Analyse qPCR survey data

2025-10-05

```
set.seed(123)
```

Analysing qPCR survey data

The models are written in NIMBLE (version 1.3.0) which must be installed prior to using this code.

```
library(nimble)
```

This document is a walkthrough to analysing qPCR survey data using the code in 'Model Codes.R'. This walkthrough will only cover analysis using Model 1, the full model accounting for contamination/inhibition and CT heteroscedasticity.

```
source('Model Codes.R')
```

Data set up

We begin by describing how the qPCR survey data needs to be formatted. We use the data set generated in 'Simulate Data Walkthrough' as an example.

```
load('Walkthrough.RData')
data_standards <- data$dataParams$data_standard # qPCR standards
data_survey <- data$dataParams$data_real # qPCR survey data
head(data_survey, n = 20)
```

```
##    Site Time Sample Plate       Ct
## 1     1    1      1     1 39.82511
## 2     1    1      1     1       NA
## 3     1    1      1     1 39.80312
## 4     1    1      1     1       NA
## 5     1    1      1     1 39.46689
## 6     1    1      2     1 38.62479
## 7     1    1      2     1 39.12887
## 8     1    1      2     1 39.67849
## 9     1    1      2     1 39.36833
## 10    1    1      2     1 39.37273
## 11    1    1      3     1       NA
## 12    1    1      3     1 39.49379
## 13    1    1      3     1 37.40572
## 14    1    1      3     1       NA
## 15    1    1      3     1 39.12627
## 16    1    1      4     1 38.84186
```

```
## 17    1    1      4     1 38.78332
## 18    1    1      4     1 37.28563
## 19    1    1      4     1 37.31741
## 20    1    1      4     1 38.54617
```

```
head(data_standards, n=20)
```

```
##    Plate Quantity       Ct
## 1      1    3e+07 14.56782
## 2      1    3e+07 14.67064
## 3      1    3e+07 14.50085
## 4      1    3e+06 22.14972
## 5      1    3e+06 18.40469
## 6      1    3e+06 18.28708
## 7      1    3e+05 22.26213
## 8      1    3e+05 22.13055
## 9      1    3e+05 22.39131
## 10     1    3e+04 26.11283
## 11     1    3e+04 26.60363
## 12     1    3e+04 26.49581
## 13     1    3e+03 33.72538
## 14     1    3e+03 29.69682
## 15     1    3e+03 30.58466
## 16     1    3e+02 33.95999
## 17     1    3e+02 37.05140
## 18     1    3e+02 38.35571
## 19     1    3e+01 38.55236
## 20     1    3e+01 37.44642
```

Survey data

Each row of data_survey corresponds to a single qPCR replicate. data_survey is a data frame of five columns:

1. Site: (integer $\in (1, n)$) The site at which the replicate was collected
2. Time: (integer $\in (1, nT)$) The time-point at which the replicate was collected
3. Sample: (integer) The sample ID of the replicate
4. Plate: (integer) The plate on which the replicate was run
5. Ct: (double > 0) The cycles to threshold for the replicate. If the cycle doesn't amplify, this reads NA.

From the output above, for example we see that sample 1 (collected at site 1 and time-point 1) was split into 5 replicates and run on plate 1. The first replicate amplified with CT 39.825, but the second replicate failed to amplify.

Standards

Each row of data_standards corresponds to a single qPCR replicate. data_standards is a data frame of 3 columns:

1. Plate: (integer $\in (1, n)$) The plate on which the replicate was run
2. Quantity: (double > 0) The concentration of DNA in the replicate

3. Ct: (double > 0) The cycles to threshold for the replicate. If the cycle doesn't amplify, this reads NA.

The Plate ID here are the same as the ID's used in the data_survey data frame. So from the output above, we see that on plate 1 (the same plate as samples collected at site 1 and time-point 1), seven different standard concentrations were used (3e+1 to 3e+7) and each split into 3 replicates.

Covariates

Alongside the qPCR data, there may be data about covariates collected at each sampling occasion and for each sample. These are to be laid out in the following way:

```
Xb <- data$dataParams$X_b # site level covariates
Xw <- data$dataParams$X_w # sample level covariates

print('Site 1 time-points 1 to 5:')
```

```
## [1] "Site 1 time-points 1 to 5:"
```

```
Xb[1,1:5,]
```

```
##            [,1] [,2]
## [1,] -0.5604756    0
## [2,]  1.2240818    1
## [3,] -1.0678237    0
## [4,]  0.4264642    0
## [5,] -0.6947070    0
```

```
print('Site 2 time-points 1 to 5:')
```

```
## [1] "Site 2 time-points 1 to 5:"
```

```
Xb[2,1:5,]
```

```
##            [,1] [,2]
## [1,] -0.2301775    0
## [2,]  0.3598138    1
## [3,] -0.2179749    0
## [4,] -0.2950715    1
## [5,] -0.2079173    0
```

```
print('Sampling covariates')
```

```
## [1] "Sampling covariates"
```

```
head(Xw)
```

```
##            [,1] [,2]
## [1,] -0.7152422    1
## [2,] -0.7526890    1
## [3,] -0.9385387    0
## [4,] -1.0525133    1
## [5,] -0.4371595    0
## [6,]  0.3311792    0
```

Site level covariates Xb is an array with i,j-th row corresponding to covariate observations from site i and time-point j (we recommend that continuous covariates by standardized). In this data set we can see that 2 covariates (one continuous and one binary) have been collected at each site and time-point. Shown above are the covariate values for sites 1 and 2 in the first 5 time-points.

Sample level covariates Xw is a matrix with i-th row corresponding to the i-th sample in the survey. Each column corresponds to a different covariate (we recommend that continuous covariates by standardized). In this data set we can see that 2 covariates (one continuous and one binary) have been collected with each sample. Shown above are the covariate values for the first 6 samples (the first 5 of which from data_survey we know are from site 1 and time-point 1).

Note: The model codes are not currently set up to handle missing covariate observations.

MCMC set up

Now we explain how to set up the data, constants, and initial values for the MCMC.

Data

```
DEdata <- list(Ct = data_survey$Ct,
               Ct_star = data_standards$Ct,
               delta_inv = 1*(is.na(data_survey$Ct)),
               delta_star_inv = 1*(is.na(data_standards$Ct)),
               w_star = data_standards$Quantity,
               Xb = Xb,
               Xw = Xw,
               constraint_data = 1)
```

The data are stored in a list with named elements:

1. Ct: (double > 0, vector) the Ct column from data_survey
2. Ct_star: (double > 0, vector) the Ct column from data_standards
3. delta_inv: (binary, vector) i-th entry is 1 if the i-th replicate from data_survey failed to amplify (0 otherwise)
4. delta_star_inv: (binary, vector) i-th entry is 1 if the i-th replicate from data_standards failed to amplify (0 otherwise)
5. w_star: (double > 0, vector) the Quantity column from data_standards
6. Xb: (double, array) the Xb array of site level covariates
7. Xw: (double, matrix) the Xw matrix of sample level covariates
8. constraint_data: (binary) a modelling constraint that forces the number of contaminated and inhibited replicates to be fewer than the number of unaffected replicates. (Recommend leaving this as 1).

Constants

The constants for the model are variables that control the indexing over sites, time-points, samples, and replicates, as well as setting up the hyperparameters for the prior distributions for model variables. The prior distributions used in the model are shown in the table below.

Linking the samples and replicates to their respective sampling occasions is done using the following vectors: id_site_l, id_site_time, id_site, and id_sample.

Table 1: Table of prior distributions. Columns indicate the parameter name, its use in the model, and the prior distribution used. Inverse Gamma distribution uses the shape/scale parametrisation

Parameter	Use	Prior
β_b	Coefficients on site covariates	$\sim$ N(0, 1)
β_w	Coefficients on sample covariates	$\sim$ N(0, 1)
$\beta_{b,0}$	Intercept on log-DNA at $t = 1$	$\sim$ Exp-Unif(0, b.max)
τ^2	Variance of log-DNA across time	$\sim$ InvGamma(a-sigma.tau, b-sigma.tau)
τ_1^2	Variance of log-DNA across sites	$\sim$ InvGamma(a-sigma.tau1, a-sigma.tau1)
σ^2	Variance of log-DNA across samples	$\sim$ InvGamma(a-sigma, a-sigma)
$(p_c, 1 - p_c - p_h, p_h)$	Probability of contamination or inhibition	$\sim$ Dir(alpha)
a_1	CT log-variance intercept term	$\sim$ N(a0, sd-a)
a_2	CT log-variance slope term	$\sim$ N(b0, sd-b)
α_p^1	Plate regression intercepts	$\sim$ N(alpha10, sigma-alpha)
α_p^2	Plate regression slopes	$\sim$ N(alpha20, sigma-alpha)
ρ_0	AR(1) coefficient mean	$\sim$ N(1, 1)
σ_ρ^2	AR(1) coefficient noise	$\sim$ InvGamma(a-sigma.rho, b-sigma.rho)
σ_c	Standard deviation for contaminated or inhibited replicate	sd.cont

The survey has $n = 10$ sites and $nT = 20$ time-points. This gives $n \times nT = 200$ total sampling occasions. id_site_l (vector of length 200) links each sampling occasion to its site. id_site_time (vector of length 200) links each sampling occasion to its time-point

```
id_site_l <- unique(data_survey[,c('Site', 'Time')])[,'Site']
id_site_time <- unique(data_survey[,c('Site', 'Time')])[,'Time']

id_site_l[51]; id_site_time[51]
```

```
## [1] 3
```

```
## [1] 11
```

For example, sampling occasion 51 occurs at site 3 and at time-point 11.

The survey has a total of 1000 samples (each sampling occasion took 5 samples from the environment). id_site links each sample to its sampling occasion.

```
num_samples <- max(data_survey$Sample)
df <- unique(data_survey[,c('Site', 'Time', 'Sample')])
id_site <- sapply(1:num_samples, function(x){which(id_site_l == df[x,'Site'] &
                                                   id_site_time == df[x,'Time'])})

id_site[101:105]
```

```
## [1] 21 21 21 21 21
```

```
id_site_l[21];id_site_time[21]
```

```
## [1] 2
```

```
## [1] 1
```

For example, samples 101 to 105 were taken at sampling occasion 21, which corresponds to site 2 and time-point 1.

The survey has a total of 5000 replicates (each sample is split into 5 technical replicates). id_sample links each replicate to its sample (in other words this is the Sample column from data_survey).

```
id_sample <- data_survey$Sample
id_sample[1001:1005]
```

```
## [1] 201 201 201 201 201
```

For example, replicates 1001 to 1005 were taken from sample 201.

```
DEconstants = list(numP = max(data_survey$Plate),
                   nT = max(data_survey$Time),
                   ncovb = 2, ncovw = 2,
                   num_sites = max(data_survey$Site),
                   num_samples = max(data_survey$Sample),
                   num_replicates = nrow(data_survey),
                   num_replicates_star = nrow(data_standards),
                   P = data_survey$Plate, P_standards = data_standards$Plate,
                   CT.max = 40,

                   id_site = id_site,
                   id_sample = id_sample,
                   id_site_time = id_site_time,
                   id_site_l = id_site_l,

                   a_sigma.tau = 2, b_sigma.tau = 1,
                   a_sigma = 2, b_sigma = 1,
                   a_sigma.tau1 = 2, b_sigma.tau1 = 1,
                   a_sigma.rho = 2, b_sigma.rho = .1,
                   alpha10 = 0, alpha20 = 0,
                   sigma_alpha = 100,
                   a0 = 0, b0 = 0,
                   sd_a = 10, sd_b = 10,
                   sd_cont = 30,
                   b.max = exp(30),
                   alpha = c(0.01, 0.98, 0.01))
```

The constants are stored in a list with named elements:

1. numP: (integer) the number of plates in the survey
2. nT: (integer) the number of time-points in the survey
3. ncovb: (integer) the number of site level covariates
4. ncovw: (integer) the number of sample level covariates
5. num_sites: (integer) the number of sites in the survey
6. num_samples: (integer) the number of samples in the survey
7. num_replicates: (integer) the number of replicates in the survey (excluding standards)
8. num_replicates_star: (integer) the number of replicates in the standards

9. P: (integer, vector of length num_replicates) i-th entry corresponds to the plate that the replicate from the i-th row of data_real was analysed on
10. P_standards: (integer, vector of length num_replicates_star) i-th entry corresponds to the plate that the replicate from the i-th row of data_standard was analysed on
11. CT.max: (integer) the maximum threshold value, after which the qPCR stops running cycles.
12. id_site: (integer, vector of length num_samples) i-th entry denotes the sampling occasion j that sample i was collected from
13. id_sample: (integer, vector of length num_replicates) i-th entry denotes the sample that the i-th row of data_real is associated with
14. id_site_l: (integer, vector of length n x nT) i-th entry denotes the site associated with sampling occasion i
15. id_site_time: (integer, vector of length n x nT) i-th entry denotes the time associated with sampling occasion i 16-34. a_sigma.tau, b_sigma.tau, a_sigma.tau1, b_sigma.tau1, a_sigma, b_sigma, a_sigma.rho, b_sigma.rho, alpha10, alpha20, sigma_alpha, a0, b0, sd_a, sd_b, sd_cont, b.max, alpha: (double) hyperparameters as defined in the table above.

Initial values

```
ncovb <- 2; ncovw <- 2
num_sites <- max(data_survey$Site); nT <- max(data_survey$Time)
CT.max = 40

## Initial values for the missing Ct values in survey and standard data frames ##
Ct_inits = 1*(is.na(data_survey$Ct))
Ct_inits[Ct_inits == 0] = NA
Ct_inits[Ct_inits == 1] = CT.max + 2

Ct_star_inits = 1*(is.na(data_standards$Ct))
Ct_star_inits[Ct_star_inits == 0] = NA
Ct_star_inits[Ct_star_inits == 1] = CT.max + 2

###

DEinits <- function(){list(betab = rep(0, ncovb),
                           betaw = rep(0, ncovw),
                           betab0 =  6,

                           Ct = Ct_inits, Ct_star = Ct_star_inits,
                           type = rep(2, nrow(data_survey)),
                           type_star = rep(2, nrow(data_standards)),
                           pi.type = c(0.01, 0.98, 0.01),

                           tau2 = 1,
                           tau2.1 = 1,
                           sigma2 = 1,
                           rho0 = 1, sd_rho2 = .1,
                           rho = rep(1, num_sites),

                           alpha1 = rep(44, max(data_survey$Plate)),
                           alpha2 = rep(-1.7, max(data_survey$Plate)),
                           a =  0.5, b =  0,
```

```
                            l = matrix(6, nrow=num_sites, ncol = nT),
                            v = rep(6, num_samples)
)}
```

The initial values are a function that returns a list with named elements:

1. betab: (double, vector of length ncovb) initial values for site level covariates (default = 0)
2. betaw: (double, vector of length ncovw) initial values for sample level covariates (default = 0)
3. betab0: (double) initial value for mean log-DNA concentration at time-point 1
4. Ct: (double, vector of length num_replicates) initial values for unobserved Ct values in data_survey (those that fail to amplify). Entries for which the Ct is observed are set to NA. Unobserved Ct values set above CT.max (default = CT.max + 2). i-th entry corresponds to the i-th row of data_survey
5. Ct_star: (double, vector of length num_replicates_star) initial values for unobserved Ct values in data_standards (those that fail to amplify). Entries for which the Ct is observed are set to NA. Unobserved Ct values set above CT.max (default = CT.max + 2). i-th entry corresponds to the i-th row of data_standards
6. type: (integer $\in (1, 2, 3)$, vector of length num_replicates) initial values for the type of each replicate in data_survey. A value of 1 indicates replicate contamination, 3 indicates replicate inhibition, 2 indicates replicate is normal. i-th entry corresponds to the i-th row of data_survey. (default = 2)
7. type_star: (integer $\in (1, 2, 3)$, vector of length num_replicates_star) initial values for the type of each replicate in data_standards A value of 1 indicates replicate contamination, 3 indicates replicate inhibition, 2 indicates replicate is normal. i-th entry corresponds to the i-th row of data_standards. (default = 2)
8. pi.type: (double $\in (0, 1)$, vector of length 3) initial value for the probability that a replicate is contaminated, normal, and inhibited respectively. Values must sum to 1.
9. tau2: (double) initial value for variance of log-DNA concentrations across time
10. tau2.1: (double) initial value for variance of log-DNA concentrations at time-point 1
11. sigma2: (double) initial value for variance of log-DNA concentrations in samples
12. rho0: (double) initial value for AR(1) coefficient mean across sites
13. sd_rho2: (double) initial value for AR(1) coefficient standard deviation across sites
14. rho: (double, vector of length num_sites) initial values for the AR(1) coefficient for each site. i-th entry corresponds to site i. (default is 1)
15. alpha1: (double, vector of length numP) initial values for the intercept in the log-CT plate regression. i-th entry corresponds to plate i.
16. alpha2: (double, vector of length numP) initial values for the slope in the log-CT plate regression. i-th entry corresponds to plate i.
17. a: (double) initial values for the intercept in the plate Ct log-variance. This is also denoted a_1 in the model.
18. b: (double) initial values for the slope in the plate Ct log-variance. This is also denoted a_2 in the model.
19. l: (double, n x nT matrix) matrix of initial values for log-DNA concentrations across each site and time-point. The i,j-th entry denotes the initial value for site i and time-point j.
20. v: (double, vector of length num_samples) vector of initial values for the log-DNA concentration in each sample. The i-th entry denotes the initial value for sample i.

The choice of appropriate initial values (and of prior distributions) is dependent on the data set (for example: looking at average DNA concentrations throughout the survey) and on qPCR lab set-ups (for example: looking at typical intercept and slope for log-CT plate regression). The initial values given above are examples only.

Run MCMC

Once all the data, constants, and initial values have been set up, the MCMC can be run. We show an example MCMC run here, but refer you to the NIMBLE UserManual (here) for more details on NIMBLE if required.

```
DEmodel = nimbleModel(Full_Code,
                      constants = DEconstants,
                      data = DEdata,
                      inits = DEinits())
```

```
## Defining model

##   [Note] Registering 'dbetab0' as a distribution based on its use in BUGS code. If you make changes t

##   [Note] Registering 'dCt' as a distribution based on its use in BUGS code. If you make changes to th

## Building model

## Setting data and initial values

## Running calculate on model
##   [Note] Any error reports that follow may simply reflect missing values in model variables.

## Checking model sizes and dimensions

##   [Note] This model is not fully initialized. This is not an error.
##          To see which variables are not initialized, use model$initializeInfo().
##          For more information on model initialization, see help(modelInitialization).
```

The monitors in configureMCMC is a vector of the named variables that are tracked throughout the MCMC.

```
# a and b in the monitors are the a_1 and a_2 (variance heteroscedasticity
# parameters) respectively.

mcmcConf = configureMCMC(DEmodel, monitors = c('l', 'v',
                                               'betaw', 'betab', 'betab0',
                                               'tau', 'tau.1', 'sigma',
                                               'rho', 'a', 'b',
                                               'alpha1', 'alpha2',
                                               'pi.type', 'type', 'type_star'),
                                print = FALSE)

DEmcmc <- buildMCMC(mcmcConf, print = FALSE)
cDEmodel <- compileNimble(DEmodel)
```

```
## Compiling
##   [Note] This may take a minute.
##   [Note] Use 'showCompilerOutput = TRUE' to see C++ compilation details.
```

```
cDEmcmc <- compileNimble(DEmcmc, project = cDEmodel)
```

```
## Compiling
##   [Note] This may take a minute.
##   [Note] Use 'showCompilerOutput = TRUE' to see C++ compilation details.
```

```
DEresults <- runMCMC(cDEmcmc, niter = 110000, nburnin = 10000, nchains = 1,
                     thin = 10, samplesAsCodaMCMC = T)
```

```
## running chain 1...
```

```
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
```

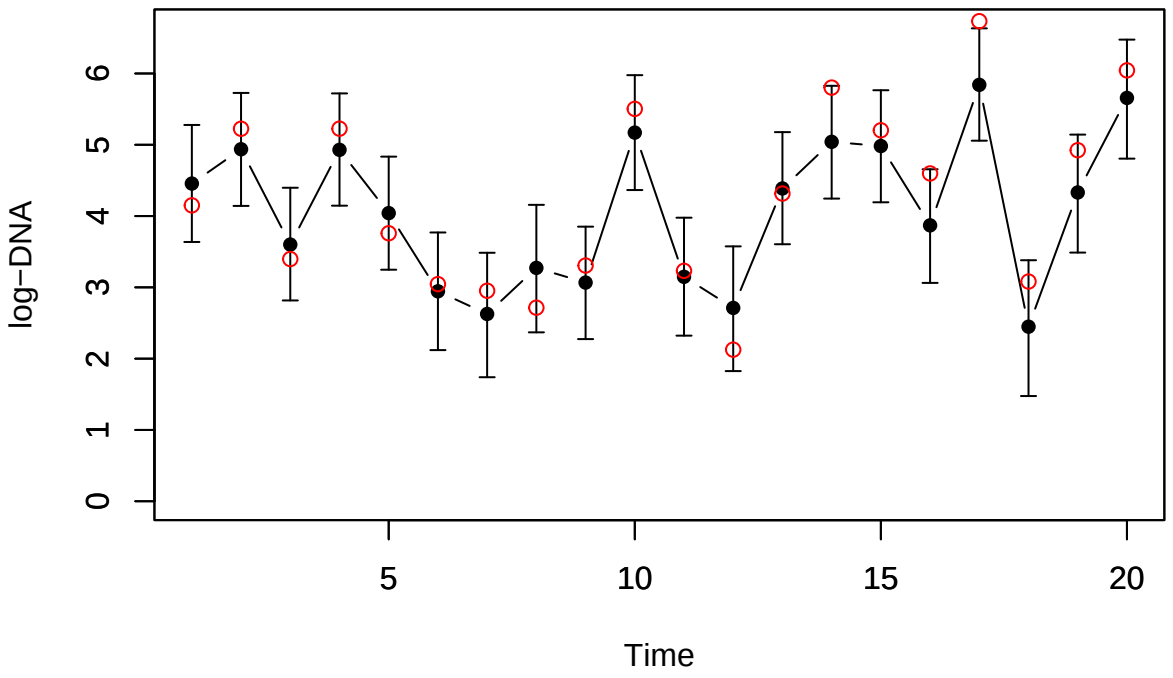
MCMC output

We can view the posterior distributions for the log-DNA concentrations across sites and time. The posterior chain for log-DNA concentrations for site i and time-point j are stored under the column with heading `l[i, j]`. Below we show the posterior means and 95% credible intervals for the concentrations across time at site 1. The posterior results are shown in black, the red shows the true values (known from the simulated data set).

```
site = '1'
posterior_names <- paste0('l[', site, ', ', as.character(1:max(data_survey$Time)), ']')

l.means <- apply(DEresults[, posterior_names], 2, mean)
l.ucl <- apply(DEresults[, posterior_names], 2, quantile, probs = 0.975)
l.lcl <- apply(DEresults[, posterior_names], 2, quantile, probs = 0.025)

errbar(1:max(data_survey$Time),l.means,
       l.ucl,l.lcl,type='b',
       xlab = 'Time', ylab = 'log-DNA',
       ylim = c(0, max(l.ucl)))
par(new =T)
plot(1:max(data_survey$Time),
     data$trueParams$l_true[1:max(data_survey$Time)],
     xlab = '', ylab = '', col = 'red',
     ylim = c(0, max(l.ucl)))
```



Similarly we can view the posterior distributions for the site and sample level covariate effects. The posterior chain for the i-th site-level covariate effect is stored under the column with heading `betab[i]`, and for the i-th sample level covariate effect, `betaw[i]`. Below we show the posterior means and 95% credible intervals for the site and sampling covariate effects alongside the true values (known fro the simulated data set).

```
betaw.means <- apply(DEresults[, c('betaw[1]', 'betaw[2]')], 2, mean)
betab.means <- apply(DEresults[, c('betab[1]', 'betab[2]')], 2, mean)

betaw.ucl <- apply(DEresults[, c('betaw[1]', 'betaw[2]')], 2, quantile, probs = 0.975)
betab.ucl <- apply(DEresults[, c('betab[1]', 'betab[2]')], 2, quantile, probs = 0.975)

betaw.lcl <- apply(DEresults[, c('betaw[1]', 'betaw[2]')], 2, quantile, probs = 0.025)
betab.lcl <- apply(DEresults[, c('betab[1]', 'betab[2]')], 2, quantile, probs = 0.025)

df <- data.frame(Mean = c(betab.means, betaw.means),
                 UCL = c(betab.ucl, betaw.ucl),
                 LCL = c(betab.lcl, betaw.lcl),
                 Truth = c(data$trueParams$beta_b_true, data$trueParams$beta_w_true))

print(df)
```

```
##                Mean        UCL        LCL Truth
## betab[1]  0.8730475  1.0119710  0.7359409     1
## betab[2] -1.0927883 -0.8420143 -1.3365640    -1
## betaw[1]  1.0275068  1.0981546  0.9571338     1
## betaw[2] -0.9601169 -0.8213474 -1.0974516    -1
```

Other results can be determined similarly.