A Test for Measurement Error Estimators

Venelin Mitov

8/22/2019

Set seed

```
set.seed(1, kind = "Mersenne-Twister", normal.kind = "Inversion")
```

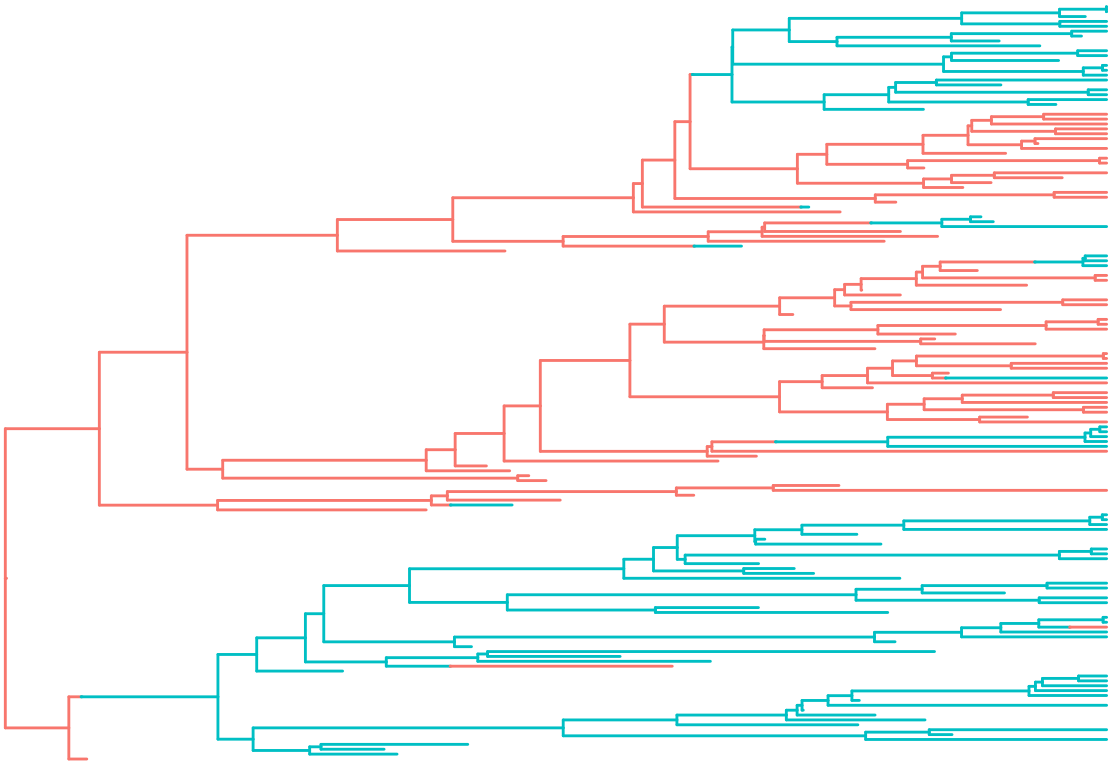
Tree

We use the tree from Fig 3 in the manuscript:

```
PCMTreePlot(dataFig3$tree)
```

```
## Loading required namespace: ggtree
```

```
## Registered S3 method overwritten by 'treeio':
##   method     from
##   root.phylo ape
```



True model

We use the parameters for the secon trait from the OU model in Fig. 3 :

regime	$X0$	H	Θ	Σ
:global:	$[5.00]$			
a		$[0.16]$	$[0.00]$	$[0.16]$
b		$[0.09]$	$[10.00]$	$[0.16]$

Simulation

We specify the following parameters for the simulation:

```
# Sample size
nSamp <- 20
#  Within species variance
VWithinSpecies <- 0.4
# number of simulations
nDatasets <- 50
```

We simulate 50 datesets. First we use PCMSim to generate the true trait values for each species; Next, for each species we generate a random sample of size 20, with mean equal to the simulated (true) trait value for the species and variance equal to 0.4:

```
# A cube, each slice corresponds to a dataset, each column corresponds to a sample
# for a species
data <- array(NA_real_, dim = c(nSamp, PCMTreeNumTips(tree), nDatasets))

for(iDataset in seq_len(nDatasets)) {
  X <- PCMSim(tree, modelOU, X0 = modelOU$X0)[, seq_len(PCMTreeNumTips(tree)), drop = FALSE]
  data[,, iDataset] <- sapply(X[1,], function(x) {

    rnorm(nSamp, x, sqrt(VWithinSpecies))
  })
}
```

Model inference

For each dataset, we fit the model using three different settings for the measurement error:

- fitSE0: SE = 0, for each species;
- fitSEDiv: SE = sqrt(sample_variance / n_i) for each species;
- fitSENoDiv: SE = sqrt(sample_variance) for each species;

```
testMEEstimators <- list()
testMEEstimators$VWithinSpecies <- VWithinSpecies
testMEEstimators$tree <- tree
testMEEstimators$model <- modelOU
testMEEstimators$data <- data

testMEEstimators$inferredModels <- list()
for(iDataset in seq_len(nDatasets)) {
  testMEEstimators$inferredModels[[iDataset]] <- list()

  cat("Fitting dataset", iDataset, "\n")

  modelForFit <- modelOU
```

```
  # Set some random initial values for the parameters
  modelForFit$H[] <- 0.0002134
  modelForFit$Theta[] <- 5.234
  modelForFit$Sigma_x[] <- 0.2342
  modelForFit$X0[] <- 1.2


  X <- matrix(colMeans(testMEEstimators$data[seq_len(nSamp),, iDataset]), nrow = 1L)
  Vemp <- matrix(apply(testMEEstimators$data[seq_len(nSamp),, iDataset], 2, var), nrow = 1L)

  fitSE0 <- PCMFit(
    X, testMEEstimators$tree, modelForFit,
    metaI = PCMInfoCpp,
    numCallsOptim = 20,
    numRunifInitVecParams = 10000, numGuessInitVecParams = 10000,
    doParallel = TRUE)

  fitSEDiv <- PCMFit(
    X, testMEEstimators$tree, modelForFit,
    SE = matrix(sqrt(Vemp/nSamp), nrow = 1L),
    metaI = PCMInfoCpp,
    numCallsOptim = 20,
    numRunifInitVecParams = 10000, numGuessInitVecParams = 10000,
    doParallel = TRUE)

  fitSENoDiv <- PCMFit(
    X, testMEEstimators$tree, modelForFit,
    SE = matrix(sqrt(Vemp), nrow = 1L),
    metaI = PCMInfoCpp,
    numCallsOptim = 20,
    numRunifInitVecParams = 10000, numGuessInitVecParams = 10000,
    doParallel = TRUE)

  testMEEstimators$inferredModels[[iDataset]] <- list(
    llTrueModelSE0 = PCMLik(X, tree, modelOU, metaI = PCMInfoCpp),
    llTrueModelSEDiv = PCMLik(X, tree, modelOU, SE = matrix(sqrt(Vemp/nSamp), nrow = 1L), metaI = PCMIn
    llTrueModelSENoDiv = PCMLik(X, tree, modelOU, SE = matrix(sqrt(Vemp), nrow = 1L), metaI = PCMInfoCp
    fitSE0 = fitSE0$modelOptim,
    llFitSE0 = fitSE0$logLikOptim,
    fitSEDiv = fitSEDiv$modelOptim,
    llFitSEDiv = fitSEDiv$logLikOptim,
    fitSENoDiv = fitSENoDiv$modelOptim,
    llFitSENoDiv = fitSENoDiv$logLikOptim
  )

usethis::use_data(testMEEstimators, overwrite = TRUE)
}
```

Infer a model with SE set to 0 but non-zero parameter Sigmae:

```
# Generate a parametrization where Sigmae
# 1. Filter the list of parametrizations to avoid generating too many S3 methods.
# (note that we could do the same type of filtering for the other parameters).
listParameterizationsOU <- PCMListParameterizations(structure(0.0, class="OU"))
```

```
listParameterizationsOU$H <- listParameterizationsOU$H[10]
listParameterizationsOU$Theta <- listParameterizationsOU$Theta[1]
listParameterizationsOU$Sigma_x <- listParameterizationsOU$Sigma_x[2]
listParameterizationsOU$Sigmae_x <- listParameterizationsOU$Sigmae_x[2]

# 2. Generate a table of parametrizations for this list:
dtParameterizations <- PCMTableParameterizations(
  structure(0.0, class="OU"), listParameterizations = listParameterizationsOU)

print(dtParameterizations)

# 3. Generate the parametrizations (optionally, we could select a subset of the
# rows in the data.table)
PCMGenerateParameterizations(structure(0.0, class="OU"),
                             tableParameterizations = dtParameterizations[])


modelOUWithSigmae <- PCM(
  paste0(
    "OU",
    "__Global_X0",
    "__Schur_Diagonal_WithNonNegativeDiagonal_Transformable_H",
    "__Theta",
    "__Diagonal_WithNonNegativeDiagonal_Sigma_x",
    "__Diagonal_WithNonNegativeDiagonal_Sigmae_x"),
  k = 1, regimes = c("a", "b"))

for(iDataset in seq_len(nDatasets)) {
  X <- matrix(colMeans(testMEEstimators$data[seq_len(nSamp),, iDataset]), nrow = 1L)

  fitSE0WithSigmae <- PCMFit(
    X, testMEEstimators$tree, modelOUWithSigmae,
    metaI = PCMInfoCpp, numCallsOptim = 20,
    numRunifInitVecParams = 10000, numGuessInitVecParams = 10000,
    doParallel = TRUE)
  testMEEstimators$inferredModels[[iDataset]]$fitSE0WithSigmae <-
    fitSE0WithSigmae$modelOptim
  testMEEstimators$inferredModels[[iDataset]]$llFitSE0WithSigmae <-
    fitSE0WithSigmae$logLikOptim
}

usethis::use_data(testMEEstimators, overwrite = TRUE)
```

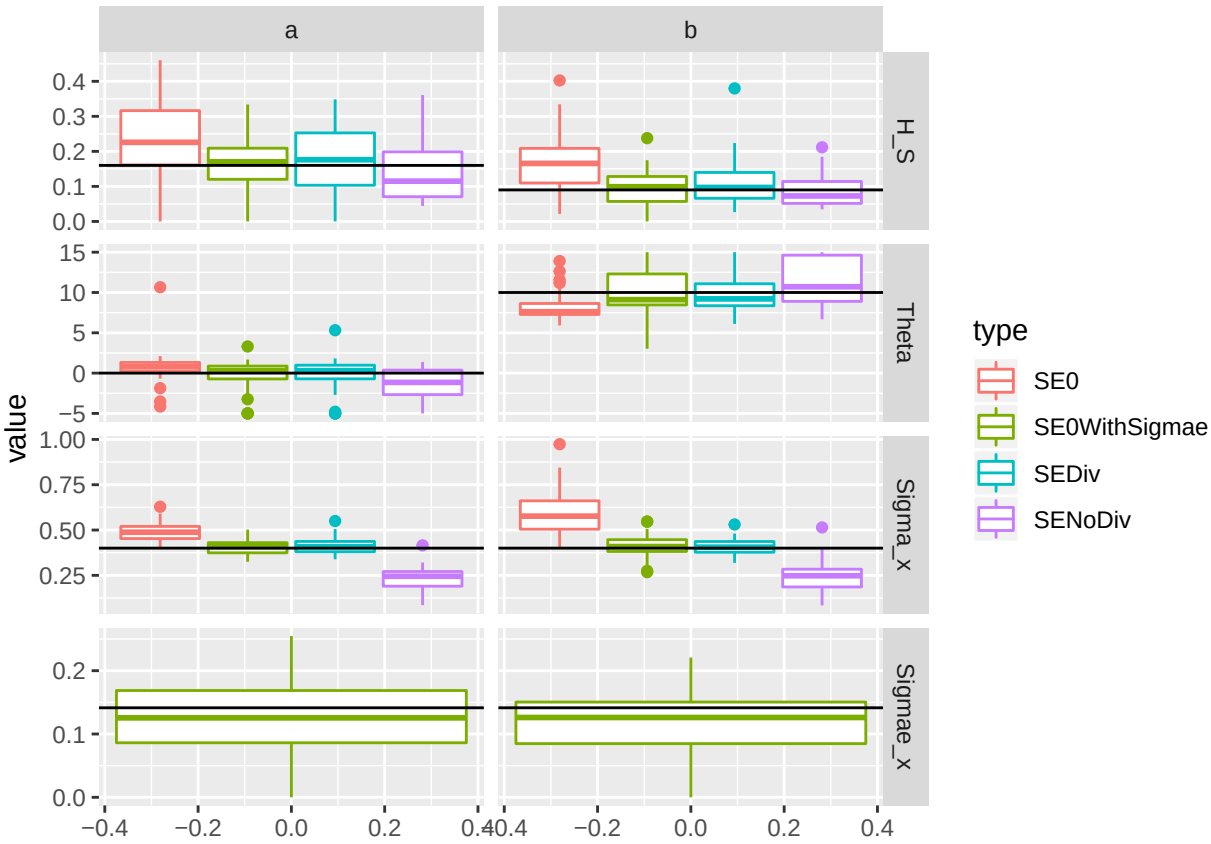
```
dt <- rbindlist(lapply(seq_len(nDatasets), function(i) {

  fits <- testMEEstimators$inferredModels[[i]]
  dt <- rbindlist(list(
    PCMTable(fits$fitSE0, addTransformed = FALSE, removeUntransformed = FALSE)[
      , c("type", "llTrue", "llFit"):=list(
      "SE0", fits$llTrueModelSE0, fits$llFitSE0)],
    PCMTable(fits$fitSEDiv, addTransformed = FALSE, removeUntransformed = FALSE)[
      , c("type", "llTrue", "llFit"):=list(
      "SEDiv", fits$llTrueModelSEDiv, fits$llFitSEDiv)],
    PCMTable(fits$fitSENoDiv, addTransformed = FALSE, removeUntransformed = FALSE)[
```

```
      , c("type", "llTrue", "llFit"):=list(
      "SENoDiv", fits$llTrueModelSENoDiv, fits$llFitSENoDiv)],
    PCMTable(fits$fitSEOWithSigmae, addTransformed = FALSE, removeUntransformed = FALSE)[
      , c("type", "llTrue", "llFit"):=list(
      "SEOWithSigmae", fits$llTrueModelSEDiv, fits$llFitSEOWithSigmae)]
  ), use.names = TRUE, fill = TRUE)

  dt[, i:=i]
}))

dt[, X0:=sapply(X0, function(x) if(is.null(x)) NA_real_ else x)]
dt[, H_S:=sapply(H_S, function(x) if(is.null(x)) NA_real_ else x)]
dt[, Theta:=sapply(Theta, function(x) if(is.null(x)) NA_real_ else x)]
dt[, Sigma_x:=sapply(Sigma_x, function(x) if(is.null(x)) NA_real_ else x)]
dt[, Sigmae_x:=sapply(Sigmae_x, function(x) if(is.null(x)) NA_real_ else x)]
```

Compare the inferred parameters against the true parameter values

```
GetParameterValueFromTrueModel <- function(variable, regime) {
  sapply(seq_len(length(variable)), function(i) {
    var <- if(variable[[i]] == "H_S") {
      "H"
    } else {
      variable[[i]]
    }
    value <- if(var %in% c("H", "Sigma_x")) {
      modelOU[[var]][,,regime[[i]]]
    } else if(var == "Sigmae_x") {
      sqrt(VWithinSpecies/nSamp)
    } else if(var %in% c("Theta")) {
      modelOU[[var]][,regime[[i]]]
    }
  })
}

pl <- ggplot(melt(dt, id = c("regime", "type", "i"))[
  regime != ":global:" & variable %in% c("H_S", "Sigma_x", "Theta", "Sigmae_x")]) +
  geom_boxplot(aes(y = value, colour = type)) +
  geom_hline(aes(yintercept = GetParameterValueFromTrueModel(variable, regime))) +
  facet_grid(variable~regime, scales = "free_y")
pl
```

```
## Warning: Removed 300 rows containing non-finite values (stat_boxplot).
```



The black lines are the true parameter values used to simulate the data. Notice the positive bias in fitSE0 and the huge negative bias for estimator SENoDiv for both regimes for paramter Sigma_x. There's also a (smaller) bias for the other parameters.