Toy example in Section 4

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/toy".

Load packages

```
# uncomment to install
# install.packages("rstan")
# install.packages("ggplot2")
# install.packages("SAVE")
library(rstan)
library(ggplot2)
library(SAVE) # package with the data
rstan_options(auto_write = TRUE)
options(mc.cores = 3) # allocate 3 cores (for each model we run 3 chains in parallel)
```

Reality and modelling choice

Note that in the paper there is a mistake in the presentation. The coefficients of the true model $\mathcal{R}$ and model M are swapped and the correct models are the following

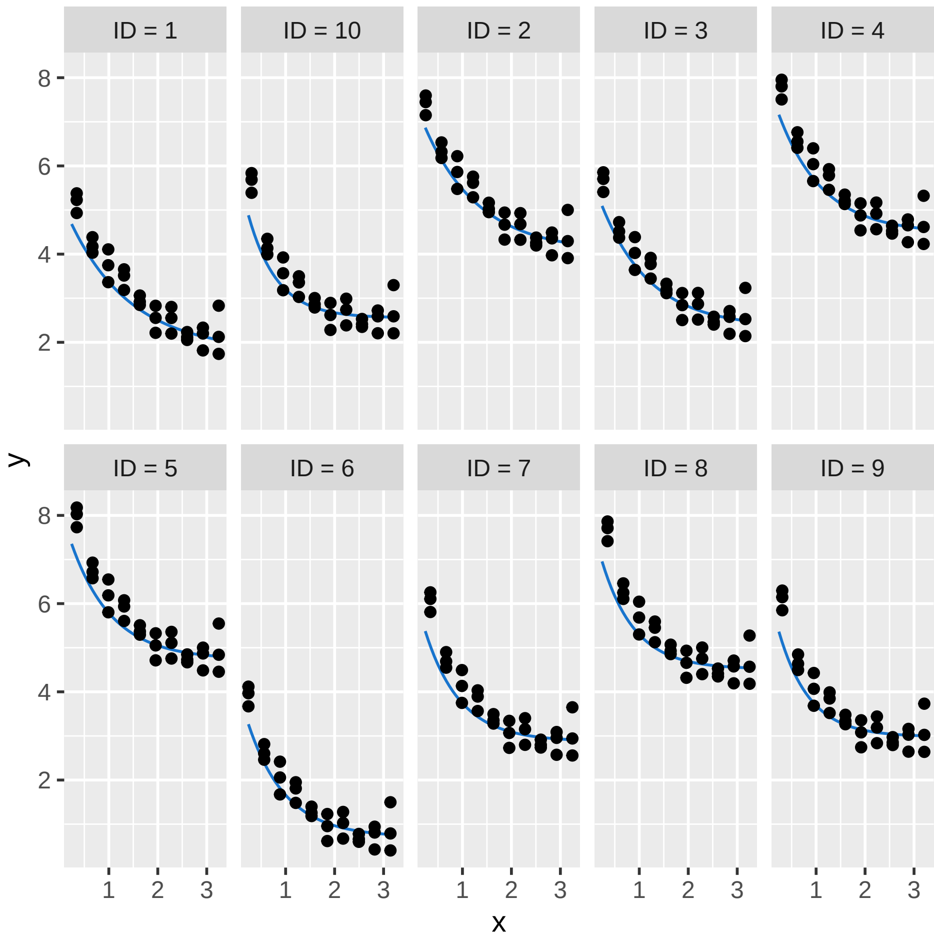
$$y^{\mathcal{R}}(x) = 3.5 \cdot \exp(-u \cdot x) + b + \varepsilon \quad \text{(the model we use to simulate data)}$$
$$\eta(x,u) = 5 \cdot \exp(-u \cdot x) \quad \text{(the misspesified model we use to fit the data)}$$

```
R = function(u,x,b) 3.5*exp(-u*x)+b
sd_noise = 0.3
```

```
data("synthfield") # data from the rpackage SAVE
X_loc = unique(synthfield$x)

# simulate data for different u val and add iid noise
u_val = seq(0.8,1.7,by=0.1)
dl=list()
set.seed(123)
offsets=runif(length(u_val),0.5,5)

xobs = c(unique(synthfield$x),unique(synthfield$x),unique(synthfield$x)); N=length(xobs); # create 3 re
X_mat = matrix(NA, nrow = length(u_val), ncol = length(xobs))
set.seed(0)
dev=c(runif(length(u_val), 0.1,0.25))

for(i in 1:nrow(X_mat)){
  X_mat[i,] = xobs+dev[i] # create different input locations for each individual
}
```

```r
for(i in seq_along(u_val)){
  set.seed(0)
  y=R(u_val[i], xobs, offsets[i])+rnorm(N,0,sd=sd_noise); # add i.i.d. N(0,0.3^2) noise
  dl[[i]] = list(x= X_mat[i,], y = y, N = N, x_pred=X_mat[i,], N_pred=N)
}
y = matrix(NA, nrow = length(u_val), ncol = length(xobs))
for(i in 1:nrow(y)) y[i,] = dl[[i]]$y

id = seq_along(u_val) # individual id
Ns = length(id) # total number of individuals
data_population = list(x= X_mat, y = y, N = N, Ns=Ns, id=id)
```

```r
# create real data (noise free) for plotting
X = seq(min(X_mat), max(X_mat), length.out = 50)
y_real = matrix(NA, nrow = Ns, ncol = length(X))
for(i in seq_along(u_val)){
  y_real[i,] = R(u_val[i],X,offsets[i]) # real data for plotting
}
id_new = paste0("ID = ", 1:Ns)
real_data = data.frame(x=rep(X,Ns), y = as.vector(t(y_real)), ID = rep(id_new, each=length(X)))
obs_data = data.frame(x=as.vector(t(X_mat)), y = as.vector(t(y)), ID = rep(id_new, each=ncol(X_mat)))

pl_obs_toy=ggplot()+
  geom_line(data=real_data, aes(x=x,y=y), color="dodgerblue3")+
  geom_point(data=obs_data, aes(x=x,y=y), color="black")+
  facet_wrap(ID~., nrow = 2)#+theme_bw()
pl_obs_toy
```



Model 1 (no-without delta in paper, Figure 3)

This is the misspecified model that does not account for model discrepancy (no-without delta in paper, Figure 3), and it is the following regression model

$$y(x_m) = 5 \cdot \exp(-u_m \cdot x_m) + \varepsilon, \text{ where } \varepsilon \sim N(0, \sigma^2).$$

This is the following probabilistic model

$$\mathbf{y} \sim N(5 \cdot \exp(-u_m \cdot \mathbf{X}_m), \sigma^2 I),$$

where we assign priors to u_m (same for each individual) and σ (for more details see Appendix).

Stan code:

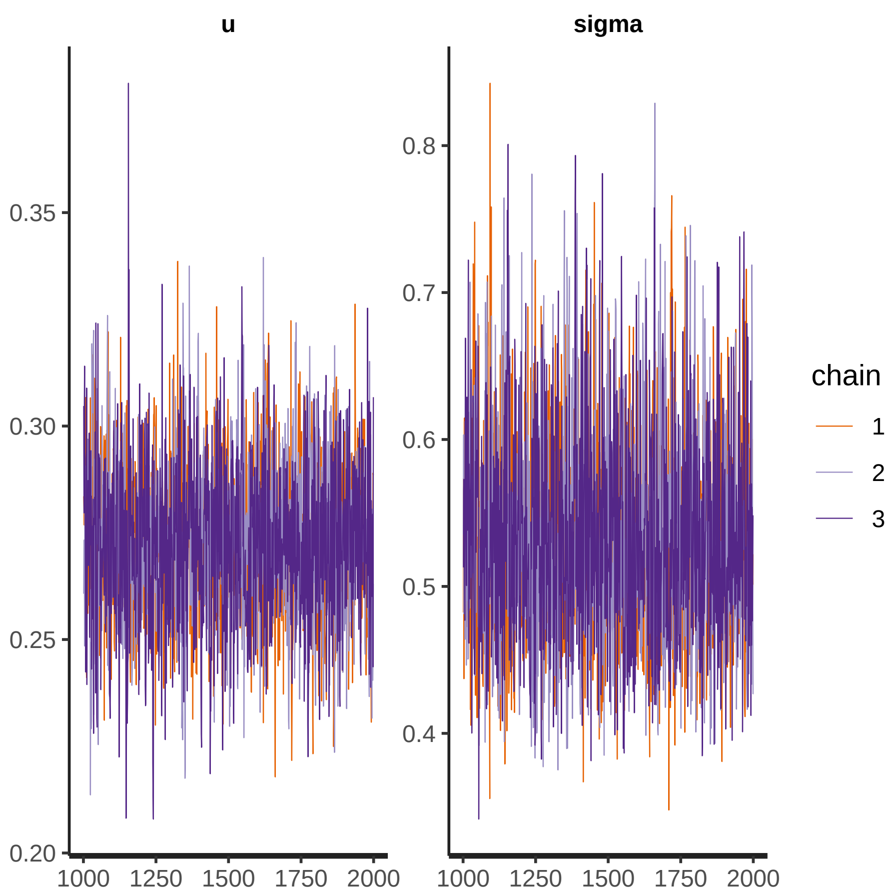
```
writeLines(readLines('STAN/toy/toy_nodelta.stan'))
```

```
data {
  int<lower=0> N;
  vector[N] x;
  vector[N] y;
}
parameters {
  real<lower=0,upper=5> u;
  real<lower=0, upper=2> sigma;
```

```
}

model {
  // priors
  u ~ normal(1,2);
  // likelihood
  y ~ normal(5*exp(-u*x), sigma);
}
```

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

```
lu_no=list()
for(i in seq_along(u_val)){
  fit_no_without_delta = stan(
    file='STAN/toy/toy_nodelta.stan', # without delta
    data=dl[[i]],
    chains=3,
    iter=2*1000,
    seed=123
  )
  lu_no[[i]] = extract(fit_no_without_delta)$u
}
stan_trace(fit_no_without_delta, size=0.2)
```



Model 2 (no-with delta in paper, Figure 3)

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 $K_\delta(x_m, x'_m) = \alpha_m^2 \exp\left(-\frac{(x_m - x'_m)^2}{2\rho_m^2}\right)$ and we have the following formulation

$$y^R(\mathbf{x}) = \eta(\mathbf{x}, \boldsymbol{\phi}) + \delta(\mathbf{x}) + \varepsilon, \text{ where } \varepsilon \sim N(0, \sigma^2).$$

This is equivalent to

$$y^{\mathcal{R}} \sim GP(5 \cdot \exp(-u_m \cdot \mathbf{X}_m), K_\delta(\mathbf{X}_m, \mathbf{X}_m \mid \boldsymbol{\omega}_m) + \sigma^2 I).$$

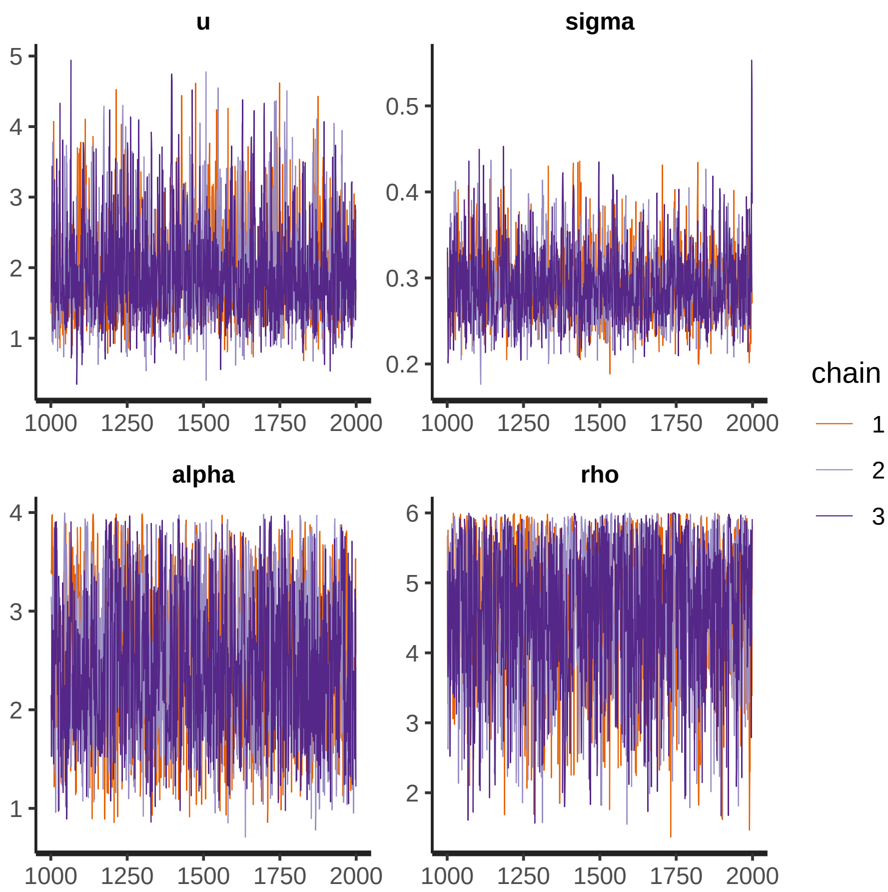
Stan code:

```
writeLines(readLines('STAN/toy/toy_delta.stan'))
```

```
functions{ // squared exponential kernel
  matrix cov_exp(vector x,
                 real alpha,
                 real rho){
    int n = rows(x);
    matrix[n, n] K;
    // KP
    for (i in 1:(n)){
      K[i,i] = pow(alpha, 0.2e1);
      for (j in (i+1):n){
        K[i,j] = exp(-pow(x[i] - x[j], 0.2e1) * pow(rho, -0.2e1));
        K[i,j] = pow(alpha, 0.2e1) * K[i,j];
        K[j,i] = K[i,j];
      }
      K[n,n] = pow(alpha, 0.2e1);
    }
    return(K);
  }
}

data {
  int<lower=0> N;
  vector[N] x;
  vector[N] y;

  int<lower=0> N_pred;
  vector[N_pred] x_pred;
}
parameters {
  real<lower=0,upper=5> u; // physical parameter
  real<lower=0, upper=2> sigma; // noise parameter
  real<lower=0, upper=4> alpha; // marginal sd (delta)
  real<lower=0, upper=6> rho; // length scale (delta)
}

model {
  matrix[N, N] cov = cov_exp(x, alpha, rho)+diag_matrix(rep_vector(sigma^2, N));
  matrix[N, N] L_cov = cholesky_decompose(cov);
  // priors
  u ~ normal(1,2);
  y ~ multi_normal_cholesky(5*exp(-u*x), L_cov);
}
```

We fit this model to each individual data set separately

```
lu=list()
for(i in seq_along(u_val)){
  fit_no_with_delta = stan(
    file='STAN/toy/toy_delta.stan', # with delta
    data=dl[[i]],
    chains=3,
    iter=2*1000,
    seed=123
  )
  lu[[i]] = extract(fit_no_with_delta)$u
}
stan_trace(fit_no_with_delta, size=0.2)
```



Model 3 (yes/common delta, Figure 3)

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/toy/toy_common_delta.stan'))
```

```
functions{ // squared exponential kernel
  matrix cov_exp(vector x,
```

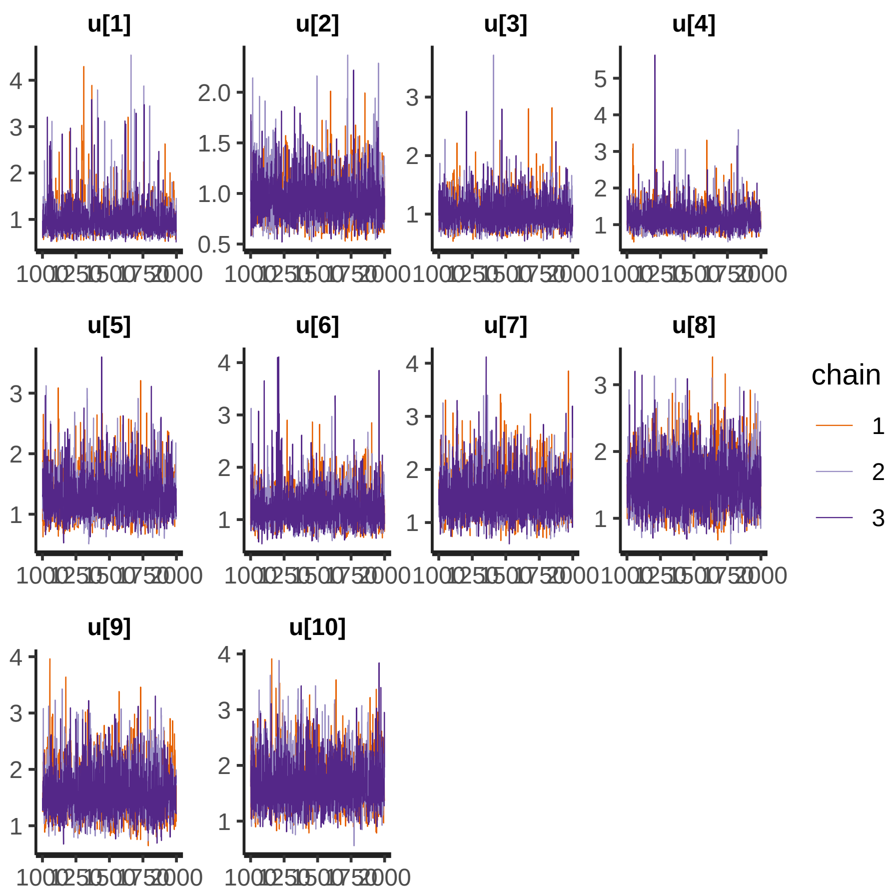
```
                 real alpha,
                 real rho){
    int n = rows(x);
    matrix[n, n] K;
    // KP
    for (i in 1:(n)){
      K[i,i] = pow(alpha, 0.2e1);
      for (j in (i+1):n){
        K[i,j] = exp(-pow(x[i] - x[j], 0.2e1) * pow(rho, -0.2e1));
        K[i,j] = pow(alpha, 0.2e1) * K[i,j];
        K[j,i] = K[i,j];
      }
      K[n,n] = pow(alpha, 0.2e1);
    }
    return(K);
  }
}
data {
  int<lower=1> N; //number of observations per individual
  int<lower=1> Ns;
  int<lower=1,upper=Ns> id[Ns]; //number of individuals
  // vector[N] x; //same across all individuals (e.g. time)
  vector[N] x[Ns]; //different across individuals (e.g. time)
  row_vector[N] y[Ns];
}
parameters {
  real<lower=0, upper=3.0> u_tilde[Ns]; // non-centered parameterization

  real<lower=0, upper=4> alpha; // marginal sd (delta)
  real<lower=0, upper=6> rho;  // length scale (delta)

  real<lower=0> sigma; // noise sd
  real<lower=0.5, upper=1.8> mu; // Global mean for u
  real<lower=1, upper=2> tau; // Global sd for u
}
transformed parameters {
  real<lower=0> u[Ns];   // individual u
  // Non-centered parameterization
  for (s in 1:Ns) {
    u[s] = mu + tau * u_tilde[s];
  }
}
model {
  matrix[N, N] cov[Ns];
  matrix[N, N] L_cov[Ns];
  for (s in 1:Ns) {
    cov[s] = cov_exp(x[s], alpha, rho)+diag_matrix(rep_vector(sigma^2, N));
    L_cov[s] = cholesky_decompose(cov[s]);
  }

  u_tilde ~ normal(0, 1);  // non-centered
  // likelihood
  for (i in 1:Ns){
    y[i] ~ multi_normal_cholesky(5*exp(-u[id[i]]*x[i]), L_cov[i]);
```

```
  }
}
```

```r
# shared u common delta model
fit_yes_common_delta = stan(file='STAN/toy/toy_common_delta.stan',
                            data=data_population,
                            chains=3,
                            iter=2*1000,
                            seed=123
)
names(fit_yes_common_delta)
```

```
 [1] "u_tilde[1]"  "u_tilde[2]"  "u_tilde[3]"  "u_tilde[4]"  "u_tilde[5]"
 [6] "u_tilde[6]"  "u_tilde[7]"  "u_tilde[8]"  "u_tilde[9]"  "u_tilde[10]"
[11] "alpha"       "rho"         "sigma"       "mu"          "tau"
[16] "u[1]"        "u[2]"        "u[3]"        "u[4]"        "u[5]"
[21] "u[6]"        "u[7]"        "u[8]"        "u[9]"        "u[10]"
[26] "lp__"
```

```r
stan_trace(fit_yes_common_delta, pars = "u", size=0.2)
```




```r
ex_ycd=extract(fit_yes_common_delta)
```

Model 4 (yes/shared delta, Figure 3)

We allow individuals to share information about both the physical parameters $u_m, m = 1, 2, \ldots, 10$ and the discrepancy through a global level parameters for bothas described in Section 3.1. The model assumes same discrepancy parameters for all individuals.

Stan code:

```r
writeLines(readLines('STAN/toy/toy_shared_delta.stan'))
```

```
functions{
  matrix cov_exp(vector x,
                 real alpha,
                 real rho){
    int n = rows(x);
    matrix[n, n] K;
    // KP
    for (i in 1:(n)){
      K[i,i] = pow(alpha, 0.2e1);
      for (j in (i+1):n){
        K[i,j] = exp(-pow(x[i] - x[j], 0.2e1) * pow(rho, -0.2e1));
        K[i,j] = pow(alpha, 0.2e1) * K[i,j];
        K[j,i] = K[i,j];
      }
      K[n,n] = pow(alpha, 0.2e1);
    }
    return(K);
  }
}
data {
  int<lower=1> N; // number of observations per individual
  int<lower=1> Ns; // number of individuals
  int<lower=1,upper=Ns> id[Ns]; // individual id
  vector[N] x[Ns]; // individual input vector
  row_vector[N] y[Ns]; // individual output vector
}
parameters {
  // non-centered parameterization parameters
  real<lower=0,upper=3.0> u_tilde[Ns];
  real rho_tilde[Ns];   // non-centered sd of rho (delta process)
  real alpha_tilde[Ns]; // non-centered sd of alpha (delta precess)
  real<lower=0> sigma;  // same noise across individuals

  // Global-level parameters for delta
  real<lower=0> rho_m;   // median of individual log-normal
  real<lower=0> rho_s;   //sd of of individual log-normal
  real<lower=0> alpha_m; // median of alpha log-normal
  real<lower=0> alpha_s; //sd of alpha log-normal

  // Global-level parameters for u
  real<lower=0.5, upper=1.8> mu;
  real<lower=1, upper=2> tau;
}
transformed parameters {
  real<lower=0> u[Ns];   // physical parameters
  real<lower=0> rho[Ns];   // length scale
  real<lower=0> alpha[Ns];  // marginal standard deviation
  // Non-centered parameterization of individual parameters
  for (s in 1:Ns) {
```

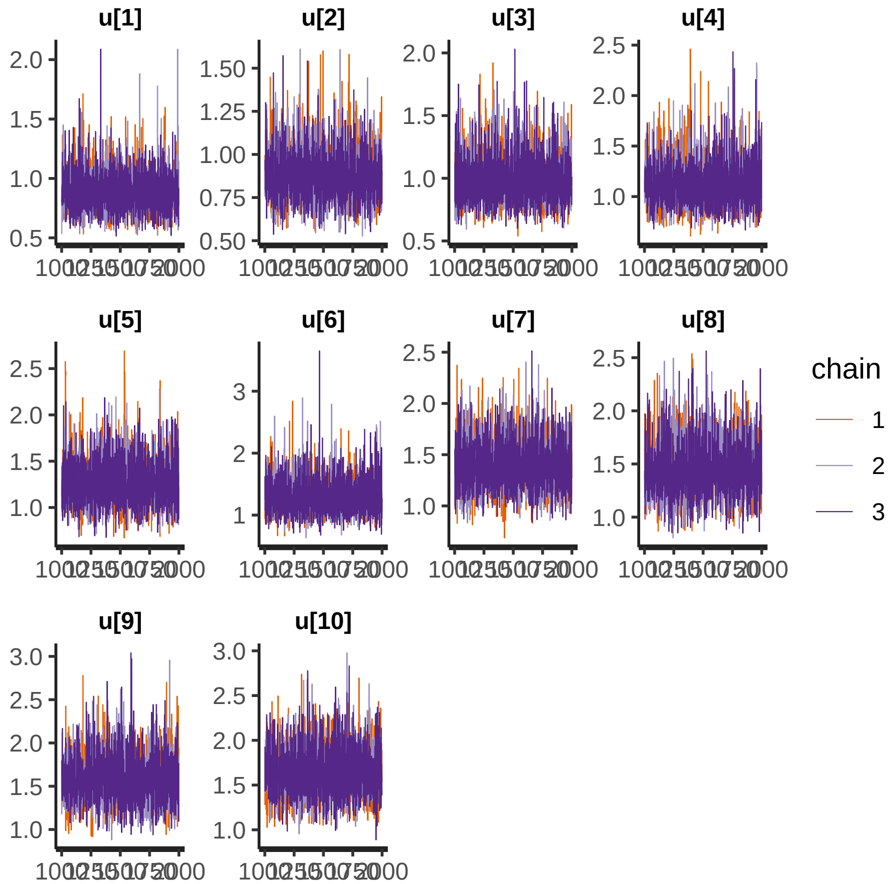
```
    rho[s] = exp(log(rho_m) + rho_s * rho_tilde[s]);
    alpha[s] = exp(log(alpha_m) + alpha_s * alpha_tilde[s]);
    u[s] = mu + tau * u_tilde[s];
  }
}
model {
  matrix[N, N] cov[Ns];
  matrix[N, N] L_cov[Ns];
  for (s in 1:Ns) { // individual covariance
    cov[s] = cov_exp(x[s], alpha[s], rho[s])+diag_matrix(rep_vector(sigma^2, N));
    L_cov[s] = cholesky_decompose(cov[s]);
  }

  // priors
  // Global parameters
  rho_m ~ inv_gamma(2, 0.5);
  alpha_m ~ normal(0,2);
  rho_s ~ normal(0, 0.5);
  alpha_s ~ normal(0, 0.5);

  // non-centered parameterization of individual parameters
  rho_tilde ~ normal(0, 1);
  alpha_tilde ~ normal(0, 1);
  u_tilde ~ normal(0, 1);

  // likelihood
  for (i in 1:Ns){
    y[i] ~ multi_normal_cholesky(5*exp(-u[id[i]]*x[i]), L_cov[id[i]]);
  }
}
```

```r
fit_yes_shared_delta = stan(file='STAN/toy/toy_shared_delta.stan',
                            data=data_population,
                            chains=3,
                            iter=2*1000,
                            seed=123
)
names(fit_yes_shared_delta)
```

```
 [1] "u_tilde[1]"      "u_tilde[2]"      "u_tilde[3]"      "u_tilde[4]"
 [5] "u_tilde[5]"      "u_tilde[6]"      "u_tilde[7]"      "u_tilde[8]"
 [9] "u_tilde[9]"      "u_tilde[10]"     "rho_tilde[1]"    "rho_tilde[2]"
[13] "rho_tilde[3]"    "rho_tilde[4]"    "rho_tilde[5]"    "rho_tilde[6]"
[17] "rho_tilde[7]"    "rho_tilde[8]"    "rho_tilde[9]"    "rho_tilde[10]"
[21] "alpha_tilde[1]"  "alpha_tilde[2]"  "alpha_tilde[3]"  "alpha_tilde[4]"
[25] "alpha_tilde[5]"  "alpha_tilde[6]"  "alpha_tilde[7]"  "alpha_tilde[8]"
[29] "alpha_tilde[9]"  "alpha_tilde[10]" "sigma"           "rho_m"
[33] "rho_s"           "alpha_m"         "alpha_s"         "mu"
[37] "tau"             "u[1]"            "u[2]"            "u[3]"
[41] "u[4]"            "u[5]"            "u[6]"            "u[7]"
[45] "u[8]"            "u[9]"            "u[10]"           "rho[1]"
[49] "rho[2]"          "rho[3]"          "rho[4]"          "rho[5]"
[53] "rho[6]"          "rho[7]"          "rho[8]"          "rho[9]"
[57] "rho[10]"         "alpha[1]"        "alpha[2]"        "alpha[3]"
[61] "alpha[4]"        "alpha[5]"        "alpha[6]"        "alpha[7]"
```

```
[65] "alpha[8]"         "alpha[9]"         "alpha[10]"        "lp__"
```

```
stan_trace(fit_yes_shared_delta, pars = "u", size=0.2)
```

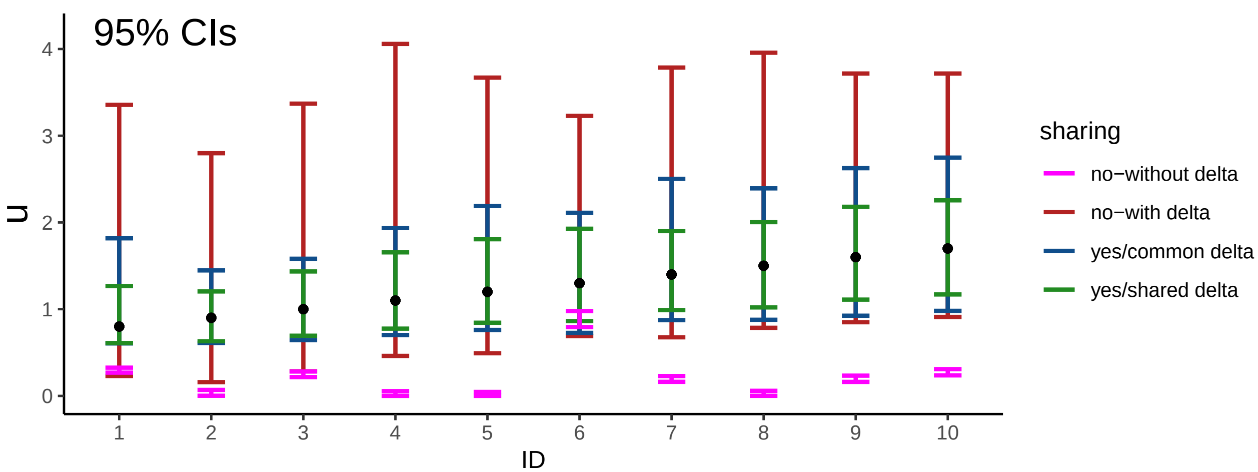



```
ex_ysd=extract(fit_yes_shared_delta)
```

Plot 95% CIs for all methods (Figure 3 in the paper)

```
nodelta_cis=m_nwd=matrix(NA,length(u_val),2)
fn_s = function(x) c(quantile(x, probs = c(0.025,0.975)))

for(i in seq_along(u_val)){
  nodelta_cis[i,] = fn_s(lu_no[[i]])
  m_nwd[i,] = fn_s(lu[[i]])
}

ysd_cis=data.frame(t(apply(ex_ysd$u,2,quantile,probs=c(0.025,0.975))))
yes_common_delta=data.frame(t(apply(ex_ycd$u,2,quantile,probs=c(0.025,0.975))))

m_nwd=data.frame(m_nwd)
colnames(m_nwd) = colnames(ysd_cis)
nodelta_cis=data.frame(nodelta_cis)
colnames(nodelta_cis) = colnames(ysd_cis)
m_nwd$u_true = u_val
m_nwd$sharing= "no-with delta"
nodelta_cis$u_true = u_val
```

```r
nodelta_cis$sharing= "no-without delta"
yes_common_delta$u_true=u_val
yes_common_delta$sharing="yes/common delta"
ysd_cis$u_true=u_val
ysd_cis$sharing="yes/shared delta"

df_CIs = rbind(m_nwd,yes_common_delta,ysd_cis,nodelta_cis)
colnames(df_CIs)[1:2] = c("lower","upper")
df_CIs$id = rep(id,4)

ggplot(df_CIs,aes(x = as.factor(id), y = u_true, ymin = lower, ymax = upper, color=sharing))+
    geom_errorbar(width = 0.3, size=0.9) +
    theme_classic()+
    geom_point(size = 1.5, color="black")+
    ylab("u")+xlab("ID")+
    annotate("text", x=1.5, y=4.2, label= "95% CIs", size=6)+
    theme(axis.title.y = element_text(size = rel(1.5)))+
    scale_color_manual(
      breaks=c('no-without delta', 'no-with delta', "yes/common delta", "yes/shared delta"),
      values=c("magenta","firebrick","dodgerblue4", "forestgreen"))
```



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] SAVE_1.0             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          DiceKriging_1.6.0  purrr_0.3.4
 [5] lattice_0.20-45    colorspace_2.0-2   vctrs_0.3.8        generics_0.1.2
 [9] htmltools_0.5.2    stats4_4.0.3       loo_2.4.1          yaml_2.2.2
[13] utf8_1.2.2         rlang_1.0.0        pkgbuild_1.3.1     pillar_1.7.0
[17] glue_1.6.1         withr_2.4.3        DBI_1.1.2          matrixStats_0.61.0
[21] lifecycle_1.0.1    stringr_1.4.0      munsell_0.5.0      gtable_0.3.0
[25] codetools_0.2-18   coda_0.19-4        evaluate_0.14      labeling_0.4.2
[29] inline_0.3.19      knitr_1.37         callr_3.7.0        fastmap_1.1.0
[33] ps_1.6.0           parallel_4.0.3     fansi_1.0.2        Rcpp_1.0.8
[37] scales_1.1.1       RcppParallel_5.1.5 farver_2.1.0       gridExtra_2.3
[41] digest_0.6.29      stringi_1.7.6      processx_3.5.2     dplyr_1.0.7
[45] grid_4.0.3         cli_3.1.1          tools_4.0.3        magrittr_2.0.2
[49] tibble_3.1.6       crayon_1.4.2       pkgconfig_2.0.3    ellipsis_0.3.2
[53] prettyunits_1.1.1  assertthat_0.2.1   rmarkdown_2.11     rstudioapi_0.13
[57] R6_2.5.1           compiler_4.0.3
```