# A Practical Guide to Estimating the Heritability of Pathogen Traits - Toy model simulations

*Venelin Mitov, Tanja Stadler*

In this document, we provide details and the R scripts for setting up and running the toy model.

## Technical description of the toy model

We introduced the toy model in the main text. This section contains more technical details.

At a time $t$, the value $z_i(t)$ for an infected host $i$ is defined as a function of its type, $\mathbf{y}_i \in \{1, 2\}$, the currently carried strain $\mathbf{x}_i(t) \in \{1, ..., 6\}$ and the host's specific effect for this strain $e_i[\mathbf{x}_i(t)] \sim \mathcal{N}(0,\, 0.36)$ drawn at random for each strain in each host. We call a (type **y-x**) **general effect** the expected trait value of type-**y** hosts carrying strain **x** in an infected population: $GE[\mathbf{y}, \mathbf{x}] = \mathrm{E}[z|\mathbf{y}, \mathbf{x}]$. For a set of fixed general effects, $z_i(t)$ is constructed according to the equation:

$$z_i(t) = GE[\mathbf{y}_i, \mathbf{x}_i(t)] + e_i[\mathbf{x}_i(t)] \tag{1}$$

We use a fixed set of general effects drawn from the uniform distribution $\mathcal{U}(2, 4)$ for the twelve **y-x** combinations (fig. 2A).

At the between-host level, the phenomena of birth, contact, transmission, recovery and death define the dynamics between the compartments of susceptibles, infected and recovered individuals - $X$, $Y$ and $Z$. The natural birth rate, $\lambda$, and the natural per capita death rate, $\mu$, are defined as constants satisfying $\lambda = \mu N_0$, so that the average lifespan of an uninfected individual equals $1/\mu = 850$ (arbitrary) time units and in a disease-free population the total number of alive individuals equilibrates at $N_0 = 10^5$. An epidemic starts with the migration of an individual with random immune system type carrying pathogen strain 1:11 to a fully susceptible population of $N_0$ individuals. Each individual has contacts with other individuals occurring randomly at a constant rate, $\kappa$. A transmission can occur upon a contact involving an infected and a susceptible individual, here, called a "risky" contact. It is assumed that the probability of transmission per risky contact, $\gamma$, is either a constant (black on fig. 2B) or a function of the value $z$ (magenta on Fig. 2B) of the infected host and does not depend on the uninfected individual. Once infected, a host starts transmitting its currently dominant pathogen strain at a rate defined as the product of $\gamma$, $\kappa$, and the current proportion of susceptible individuals in the population, $S = X/N$. Thus, for fixed $\kappa$, the transmission rate of an infected host is a function of the global variable $S$ and the constant or variable $\gamma$. This transmission process continues until recovery or death of the host. Recovery has the meaning of a medical check occurring at a constant per capita rate, $\rho$, followed by immediate therapy and immunity. Due to the virulence of the pathogen, an infected host has an increased (per capita) death rate, $\delta$, which is defined either as a constant or as a function of $z$. Based on their scope of action, we call "between-host" the parameters $\lambda$, $\mu$, $\kappa$, $\gamma$, $\rho$and $\delta$.

Within a host, mutants of the dominant strain can appear at any time as a result of random single-locus mutations, which occur at a constant or $z$-dependent rate, $\nu$. It is important to make a distinction between a mutation and a substitution of a mutant strain for a dominant strain within a host, because a mutation doesn't necessarily lead to a substitution. For example, when $z$ is (or correlates with) the within-host reproductive fitness of the pathogen, substitutions would result only from mutations causing an increase in $z$. The rate of substitution of a mutant strain $\mathbf{x}_j$ for a dominant strain $\mathbf{x}_i$, differing by a single nucleotide at a locus $l$, is denoted $\xi_{l,i\leftarrow j}$ and defined as a function of $\nu$, the number of alleles at the locus, $M_l$, and the presence or absence of within-host selection with respect to $z$. No substitution can occur between strains differing at more than one locus, although, the same effect can result from two or more consecutive substitutions. Based on their scope of action, we call "within-host" the parameters $\nu$ and $\xi$.

Each simulation was run for $\min(4t_{10k}, 2400)$ time-units, where $t_{10k}$ denotes the time for the simulation until reaching 10,000 diagnosed individuals. The data generated after reaching 10,000 diagnoses has not been used in this study but it is intended for future analysis of post-outbreak dynamics, i.e. epidemic waves occurring after exhaustion of the susceptible pool.

The code chunks below represent the R-script used to execute the toy-model simulations. It is not supposed to be executed at once, but, rather by running the chunks manually one after another, and checking the intermediate results.
All toy-model simulations were executed on parallel cluster, thus the need to generate scripts that launch parallel jobs using the accompanying package benchtable.

## Setting-up the toy-model simulations

```
source('ToyModelSetup.R')
```

## Simulation on a parallel cluster

```
# this is done for all four scenarios (here, shown only for select/select - simulEpidemicSIRSS)
ids <- simulEpidemicSIRSS[, id]

# this generates a shell scripts that have to be copied and executed on the parallel cluster
genBenchJobs(simulateAndFit10k, params='doTreeAndFit=TRUE', script.file='simulateAndFitSIRSS',
             table.file='DATA/ToyModelSIR/SS.RData',
             ids=ids,
             perJob=1,
             type='bsub',
             requires=c('ape', 'patherit', 'Rmpfr', 'data.table'),
             sources='ToyModelSetup.R',
             bsub.W='12:00',
             bsub.mem=1000,
             sleep.every=420)

# copy scripts and data to the cluster
system('rsync -cv --progress ToyModelSetup.R vmitov@brutus:~/SPVL/SCRIPT/R/')
system('rsync -cv -a --progress DATA/ToyModelSIR/*.RData vmitov@brutus:~/SPVL/DATA/ToyModelSIR/')
system('rsync -cv --progress --remove-source-files j_* vmitov@brutus:~/SPVL/SCRIPT/R')

# once the execution is finished, copy the resulting job*.RData files from the cluster for analysis
system('rsync -cv --progress --remove-source-files vmitov@brutus:~/SPVL/SCRIPT/R/job*.RData DATA/ToyMod
```

## Collect results after the simulation and update the files NN.RData, etc.

```
i <- 0
for(i in 0:3) {
  ids <- i*60+(1:60)
  if(i==0) {
```

```r
  data <- simulEpidemicSIRNN
} else if(i==1) {
  data <- simulEpidemicSIRSN
} else if(i==2) {
  data <- simulEpidemicSIRNS
} else if(i==3) {
  data <- simulEpidemicSIRSS
}

data <- collectBenchRes('simulateAndFit10k', data,
                        dir.res='DATA/ToyModelSIR/20151212/',
                        ids=ids, res.ids.only=F, verbose=T)

# do some data extraction for later analysis
data[id%in%ids & sapply(simulateAndFit10k, is.list),
     epidemic:=lapply(simulateAndFit10k, function(.) .$epidemic)]

data[id%in%ids & sapply(simulateAndFit10k, is.list),
     treeAll:=lapply(simulateAndFit10k, function(.) .$treeAll)]
data[id%in%ids & sapply(simulateAndFit10k, is.list),
     vAll:=lapply(simulateAndFit10k, function(.) .$vAll)]
data[id%in%ids & sapply(simulateAndFit10k, is.list),
     tree10k:=lapply(simulateAndFit10k, function(.) .$tree10k)]
data[id%in%ids & sapply(simulateAndFit10k, is.list),
     v10k:=lapply(simulateAndFit10k, function(.) .$v10k)]

data[id%in%ids & sapply(simulateAndFit10k, is.list),
     nTips10k:=sapply(
       simulateAndFit10k,
       function(.) if(!is.null(.$nTips10k)) .$nTips10k else 0)]
data[id%in%ids & sapply(simulateAndFit10k, is.list),

     mlOUTree10k:=lapply(
       simulateAndFit10k,
       function(.) .$mlOUTree10k)]

data[id%in%ids & sapply(simulateAndFit10k, is.list),
     mlBMTree10k:=lapply(
       simulateAndFit10k,
       function(.) .$mlBMTree10k)]

data[, simulateAndFit10k:=NULL]
data[, t10k:=s.(epidemic, max(which(.$counts[,'nTips']<=10000)))]

data[rateSample==1/48&nTips10k>1000,
     sampPP:=lapply(ll.(
       lapply(ll.(epidemic, tree10k, list(epidemic=.[[1]],
                                          tree10k=.[[2]])),

              function(ep) {
                if(is.list(ep$epidemic)) {
                  pop <- extractPop(epidemic=ep$epidemic,
                                    ids=ep$tree10k$tip.label)
```

```
                    setkey(pop, id)
                    decomposeTrait(pop, GEVs, copy=FALSE)
                  } else {
                    NULL
                  }
                }), tree10k, list(sp=.[[1]], tr=.[[2]])),
         function(sptr) {
           pp <- extractPP(sptr$tr)
           pp[, z:=sptr$sp[J(sptr$tr$tip.label[i]), z]]
         })]
  if(i==0) {
    simulEpidemicSIRNN <- data
    save(simulEpidemicSIRNN, file='DATA/ToyModelSIR/NN.RData')
  } else if(i==1) {
    simulEpidemicSIRSN <- data
    save(simulEpidemicSIRSN, file='DATA/ToyModelSIR/SN.RData')
  } else if(i==2) {
    simulEpidemicSIRNS <- data
    save(simulEpidemicSIRNS, file='DATA/ToyModelSIR/NS.RData')
  } else if(i==3) {
    simulEpidemicSIRSS <- data
    save(simulEpidemicSIRSS, file='DATA/ToyModelSIR/SS.RData')
  }
  gc()
}
```

## Loading the data

```
# set eval=TRUE if you wish to load the data from previous execution
# note that the data files are really big (about 2GB), so executing this snippet m
# may result in a memory issue on a computer with little memory (e.g. smaller
# than 16GB) or many open applications.
load('DATA/ToyModelSIR/NN.RData')
load('DATA/ToyModelSIR/NS.RData')
load('DATA/ToyModelSIR/SN.RData')
load('DATA/ToyModelSIR/SS.RData')

if(file.exists('DATA/ToyModelSIR/h2statsAll.RData')) {
  load('DATA/ToyModelSIR/h2statsAll.RData')
}
```

## Analysis: heritability estimation on the simulated epidemic

```
# h2stats from anH2 analysis with patherit v0.8.
extractStat <- function(object, meth, st) {
  if(!'H2Analysis' %in% class(object)) {
    NA
  } else {
    if(meth == "PP") {
      smm <- summary(object$fits$PP)
      val <- smm[tauQuantile == "D[0%,10%]", est]
    } else {
      smm <- summary(object$fits[[meth]])
      val <- smm[stat == st, MLE[1]]
    }
    if(length(val) == 1) {
      val
    } else {
      NA
    }
  }
}

# Somethign went wrong with the cache here, so we do it manually
h2statsAll3 <- NULL
for(dataName in c('simulEpidemicSIRNN', 'simulEpidemicSIRSN', 'simulEpidemicSIRNS', 'simulEpidemicSIRSS
  print(dataName)
  data <- get(dataName)
  h2statsAll3 <- rbindlist(list(h2statsAll3,
                              data[rateSample == 1/48 & nTips10k > 1000, {
    sampCouples <- lapply(ll.(epidemic, tree10k, list(epidemic=.[[1]], tree10k=.[[2]])),
                          function(ep) {
                            if(is.list(ep$epidemic)) {
                              data <- extractDRCouples(
                                epidemic=ep$epidemic, ids=ep$tree10k$tip.label)
                              b(data = data, GEValues = GEVs)
                              data
                              } else {
                                NULL
                                }
                            })
    sampCouples2 <- lapply(sampCouples, function(.) {
      if(!is.null(.)) {
        rbind(.[, list(id, gene=gR0, z = zR0)],
              .[, list(id, gene=gR0, z = zD0)])
      } else {
        NULL
      }
    })
    cat('.')

    sampPop <- lapply(ll.(epidemic, tree10k,
                          list(epidemic=.[[1]], tree10k=.[[2]])),
                      function(ep) {
                        if(is.list(ep$epidemic)) {
                          pop <- extractPop(epidemic=ep$epidemic,
                                            ids=ep$tree10k$tip.label)
                          setkey(pop, id)
                          decomposeTrait(pop, GEVs, copy=FALSE)
                          } else {
                            NULL
```

```
                             }
                         })

    list(id=id, process=process, rateContact=rateContact,
         cov.zDzR = s.(sampCouples, if(!is.null(.)) .[, cov(zR, zD)] else NA),
         var.zD = s.(sampCouples, if(!is.null(.)) .[, var(zD)] else NA),
         var.zR = s.(sampCouples, if(!is.null(.)) .[, var(zR)] else NA),
         var.GD = s.(sampCouples, if(!is.null(.)) {
             data <- .
             data[, GD:=mean(zD), by=gD]
             data[, var(GD)]
           } else NA),
         var.GR = s.(sampCouples, if(!is.null(.)) {
             data <- .
             data[, GR:=mean(zR), by=gR]
             data[, var(GR)]
           } else NA),
         nTips10k=nTips10k, rateSample=rateSample, t10k=t10k,
         meanTau10k=s.(sampCouples, .[, mean(taum)*timeStep]),
         meanTauR10k=s.(sampCouples, .[, mean(tauR)*timeStep]),
         meanTauD10k=s.(sampCouples, .[, mean(tauD)*timeStep]),
         R2adj=s.(sampPop, if(!is.null(.)) R2adj(data=.) else NA),
         var.G = s.(sampPop, if(!is.null(.)) .[, var(G)] else NA),
         R2adj.0=s.(sampCouples2, if(!is.null(.))
           R2adj(data = .) else NA),
         R2adj.R0=s.(sampCouples2, if(!is.null(.))
           R2adj(data = .[1:(.N/2)]) else NA),
         R2adj.D0=s.(sampCouples2, if(!is.null(.))
           R2adj(data = .[(.N/2+1):.N]) else NA),
         var.GR0 = s.(sampCouples2, if(!is.null(.)) {data2 <- .[1:(.N/2)]
         data2[, g:=mean(z), by=gene]
         data2[, var(g)]
         } else NA),
         var.zR0 = s.(sampCouples2, if(!is.null(.))
           .[1:(.N/2), var(z)] else NA),
         var.GD0 = s.(sampCouples2, if(!is.null(.)) {data2 <- .[(.N/2+1):.N]
         data2[, g:=mean(z), by=gene]
         data2[, var(g)]
         } else NA),
         var.zD0 = s.(sampCouples2, if(!is.null(.))
           .[(.N/2+1):.N, var(z)] else NA),
         var.z0 = s.(sampCouples2, if(!is.null(.)) .[, var(z)] else NA),
         var.G0 = s.(sampCouples2, if(!is.null(.)) {data2 <- .
         data2[, g:=mean(z), by=gene]
         data2[, var(g)]
         } else NA),
         cov.zD0zR0 = s.(sampCouples, if(!is.null(.))
           .[, cov(zR0, zD0)] else NA),
         rA.hat.0 = s.(anH2, if(!is.null(.))
           summary(.$fits$PP)[tauQuantileType=="D" & stat=="rA",
                              coef(lm(est~tauMean))[1]] else NA),
         rA.hat.exp.0 = s.(anH2, if(!is.null(.))
           summary(.$fits$PP)[tauQuantileType=="D" & stat=="rA",
                           exp(coef(lm(log(est)~tauMean))[1])] else NA),
rA.hat.exp.V.0 = s.(anH2, if(!is.null(.))
  summary(.$fits$PP)[tauQuantileType=="V" & stat=="rA",
                           exp(coef(lm(log(est)~tauMean))[1])] else NA)
rA.0 = s.(sampCouples2, if(!is.null(.))
  rA(data = ., by = "gene")),
rA.R0 = s.(sampCouples2, if(!is.null(.))
  rA(data = .[1:(.N/2)], by = "gene")),
rA.D0 = s.(sampCouples2, if(!is.null(.))
  rA(data = .[(.N/2+1):.N], by = "gene")),
rA.id0 = s.(sampCouples2, if(!is.null(.)) rA(data = ., by = "id")),
rA.id=s.(sampPop, if(!is.null(.)) rA(data=.) else NA),
rA.PP=s.(sampPP, if(!is.null(.)) rA(data=., by='idPair') else NA),
rA.CPP.200=s.(sampPP, if(!is.null(.))
  rA(data=.[d<=sort(d)[min(length(d),200)]], by='idPair') else NA),
rA.CPP.400=s.(sampPP, if(!is.null(.))
  rA(data=.[d<=sort(d)[min(length(d),400)]], by='idPair') else NA),
rA.CPP.800=s.(sampPP, if(!is.null(.))
  rA(data=.[d<=sort(d)[min(length(d),800)]], by='idPair') else NA),
rA.CPP.1600=s.(sampPP, if(!is.null(.))
  rA(data=.[d<=sort(d)[min(length(d),1600)]], by='idPair') else NA),
rA.CPP.D1=s.(sampPP, if(!is.null(.))
  rA(data=.[d<=quantile(d,
                        probs=0.1)], by='idPair') else NA),
rA.CPP.V1=s.(sampPP, if(!is.null(.))
  rA(data=.[d<=quantile(d, probs=0.05)], by='idPair') else NA),
rSp.CPP.D1=s.(sampPP, if(!is.null(.))
  .[d<=quantile(d,probs=0.1), list(.N, z1=z[1], z2=z[2]),
    by=idPair][N==2, cor(z1,z2,method="spearman")])

a.PP=s.(sampPP, if(!is.null(.)) {
  pp <- .[, list(i, j, K=.N, tau=d, z), keyby=idPair][K==2]
  pp[, muz:=mean(z)]
  pp[, varz:=var(z)]
  pp.corr <- pp[, list(tau=unique(tau),
                       y=unique((z[1]-muz)*(z[2]-muz)/varz)), by=idPair]

  linmod <- pp.corr[, lm(y~tau)]

  coef(linmod)[1]
} else {NA}),
N.PP=s.(sampPP, if(!is.null(.)) nrow(.) else NA),
N.CPP.200=s.(sampPP, if(!is.null(.))
  nrow(.[d<=sort(d)[min(length(d),200)]]) else NA),
N.CPP.400=s.(sampPP, if(!is.null(.))
  nrow(.[d<=sort(d)[min(length(d),400)]]) else NA),
N.CPP.800=s.(sampPP, if(!is.null(.))
  nrow(.[d<=sort(d)[min(length(d),800)]]) else NA),
N.CPP.1600=s.(sampPP, if(!is.null(.))
  nrow(.[d<=sort(d)[min(length(d),1600)]]) else NA),
N.CPP.D1=s.(sampPP, if(!is.null(.))
  nrow(.[d<=quantile(d, probs=0.1)]) else NA),
N.CPP.V1=s.(sampPP, if(!is.null(.))
```

```r
      nrow(.[d<=quantile(d, probs=0.05)]) else NA),
    b=s.(sampCouples, b(data=., GEValues=GEVs, atInfection=FALSE)),
    b.D1=s.(sampCouples, b(data=.[taum<=quantile(taum, probs=0.1)],
                           GEValues=GEVs, atInfection=FALSE)),
    b.V1=s.(sampCouples, b(data=.[taum<=quantile(taum, probs=0.05)],
                           GEValues=GEVs, atInfection=FALSE)),
    N.b=s.(sampCouples, nrow(.)),
    N.b.D1=s.(sampCouples, nrow(.[taum<=quantile(taum, probs=0.1)])),
    N.b.V1=s.(sampCouples, nrow(.[taum<=quantile(taum, probs=0.05)])),
    b.0=s.(sampCouples, b(data=., GEValues=GEVs, atInfection=TRUE)),
    corr=s.(sampCouples, b(data=., GEValues=GEVs, atInfection=FALSE,
                           corr=TRUE)),
    corr.0=s.(sampCouples, b(data=., GEValues=GEVs, atInfection=TRUE,
                             corr=TRUE)),

    anH2.rAD1 = sapply(anH2, extractStat, "PP", "rA"),

    anH2.PMM.H2e = sapply(anH2, extractStat, "PMM", "H2e"),
    anH2.PMM.H2tMean = sapply(anH2, extractStat, "PMM", "H2tMean"),
    anH2.PMM.H2tInf = sapply(anH2, extractStat, "PMM", "H2tInf"),
    anH2.PMM.loglik = sapply(anH2, extractStat, "PMM", "loglik"),
    anH2.PMM.AIC = sapply(anH2, extractStat, "PMM", "AIC"),
    anH2.PMM.AICc = sapply(anH2, extractStat, "PMM", "AICc"),
    anH2.PMM.alpha = sapply(anH2, extractStat, "PMM", "alpha"),
    anH2.PMM.theta = sapply(anH2, extractStat, "PMM", "theta"),
    anH2.PMM.sigma = sapply(anH2, extractStat, "PMM", "sigma"),
    anH2.PMM.sigmae = sapply(anH2, extractStat, "PMM", "sigmae"),
    anH2.PMM.sigmaG2tMean = sapply(anH2, extractStat, "PMM",
                                   "sigmaG2tMean"),
    anH2.PMM.sigmae2 = sapply(anH2, extractStat, "PMM", "sigmae")^2,
    anH2.PMM.g0 = sapply(anH2, extractStat, "PMM", "g0"),
    corrProfile.PMM.varZ = sapply(corrProfile, function(.) {
      mean(sapply(.$simulatedData$PMM, var))
    }),

    anH2.POUMM.H2e = sapply(anH2, extractStat, "POUMM", "H2e"),
    anH2.POUMM.H2tMean = sapply(anH2, extractStat, "POUMM", "H2tMean"),
    anH2.POUMM.H2tInf = sapply(anH2, extractStat, "POUMM", "H2tInf"),
    anH2.POUMM.loglik = sapply(anH2, extractStat, "POUMM", "loglik"),
    anH2.POUMM.AIC = sapply(anH2, extractStat, "POUMM", "AIC"),
    anH2.POUMM.AICc = sapply(anH2, extractStat, "POUMM", "AICc"),
    anH2.POUMM.alpha = sapply(anH2, extractStat, "POUMM", "alpha"),
    anH2.POUMM.theta = sapply(anH2, extractStat, "POUMM", "theta"),
    anH2.POUMM.sigma = sapply(anH2, extractStat, "POUMM", "sigma"),
    anH2.POUMM.sigmae = sapply(anH2, extractStat, "POUMM", "sigmae"),
    anH2.POUMM.sigmaG2tMean = sapply(anH2, extractStat, "POUMM", "sigmaG2tMean"),
    anH2.POUMM.sigmae2 = sapply(anH2, extractStat, "POUMM", "sigmae")^2,
    anH2.POUMM.g0 = sapply(anH2, extractStat, "POUMM", "g0")
    meanZ = sapply(v10k, mean),
    varZ = sapply(v10k, var),
    corrProfile.POUMM.varZ = sapply(corrProfile, function(.) {
       mean(sapply(.$simulatedData$POUMM, var))
    })
    )
    }
    ]
    ))
  }

#h2statsAll <- merge(h2statsAll, h2statsAll3, by="id")
#save(h2statsAll, GEVs, file='DATA/ToyModelSIR/h2statsAll.RData')
```

## Analysing with patherit version 0.8 (combined ANOVA-CPP, POUMM and PMM)

```
simulEpidemicSIRNN[, anH2:=lapply(1:.N, function(i) {
  if(!is.null(tree10k[[i]])) {
    cat("Epidemic ", i, '\n')
    patherit::estimateH2(v10k[[i]], tree10k[[i]],
               methods = list(PP=TRUE,
                              POUMM=list(nSamplesMCMC = 0, verbose=TRUE),
                              PMM=list(nSamplesMCMC = 0, verbose=TRUE)),
               verbose = TRUE, usempfr=-2)
  } else {
    NULL
  }
})]

simulEpidemicSIRNS[, anH2:=lapply(1:.N, function(i) {
  if(!is.null(tree10k[[i]])) {
    cat("Epidemic ", i, '\n')
    patherit::estimateH2(v10k[[i]], tree10k[[i]],
               methods = list(PP=TRUE,
                              POUMM=list(nSamplesMCMC = 0, verbose=TRUE),
                              PMM=list(nSamplesMCMC = 0, verbose=TRUE)),
               verbose = TRUE, usempfr=-2)
  } else {
    NULL
  }
})]

simulEpidemicSIRSN[, anH2:=lapply(1:.N, function(i) {
  if(!is.null(tree10k[[i]])) {
    cat("Epidemic ", i, '\n')
    patherit::estimateH2(v10k[[i]], tree10k[[i]],
               methods = list(PP=TRUE,
                              POUMM=list(nSamplesMCMC = 0, verbose=TRUE),
                              PMM=list(nSamplesMCMC = 0, verbose=TRUE)),
               verbose = TRUE, usempfr=-2)
  } else {
    NULL
  }
})]
```

```r
# for id==114 there was a stucking in a local minima, so we rerun it with different bounds.
simulEpidemicSIRSN[id==114, anH2:=list(list(
  estimateH2(v10k[[1]], tree10k[[1]],
             methods = list(PP=TRUE,
                            POUMM=list(parUpper=c(alpha=5, theta=6, H2tMean=1, sigmae=2, g0=12), nSampl
                            PMM=list(nSamplesMCMC = 0, verbose=TRUE)),
             verbose = TRUE, usempfr=-2)))]


simulEpidemicSIRSS[, anH2:=lapply(1:.N, function(i) {
  if(!is.null(tree10k[[i]])) {
    cat("Epidemic ", i, '\n')
    patherit::estimateH2(v10k[[i]], tree10k[[i]],
               methods = list(PP=TRUE,
                              POUMM=list(nSamplesMCMC = 0, verbose=TRUE),
                              PMM=list(nSamplesMCMC = 0, verbose=TRUE)),
               verbose = TRUE, usempfr=-2)
  } else {
    NULL
  }
})]
```

```r
simulEpidemicSIRNN[, corrProfile:=lapply(1:.N, function(i) {
  if(!is.null(anH2[[i]])) {
    cat("Epidemic ", i, '\n')
    print(summary(anH2[[i]]))
    corrProfile(anH2[[i]], verbose = TRUE)
  } else {
    NULL
  }
})]

simulEpidemicSIRNS[, corrProfile:=lapply(1:.N, function(i) {
  if(!is.null(anH2[[i]])) {
    cat("Epidemic ", i, '\n')
    print(summary(anH2[[i]]))
    corrProfile(anH2[[i]], verbose = TRUE)
  } else {
    NULL
  }
})]

simulEpidemicSIRSN[, corrProfile:=lapply(1:.N, function(i) {
  if(!is.null(anH2[[i]])) {
    cat("Epidemic ", i, '\n')
    print(summary(anH2[[i]]))
    corrProfile(anH2[[i]], verbose = TRUE)
  } else {
    NULL
  }
})]

simulEpidemicSIRSS[, corrProfile:=lapply(1:.N, function(i) {
  if(!is.null(anH2[[i]])) {
    cat("Epidemic ", i, '\n')
    print(summary(anH2[[i]]))
    corrProfile(anH2[[i]], verbose = TRUE)
  } else {
    NULL
  }
})]
```