Assignment 8

Hui Jeong (HJ) Jung, Gudmundur Björgvin Magnusson, Jieran Sun

2023-04-26

Problem 20 Classical NEMs

For this question we define our the adjacency phi matrices and compute the transitive closures using matrix exponentiation. We also define our rho matrices and compute the F matrix using matrix multiplication.

Here we define a function to compute the transitive closure of a graph. Instead of using a convergence check we just run for a couple of iterations.

1. Define variables

```
Sgenes <- c("S1","S2","S3","S4","S5")

Egenes <- c("E1","E2","E3","E4","E5","E6")

aphi <-   t(matrix(c(1,0,1,1,0,
                     0,1,0,0,1,
                     0,0,1,1,1,
                     0,0,0,1,1,
                     0,0,0,0,1),
                   nrow = 5,
                   dimnames = list(Sgenes,Sgenes)))
bphi <-   t(matrix(c(1,0,0,1,0,
                     0,1,0,0,1,
                     1,0,1,1,1,
                     0,0,0,1,1,
                     0,0,0,0,1),
                   nrow = 5,
                   dimnames = list(Sgenes,Sgenes)))
atheta <- (matrix(c(0,0,0,0,0,0,
                    0,0,0,1,0,1,
                    1,1,0,0,0,0,
                    0,0,1,0,0,0,
                    0,0,0,0,1,0),
                  ncol = 6,
                  dimnames = list(Sgenes,Egenes),
                  byrow = T
))
btheta <- (matrix(c(1,1,0,0,0,0,
                    0,0,0,1,0,1,
                    0,0,0,0,0,0,
                    0,0,1,0,0,0,
                    0,0,0,0,1,0),
                  ncol = 6,
                  dimnames = list(Sgenes,Egenes),
                  byrow=T))
# Transitive closure

transitive <- function(mat,iter) {
  n = 2
  temp <- mat
  matexp = mat
  while (n < iter) {
    matexp = mat%^%n
    temp = temp + matexp
    n = n + 1
  }
  return(ceiling(temp/max(temp)))
}

atphi = transitive(aphi,20)
btphi = transitive(bphi,20)

# Effect Pattern matrix F

aF = atphi%*%atheta
bF = btphi%*%btheta

aF
```

```
##    E1 E2 E3 E4 E5 E6
## S1  1  1  1  0  1  0
## S2  0  0  0  1  1  1
## S3  1  1  1  0  1  0
## S4  0  0  1  0  1  0
## S5  0  0  0  0  1  0
```

```
bF
```

```
##    E1 E2 E3 E4 E5 E6
## S1  1  1  1  0  1  0
## S2  0  0  0  1  1  1
## S3  1  1  1  0  1  0
## S4  0  0  1  0  1  0
## S5  0  0  0  0  1  0
```

2. Since we have a noiseless, discrete case the Data matrix is the transpose of the effect matrix

```
#(a)
(aD = t(aF))
```

```
##    S1 S2 S3 S4 S5
## E1  1  0  1  0  0
## E2  1  0  1  0  0
## E3  1  0  1  1  0
## E4   0  1  0  0  0
## E5   1  1  1  1  1
## E6   0  1  0  0  0
```

```
#(b)
(bD = t(bF))
```

```
##     S1 S2 S3 S4 S5
## E1   1  0  1  0  0
## E2   1  0  1  0  0
## E3   1  0  1  1  0
## E4   0  1  0  0  0
## E5   1  1  1  1  1
## E6   0  1  0  0  0
```

The data matrices are the same for both graphs, meaning we cannot discern between the two models.

```
all(aD == bD)
```

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

3. We can compute the scores by feeding out Data matrices from the previous section into the mnem::scorAdj() function from the mnem package.

```
aDscore <- mnem::scoreAdj(D = aD,adj = aphi, fpfn = c(0.05, 0.01))
bDscore <- mnem::scoreAdj(D = bD,adj = bphi, fpfn = c(0.05, 0.01))
```

Score for the (a) model:

```
aDscore$score
```

```
## [1] 14
```

Score for the (b) model:

```
bDscore$score
```

```
## [1] 14
```

Problem 21 Hidden Markov NEMs

1. Implement Hidden markov model in the transitional probability between the graphs.

```
adjMatU <- matrix(c(1,1,1,0,
                    0,1,1,1,
                    0,0,1,1,
                    0,0,0,1),
                  nrow = 4, byrow = TRUE)
```

```r
adjV1 <- matrix(c(1,1,1,0,
                  0,1,1,1,
                  0,0,1,0,
                  0,0,0,1),
                nrow = 4, byrow = TRUE)

adjV2 <- matrix(c(1,0,0,0,
                  1,1,1,0,
                  1,0,1,0,
                  1,0,0,1),
                nrow = 4, byrow = TRUE)

# endAdj <- list(adjV1, adjV2)

allGraphs <- mnem:::enumerate.models(c("S1","S2","S3","S4"))
```

```
## Generated 355 unique models ( out of 4096 )
```

```r
distFunc <- function(startAdj, endAdj, lambda){
  Suv <- sum(abs(endAdj - startAdj))
  Tuv <- lambda * ((1 - lambda)^Suv)
  return(Tuv)
}


transitionProb <- function(startAdj, endAdj, lambda, allGraphs){
  a <- distFunc(startAdj,endAdj,lambda)
  Cu<- 0
  for (graph in allGraphs) {
    Cu <- Cu + distFunc(startAdj,graph,lambda)
  }
  return(a/Cu)
}


transProb1 <- lapply(seq(0.1, 0.9, by = 0.1), function(lambda){
  transitionProb(startAdj = adjMatU, endAdj = adjV1, lambda = lambda, allGraphs)
}) %>% as.data.frame() %>% t() %>% as.data.table()

transProb2 <- lapply(seq(0.1, 0.9, by = 0.1), function(lambda){
  transitionProb(startAdj = adjMatU, endAdj = adjV2, lambda = lambda, allGraphs)
}) %>% as.data.frame() %>% t() %>% as.data.table()
```

```r
transProb <- cbind(V1= transProb1, V2= transProb2, lambda=seq(0.1, 0.9, by = 0.1))
```

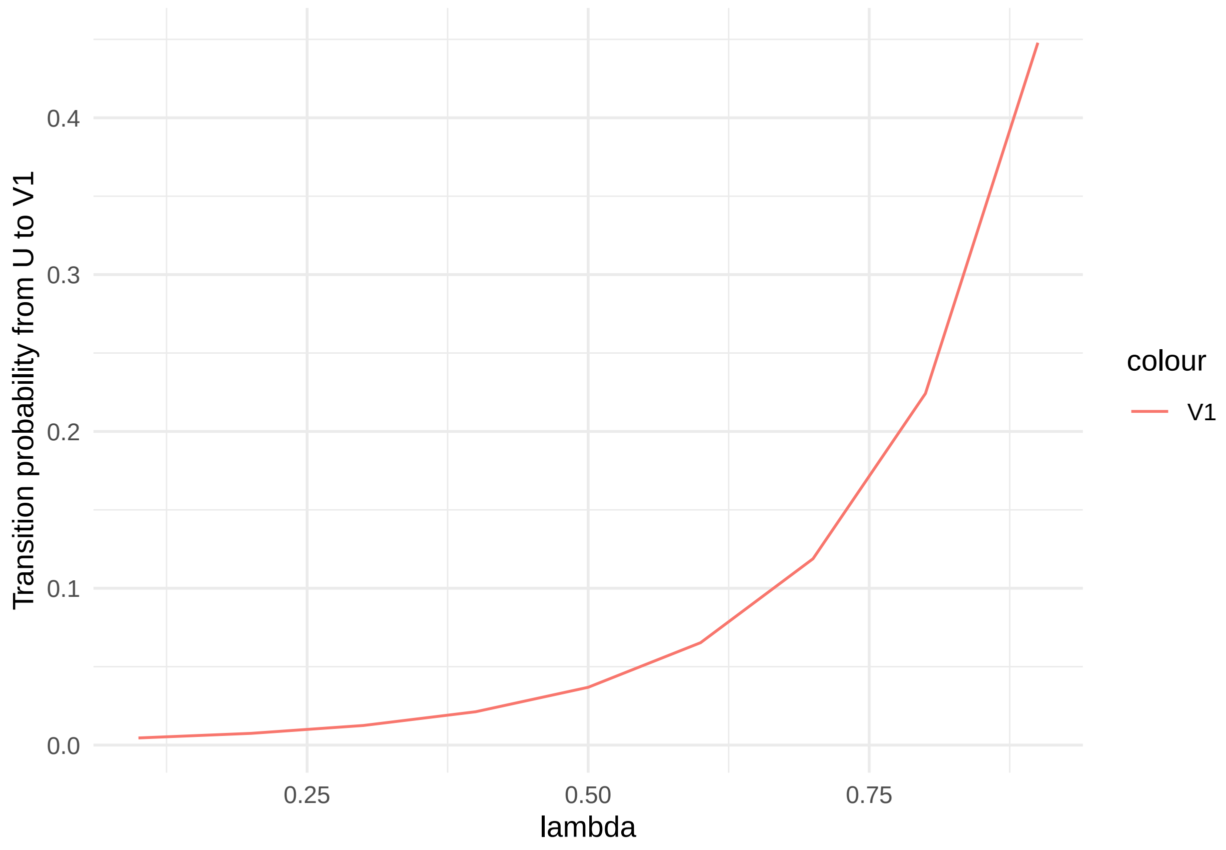
```r
knitr::kable(transProb)
```

V1.V1	V2.V1	lambda
0.0045546	0.0024205	0.1
0.0074835	0.0019618	0.2

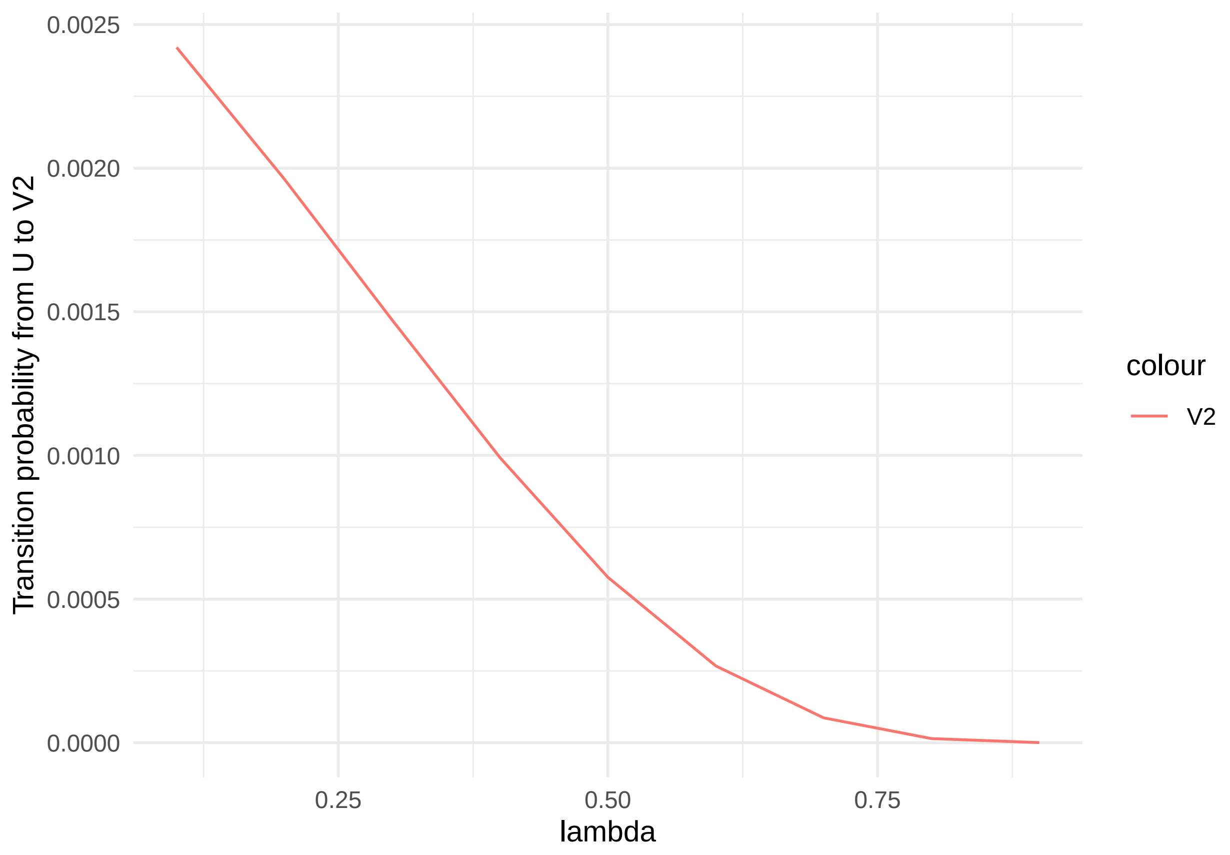
V1.V1	V2.V1	lambda
0.0125031	0.0014710	0.3
0.0212615	0.0009920	0.4
0.0368544	0.0005759	0.5
0.0652922	0.0002674	0.6
0.1188105	0.0000866	0.7
0.2241996	0.0000143	0.8
0.4478655	0.0000004	0.9

2. Plot out the results

```
ggplot(data = transProb) +
  geom_line(aes(x = lambda, y = V1.V1, col = "V1")) +
  ylab("Transition probability from U to V1") +
  theme_minimal()
```



```
ggplot(data = transProb) +
  geom_line(aes(x = lambda, y = V2.V1, col = "V2")) +
  ylab("Transition probability from U to V2") +
  theme_minimal()
```



The larger the lambda, the more the hamming distance matters and thus more heavily penalized. That's why the larger hamming distance have a lower probability to come after u while networks with a smaller distance have a higher probability. Therefore it makes sense that the transition probability from U to V1 is generally higher than the probability for U to V2, and it also explains why for V2 the transition probability continuously decreases while the lambda increases while for V1 it increases as the hamming distance is very low.

Problem 22 Mixture NEMs

1. Given the circumstances, the perturbation map rho

```
rho <- matrix(c(1,0,1,0,
                0,1,1,1),
              nrow = 2, byrow = TRUE)

print("rho: ")
```

```
## [1] "rho: "
```

```
print(rho)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    1    0
## [2,]    0    1    1    1
```

2. (a) Given the graphs shown in F1 and F2, the expected effect pattern for both cases are

```r
FList <- list()

FList[["K1"]] <- matrix(c(1,0,1,0,
                          1,1,1,1), nrow = 2, byrow = TRUE)
FList[["K2"]] <- matrix(c(0,1,1,1,
                          1,1,1,1), nrow = 2, byrow = TRUE)

FList
```

```
## $K1
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    1    0
## [2,]    1    1    1    1
##
## $K2
##      [,1] [,2] [,3] [,4]
## [1,]    0    1    1    1
## [2,]    1    1    1    1
```

2 (b) Given c1 and c2 are from F1, and c3 and c4 are from F2. The final R matrix is

```r
RMat <- as.matrix(cbind(FList$K1[,c(1,2)],
                        FList$K2[,c(3,4)]))
RMat[RMat == 0 ] <- -1
RMat
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1   -1    1    1
## [2,]    1    1    1    1
```

3 Calculating the EM algorithm to update the weight matrix

```r
logList <- list()
logList[["K1"]] <- diag(t(FList$K1) %*% RMat)
logList[["K2"]] <- diag(t(FList$K2) %*% RMat)
logList
```

```
## $K1
## [1] 2 1 2 1
##
## $K2
## [1] 1 0 2 2
```

Then we calculate the responsibility

```r
pi <- c(0.44, 0.56)

resp <- rbind(pi[1]*exp(logList$K1),
              pi[2]*exp(logList$K2)) %>% as.data.table()

resp <- resp[, lapply(.SD, function(x) x/sum(x)), .SD = colnames(resp)]
resp
```

```
##             V1        V2   V3        V4
## 1: 0.6811014 0.6811014 0.44 0.2242338
## 2: 0.3188986 0.3188986 0.56 0.7757662
```

Based on the responsibility, we update the weight as

```
(newWeight <- rowSums(resp)/sum(resp))
```

```
## [1] 0.5066091 0.4933909
```