`branchCorr`: simulation and inference for partially observed stochastic compartmental models

Jason Xu

October 20, 2016

1 Simulation from branching process/stochastic compartmental model

The `R` package `branchCorr` provides functions to forward-simulate using the Gillespie algorithm from a general class of stochastic compartmental models, as well as capabilities to produce discretely observed datasets and partially observed datasets given a sampling distribution.

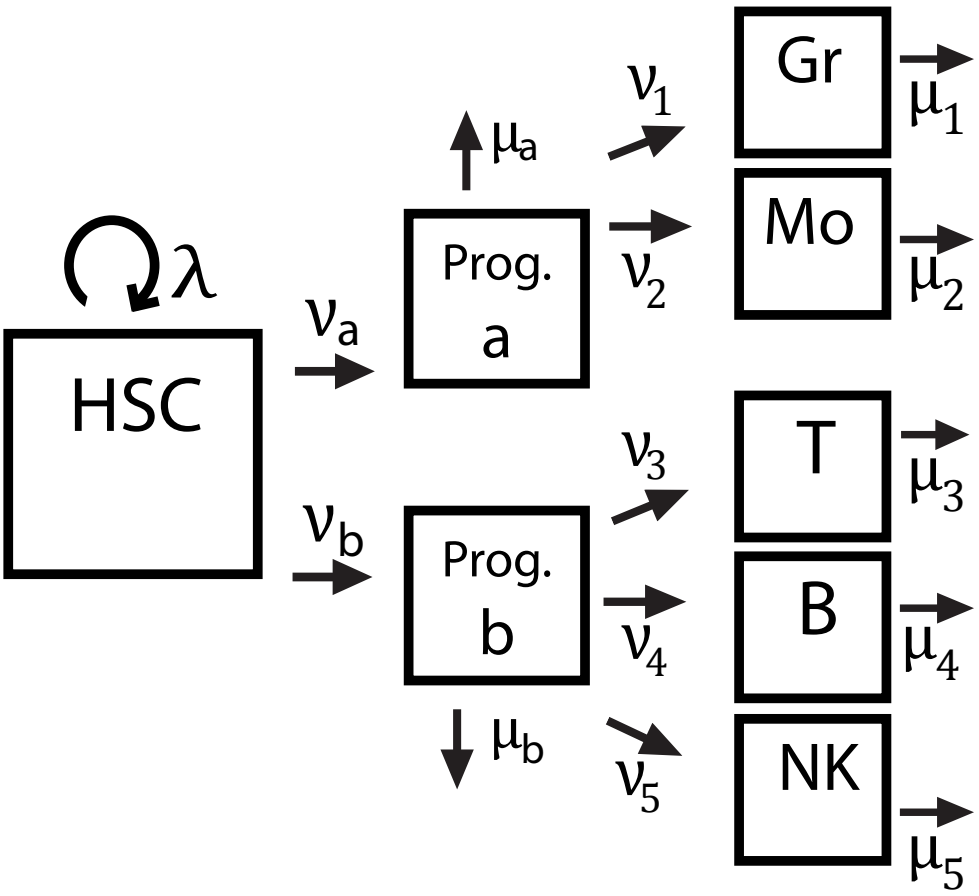
Here we step through how to simulate from a stochastic compartmental model. The following specifies a model with 5 mature types and 2 intermediate progenitor types, i.e. Figure 2(d) in the manuscript, included again below:

```
library(branchCorr)
# Specify a model with 5 mature types as in Figure 2(d)
progStructure <- c(1,1,1,2,2)
totalTypes <- 8
rates <- c(.0285,.014,.007,.005, .004,
        36, 18, 10, 20, 12, .26,.13,.11,.16,.09)
```

First, let's straightforwadly simulate from the model and check functions that compute the model-based moment expressions. We simulate 2000 independent processes, each beginning with one HSC cell, for time $t = 4$ (months for instance).

```
t <- 10
N <- 5000
initPopulation <- c(1,0,0,0,0,0,0,0)
set.seed(99999)
dat <- t( replicate(N, simCompartments(t, initPopulation, rates, progStructure )))
head(dat)

##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    1    0    0    0    0    0    0    0
```


λ
HSC
ν_a
ν_b
Prog. a
Prog. b
μ_a
μ_b
ν_1
ν_2
ν_3
ν_4
ν_5
Gr
Mo
T
B
NK
μ_1
μ_2
μ_3
μ_4
μ_5

```
## [2,]    2    0    0    0    0    0    0    0
## [3,]    1    0    0    0    0    0    0    0
## [4,]    1    0    0    0    0    0    0    0
## [5,]    1    1    0    9    5    3    0    0
## [6,]    1    0    0    0    0    0    0    0
```

Now let's check that observed standard deviations and correlations are close to the model-based expressions given the true rates used to generate the data.

```
numProgs <- length(unique(progStructure))
matureTypes <- length(progStructure)
# Check standard deviations:
for(i in 1:matureTypes){
  print( c(i, sd(dat[,i+numProgs+1]),
  sqrt(Var_xFrom1(t,rates,progStructure[i],
                  i,progStructure) ) ) )
}

## [1]  1.00000 33.65709 34.07248
## [1]  2.00000 23.52464 24.03103
## [1]  3.00000 14.09181 14.25257
## [1]  4.00000 17.64353 17.84673
## [1]  5.00000 13.20171 13.21819

# Check correlations:
combs <- combn(matureTypes,2)
for(ind in 1:choose(matureTypes,2)){
  i <- combs[1,ind]; j <- combs[2,ind]
  print(paste("Mature type pair: ", i,j))
  print(c( cor(dat[,i+1+numProgs], dat[,j+1+numProgs]),
    Cor_xyFrom1(t,rates,progStructure[i],progStructure[j],
                i,j, progStructure)) )
}

## [1] "Mature type pair:  1 2"
## [1] 0.9811784 0.9825601
## [1] "Mature type pair:  1 3"
## [1] 0.9744303 0.9758301
## [1] "Mature type pair:  1 4"
## [1] -0.07704518 -0.07944525
## [1] "Mature type pair:  1 5"
## [1] -0.07579880 -0.07808959
## [1] "Mature type pair:  2 3"
## [1] 0.9787504 0.9799365
## [1] "Mature type pair:  2 4"
```

```
## [1] -0.07607191 -0.07752933
## [1] "Mature type pair:  2 5"
## [1] -0.07457813 -0.07617646
## [1] "Mature type pair:  3 4"
## [1] -0.07578786 -0.07677907
## [1] "Mature type pair:  3 5"
## [1] -0.07430359 -0.07543402
## [1] "Mature type pair:  4 5"
## [1] 0.9811605 0.9821754
```

Note that one can repeat the above beginning with a progenitor type cell and find similar results. We will move forward allowing the initial population to be randomly sampled. Let's generate a discretely observed dataset, and also introduce an initial distribution so that each process is descended from an HSC or either of the two progenitors at random.

Now, let's generate a discretely observed dataset, and also introduce an initial distribution so that each process is descended from an HSC or either of the two progenitors at random. We must specify the initial distribution, observation times, and number of independent processes to be simulated: in the hematopoiesis example, N would correspond to the number of independent barcode lineages in the dataset. The function `randInit` used within the simulation call samples the initial population according to `initProbs`.

```
N <- 10000
obsTimes <- 6*c(2,3,4,6)
t <- obsTimes[length(obsTimes)] #simulation length
initProbs <- c(.1,.6,.3)
synthData <- t( replicate(N, sim.once( t, randInit(initProbs,
      totalTypes), rates, progStructure,obsTimes, vecOutput=T )))
```

Next, let's check that the model-based correlations are close to the empirical correlations in the simulated dataset.

```
numProgs <- length(unique(progStructure)) #number of unique progenitors
matureTypes <- length(progStructure)

synthCorr <- getObsCorr(synthData,obsTimes)

pairs <- combn(seq(1:matureTypes),2) #indices of final types
Corr <- synthCorr #matrix of same size for model-based corr

for( j in 1:dim(pairs)[2]){ #loop over number of pairwise combinations:
  Corr[j,] <- marginalizedCor_xy(obsTimes,
      rates, initProbs, pairs[1,j], pairs[2,j], progStructure)
}
```

The differences between model-based correlations evaluated at the true parameters and observed correlations is small, and one can check that it will decrease with the Monte Carlo error as we increase N: instead of printing out all pairs as before, let's directly examine the difference:

```
round(Corr-synthCorr,3)
```

```
##          [,1]   [,2]  [,3]  [,4]
##  [1,] -0.001  0.000 0.000 0.001
##  [2,]  0.000  0.000 0.000 0.000
##  [3,]  0.006  0.006 0.004 0.008
##  [4,]  0.006  0.006 0.004 0.006
##  [5,] -0.001 -0.001 0.000 0.000
##  [6,]  0.006  0.005 0.005 0.007
##  [7,]  0.006  0.005 0.005 0.006
##  [8,]  0.005  0.005 0.004 0.007
##  [9,]  0.005  0.005 0.004 0.006
## [10,] -0.001  0.000 0.000 0.000
```

Now, let's introduce noise via hypergeometric sampling from the discretely observed dataset we've created, forming a partially observed dataset. We first make sure we do not try to set the sample size greater than any type's total population:

```
sampSize <- min(colSums(synthData))-1000
sampledData <- hyperGeoSample(synthData, sampSize)
synthCorrSamp <- getObsCorr(sampledData,obsTimes)
sampCorr <- synthCorrSamp #to store model-based sampling corr

# parameters for sampling correlation computations
totalPopulations <- colSums(synthData)
nTot <- matrix(totalPopulations, ncol = length(obsTimes), byrow = T)
ll <- length(obsTimes)
# Calculate model-based correlations
for( j in 1:dim(pairs)[2]){ #loop over number of pairwise combinations:
  sampCorr[j,] <- samplingCor_xy(obsTimes,rates,initProbs, pairs[1,j],
                pairs[2,j], sampSize, sampSize, totalPopulations[1:ll],
                totalPopulations[(ll+1):(2*ll)], progStructure)
}
```

Check the difference between observed and model-based:

```
round(sampCorr-synthCorrSamp,3)
```

```
##          [,1]   [,2]   [,3]   [,4]
##  [1,] -0.002  0.000 -0.001  0.000
```

```
##  [2,] -0.015 -0.015 -0.015 -0.011
##  [3,]  0.012  0.012  0.011  0.014
##  [4,]  0.014  0.013  0.011  0.012
##  [5,] -0.022 -0.018 -0.015 -0.011
##  [6,]  0.016  0.013  0.012  0.013
##  [7,]  0.019  0.015  0.013  0.012
##  [8,]  0.030  0.025  0.023  0.022
##  [9,]  0.032  0.027  0.023  0.021
## [10,] -0.023 -0.020 -0.018 -0.016
```

Finally, we are ready to do inference on the dataset. The function `optimizeSamplingCorrNLMINB` makes this easy; we simply specify the number of restarts. Note that the best estimate for a particular synthetic dataset is not always close to the true rates, as can be seen from the plotted estimates in Figure 3 of the manuscript. Indeed, the loss function surface may attain its optimum at a point other than the true rates for a given finite dataset. However, if we repeat the entire procedure of simulating the sampled dataset and inferring the best fitting rates, the median of the best estimates shows empirical unbiasedness. Repeating this procedure outlined in the vignette over 400 simulated datasets reproduces the simulation studies and Figures 3 and 4.

```
par <- c(rates, log(6), log(3))
print( samplingCorrObjective(par, obsTimes,
      synthCorrSamp, rep(sampSize,5), nTot, progStructure))

## [1] 0.04974427

set.seed(1234)
numRestarts <- 500
best <- optimizeSamplingCorrNLMINB(sampledData, numRestarts, rates, obsTimes,
                rep(sampSize,5), nTot, c(.26,.13,.11,.16,.09), progStructure) #return 3
```

Now we compare the model-based correlations computed with the true rates and the best estimates; we see that visually the fit is close, as one would expect due to the small value of objective function at convergence. The following plot code relies on libraries `RColorBrewer` and `scales`. This dataset features less interesting correlation profiles than the simulation study in the paper, which simulates a dataset over a more realistic time scale. While runtime required for estimation does not change much, simulating such a dataset requires significant computing time.

```
library(RColorBrewer)
library(scales)
pal <- brewer.pal(10 ,"Set3")
pal <- sample(pal)

pairs <- combn(5,2)
```

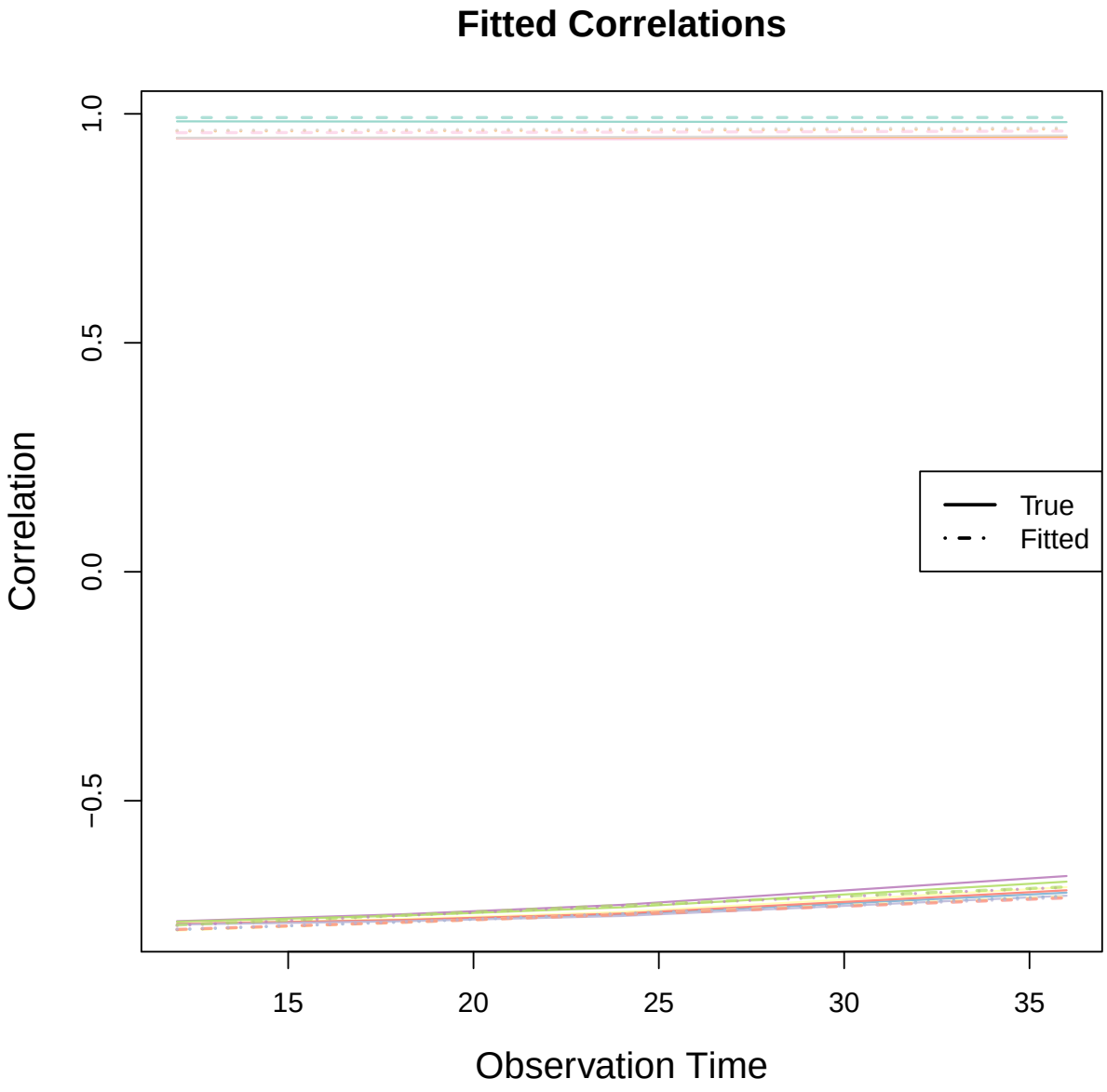
```r
corrs <- matrix(NA, 10,length(obsTimes)) #true correlations
for( j in 1:dim(pairs)[2]){ #loop over number of pairwise combinations:
  corrs[j,] <- samplingCor_xy(obsTimes, rates, initProbs, pairs[1,j], pairs[2,j],
        sampSize, sampSize, nTot[pairs[1,j],], nTot[pairs[2,j],], progStructure)
}

Corr <- matrix(NA, 10,length(obsTimes)) #will hold correlation from estimates
par <- best[-length(best)]
rate <- unlist(par[1:(length(par)-numProgs)] )
rate <- as.vector(rate)
initProbTemp <- par[(length(par)-numProgs+1):length(par)]
initProb <- unlist(c(1-sum(initProbTemp), initProbTemp))
print(initProb) #estimated initial distribution, original scale

## [1] 0.07287864 0.70842670 0.21869466

for( j in 1:dim(pairs)[2]){
  Corr[j,] <- samplingCor_xy(obsTimes,rate,initProb, pairs[1,j], pairs[2,j],
      sampSize, sampSize, nTot[pairs[1,j],], nTot[pairs[2,j],], progStructure)
}

#plotting
plot(obsTimes, corrs[1,], ylim = c(-.76,.98), type = "l", lty = 1,
     lwd = 1.4, col = pal[1], ylab = "Correlation", xlab = "Observation Time",
     main = "Fitted Correlations", cex.lab = 1.35, cex.main = 1.35)
for(i in 2:dim(pairs)[2]){
  lines(obsTimes, corrs[i,], lty = 1, lwd = 1.2, col = alpha(pal[i],.9) )
}
#fitted curves
for(i in 1:dim(pairs)[2]){
  lines(obsTimes, Corr[i,], col = alpha(pal[i],.7),
        lwd = 1.8, lty = (2 + i%%2) )
}
legend("right", legend=c("True", "Fitted"), bty="o", lty = c(1,4), lwd = 2)
```


Fitted Correlations
Correlation
1.0
0.5
0.0
−0.5
15
20
25
30
35
Observation Time
True
Fitted