The nested Dirichlet distribution and incomplete categorical data analysis (S-plus codes)

Kai Wang Ng[†], **Man-Lai Tang**[‡], **Guo-Liang Tian**[§] and **Ming Tan**[§,*]

[†]*Department of Statistics and Actuarial Science*
The University of Hong Kong, Pokfulam Road, Hong Kong, P. R. China

[‡]*Department of Mathematics, Hong Kong Baptist University,*
Kowloon Tong, Hong Kong, P. R. China

[§]*Division of Biostatistics, University of Maryland Greenebaum Cancer Center*
22 South Greene Street, Baltimore, Maryland 21201, U.S.A.
[*]Corresponding author's email: mtan@umm.edu

In this supplementary document, we provide fives main S-plus functions to facilitate the incomplete categorical data analysis based on the *nested Dirichlet distribution* (NDD).

- `x <- NDD.mode(a, b)`. This S-plus function finds the mode of the NDD defined by (2.1) or the MLE of $\boldsymbol{\theta}$ when the likelihood function takes the form of NDD;

- `theta.std <- NDD.std(a, b)`. This function calculates the standard errors of the MLE $\hat{\boldsymbol{\theta}}$, where $\hat{\boldsymbol{\theta}} = \arg\max L(\boldsymbol{\theta}|Y_{\text{obs}}) = \arg\max \text{ND}_{n,n-1}(\boldsymbol{\theta}|\mathbf{a}, \mathbf{b})$;

- `x <- rNDirichlet(a, b)`. It generates one random sample from $\text{ND}_{n,n-1}(\boldsymbol{\theta}|\mathbf{a}, \mathbf{b})$, which plays a crucial role in Bayesian analysis for incomplete categorical data;

- `x <- rmultinomial(n, p)`. It generates a random vector from the multinomial distribution with parameters n and $p = (p_1, \dots, p_d)^\top$. This function will be used in the I-step of the DA algorithm when the likelihood function takes the form of (3.5) based on NDD;

- `y <- rDirichlet(a)`. It generates a random vector from the Dirichlet distribution with parameters $a = (a_1, \dots, a_s)^\top$. This function will be used in the I-step of the DA algorithm when the likelihood function takes the form of (3.2) or (3.5) based on Dirichlet distribution.

```
function(a, b)
{
    # x <- NDD.mode(a, b)
    # Aim:     Finding the mode of ND_{n, n-1}(a, b) defined by (2.1)
    # Method:  Using Proposition 5
```

```
    # Input:   a = c(a_1, ..., a_n), b = c(b_1, ..., b_{n-1})
    # Output:  x = x(x_1, ..., x_n) is the mode of ND_{n, n-1}(a, b)
    n <- length(a)
    d <- rep(0, n - 1)
    x <- rep(0, n)
    for(j in 1:(n - 1)) {
       d[j] <- sum(a[1:j]) + sum(b[1:j])
    }
    x[n] <- (a[n]-1)/(d[n-1] + a[n] - n)
    for(j in 2:(n - 1)) {
       i <- n - j + 1
       x[i] <- ((a[i]-1) * (1-sum(x[(i+1):n])))/(d[i-1]+a[i]-i)
    }
    x[1] <- 1 - sum(x[2:n])
    return(x)
}
```

```
function(a, b)
{
    # theta.std <- NDD.std(a, b)
    # Aim:    Finding the standard error (std) of the MLE \hat{theta},
    #         where, \hat{theta} = arg max L(theta | Y_obs),
    #         L(theta | Y_obs) = ND_{n, n-1}(theta|a, b) given by (3.2)
    # Method: Using formulae (3.3) and (3.4)
    # Input:  a = c(a_1, ..., a_n), b = c(b_1, ..., b_{n-1})
    # Output: theta.std = c(theta.std_1, ..., theta.std_n)
    theta.MLE <- NDD.mode(a, b)
    # ----- Calculate the MLE of theta ----------------------------------
    n <- length(a)
    n1 <- n - 1
    Iobs <- diag((a[1:n1] - 1)/(theta.MLE[1:n1])^2) +
       (a[n] - 1)/(theta.MLE[ n])^2 * matrix(1, n1, n1)
    # --- Calculate the first two parts of I_obs(theta) defined by (3.3)
    psi <- rep(0, n1)
    for(k in 1:n1) {
       cc <- sum(theta.MLE[1:k])
       psi[k] <- b[k]/(cc * cc)
    }
```

```
    ID <- matrix(1, n1, n1)
    for(j in 1:(n1 - 1)) {
       ID[(j + 1):n1, j] <- 0
    }
    A <- psi %*% t(rep(1, n1))
    for(j in 2:n1) {
       A[1:(j - 1), j] <- 0
    }
    Iobs <- Iobs + ID %*% A
    B <- solve(Iobs)
    one <- rep(1, n1)
    theta.std <- c(sqrt(diag(B)), sqrt(t(one) %*% B %*% one))
    return(theta.std)
}
```

```
function(a, b)
{
    # x <- rNDirichlet(a, b)
    # Aim:    Generating one random sample from ND_{n, n-1}(a, b)
    #         defined by (2.1)
    # Method: Using Proposition 1
    # Input:  a = c(a_1, ..., a_n), b = c(b_1, ..., b_{n-1})
    # Output: x = c(x_1, ..., x_n) follows ND_{n, n-1}(a, b)
    n <- length(a)
    y <- rep(0, n - 1)
    for(j in 1:(n - 1)) {
       y[j] <- rbeta(1, sum(a[1:j]) + sum(b[1:j]), a[j + 1])
    }
    x <- rep(0, n)
    x[1] <- prod(y)
    for(i in 2:(n - 1)) {
       x[i] <- (1 - y[i - 1]) * prod(y[i:(n - 1)])
    }
    x[n] <- 1 - y[n - 1]
    return(x)
}
```

```
function(n, p)
{
    # x <- rmultinomial(n, p)
    # Aim:    Generating a random vector from the multinomial
    #         distribution with parameters n and p
    # Method: Using conditional sampling procedure
    # Input:  n, p = c(p_1, ..., p_d), where d >= 3
    # Output: x = c(x_1, ..., x_d) follows Multinomial(n, p)
    d <- length(p)
    N <- n
    S <- 1
    x <- rep(0, d)
    for(i in 1:(d - 1)) {
       if(n == 0) {
           x[i] <- 0
       }
       else {
           x[i] <- rbinom(1, n, p[i]/S)
       }
       n <- n - x[i]
       S <- S - p[i]
    }
    x[d] <- N - sum(x)
    return(x)
}
```

```
function(a)
{
    # y <- rDirichlet(a)
    # Aim:    Generating a random vector from y ~ D_s(a),
    #         where y_1 + ... + y_s = 1, s >= 3
    # Method: Using conditional sampling procedure
    # Input:  a = c(a_1, ..., a_s)
    # Output: y = c(y_1, ..., y_s) follows D_s(a)
    s <- length(a)
    b <- rep(0, s - 1)
    for(i in 1:(s - 1)) {
        b[i] <- rbeta(1, sum(a[1:i]), a[i + 1])
```

```
    }
    y <- rep(0, s)
    y[1] <- prod(b)
    for(i in 2:(s - 1)) {
       y[i] <- (1 - b[i - 1]) * prod(b[i:(s - 1)])
    }
    y[s] <- 1 - sum(y[1:(s - 1)])
    return(y)
}
```