Multiple flow FeNO in population research: R code methods demo

The following demonstration provides example R codes to implement six methods for cross-sectional studies with multiple flow exhaled nitric oxide (FeNO) measurements, where the goal is to relate estimated NO parameters to factors of interest (i.e., covariate(s) X) in the study population.

0. Preliminaries: set up your R session

IMPORTANT! To apply out bayesian approaches, please install JAGS, if not already installed, follow instructions at: https://mcmc-jags.sourceforge.io/

Load the following required packages, installing first if not already available. To use the R2jags package, JAGS must be pre-installed.

```
# set working directory
setwd("K:/paper/newpackage/PopFeNO")
# load required packages
require(MASS)
require(lme4)
require(nlme)
require(reshape2)
require(R2jags)
require(knitr)
```

1. FeNO data preparation

We use the long format FeNO data for all methods. The long format refers to the data structure with rows for maneuvers and columns for variables, e.g. measured FeNO, flow rates, and environmental factor/s. Our example was based on single factor X, while the models were also available to deal with multiple linear regression models or other more complex models with some modification. We also impute the measured FeNO with natural logarithmic transformation to adapt its increased randomness (ref). Each participant may took different numbers of maneuver at multiple exhale flow rates. There is an addition column, "target flow rates" to help categorizing the flow rates into low, median and high levels for Högman & Meriläneen Algorithm (HMA) method. The flow rates were not necessary been exactly the same within categories, since it was not realistic during test. The popular "target flow rates" were 30, 50, 100 and 300ml/s where HMA methods only used 30, 50, and 300 ml/s. Users may input their own collected data or generate it via our simulation function (**DataGeneratorCS**) based on the two compartment (2CM) FeNO function.

$$FeNO = C_{aw} + (C_A - C_{aw}) \times e^{-\frac{D_{aw}}{flow}}$$

Along with the available real world FeNO data, we also provide data simulation functions to mimic the actual data based on the two compartment model. Here we modified the 2CM model to log transform both sides of it and used the logged CawNO and DawNO form. We wrote the distribution function as follows:

$$log(FeNO) \sim \mathcal{N}\left(exp(logC_{aw}) + (C_A - exp(logC_{aw})) \times e^{-\frac{exp(logD_{aw})}{flow}}, \sigma^2\right)$$

Several parameters were required for simulation FeNO data. We set the population mean, variances and correlations of NO parameters as default with the values measures from part of CHS (ref) data. Users can update them with their own numbers.

Parameters required for simulate FeNO data are:

1: Number of Participants: **Ndat**

2: The variance covariance matrix of NO parameters [Ca, log(CawNO), log(DawNO)]: **NOcov**, a 3 by 3 matrix, which should be positive definite.

3: Flow rates for each maneuver: **flow**, a vector, length equals to total number of observations (maneuver), ordered by the participant id

4: Standard deviation of measurement error: **SDerror**, we defined the measurement error follows normal distribution with mean 0 and standard deviation

5: Environmental factor: **X**, a vector, length equals to total number of participants, ordered by the participant id.

6: Coefficient: **beta**, a length 3 vector, the effect sizes for each of the three NO parameters' response to environmental factor **X**

Data simulation Process

1: set a random seed for your work.

```
set.seed(2022)
```

2: Demonstration of the Data generation function:

Arguments:

Ndat: 500 participants

Nocov: The covariance matrix of NO parameters were constructed with:

SD Ca	SD logCaw	SD logDaw	Cor(Ca, logCaw)	Cor(Ca, logDaw)	Cov(logCaw, logDaw)
0.45	0.65	0.55	0.66	-0.38	-0.35

Flow rates: With a dropping/missing rate of 0.1 and random deviation taken the form of standard normal distribution, each participant was assigned a target flow rates of 30, 50, 100, and 300 ml/s

SDerror: The standard deviation of measurement error equals to 0.1

X: identically independently distributed vector, generated by rnorm(Ndat,0,1)

beta: The covariate coefficients equal to 0.1 for each NO parameter

In this example, we generated data for 500 participants with a single vector X (Environmental factor, length=500), which had a standard normal distribution (though in practice, X can have any distribution) and multiple flow FeNO (8 maneuvers per participant, two each at the target flow rates of 30, 50, 100, and 300 ml/s). Participant-level NO parameters (CANO, CawNO and DawNO) are generated with linear functions of X, except that rather than CawNO or DawNO, we generate logCawNO and logDawNO to better replicate the approximately log normal distribution observed in practice and to avoid negative estimates in later modeling. For example, participant-level $CANO_i = 1.5 + 0.1 \times X_i + \epsilon_i$ where the standard deviation of the random normal error ϵ_i is a normal distributed: $\epsilon \sim \mathcal{N}(0, 0.45)$. While we could generate each of the

three NO parameters separately, we instead generate the three NO parameters from a multivariate normal distribution to introduce correlation between NO parameters. Given the NO parameters, FeNO at each maneuvers can then be generated according to the two compartment model mentioned above.

3: Allow unbalance and randomness for flow rates

To mimic the real world data, we allows for unbalance and randomness of flow rates in our simulated data by a given probability of dropping/missing of maneuvers and random noises as the deviates from the target flow during test. The final FeNO data could have different numbers of maneuvers for each participant and the flow rates could be slightly different from the target flow rates.

Example 1: Assume every participants has same number of maneuvers and the flow rates were perfectly as same as the target rate, the flow rate matrix for 10 participants and target flow rates: two each of 30, 50, 100, and 300 ml/s will be:

```
FlowFun(Ndat = 5, flowTarget = c(30, 30, 50, 50, 100, 100, 300,
    300), droprate = 0, sigma = 0)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]   30   30   50   50  100  100  300  300
## [2,]   30   30   50   50  100  100  300  300
## [3,]   30   30   50   50  100  100  300  300
## [4,]   30   30   50   50  100  100  300  300
## [5,]   30   30   50   50  100  100  300  300
```

Example 3: Based on example 1, assume there was a 10% drop rate over all observations:

```
FlowFun(Ndat = 5, flowTarget = c(30, 30, 50, 50, 100, 100, 300,
    300), droprate = 0.1, sigma = 0)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]   30   30   50   50  100  100  300  300
## [2,]   30   30   50   50  100  100  300  300
## [3,]   30   30   50   50  100  100  300  300
## [4,]   30    0   50    0  100  100  300  300
## [5,]   30   30   50   50  100  100  300  300
```

Example 2: Based on example 1, assume the flow rate at measurement were not perfectly as same as the target flow rates, the variation has noises followed a standard normal distribution.

```
FlowFun(Ndat = 5, flowTarget = c(30, 30, 50, 50, 100, 100, 300,
    300), droprate = 0, sigma = 1)
```

```
##          [,1]     [,2]     [,3]     [,4]      [,5]      [,6]     [,7]     [,8]
## [1,] 30.36446 30.38365 51.11341 51.21151  99.65167  99.14045 300.6500 300.3281
## [2,] 29.48205 29.76102 50.11778 50.83152  98.44108  99.77948 299.1828 301.0767
## [3,] 31.07966 30.14213 50.15698 49.83128  99.73096 100.80777 298.8753 298.5692
## [4,] 30.06036 29.20702 50.34028 49.74053  98.69515 100.36817 301.6932 300.9958
## [5,] 30.18675 31.23834 50.30937 50.63572 100.02318 101.17786 299.5465 300.4159
```

This is just our pre-determined form of flow rates. Users can construct own flow rate matrix for the data simulation function. But the target flow vector and the flow matrix can not be NULL at the same time. If the flow matrix is not provided, we assume you would like to simulate it via target flow vector together with drop rate and noise argument (which were set to be 0 as default). If the target flow is not provided, it does not affect the data simulation process, you just need to manually add it when applying HMA approach.

```
# Simulate a multiple flow FeNO dataset in a study population
# Users can use their own flow data, with labels of target flows as "low", "medium" and "high"
# The input flow data should be in matrix form, each row represent a participant.
# Labels of target flow can be attached at last or been manually added after the data generated.
# The number of flows for each participant are not necessary equal.

source("DataG.R")       # code to generate multiple flow FeNO datasets
flow=c(rep(30,2),rep(50,2),rep(100,2),rep(300,2))
truebeta=c(0.1,0.1,0.1)
out <- DataGeneratorCS(Ndat       = 100,
                       # number of study participants
                       popmean    = c(1.5,3.5,2.5),
                       # vector of NO parameter population means,
                       # when X=0, in this order: CA, logCaw, logDaw,
                       beta= truebeta,
                       # vector of regression coefficients relating X to NO parameters,
                       # In this order: CA, logCaw, logDaw
                       flowTarget = flow,
                       # Targeted flow rate for the test
                       # Or used for generating the flow rate for simulation
                       Flow       = NULL,
                       # Must be a Ndat*max length of flow matrix, missing values were 0s
                       noise      = 0.1,
                       droprate   = 0.1,
                       SD                    =0.1,   #SD of the error
                       sdalphaCa             = 0.45, # population SD of CA
                       sdalphalogCaw         = 0.65, # population SD of logCaw
                       sdalphalogDaw         = 0.55, # population SD of logDaw
                       coralphalogCawCa      = 0.66, # correlation of logCaw, CA
                       coralphalogCawlogDaw = -0.35,# correlation of logCaw, logDaw
                       coralphalogDawCa      = -0.38 # correlation of logDaw, CA
                     )
dat<-out$dat
datX<-out$datX
datNOparam<-out$datNOparam
```

4: The resultant data set is in the usual 'long' data format:

	id	eno	logeno	flow	flowTarget
1	1	9.499030	2.2511897	29.94180	30
3	1	6.813948	1.9189717	49.94722	50
4	1	5.947526	1.7829753	49.86257	50
5	1	4.070802	1.4038401	99.80663	100
6	1	4.087459	1.4079236	100.09798	100
7	1	2.474126	0.9058872	299.85043	300
8	1	2.595099	0.9536246	300.17008	300
9	2	7.650895	2.0348226	29.85990	30
10	2	6.985697	1.9438648	29.95155	30
11	2	5.209691	1.6505205	49.87838	50

5: Additional: If we can measure the NO parameters directly as the NO parameters obtained in data simulation process.

We can get the the population mean and coefficients as follows. These results can be used for a more "concise" comparison with the estimation.

	Ca	logCaw	logDaw
(Intercept)	1.4901651	3.5249929	2.5815233
out$datX[, 2]	0.0807897	0.1478774	0.1144271

2. Estimate NO parameter and their associations with X

We are interested in discovering the impact of environmental factor(s) to the NO parameters. Those values were usually estimated in the so called "two-stage" approaches, which use the estimated NO parameters as the dependent variables. The corresponding methods was "unified" approaches, which use the NO parameters as intermediate variables. Here, we presented 4 two-stage methods and 2 corresponding unified methods.

2.1 TS_NLS: Two-stage nonlinear least squares

R Function

TS_NLS_StageI: Estimated NO parameters ordered by id, NA would be reported if the regression can not converge (due to the small sample size for the participant's data).

Steps

1: Estimated NO parameters for each participants. (NLS_StageI)

2: Combine NO parameters and covariate X by id. (NLS_StageIX)

3: Fit individual grouped linear regression models for each NO parameters

```
source("TS_NLS.R")

# 1. Stage I
NLS_StageI <- TS_NLS_StageI(dat)

# 2. Combine NO parameters and covariate X by id
NLS_StageIX <- merge(NLS_StageI, datX, by = "id")

# 3. Stage II - edit to include any additional Stage II
# covariates (e.g., confounder adjustments)
TS_NLS_Ca <- lme(Ca ~ X, random = ~1 | id, data = NLS_StageIX,
    na.action = na.omit)
TS_NLS_logCaw <- lme(logCaw ~ X, random = ~1 | id, data = NLS_StageIX,
    na.action = na.omit)
TS_NLS_logDaw <- lme(logDaw ~ X, random = ~1 | id, data = NLS_StageIX,
    na.action = na.omit)
```

Interpretation Example:

1: CANO

The population mean CANO is 1.24 (95% CI: 1.07,1).

For 1 unit increase in covariate X, CANO increases 0.04, (95% CI: -0.12, 0.19).

2. CawNO and DawNO from their log version:

The population mean CawNO is 3.41 (95% CI: 3.23,4).

For 1 unit increase in covariate X, CawNO increases 21.14%, (95% CI: 3.12%, 42.32%).

The population mean DawNO is 2.85 (95% CI: 2.66,3).

For 1 unit increase in covariate X, DawNO increases 7.03%, (95% CI: -9.15%, 26.09%).

2.2 TS_HMA: Two-stage Högman & Merilänen Algorithm

HMA algorithm only uses flow rates in three categories. In our example, we labeled low, median and high for target flow rates 30,100 and 300 ml/s.

The interpretations of results were the same as those in TS_NLS.

R Function

TS_HMA_StageI: Estimated NO parameters ordered by id, NA would be reported if the regression can not converge

```
source("TS_HMA.R")

# 1. Stage I, specify target flow rates for HMA (low,
# medium, high)
HMA_StageI <- TS_HMA_StageI(dat, flowLMH = c(30, 100, 300))

# 2. Combine NO parameters and covariate X by id
HMA_StageIX <- merge(HMA_StageI, datX, by = "id")

# 3. Stage II - edit to include any additional Stage II
# covariates (e.g., confounder adjustments)
TS_HMA_Ca <- lme(Ca ~ X, random = ~1 | id, data = HMA_StageIX,
    na.action = na.omit)
TS_HMA_logCaw <- lme(logCaw ~ X, random = ~1 | id, data = HMA_StageIX,
    na.action = na.omit)
TS_HMA_logDaw <- lme(logDaw ~ X, random = ~1 | id, data = HMA_StageIX,
    na.action = na.omit)
```

2.3 TS_NLME: Two-stage nonlinear mixed effects model

NLME methods may iterate for many rounds and take several minutes to converge. The interpretations of results were the same as those in TS_NLS.

R Functions

TS_NLME_StageI: Estimated NO parameters ordered by id, NA would be reported if the regression can not converge

```
if (!file.exists("TSNLME.Rdata")) {
    set.seed(2020)
    source("TS_NLME.R")
    # Stage I
    TSNLME_StageIout <- TS_NLME_StageI(dat, tol1 = 0.1, tol2 = 0.1,
        outputFit = TRUE)  # include X for later unified version
    save(TSNLME_StageIout, file = "TSNLME.Rdata")
} else {
```

```
    load("TSNLME.Rdata")
}
TSNLME_StageI <- TSNLME_StageIout$ests
TSNLME_StageIfit <- TSNLME_StageIout$fit  # save fit to speed up U_NLME

# create dataset including both Stage I estimates and X
TSNLME_StageIX <- merge(TSNLME_StageI, datX, by = "id")

# Stage II - edit to include any additional Stage II
# covariates (e.g., confounder adjustments)
TS_NLME_Ca <- lme(Ca ~ X, random = ~1 | id, data = TSNLME_StageIX,
    na.action = na.omit)
TS_NLME_logCaw <- lme(logCaw ~ X, random = ~1 | id, data = TSNLME_StageIX,
    na.action = na.omit)
TS_NLME_logDaw <- lme(logDaw ~ X, random = ~1 | id, data = TSNLME_StageIX,
    na.action = na.omit)
```

2.4 UNLME: Unified nonlinear mixed effects model

Notation

There are two different approaches to fit the unified NLME model. The "Direct" approach fit the model with initial values via approximation; the "Update" approach utilize the fitted two-stage version and update it with added regressions between NO parameters and X.

```
if (!file.exists("UNLME.Rdata")) {
    set.seed(2020)
    # The function is for single X. If you want to fit with
    # multiple X, just modify the 'fixed' and start
    # statement of the function
    source("U_NLME.R")
    # direct approach
    U_NLMEout <- U_NLME_direct(dat, datX, tol = 0.1)
    # update approach
    # U_NLMEout_u<-U_NLME_update(TSNLME_StageIout,dat,datX,tol=0.1)
    # anova(U_NLMEout,U_NLMEout_u) # compare two approaches
    save(U_NLMEout, file = "UNLME.Rdata")
} else {
    load("UNLME.Rdata")
}
```

2.5 TS_HB: Two-stage Hierarchical Bayesian method

The HB methods usually take hours to finish, so here we loaded the already fitted results. TS_HB methods was the alternative version for the U_HB to have a quicker pilot results, especially when the regression models between NO parameters and X become complex. It is also a "fair" comparison with other two stage methods.

The MCMC process in HB methods has three phase. The first phase runs a main jags function with one adaptation phase and update phase. Rhats were checked for parameters specified in the tracing vector, if they are smaller than the criteria, a final (third) phase runs to get the posterior distribution of parameters. If Rhats were larger than the criteria, a second phase runs with rounds and rounds of update function untill Rhats met the criteria.

Important parameters

N.iterT: The total number of iteration in the first phase. N.terT-N.burnin is the iteration number of update phase in the first phase.

N.burnin: The iteration number for adaptation phase in the first phase.

addon.iter: Iteration number for updating rounds

Max_update: Maximum rounds allowed in the second phase.

n.final: Iteration number in the third phase.

rhat: Rhat criteria

tracing: Parameters to be traced, recorded and checked for Rhat.

N.thinM: Thining rate, n.final/N.thinM >= 2000 usually

beta0_prior: the means of prior distribution for population level NO parameters

```
# Stage I
if (!file.exists("TSHB_cc.Rdata")) {
    set.seed(2020)
    source("TS_HB.R")

    TSHB_S1 <- TSHB_iter_UB(beta0_prior = c(2, 4, 3), rhat = 1.1,
        addon.iter = 6000, Max_update = 10, n.final = 6000, N.iterT = 45000,
        N.burnin = 40000, N.thinM = 20, N.chain = 3, flow = flow,
        dat = dat, tracing = c("beta0_Ca", "beta0_logCaw", "beta0_logDaw",
            "sdCa", "sdlogCaw", "sdlogDaw", "corlogCawCa", "corlogCawlogDaw",
            "corlogDawCa", "sigma_c"))
    save(TSHB_S1, file = "TSHB_cc_ub.Rdata")
} else {
    load("TSHB_cc.Rdata")
}

TSHB_S1_dat <- data.frame(Ca = TSHB_S1$summary[grepl("^Ca", rownames(TSHB_S1$summary)),
    1], logCaw = TSHB_S1$summary[grepl("^logCaw", rownames(TSHB_S1$summary)),
    1], logDaw = TSHB_S1$summary[grepl("^logDaw", rownames(TSHB_S1$summary)),
    1])
TSHB_S1_dat$id <- as.numeric(unlist(lapply(rownames(TSHB_S1_dat),
    function(x) strsplit(strsplit(x, "\\[")[[1]][2], "\\]")[[1]])))
TSHB_S1_dat <- TSHB_S1_dat[order(TSHB_S1_dat$id), ]
TSHB_StageIX <- cbind(TSHB_S1_dat, datX)

# Stage II - edit to include any additional Stage II
# covariates (e.g., confounder adjustments)
TS_HB_Ca <- lme(Ca ~ X, random = ~1 | id, data = TSHB_StageIX,
    na.action = na.omit)
TS_HB_logCaw <- lme(logCaw ~ X, random = ~1 | id, data = TSHB_StageIX,
    na.action = na.omit)
TS_HB_logDaw <- lme(logDaw ~ X, random = ~1 | id, data = TSHB_StageIX,
    na.action = na.omit)
```

Convergence diagnostic

We used Rhat (See Gelman and Rubin (1992), Brooks and Gelman (1998)]) to assess the convergence of the MCMC simulation mdoel.

Here we printed out the estimations (95% CL) and their Rhats (converge if <1.1)

	2.5%	mean	97.5%	Rhat
beta0_Ca	1.5080079	1.5481513	1.5885954	1.015909
beta0_logCaw	3.4062866	3.4864066	3.5702279	1.053775
beta0_logDaw	2.4312928	2.5246991	2.6151845	1.058032
sdCa	0.4215206	0.4546784	0.4895051	1.022329
sdlogCaw	0.5771957	0.6380713	0.7042129	1.012016
sdlogDaw	0.4055969	0.4860926	0.5773137	1.034582
corlogCawCa	0.5483103	0.6367438	0.7206984	1.026245
corlogCawlogDaw	-0.4052003	-0.2362898	-0.0557253	1.024703
corlogDawCa	-0.4233883	-0.2356216	-0.0349502	1.026120
sigma_c	0.0989450	0.1007883	0.1026800	1.002020

2.6 U_HB: Unified Hierarchical Bayesian method Here we also loaded the already fitted U_HB results due to the large amount of time for simulation.

Importatn parameters

Other than the TS_HB:

betaC_prior: the prior means for beta coefficients

```
source("U_HB.R")

if (!file.exists("UHB_cc.Rdata")) {
    set.seed(2020)
    UHB_sim <- UHB_iter_UB(beta0_prior = c(2, 4, 3), betaC_prior = c(0.1,
        0.1, 0.1), rhat = 1.1, addon.iter = 12000, Max_update = 10,
        n.final = 6000, N.iterT = 90000, N.burnin = 80000, N.thinM = 20,
        N.chain = 3, flow = flow, dat = dat, X = datX[, 2], tracing = c("beta0_Ca",
            "beta0_logCaw", "beta0_logDaw", "beta1_Ca", "beta1_logCaw",
            "beta1_logDaw", "sdCa", "sdlogCaw", "sdlogDaw", "corlogCawCa",
            "corlogCawlogDaw", "corlogDawCa", "sigma_c"))
    save(UHB_sim, file = "UHB_cc_ub.Rdata")
} else {
    load("UHB_cc.Rdata")
}
```

Convergence diagnostic

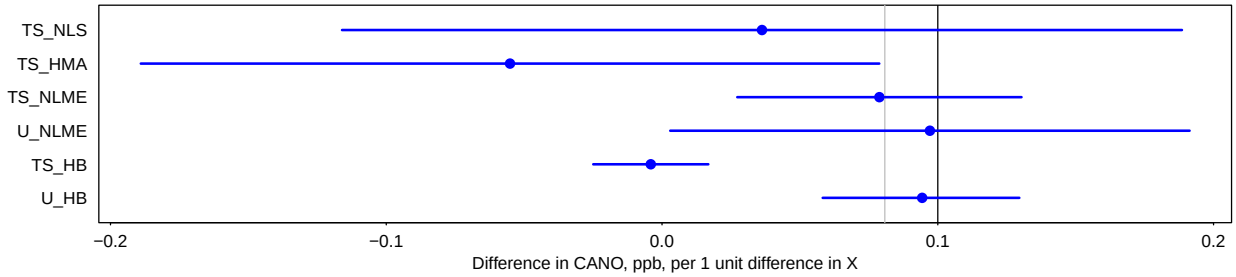
We print out the estimation (95% CL) and Rhat (converge if <1.1)

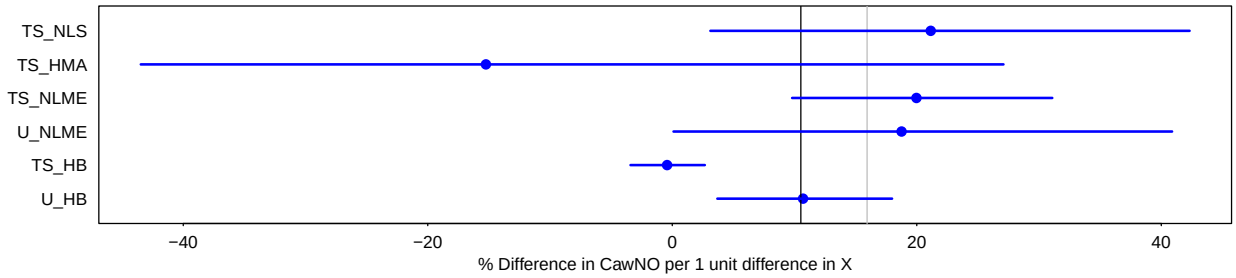
	2.5%	mean	97.5%	Rhat
beta0_Ca	1.5111985	1.5509132	1.5915931	1.005527
beta0_logCaw	3.4085700	3.4942062	3.5963433	1.033232
beta0_logDaw	2.4031259	2.5224750	2.6206695	1.032590
beta1_Ca	0.0582630	0.0943350	0.1295614	1.001966
beta1_logCaw	0.0362116	0.1016380	0.1653405	1.002494
beta1_logDaw	0.0150388	0.0835517	0.1558483	1.004417
sdCa	0.4154042	0.4468857	0.4805696	1.005201
sdlogCaw	0.5790614	0.6414866	0.7056561	1.023145
sdlogDaw	0.3833122	0.4899050	0.6050677	1.065012

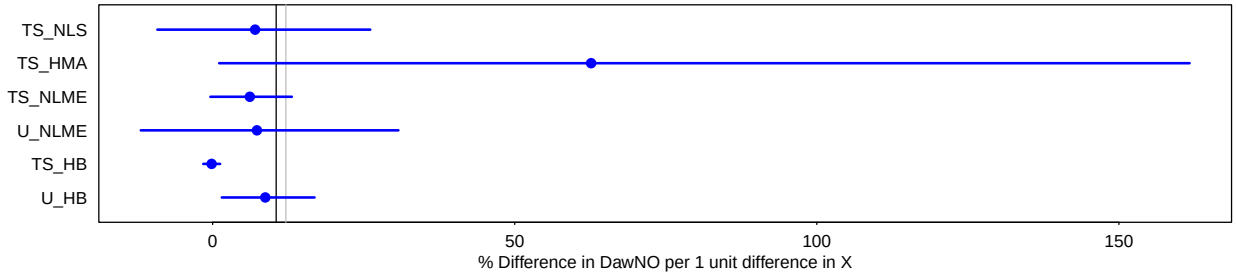
	2.5%	mean	97.5%	Rhat
corlogCawCa	0.5382412	0.6325284	0.7250663	1.010522
corlogCawlogDaw	-0.4646919	-0.3068066	-0.1044257	1.049161
corlogDawCa	-0.4791551	-0.3068774	-0.1284974	1.014665
sigma_c	0.0988964	0.1007370	0.1026550	1.007205

3. Create plot comparing estimated NO parameter associations across 6 methods

- Y axis: 6 methods
- X axis: coefficient effect size: The values used in simulation were all equaled to 0.1 for CANO, logCawNO and logDawNO. Which means for 1 unit increase in the covariate X, the corresponding NO paramters CANO, logCawNO and logDawNO increase 0.1 unit. The geometric interpretation for CawNO was that it was (exp(0.1)-1) times higher for 1 unit increase in the covariate, so was for DawNO
- Refernce lines: Black lines indicate the values used in data simulation. Gray lines indicate the values obtained by regression on the simulated NO parameters and Xs.







4. Problems may happen:

Due to the feature of some approaches, you may get error message or sometimes difficult to get the results. For HMA method, if participants did not take maneuver in all three target flows, an error message may

pop out as “replacement has length zero”. And if the sample size was too large or for data set generated with different random seed, U_NLME method may not converge in limited iterations. You may want to change the maximum iteration number in the script: UNLME.R. Usually for simulations, you can change the random seed or use a different sample size. TS_HB and U_HB methods need hours to converge. Usually it is more like to converge with longer adaptation phase.