536	1.5635764	C	3	1.10	no-with delta
13	0.5915794	1.8254503	C	4	0.95	no-with delta

14	0.6247050	2.1310589	C	5	0.95	no-with delta
15	0.6314045	1.8455638	C	6	0.95	no-with delta
16	0.6889692	2.6351526	C	7	1.25	no-with delta
17	0.6350172	2.6579013	C	8	1.25	no-with delta
18	0.7256722	2.4838631	C	9	1.25	no-with delta
R.1	0.8162363	1.4811896	R	1	1.04	yes/shared delta
R.2	0.8988965	1.6764984	R	2	1.19	yes/shared delta
R.3	0.9828948	1.8871914	R	3	1.33	yes/shared delta
R.4	0.8044930	1.5205505	R	4	1.07	yes/shared delta
R.5	0.8770352	1.6816121	R	5	1.21	yes/shared delta
R.6	0.9847210	1.7824699	R	6	1.35	yes/shared delta
R.7	0.8109141	1.5493231	R	7	1.07	yes/shared delta
R.8	0.8661692	1.7971734	R	8	1.25	yes/shared delta
R.9	0.9738889	1.9748676	R	9	1.36	yes/shared delta
C.1	0.7417057	1.4393457	C	1	1.10	yes/shared delta
C.2	0.7482260	1.4557823	C	2	1.10	yes/shared delta
C.3	0.7578428	1.3937755	C	3	1.10	yes/shared delta
C.4	0.6702127	1.3126996	C	4	0.95	yes/shared delta
C.5	0.6712563	1.3025081	C	5	0.95	yes/shared delta
C.6	0.6979896	1.2912788	C	6	0.95	yes/shared delta
C.7	0.7438852	1.5103079	C	7	1.25	yes/shared delta
C.8	0.7050344	1.5552484	C	8	1.25	yes/shared delta
C.9	0.7506893	1.5159890	C	9	1.25	yes/shared delta
R.11	0.8032168	1.5810382	R	1	1.04	yes/common delta
R.21	0.8942143	1.7071695	R	2	1.19	yes/common delta
R.31	0.9640072	1.9190497	R	3	1.33	yes/common delta
R.41	0.8234617	1.5026749	R	4	1.07	yes/common delta
R.51	0.8719202	1.6107578	R	5	1.21	yes/common delta
R.61	0.9683857	1.7971158	R	6	1.35	yes/common delta
R.71	0.8155684	1.5618177	R	7	1.07	yes/common delta
R.81	0.8991857	1.6773592	R	8	1.25	yes/common delta
R.91	0.9557937	1.8781220	R	9	1.36	yes/common delta
C.11	0.7248117	1.4923104	C	1	1.10	yes/common delta
C.21	0.7350571	1.4744067	C	2	1.10	yes/common delta
C.31	0.7388559	1.4682567	C	3	1.10	yes/common delta
C.41	0.6638033	1.2865223	C	4	0.95	yes/common delta
C.51	0.6822117	1.3294336	C	5	0.95	yes/common delta
C.61	0.6863748	1.3403039	C	6	0.95	yes/common delta
C.71	0.7343834	1.4993804	C	7	1.25	yes/common delta
C.81	0.7224355	1.4364309	C	8	1.25	yes/common delta
C.91	0.7524487	1.5601956	C	9	1.25	yes/common delta
19	0.8415313	1.0308030	R	1	1.04	no-without delta
21	0.9626938	1.1700824	R	2	1.19	no-without delta
31	1.1253319	1.3066564	R	3	1.33	no-without delta
41	0.8289461	1.0350052	R	4	1.07	no-without delta
51	0.9513089	1.1719719	R	5	1.21	no-without delta
61	1.0830271	1.3190403	R	6	1.35	no-without delta
71	0.7710298	1.0287726	R	7	1.07	no-without delta
81	0.8236080	1.1872510	R	8	1.25	no-without delta
91	1.0037853	1.3392017	R	9	1.36	no-without delta
110	0.8288485	1.1535763	C	1	1.10	no-without delta
22	0.8586015	1.1481633	C	2	1.10	no-without delta
32	0.8833197	1.1377226	C	3	1.10	no-without delta
42	0.6750889	0.9766766	C	4	0.95	no-without delta

```
52   0.7060090 1.0013359   C  5 0.95 no-without delta
62   0.7342586 1.0030787   C  6 0.95 no-without delta
72   0.8273293 1.3279104   C  7 1.25 no-without delta
82   0.7524619 1.4181516   C  8 1.25 no-without delta
92   0.8812200 1.4314809   C  9 1.25 no-without delta
```



Session information

```r
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] rstan_2.21.3         ggplot2_3.3.5        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
```