Dissertation Topic Part 1: MISSING DATA AND PARAMETER ESTIMATES IN MULTIDIMENSIONAL ITEM RESPONSE MODELS

Maximum Likelihood Methods Using the Expeactation-Maximization Algorithm

SALVADOR CASTRO

Contents

1 Computational Linear Algebra

Preamble

```
# clear memory
rm(list = ls())
ls() # check memory
```

```
## character(0)
```

```
# set the default working directory
setwd("/Users/salvadorcastro/Desktop/R Files/EM")

# the number of digits to print
options(digits = 5)

# turn off warning messages
options(warn = -1)
```

The Data:

```
# Data matrix: Taken from Enders (2010)
A <-
  data.matrix(
    cbind(X = c(78, 84, 84, 85, 87, 91, 92, 94, 94, 96, 99, 105, 105, 106,
                108, 112, 113, 115, 118, 134, NA),
          Y = c(13, 9, 10, 10, NA, 3, 12, 3, 13, NA, 6, 12, 14, 10, NA, 10,
                14, 14, 12, 11, NA),
          Z = c(NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 7, 10, 11, 15, 10,
                10, 12, 14, 16, 12, NA)
    )
  )

knitr::kable(A, format = "pandoc",
             caption = 'Data matrix: Taken from Enders (2010).')
```

Table 1: Data matrix: Taken from Enders (2010).

X	Y	Z
78	13	NA
84	9	NA
84	10	NA
85	10	NA
87	NA	NA
91	3	NA
92	12	NA
94	3	NA
94	13	NA
96	NA	NA
99	6	7
105	12	10
105	14	11
106	10	15
108	NA	10
112	10	10
113	14	12
115	14	14
118	12	16
134	11	12
NA	NA	NA

```
# save data
save("A", file = "A.Rdata")

# remove trivial cases that have no observed values
A1 <- data.matrix(A[apply(A, 1, function(x){!all(is.na(x))}),])

knitr::kable(A1, format = "pandoc",
             caption = 'The trivial patern with all variables missing
             ommited from consideration because it contributes nothing to
```

```
            the observed-data likelihood and would only slow the
            convergence of EM by increasing the fractions of missing
            information')
```

Table 2: The trivial patern with all variables missing ommited from consideration because it contributes nothing to the observed-data likelihood and would only slow the convergence of EM by increasing the fractions of missing information

X	Y	Z
78	13	NA
84	9	NA
84	10	NA
85	10	NA
87	NA	NA
91	3	NA
92	12	NA
94	3	NA
94	13	NA
96	NA	NA
99	6	7
105	12	10
105	14	11
106	10	15
108	NA	10
112	10	10
113	14	12
115	14	14
118	12	16
134	11	12

```r
# omit cases with any missing values
A2 <- A[apply(A, 1, function(x)!any(is.na(x))), , drop = FALSE]

knitr::kable(A2, format = "pandoc",
             caption = 'Listwise deletion is a method for handling missing
             data in which an entire record is excluded from analysis if
             any single value is missing')
```

Table 3: Listwise deletion is a method for handling missing data in which an entire record is excluded from analysis if any single value is missing

X	Y	Z
99	6	7
105	12	10
105	14	11
106	10	15
112	10	10
113	14	12
115	14	14
118	12	16
134	11	12

1.1 Some Utility Functions

1.1.1 Column Sums

```r
SUMS <-
  function(Y) {
    # No missing values
    Y <- Y[apply(Y, 1, function(x)!any(is.na(x))), , drop = FALSE]
    ones <- rep(1, nrow(Y))
    return(t(ones %*% Y))
```

```
  } # SUMS

SUMS(A)
```

```
##    [,1]
## X 1007
## Y  103
## Z  107
```

```
# compare
colSums(A1)
```

```
##    X    Y    Z
## 2000   NA   NA
```

```
# compare
colSums(A, na.rm = TRUE)
```

```
##    X    Y    Z
## 2000  176  117
```

1.1.2 Column Means

```
MEANS <-
  function(Y){
    source("mean.imputation.R")

    # replace the missing values with their respective variable means
    Y <- mean.imputation(Y)
    n <- nrow(Y)

    # a vector of ones
    ones <- rep(1, n)
```

```
    return(t(ones %*% Y * (1/n)))
  } # MEANS

# save function
dump("MEANS", file = "MEANS.R")

MEANS(A)
```

```
##      [,1]
## X 100.000
## Y  10.353
## Z  11.700
```

```
# compare
colMeans(A, na.rm = TRUE)
```

```
##       X       Y       Z
## 100.000  10.353  11.700
```

```
# compare
colMeans(A1)
```

```
##   X   Y   Z
## 100  NA  NA
```

1.1.3 Matrix Equality

For two matrices to be equal, they must have the same dimensions and corresponding elements. In other words, let $A_{n\times m} = a[i,j]$ and $B_{p\times r} = b[i,j]$ be two matrices. $A = B$ if and only if:

$n = p$ and $m = r$

$a[i,j] = b[i,j]$

```
is.identical <-
  function(M, N){
    M <- data.matrix(M)
    N <- data.matrix(N)
    return(is.matrix(M) && is.matrix(N) && dim(M) == dim(N) && all(M == N))
  } # is.identical
dump("is.identical", file = "is.identical.R")
```

1.2 Matrices

1.2.1 Deviation Scores Matrix

A deviation score is the difference between a raw score and the mean.

$$d_i = x_i - \bar{x}$$

where

d_i is the deviation score for the i^{th} observation

x_i is the raw score for the i^{th} observation

$\bar{x}$ is the mean of all the observations

```
deviation.scores <-
  function(x){
    # No missing values
    x <- x[apply(x, 1, function(x)!any(is.na(x))), , drop = FALSE]
    n <- nrow(x)

    # add ones column
    ones <- as.matrix(rep(1, n))

    return(x - ones %*% t(ones) %*% x * (1/n))
  }
```

```r
# save function
dump("deviation.scores", file = "deviation.scores.R")

deviation.scores(A)
```

```
##                X        Y        Z
##  [1,] -12.88889 -5.44444 -4.88889
##  [2,]  -6.88889  0.55556 -1.88889
##  [3,]  -6.88889  2.55556 -0.88889
##  [4,]  -5.88889 -1.44444  3.11111
##  [5,]   0.11111 -1.44444 -1.88889
##  [6,]   1.11111  2.55556  0.11111
##  [7,]   3.11111  2.55556  2.11111
##  [8,]   6.11111  0.55556  4.11111
##  [9,]  22.11111 -0.44444  0.11111
```

```r
# compare
residuals <- apply(mean.imputation(A1), 2, function(x){x - mean(x)})
round(residuals, 5)
```

```
##          X        Y    Z
##  [1,] -22  2.64706  0.0
##  [2,] -16 -1.35294  0.0
##  [3,] -16 -0.35294  0.0
##  [4,] -15 -0.35294  0.0
##  [5,] -13  0.00000  0.0
##  [6,]  -9 -7.35294  0.0
##  [7,]  -8  1.64706  0.0
##  [8,]  -6 -7.35294  0.0
##  [9,]  -6  2.64706  0.0
## [10,]  -4  0.00000  0.0
## [11,]  -1 -4.35294 -4.7
## [12,]   5  1.64706 -1.7
## [13,]   5  3.64706 -0.7
```

```
## [14,]   6 -0.35294  3.3
## [15,]   8  0.00000 -1.7
## [16,]  12 -0.35294 -1.7
## [17,]  13  3.64706  0.3
## [18,]  15  3.64706  2.3
## [19,]  18  1.64706  4.3
## [20,]  34  0.64706  0.3
```

1.2.2 Sum of Squares and Cross Products Matrix

$$SSCP(X_{m\times n}) = X^TX = \begin{bmatrix} \sum x_1^2 & \sum x_1x_2 & \sum x_1x_3 & \dots & \sum x_1x_n \\ \sum x_2x_1 & \sum x_2^2 & \sum x_2x_3 & \dots & \sum x_2x_n \\ \sum x_3x_1 & \sum x_3x_2 & \sum x_3^2 & \dots & \sum x_3x_n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum x_mx_1 & \sum X_mx_2 & \sum x_mx_3 & \dots & \sum x_m^2 \end{bmatrix}$$

Function: Given a matrix Y, computes the SSCP matrix. The diagonal elements are sums of squares, and the off-diagonal elements are cross products.

```
SSCP <-
  function(Y) {
    # listwise deletion
    Y <- Y[apply(Y, 1, function(x)!any(is.na(x))), , drop = FALSE]

    return(t(Y) %*% Y)
  } # SSCP

# save function
```

```r
dump("SSCP", file = "SSCP.R")

SSCP(A)
```

```
##         X     Y     Z
## X 113505 11586 12070
## Y  11586  1233  1254
## Z  12070  1254  1335
```

```r
# compare
crossprod(A1)
```

```
##         X  Y  Z
## X 203792 NA NA
## Y     NA NA NA
## Z     NA NA NA
```

1.2.3 The Deviation Score Sums of Squares Matrix

$$CSSCP(X_{m\times n}) = [x_i - \bar{X}]^T[x_i - \bar{X}] = \begin{bmatrix} \sum d_1^2 & \sum d_1 d_2 & \sum d_1 d_3 & \dots & \sum d_1 d_n \\ \sum d_2 d_1 & \sum d_2^2 & \sum d_2 d_3 & \dots & \sum d_2 d_n \\ \sum d_3 d_1 & \sum d_3 d_2 & \sum d_3^2 & \dots & \sum d_3 d_n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum d_m d_1 & \sum d_m d_2 & \sum d_m d_3 & \dots & \sum d_m^2 \end{bmatrix}$$

where

$d_i = [x_i - \bar{X}]$

```
CSSCP <-
  function(Y){
    # listwise missing data deletion
    Y <- Y[apply(Y, 1, function(x)!any(is.na(x))), , drop = FALSE]
    n <- nrow(Y) # number of cases

    # residuals
    ones <- rep(1, n) # a vector of ones
    ONES <- ones  %*% t(ones) # 1's square-matrix
    R <- Y - ONES %*% Y * (1/n) # corrected residuals X-mean(X)/n

    return(t(R) %*% R)
  } # CSSCP

# save function
dump("CSSCP", file = "CSSCP.R")

CSSCP(A)
```

```
##         X      Y      Z
## X 832.889 61.444 97.889
## Y  61.444 54.222 29.444
## Z  97.889 29.444 62.889
```

```
# compare
crossprod(deviation.scores(A1))
```

```
##         X      Y      Z
## X 832.889 61.444 97.889
## Y  61.444 54.222 29.444
## Z  97.889 29.444 62.889
```

1.2.4 Variance-covariance Matrix

The variances appear along the diagonal and covariances appear in the off-diagonal elements

```r
COV <-
  function(A){
    source("CSSCP.R")
    source("mean.imputation.R")

    # replace the missing values with their respective variable means
    A <- mean.imputation(A)
    n <- nrow(A)

    # residuals
    ones <- rep(1, n) # vector of ones
    ONES <- ones  %*% t(ones) # a square-matrix of ones
    R <- A - ONES %*% A * (1/n) # residuals X-mean(X)

    if (n > 1) {
      return(CSSCP(A)/(n - 1))
    }
    else{return(CSSCP(A))}
  } # COV

# save function
dump("COV", file = "COV.R")

COV(A)
```

```
##         X       Y     Z
## X 189.600 11.6912 5.225
## Y  11.691  9.5941 1.565
## Z   5.225  1.5650 3.305
```

```r
# compare
cov(mean.imputation(A), use = "everything")
```

```
##         X       Y     Z
## X 189.600 11.6912 5.225
## Y  11.691  9.5941 1.565
## Z   5.225  1.5650 3.305
```

```
# compare
cov(A, use = "na.or.complete")
```

```
##          X      Y       Z
## X 104.1111 7.6806 12.2361
## Y   7.6806 6.7778  3.6806
## Z  12.2361 3.6806  7.8611
```

1.2.5 Variance-covariance Matrix (second method)

```
SIGMA <-
  function(A){
    source("MEANS.R")
    source("mean.imputation.R")

    # remove trivial cases that are all NAs and ensure proper data matrix
    y <- data.matrix(A[apply(A, 1, function(x){!all(is.na(x))}),])

    # replace the missing values with their respective variable means
    x <- mean.imputation(y)
    n <- nrow(x)

    return((1/n) * t(x) %*% x - MEANS(x) %*% t(MEANS(x)))
  } # SIGMA

# save function
dump("SIGMA", file = "SIGMA.R")

SIGMA(A)
```

```
##         X       Y     Z
## X 189.600 11.6912 5.225
## Y  11.691  9.5941 1.565
## Z   5.225  1.5650 3.305
```

1.2.6 Trace of a Square Matrix

The trace of a square matrix is defined as the sum of the elements on the main diagonal. The trace of a matrix is the sum of the (complex) eigenvalues, and it is invariant with respect to a change of basis.

A square matrix for testing:

```
B <- matrix(c(6, 12, 0, 12,
              12, 28, 0, 25,
              0, 0, 6, 2,
              12, 25, 2, 28),
            4, 4, byrow = TRUE)

B
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    6   12    0   12
## [2,]   12   28    0   25
## [3,]    0    0    6    2
## [4,]   12   25    2   28
```

```
TRACE <-
  function(Y) {
    # listwise missing data deletion
    Y <- as.matrix(Y[apply(Y, 1, function(x)!any(is.na(x))), , drop = FALSE])

    return(sum(diag(Y)))
  } # TRACE
```

```
# save function
dump("TRACE", file = "TRACE.R")

TRACE(B)
```

```
## [1] 68
```

```
# compare
psych::tr(B)
```

```
## [1] 68
```

1.2.7 SWEEP Operator

The sweep operator (Goodnight 1979) is closely related to Gauss-Jordan elimination and the Forward Doolittle procedure. A sweep operation can produce a generalized inverse by in-place mapping with minimal storage. The operation of sweeping on a variable turns that variable from an outcome variable into a predictor variable.

A tutorial on the SWEEP operator (Goodnight 1979):

The SWEEP operator can be programmed to produce generalized inverses and create, as by-products, such items as the Forward Doolittle matrix, the Cholesky decomposition matrix, the Hermite canonical form matrix, the determinant of the original matrix, Type I sums of squares, the error sum of squares, a solution to the normal equations, and the general form of estimable functions.

```
# As decribed in Goodnight (1979)
SWEEP <-
  function(A, k){
    a <- as.matrix(A)
    n <- nrow(a)

    # step 1
    D <- a[k,k]
```

```
    # step 2
    a[k,] <- a[k,]/D

    # step 3
    for (i in 1:n) {
      if (i != k) {
        B <- a[i, k]
        a[i,] <- a[i,] - B*a[k,]
        a[i,k] <- -B/D
      }
    }

    # step 4
    a[k,k] <- 1/D

    return(a)
  } # SWEEP

# save function
dump("SWEEP", file = "SWEEP.R")

SWEEP(B, 4)
```

```
##          [,1]     [,2]      [,3]      [,4]
## [1,]  0.85714  1.28571 -0.857143 -0.428571
## [2,]  1.28571  5.67857 -1.785714 -0.892857
## [3,] -0.85714 -1.78571  5.857143 -0.071429
## [4,]  0.42857  0.89286  0.071429  0.035714
```

Sweep as described in Little and Rubin (2002, 148):

```
# As described in Little and Rubin (2002, p. 148)
SWP <-
  function(A, k){
```

```
    h <- g <- as.matrix(A)
    n <- nrow(g)

    # step 1
    h[k,k] <- -1/g[k,k]

    # step 2
    for (j in 1:n) {
      if (j != k) {
        h[k,j] <- h[j,k] <- -g[j,k]*h[k,k]
      }
    }

    # step 3
    for (j in 1:n) {
      for (l in 1:n) {
        if (j != k && l != k) {
          h[j,l] <- g[j,l] - h[j,k]*g[k,l]
        }
      }
    }

    return(h)
  } # SWP

# save function
dump("SWP", file = "SWP.R")

SWP(B, 1)
```

```
##          [,1] [,2] [,3] [,4]
## [1,] -0.16667    2    0    2
## [2,]  2.00000    4    0    1
## [3,]  0.00000    0    6    2
```

```
## [4,]  2.00000    1    2    4
```

1.2.8 Reverse Sweep

```
RSW <-
  function(A, k){
    h <- g <- as.matrix(A)
    n <- nrow(g)

    # step 1
    h[k,k] <- -1/g[k,k]

    # step 2
    for (j in 1:n) {
      if (j != k) {
        h[k,j] <- h[j,k] <- g[j,k]*h[k,k]
      }
    }

    # step 3
    for (j in 1:n) {
      for (l in 1:n) {
        if (j != k && l != k) {
          h[j,l] <- g[j,l] + h[j,k]*g[k,l]
        }
      }
    }

    return(h)
  } # RSW

# save function
dump("RSW", file = "RSW.R")
```

```
RSW(B, 1)
```

```
##          [,1] [,2] [,3] [,4]
## [1,] -0.16667   -2    0   -2
## [2,] -2.00000    4    0    1
## [3,]  0.00000    0    6    2
## [4,] -2.00000    1    2    4
```

1.2.9 Properties of the SWEEP Operator

- Sweeping is reversible (i.e., sweeping twice on row r is equivalent to no sweep).

```
# Sweeping is reversible
SWEEP(SWEEP(B, 1), 1)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    6   12    0   12
## [2,]   12   28    0   25
## [3,]    0    0    6    2
## [4,]   12   25    2   28
```

```
RSW(SWP(B, 1), 1)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    6   12    0   12
## [2,]   12   28    0   25
## [3,]    0    0    6    2
## [4,]   12   25    2   28
```

- Sweeping is commutative (i.e., sweeping on row r and then on row s is equivalent to sweeping on s and then sweeping on r).

```r
# Sweeping is commutative
SWP(SWP(B, 3), 1)
```

```
##          [,1] [,2]     [,3]    [,4]
## [1,] -0.16667    2  0.00000 2.00000
## [2,]  2.00000    4  0.00000 1.00000
## [3,]  0.00000    0 -0.16667 0.33333
## [4,]  2.00000    1  0.33333 3.33333
```

```r
SWP(SWP(B, 1), 3)
```

```
##          [,1] [,2]     [,3]    [,4]
## [1,] -0.16667    2  0.00000 2.00000
## [2,]  2.00000    4  0.00000 1.00000
## [3,]  0.00000    0 -0.16667 0.33333
## [4,]  2.00000    1  0.33333 3.33333
```

1.2.10 Algorithmic efficiency: Comparing CPU Time

```r
# create a large symmetric matrix (1000x1000)
n <- 1e3
sym.mat <- matrix(rep(NA, n*n), nrow = n) # a 1000x1000 matrix

s <- sample(1:n, (n^2-n)/2, replace = TRUE) # a random sample of 1000^2 integers (1-100
d <- sample(1:n, n, replace = TRUE) # a random sample of 1000 integers (1-1000)

# populate lower triangle
sym.mat[lower.tri(sym.mat)] <- s
# transpose the matrix
sym.mat <- t(sym.mat)
# and populate the lower triangle, which was formerly the upper triangle
sym.mat[lower.tri(sym.mat)] <- s
# populate the diagonal
```

```
diag(sym.mat) <- d

# check for symmetry
isSymmetric(sym.mat, tol = 1e-5) # Differences smaller than tolerance are not reported.
```

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

```
# Check time: SWEEP
ptm <- proc.time()
S1 <- SWEEP(sym.mat, n)
proc.time() - ptm
```

```
##    user  system elapsed
##   0.057   0.012   0.070
```

```
# Check time: SWP
ptm <- proc.time()
S2 <- SWP(sym.mat, n)
proc.time() - ptm
```

```
##    user  system elapsed
##   2.680   0.028   2.713
```

1.2.11 Inverse of a Matrix

Definition: Let $A_{n\times n}$ be a square matrix. If a matrix A^{-1} can be found such that $AA^{-1} = A^{-1}A = I$ then A^{-1} is called the *inverse* of A.

```
INV <-
  function(A){
    source("SWEEP.R")

    A0 <- as.matrix(A)
    r <- ncol(A0)
```

```
    A1 <- SWEEP(A0, 1)

    if (r < 2) {
      return(A1)
    }
    else{
      for (k in 2:r) {
        A2 <- SWEEP(A1, k)
        A1 <- A2
      }
    }
    return(A1)
  } # INV

# save function
dump("INV", file = "INV.R")

INV(B)
```

```
##          [,1]      [,2]      [,3]      [,4]
## [1,]  1.89640 -0.378378  0.162162 -0.486486
## [2,] -0.37838  0.270270  0.027027 -0.081081
## [3,]  0.16216  0.027027  0.202703 -0.108108
## [4,] -0.48649 -0.081081 -0.108108  0.324324
```

```
# compare
solve(B)
```

```
##          [,1]      [,2]      [,3]      [,4]
## [1,]  1.89640 -0.378378  0.162162 -0.486486
## [2,] -0.37838  0.270270  0.027027 -0.081081
## [3,]  0.16216  0.027027  0.202703 -0.108108
## [4,] -0.48649 -0.081081 -0.108108  0.324324
```

```
# check
round(B %*% INV(B), 2) # should return the identity element for multiplication
```

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

1.2.12 Correlation Matrix

```
COR <-
  function(Y){
    source("COV.R")
    source("INV.R")
    source("CSSCP.R")

    # listwise missing data deletion
    Y <- as.matrix(Y[apply(Y, 1, function(x)!any(is.na(x))), , drop = FALSE])
    n <- nrow(Y) # number of cases
    p <- ncol(Y)

    V <- diag(COV(Y))
    S <- (diag(sqrt(V), p, p))
    C <- CSSCP(Y)/(n - 1)
    return(INV(S) %*% C %*% INV(S))
  } # COR

# save function
dump("COR", file = "COR.R")

COR(A)
```

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

```
## [1,] 1.00000 0.28913 0.42771
## [2,] 0.28913 1.00000 0.50423
## [3,] 0.42771 0.50423 1.00000
```

```
# compare
cor(A, use = "na.or.complete", method = "pearson")
```

```
##         X       Y       Z
## X 1.00000 0.28913 0.42771
## Y 0.28913 1.00000 0.50423
## Z 0.42771 0.50423 1.00000
```

```
# compare
cor(A, use = "pairwise.complete.obs", method = "pearson")
```

```
##         X       Y       Z
## X 1.00000 0.28378 0.44178
## Y 0.28378 1.00000 0.50423
## Z 0.44178 0.50423 1.00000
```

1.2.13 Determinant of a Matrix

Definition: For an $n \times n$ matrix $A = [a_{ij}]$, the determinant of A is defined to be the scalar

$$\| A \| = \sum_{p} \sigma(p) a_{1_{p_1}} a_{2_{p_2}} \dots a_{n_{p_n}}$$

where the sum is taken over the $n!$ permutations $p = (p_1, p_2, \dots, p_n) of (1, 2, \dots, n)$.

Determinants are defined only for square matrices. If the determinant of a matrix is 0, the matrix is said to be singular. The determinant of a triangular matrix is the product of its diagonal entries.

```
DET <-
  function(S){
    source("SWEEP.R")
```

```
    A <- as.matrix(S)
    r <- ncol(A)

    D <- A[1,1]

    if (r > 2) {
      B1 <- SWEEP(A, 1)
      for (k in 2:r) {
        D <- D * B1[k, k]
        B2 <- SWEEP(B1, k)
        B1 <- B2
      }
    }
    return(D)
  } # DET

# save function
dump("DET", file = "DET.R")

DET(B)
```

```
## [1] 444
```

```
# compare
det(B)
```

```
## [1] 444
```

```
# COMPARE
DET(as.matrix(189.6))
```

```
## [1] 189.6
```

```r
det(as.matrix(189.6))
```

```
## [1] 189.6
```

Determinant of variance-covariance matrix (Generalized Variance) Covariance matrix is always positive semi definite. That means the determinant must be positive. When it is zero, then the matrix is rank deficient. A matrix is said to have full rank if its rank equals the largest possible for a matrix of the same dimensions, which is the lesser of the number of rows and columns. A matrix is said to be rank deficient if it does not have full rank.

```r
# check

# A rank deficient matrix: Row 3 = Row 1 + Row 2
# i.e. the third row is a linear combination of the first two rows
D <- matrix(c(1, 2, 3,
              3, 4, 6,
              4, 6, 9), 3, 3, byrow = FALSE)

D
```

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

```r
DET(D)
```

```
## [1] 0
```

1.2.14 Cholesky Decomposition

Definition: Let A be a positive definite matrix. Then A can be decomposed in exactly one way into a product $A = U^T U$ such that U is upper triangular and has all main diagonal entries positive. U and and its conjugate transpose are called the Cholesky factors of A.

```r
# as described in Watkins (2002, p. 38)
cholesky <-
  function(A){
    a <- as.matrix(A)
    n <- nrow(a)

    for (i in 1:n) {
      for (k in 1:(i-1)) {
        if (i != 1) {
          a[i,i] <- a[i,i] - a[k,i]^2
          if (a[i,i] <= 0) {
            return("A is not positive definite")
          }
        }
      }

      # entire SWEEP computation is performed in-place
      a[i,i] <- sqrt(a[i,i]) # this is L_ii
      for (j in (i+1):n) {
        if (i != n) {
          for (k in 1:(i-1)) {
            if (i != 1) {
              a[i,j] <- a[i,j] - a[k,i]*a[k,j]
            }
          }
          a[i,j] <- a[i,j]/a[i,i] # this is L_ij
          a[j,i] <- 0 # set lower triangle to zero
        }
      }
    }

    return(a)
  }
```

```r
# save function
dump("cholesky", file = "cholesky.R")
```

```r
C <- matrix(c(4, -2, 4, 2,
              -2, 10, -2, -7,
              4, -2, 8, 4,
              2, -7, 4, 7), 4, 4, byrow = TRUE)
C
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    4   -2    4    2
## [2,]   -2   10   -2   -7
## [3,]    4   -2    8    4
## [4,]    2   -7    4    7
```

```r
b <- c(8, 2, 16, 6)

U <- cholesky(C); U # upper triangular matrix
```

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

```r
L <- t(U); L # the conjugate transpose of U, lower triangular matrix
```

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

```
# check
L %*% U # should return the original matrix, C
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    4   -2    4    2
## [2,]   -2   10   -2   -7
## [3,]    4   -2    8    4
## [4,]    2   -7    4    7
```

The Cholesky decomposition is mainly used for the numerical solution of linear equations $Ax = b$. If A is symmetric and positive definite (the normal equations in linear least squares problems are of this form), then we can solve for x by:

- first computing the Cholesky decomposition $A = U^T U$ and substituting A for $U^T U$, so $U^T U x = b$,
- then substituting Ux for y and solving $U^T y = b$ for y by forward substitution,
- and finally solving $Ux = y$ for x by back substitution.

1.2.15 Forward Substitution

Forward substitution is an iterative process to solve matrix equations in the form $\mathbf{Lx} = \mathbf{b}$ or $\mathbf{Ux} = \mathbf{b}$ for lower triangular matrices. Analogously back substitution can be used for upper triangular matrices.

```
# As described in Watkins (2002, p. 38)
forward.solve <-
  function(L, b){
    # Forward solve Ly = b
    n <- nrow(L)

    for (i in 1:n) {
      for (j in 1:(i-1)) {
        if (i != 1)
```

```
        b[i] = b[i] - L[i, j] * b[j]
      }
      if (L[i,i] == 0) {
        return("Error")
      }
      b[i] = b[i] / L[i, i]
    }
    return(as.matrix(b))
  }

# save function
dump("forward.solve", file = "forward.solve.R")

y <- forward.solve(L, b); y
```

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

```
# compare
forwardsolve(L, b)
```

```
## [1] 4 2 4 2
```

1.2.16 Back Substitution

```
backward.solve <-
  function(U, y){
    # Backward solve Ux = y
    n <- nrow(U)
```

```
    for (i in (n-1):1) {
      for (j in (i+1):n) {
        if (i < n)
        y[i] = y[i] - U[i, j] * y[j]
      }
      if (U[i,i] == 0) {
        return("Error")
      }
      y[i] = y[i] / U[i, i]
    }
    return(as.matrix(y))
  }

# save function
dump("backward.solve", file = "backward.solve.R")

x <- backward.solve(U, y); x
```

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

```
# compare
backsolve(U, y)
```

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

```
# check
b
```

```
## [1]  8  2 16  6
```

```
C %*% x # should return b as Cx=b
```

```
##      [,1]
## [1,]    8
## [2,]    2
## [3,]   16
## [4,]    6
```

1.2.17 The inverse of a Matrix by Cholesky decomposition:

The Cholesky decomposition can also be used for matrix inversion. It is much easier to compute the inverse and the determinant of a triangular matrix.

$$
\begin{aligned}
A &= U^T U \\
A^{-1} &= (U^T U)^{-1} \\
&= U^{-1} U^{T^{-1}} \\
&= U^{-1} U^{-1^T}
\end{aligned}
$$

```
V <- INV(U)
V %*% t(V)
```

```
##          [,1]     [,2]     [,3]     [,4]
## [1,]  0.63889  0.27778 -0.41667  0.33333
## [2,]  0.27778  0.55556 -0.33333  0.66667
## [3,] -0.41667 -0.33333  0.50000 -0.50000
## [4,]  0.33333  0.66667 -0.50000  1.00000
```

```
# same as
t(INV(L)) %*% INV(L)
```

```
##          [,1]     [,2]     [,3]     [,4]
## [1,]  0.63889  0.27778 -0.41667  0.33333
## [2,]  0.27778  0.55556 -0.33333  0.66667
## [3,] -0.41667 -0.33333  0.50000 -0.50000
## [4,]  0.33333  0.66667 -0.50000  1.00000
```

```
# compare
INV(C)
```

```
##          [,1]     [,2]     [,3]     [,4]
## [1,]  0.63889  0.27778 -0.41667  0.33333
## [2,]  0.27778  0.55556 -0.33333  0.66667
## [3,] -0.41667 -0.33333  0.50000 -0.50000
## [4,]  0.33333  0.66667 -0.50000  1.00000
```

1.2.18 The determinant of a Matrix by Cholesky decomposition:

$$
\begin{aligned}
A &= U^T U \\
\| A \| &= \| U^T U \| \\
&= \| U^T \| \| U \| \\
&= \left(\prod_{i=1}^{n} a_{ii} \right)^2
\end{aligned}
$$

```
# determinant of C
prod(diag(L))^2
```

```
## [1] 144
```

```
# compare
DET(C)
```

```
## [1] 144
```

Moreover, in the Monte Carlo method for simulating systems with multiple correlated variable, the Cholesky decomposition is used to decompose the correlation matrix to give the lower-triangular L. Then, this Cholesky factor is applied to a vector of uncorrelated samples to produce a sample vector with the covariance properties of the system being modeled.

2 Computational Multivariate Statistics Using Matrices

Data:

Numerical Example. Rubin (1976c) taken from Draper and Smith (1981).

```
B1 <- matrix(c(7, 26, 6, 60, 78.5,
               1, 29, 15, 52, 74.3,
               11, 56, 8, 20, 104.3,
               11, 31, 8, 47, 87.6,
               7, 52, 6, 33, 95.9,
               11, 55, 9, 22, 109.2,
               3, 71, 17, 6, 102.7,
               1, 31, 22, 44, 72.5,
               2, 54, 18, 22, 93.1,
               21, 47, 4, 26, 115.9,
               1, 40, 23, 34, 83.8,
               11, 66, 9, 12, 113.3,
               10, 68, 8, 12, 109.4), 13, 5, byrow = TRUE)

B1
```

```
##       [,1] [,2] [,3] [,4]  [,5]
##  [1,]    7   26    6   60  78.5
##  [2,]    1   29   15   52  74.3
##  [3,]   11   56    8   20 104.3
##  [4,]   11   31    8   47  87.6
##  [5,]    7   52    6   33  95.9
##  [6,]   11   55    9   22 109.2
##  [7,]    3   71   17    6 102.7
##  [8,]    1   31   22   44  72.5
##  [9,]    2   54   18   22  93.1
## [10,]   21   47    4   26 115.9
## [11,]    1   40   23   34  83.8
```

```
## [12,]   11   66    9   12 113.3
## [13,]   10   68    8   12 109.4
```

```
B2 <- matrix(c(7, 26, 6, 60, 78.5,
               1, 29, 15, 52, 74.3,
               11, 56, 8, 20, 104.3,
               11, 31, 8, 47, 87.6,
               7, 52, 6, 33, 95.9,
               11, 55, 9, 22, 109.2,
               3, 71, 17, NA, 102.7,
               1, 31, 22, NA, 72.5,
               2, 54, 18, NA, 93.1,
               NA, NA, 4, NA, 115.9,
               NA, NA, 23, NA, 83.8,
               NA, NA, 9, NA, 113.3,
               NA, NA, 8, NA, 109.4), 13, 5, byrow = TRUE)

B2
```

```
##       [,1] [,2] [,3] [,4]  [,5]
##  [1,]    7   26    6   60  78.5
##  [2,]    1   29   15   52  74.3
##  [3,]   11   56    8   20 104.3
##  [4,]   11   31    8   47  87.6
##  [5,]    7   52    6   33  95.9
##  [6,]   11   55    9   22 109.2
##  [7,]    3   71   17   NA 102.7
##  [8,]    1   31   22   NA  72.5
##  [9,]    2   54   18   NA  93.1
## [10,]   NA   NA    4   NA 115.9
## [11,]   NA   NA   23   NA  83.8
## [12,]   NA   NA    9   NA 113.3
## [13,]   NA   NA    8   NA 109.4
```

2.1 Regression

2.1.1 Regression Coefficients

```
b <-
  function(X, Y = NULL) {
    source("INV.R")

    # listwise missing data deletion
    X <- as.matrix(X)
    X <- as.matrix(X[apply(X, 1, function(x)!any(is.na(x))), , drop = FALSE])

    if (is.null(Y)) {
      p <- ncol(X)
      Y <- as.matrix(X[,p])
      X <- as.matrix(X[,1:(p - 1)])
    }
    else{
      X <- as.matrix(X)
      Y <- as.matrix(Y)
    }

    X1 <- cbind(rep(1, nrow(X)), X)  # design matrix
    XX <- t(X1) %*% X1 # X-prime*X
    XY <- t(X1) %*% Y # X-prime*Y

    B <- INV(XX) %*% XY

    return(B)
  } # b

# save function
dump("b", file = "b.R")
```

```
b(B1)
```

```
##           [,1]
## [1,] 62.40537
## [2,]  1.55110
## [3,]  0.51017
## [4,]  0.10191
## [5,] -0.14406
```

```
# compare
p <- ncol(B1)
coefficients <- as.matrix(coefficients(glm(B1[, p] ~ B1[, 1:(p - 1)])))
dimnames(coefficients) <- NULL
coefficients
```

```
##           [,1]
## [1,] 62.40537
## [2,]  1.55110
## [3,]  0.51017
## [4,]  0.10191
## [5,] -0.14406
```

2.1.2 Hat Matrix

The hat matrix (also called the influence matrix or projection matrix), maps the vector of response values (dependent variable values) to the vector of fitted values (or predicted values).

$$\hat{y} = Hy$$

The element in the i^{th} row and j^{th} column of H is equal to the covariance between the j^{th} response value and the i^{th} fitted value, divided by the variance of the former:

$$h_{ij} = \operatorname{cov}[\hat{y}_i, y_j] / \operatorname{var}[y_j]$$

Properties of H and (I-H) matrices:

- Symmetric: $H = H^T$ and $(I - H)^T = (I - H)$
- Idempotent: $H^2 = H$ and $(I - H)(I - H) = (I - H)$
- X is invariant under H : $HX = X$, hence $(I - H)X = 0$

```
HAT <-
  function(Y){
    source("INV.R")

    # listwise missing data deletion
    Y <- Y[apply(Y, 1, function(x)!any(is.na(x))), , drop = FALSE]
    n <- nrow(Y)

    X <- cbind(rep(1, n), Y) # design matrix

    return(X %*% INV(t(X) %*% X) %*% t(X))
  } # HAT

# save function
dump("HAT", file = "HAT.R")

round(HAT(B1), 1)
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
##  [1,]  0.6  0.2 -0.1  0.2  0.3  0.0 -0.1  0.1 -0.1   0.0   0.0   0.0   0.1
##  [2,]  0.2  0.4  0.1  0.1  0.1  0.1 -0.1  0.1  0.2  -0.2   0.1  -0.1  -0.2
##  [3,] -0.1  0.1  0.6  0.3  0.0  0.0  0.1  0.0  0.2   0.1  -0.2  -0.1   0.1
##  [4,]  0.2  0.1  0.3  0.4  0.0  0.0 -0.1  0.2  0.0   0.2  -0.1  -0.1   0.0
##  [5,]  0.3  0.1  0.0  0.0  0.4  0.1  0.1 -0.1  0.0  -0.1  -0.1   0.1   0.2
##  [6,]  0.0  0.1  0.0  0.0  0.1  0.4  0.0 -0.3  0.2   0.2   0.2   0.2  -0.1
##  [7,] -0.1 -0.1  0.1 -0.1  0.1  0.0  0.4  0.1  0.2  -0.1   0.1   0.2   0.3
##  [8,]  0.1  0.1  0.0  0.2 -0.1 -0.3  0.1  0.6  0.0   0.1   0.3  -0.1   0.1
##  [9,] -0.1  0.2  0.2  0.0  0.0  0.2  0.2  0.0  0.3  -0.1   0.2   0.0  -0.1
## [10,]  0.0 -0.2  0.1  0.2 -0.1  0.2 -0.1  0.1 -0.1   0.7   0.1   0.2   0.1
## [11,]  0.0  0.1 -0.2 -0.1 -0.1  0.2  0.1  0.3  0.2   0.1   0.5   0.1  -0.1
```

```
## [12,]  0.0 -0.1 -0.1 -0.1  0.1  0.2  0.2 -0.1  0.0  0.2  0.1  0.3  0.2
## [13,]  0.1 -0.2  0.1  0.0  0.2 -0.1  0.3  0.1 -0.1  0.1 -0.1  0.2  0.4
```

```
# check
X <- cbind(B1[,1:4])
Y <- as.matrix(B1[,5])

(design.matrix <- cbind(rep(1, nrow(X)), X))
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]    1    7   26    6   60
##  [2,]    1    1   29   15   52
##  [3,]    1   11   56    8   20
##  [4,]    1   11   31    8   47
##  [5,]    1    7   52    6   33
##  [6,]    1   11   55    9   22
##  [7,]    1    3   71   17    6
##  [8,]    1    1   31   22   44
##  [9,]    1    2   54   18   22
## [10,]    1   21   47    4   26
## [11,]    1    1   40   23   34
## [12,]    1   11   66    9   12
## [13,]    1   10   68    8   12
```

```
data.frame("Y.Hat.1" = round(design.matrix %*% b(B1), 1), # Y-hat (method 1)
           "Y.Hat.2" = round(HAT(X) %*% Y, 1), # Y-hat (method 2)
           "Y" = Y)
```

```
##    Y.Hat.1 Y.Hat.2     Y
## 1     78.5    78.5  78.5
## 2     72.8    72.8  74.3
## 3    106.0   106.0 104.3
## 4     89.3    89.3  87.6
## 5     95.6    95.6  95.9
## 6    105.3   105.3 109.2
```

```
## 7    104.1   104.1 102.7
## 8     75.7    75.7  72.5
## 9     91.7    91.7  93.1
## 10   115.6   115.6 115.9
## 11    81.8    81.8  83.8
## 12   112.3   112.3 113.3
## 13   111.7   111.7 109.4
```

The diagonal elements of the hat matrix, h_{ii}, are the leverages which are an indicator of the effects of observations on their own predicted values. In other words, the element, h_{ij} gives the influence of the j^{th} observation on the i^{th} predicted value, $\hat{y}_i$. Ideally, all of the elements of the diagonal are the same, which would mean that there aren't any outliers. The sum of the diagonal elements is the number of parameters to be estimated. If an element in the hat matrix diagonal is twice as great as that average value, then that observation is considered a leverage point

```
# diagonals
H <- HAT(B1)
diag(H)
```

```
##  [1] 0.55029 0.38096 0.63528 0.35756 0.35892 0.44609 0.41092 0.61915
##  [9] 0.33400 0.70206 0.50833 0.28276 0.41370
```

```
# compare
stats::hat(B1)
```

```
##  [1] 0.55029 0.38096 0.63528 0.35756 0.35892 0.44609 0.41092 0.61915
##  [9] 0.33400 0.70206 0.50833 0.28276 0.41370
```

```
# the number of parameters to be estimated
sum(diag(H))
```

```
## [1] 6
```

```r
# leverage points
any(diag(H) > 2*mean(diag(H)))
```

```
## [1] FALSE
```

2.1.3 Residuals

$$e = Y - \hat{Y} = Y - HY = (I - H)Y$$

```r
RESID <-
  function(Y, X = NULL){
    source("HAT.R")

    if (is.null(X)) {
      p <- ncol(Y)
      X <- as.matrix(Y[,1:(p - 1)])
      Y <- as.matrix(Y[,p])
    }
    else{
      X <- as.matrix(X)
      Y <- as.matrix(Y)
    }

    n <- nrow(Y)
    I <- diag(n)
    H <- HAT(X)
    return((I - H) %*% Y)
  } # RESID

# save function
dump("RESID", file = "RESID.R")

e1 <- round(RESID(B1), 4)
```

```r
# compare
k <- ncol(B1)
e2 <- as.matrix(round(residuals(glm(B1[, k] ~ B1[, 1:(k - 1)])), 4))

data.frame(e1 = e1, e2 = e2)
```

```
##          e1      e2
## 1    0.0048  0.0048
## 2    1.5112  1.5112
## 3   -1.6709 -1.6709
## 4   -1.7271 -1.7271
## 5    0.2508  0.2508
## 6    3.9254  3.9254
## 7   -1.4487 -1.4487
## 8   -3.1750 -3.1750
## 9    1.3783  1.3783
## 10   0.2815  0.2815
## 11   1.9910  1.9910
## 12   0.9730  0.9730
## 13 -2.2943 -2.2943
```

```r
# The sum and the sample mean of the residuals is zero
round(sum(e1), 10)
```

```
## [1] 0
```

```r
round(mean(e1), 10)
```

```
## [1] 0
```

2.1.4 The Covariance Matrix of the Residuals

$$
\begin{aligned}
e &= (I - H)y \\
\sigma_e^2 &= (I - H)\sigma_y^2(I - H)^T \\
&= \sigma_y^2(I - H)I(I - H) \\
&= \sigma_y^2(I - H)(I - H) \\
&= \sigma_y^2(I - H)
\end{aligned}
$$

- The variance-covariance matrix of the observed y's, equals the variance-covariance matrix of the residuals:

$$
\begin{aligned}
\text{var}(y_i) &= \text{var}(X_i\beta + e_i) \\
&= \text{var}(X_i\beta) + \text{var}(e_i) + 2\,\text{cov}(X_i\beta, e_i)
\end{aligned}
$$

where

- $\text{var}(\beta) = 0$
- $2\,\text{cov}(X_i\beta, e_i) = 0$

so

$\text{var}(y) = \text{var}(e)$

- No autocorrelation implies that the variance-covariance matrix is diagonal because it is assumed that the error terms are not correlated $\text{cov}(e_i, e_j) = 0$
- Homoskedasticity implies that error variance (each diagonal element) is the same for all observations.

```
cov.matrix <-
  function(data){
    source("HAT.R")
    source("COV.R")
    source("RESID.R")
```

```
    X <- as.matrix(data[,-ncol(data)])
    y <- as.matrix(data[ , ncol(data)])

    I <- diag(nrow(X)) # identity matrix
    H <- HAT(X) # hat matrix
    e <- RESID(X)  # residuals
    #C <- as.numeric(COV(e)) # sigma
    C <- as.numeric(var(e))

    return(round(((I-H) * C * t(I-H)), 1))
  }

round(cov.matrix(B1), 2)
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
##  [1,]  0.2  0.1  0.0  0.0  0.1  0.0  0.0  0.0  0.0   0.0   0.0   0.0   0.0
##  [2,]  0.1  0.4  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.0   0.0   0.0   0.0
##  [3,]  0.0  0.0  0.2  0.0  0.0  0.0  0.0  0.0  0.1   0.0   0.0   0.0   0.0
##  [4,]  0.0  0.0  0.0  0.5  0.0  0.0  0.0  0.0  0.0   0.0   0.0   0.0   0.0
##  [5,]  0.1  0.0  0.0  0.0  0.4  0.0  0.0  0.0  0.0   0.0   0.0   0.0   0.0
##  [6,]  0.0  0.0  0.0  0.0  0.0  0.8  0.0  0.0  0.0   0.0   0.0   0.0   0.0
##  [7,]  0.0  0.0  0.0  0.0  0.0  0.0  0.4  0.0  0.0   0.0   0.0   0.0   0.0
##  [8,]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.3  0.0   0.0   0.2   0.0   0.0
##  [9,]  0.0  0.0  0.1  0.0  0.0  0.0  0.0  0.0  0.5   0.0   0.0   0.0   0.0
## [10,]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.1   0.0   0.0   0.0
## [11,]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.2  0.0   0.0   0.3   0.0   0.0
## [12,]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.0   0.0   0.5   0.1
## [13,]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.0   0.0   0.1   0.5
```

2.2 ANOVA

ANOVA is a special case of regression.

2.2.1 Sources of Variation

The between group variation or model sum of squares is variation due to the interaction between the samples. If the sample means are close to each other (and therefore to the Grand Mean) this will be small.

$$SS_{Between} = \sum_{i=1}^{k} n_k(\bar{x}_k - \bar{x}_{grand})^2$$

where n_k is the number of cases within group k and $\bar{x}_k$ is the group mean.

The within group variation or residual (error) sum of squares is the variation due to differences within individual samples.

$$SS_{Within} = \sum_{i=1}^{n}(x_{ik} - \bar{x}_k)^2 = \sum_{k=1}^{k} s_k^2(n_k - 1)$$

where s_k^2 is the variance for group k and $(n_k - 1)$ is one less than the number of cases in that group.

The total variation (not variance) is comprised the sum of the squares of the differences of each mean with the grand mean. If the variance caused by the interaction between the samples is much larger when compared to the variance that appears within each group, then it is because the means aren't the same.

$$\begin{aligned} SS_{Total} &= SS_{Between} + SS_{Within} \\ &= \sum_{i=1}^{n}(x_i - \bar{x}_{grand})^2 \end{aligned}$$

where n is the number of cases and $\bar{x}_{grand}$ is the grand mean.

```
# data
X <- cbind(B1[,1:4])
Y <- as.matrix(B1[,5])
n <- nrow(B1)
p <- ncol(B1)

## Between Group Variation
```

```
SY <- HAT(X) %*% (Y - rep(MEANS(Y), n))
(SSB <- t(SY) %*% SY)
```

```
##          [,1]
## [1,] 2667.9
```

```
## Within Group Variation (Residual variation)
# Want to minimize sum of squared residuals.
(SSW <- t(e1) %*% e1)
```

```
##          [,1]
## [1,] 47.863
```

```
## Total Variation (not variance) SST
ST <- Y - rep(MEANS(Y), n)
(SST <- t(ST) %*% ST)
```

```
##          [,1]
## [1,] 2715.8
```

```
# check
SSB + SSW
```

```
##          [,1]
## [1,] 2715.8
```

2.2.2 Degrees of Freedom

$$df_{Total} = df_{Between} + df_{Within}$$

```
# Degrees of freedom
(dfB <- p - 1) # between
```

```
## [1] 4
```

```
(dfW <- n - p) # within
```

```
## [1] 8
```

```
(dfT <- n - 1) # total = between + within
```

```
## [1] 12
```

```
# check
dfB + dfW
```

```
## [1] 12
```

2.2.3 Mean Squares

Mean square between groups ($MS_{Between}$) is the variance due to the interaction between samples. This is the between group variation divided by its degrees of freedom.

```
MSB <- SSB/dfB; MSB # between
```

```
##         [,1]
## [1,] 666.97
```

Mean square within groups (MS_{Within}) is the variance due to the differences within individual samples. This is the within group variation divided by its degrees of freedom. It is the weighted (by the degrees of freedom) average of the variances.

```
MSW <- SSW/dfW; MSW # within
```

```
##         [,1]
## [1,] 5.9829
```

```r
# compare
MSE <- (t(e1) %*% e1)/dfW; MSE # mean squared error same as MSW
```

```
##         [,1]
## [1,] 5.9829
```

```r
RMSE <- sqrt(MSE); RMSE # root mean squared error
```

```
##        [,1]
## [1,] 2.446
```

2.2.4 F-test

$$F = \frac{MS_{Between}}{MS_{Within}}$$

- $H_0 : \beta_1 = \beta_2 = \cdots = \beta_{p-1} = 0$ (all regression coefficients are zero)
- $H_A : \beta_k \neq 0$, for k = 1, . . . , p-1 (at least one of the regression coefficients is not zero)
- Reject H_0 if F is larger than critical value; reject H_0 if p $< \alpha$ (0.05)

```r
# f-statistic
(F <- MSB/MSW)
```

```
##         [,1]
## [1,] 111.48
```

```r
# significance (p-value)
pf(F, df1 = dfB, df2 = dfW, lower.tail = FALSE)
```

```
##             [,1]
## [1,] 4.756e-07
```

```
# compare
# One Way Anova (Completely Randomized Design)
fit <- aov(Y ~ X)
summary(fit)
```

```
##             Df Sum Sq Mean Sq F value  Pr(>F)
## X            4   2668     667     111 4.8e-07 ***
## Residuals    8     48       6
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2.2.5 The squared multiple regression correlation (R^2)

R^2 gives the proportion of variation in the response variable explained by the explanatory variables.

```
R2 <- SSB/SST; R2
```

```
##         [,1]
## [1,] 0.98238
```

```
# same as
1 - (SSW/SST)
```

```
##         [,1]
## [1,] 0.98238
```

```
# The squared multiple correlation of a variable with the remaining
# variables in a matrix

# compare
1 - (1/(diag(INV(COR(B1)))))
```

```
## [1] 0.98316 0.99630 0.97871 0.99648 0.98238
```

```r
# compare
require(psych)
```

```
## Loading required package: psych
##
## Attaching package: 'psych'
##
## The following object is masked from 'package:eRm':
##
##     sim.rasch
##
## The following object is masked from 'package:irtoys':
##
##     sim
##
## The following object is masked from 'package:ltm':
##
##     factor.scores
##
## The following object is masked from 'package:polycor':
##
##     polyserial
```

```r
smc(B1)
```

```
##      V1      V2      V3      V4      V5
## 0.98316 0.99630 0.97871 0.99648 0.98238
```

```r
# F and R2 are related:
F <- (R2/(p - 1))/((1 - R2)/(n - p)); F
```

```
##        [,1]
## [1,] 111.48
```

3 An Introduction to Missing Data

Modern, principled missing data analysis methods, which are developed in the 1970s, are based on maximum likelihood estimation (MLE) such as Expectation-Maximization (EM) algorithm and Bayesian inference such as Markov chain Monte Carlo (MCMC) algorithms (Beale and Little 1975; Dempster, Laird, and Rubin 1977; Rubin 1976; Rubin 1987; Enders 2010, ~1; Schafer and Graham 2002). These missing data handling techniques present a cost-benefit trade-off between their theoretical superiority and ease of use as these methods are computationally intensive and require specialized software, which had not become widely available in statistical packages until late 1990s (Enders 2010, ~1). Additionally, these techniques and software require high technical expertise which also presents another significant obstacle for the general use of these methods (Enders 2010, ~1). As a result, despite their theoretical appeal and "state of the art" designation (Schafer and Graham 2002), there has been some hesitation among researchers in adopting these methods (Bodner 2006; Peugh and Enders 2004). However, these methods make relatively weaker assumptions concerning the cause of missing data; therefore, they produce less biased parameter estimates with greater power. (Enders 2010, ~1; Rubin 1976). Furthermore, they account for the missing data and the uncertainty introduced by it eliminating the need for ad-hoc analysis applicable to only a particular case (Schafer 1997, ~2). They can be applied more or less routinely to a wide variety of problems reducing the need for new technology unique to each problem (Schafer 1997, ~2).

Where as, traditional imputation methods have many disadvantages despite their ease of implementation. For example, replacing the missing values with their respective variable means preserves the observed sample means; however, it distorts the covariance structure by deflating dispersion (Schafer 1997, ~2). On the other hand, using regression models to predict imputation values tends to inflate correlations (Schafer 1997, ~2). What is more, using standard complete-data methods to compute measures of uncertainty such as standard errors or p-values may be misleading because they fail to account for the uncertainty due to imputed missing-data (Schafer 1997, ~2). Lastly, when the missing data pattern is complex in a multivariate data, it can be a challenging task to construct an ad-hoc imputation scheme that preserves crucial aspects of the joint distribution of the variables (Schafer 1997, ~2).

While it is preferable to prevent or limit the rate of missing data by experimental design before the actual data gathering takes place, nonresponse in surveys and attrition in social research experiments are often inevitable and present a threat to the validity of a statistical

analyses [Rubin (1976)}[p.~2]{Enders:2010}. As a result the researcher is often advised to plan to use methods of data analysis which are robust to bias due to incomplete cases in situations where missing data is unavoidable (Rubin 1976). For example, when a sample is drawn from a finite population; that is to say, values of variables for unsampled units are missing (Cochran 1977, ~18; Rubin 1976). In experiments with random assignment of the units to different treatment groups, each group is missing the values for the units which would have been observed had they not been assigned to other groups (Hinkelmann and Kempthorne 2007, 1: Introduction to Experimental Design:~138–9; Rubin 1978; Rubin 1976). Lastly, some observed values may be judged to be outliers and discarded (Rubin 1976). On the other hand, missing data may be created as an intentional by-product of data collection to solve a number of practical problems by taking advantage of the fact that in some situations missing data are relatively benign. (Enders 2010, ~2). Combined with modern missing data handling techniques, these planned missing data designs provide a means for reducing the cost of data collection (Enders 2010, ~2).

An analysis is robust when negligible violations of the key assumptions of an analysis produce little or no bias that would lead to distortion in the conclusions drawn about the population in question. However, ad-hoc techniques for handling missing data such as case-deletion and data-imputation require a relatively strict assumption about the cause of missing data (Enders 2010, ~1), and can potentially distort the representativeness of the sample data (Schafer 1997, ~1; Enders 2010, ~1). Most methods of missing-data handling assume that the mechanisms that lead to missing values in the data are accidental and random" therefore, each value in the sample data is equally likely to be missing [Afifi and Elashoff (1966); Anderson (1957); Hartley and Hocking (1971); Hocking and Smith (1968); Wilks (1932); Rubin (1976)}. On the other hand, the assumptions seams to be that only the values of dependent variables are missing at random with the analyses which deal with the analysis of variance (Hartley 1956; Healy and Westmacott 1956; Rubin 1972; Rubin 1976; Wilkinson 1958). If the values of a random variable are missing at random, then the data may still be representative of the population. However, if they are missing systematically, then the measurements of it is biased and the data sample does not represent the population. Therefore, missing values may lessen the representativeness of the sample data, therefore can lead to biased parameter estimations and false inferences about the population if the incomplete cases systematically differ from the complete cases (Rubin 1976, ~2; Schafer 1997). For example, omitting the incomplete cases may introduce bias in estimation of population parameters to the extent that the incomplete cases differ from the complete cases especially when the missing cases

make up more than 5% of all cases, which is often the situation with multivariate data with missing values in more than one variable (Schafer 1997, ~1). As a result, the researcher has to support his or her assumption that the probability of an observation being missing does not depend on observed or unobserved measurements, in order to be able to conduct unbiased statistical analyses.

3.1 Single Imputation Techniques

3.1.1 Replacing the missing values with their respective variable means

Replacing the missing values with their respective variable means preserves the observed sample means; however, it distorts the covariance structure by deflating dispersion.

```
mean.imputation <-
  function(Y){
    # Replace the missing values with their respective variable means
    n <- nrow(Y)
    k <- ncol(Y)
    z <- matrix(apply(Y, 2, function(x){replace(x, is.na(x), mean(x, na.rm = TRUE))}),
                nrow = n, ncol = k)
    colnames(z) <- colnames(Y)
    return(z)
  }

# save function
dump("mean.imputation", file = "mean.imputation.R")

A3 <- round(mean.imputation(A), 2); A3
```

```
##         X     Y    Z
##  [1,]  78 13.00 11.7
##  [2,]  84  9.00 11.7
##  [3,]  84 10.00 11.7
##  [4,]  85 10.00 11.7
##  [5,]  87 10.35 11.7
```

```
##  [6,]  91  3.00 11.7
##  [7,]  92 12.00 11.7
##  [8,]  94  3.00 11.7
##  [9,]  94 13.00 11.7
## [10,]  96 10.35 11.7
## [11,]  99  6.00  7.0
## [12,] 105 12.00 10.0
## [13,] 105 14.00 11.0
## [14,] 106 10.00 15.0
## [15,] 108 10.35 10.0
## [16,] 112 10.00 10.0
## [17,] 113 14.00 12.0
## [18,] 115 14.00 14.0
## [19,] 118 12.00 16.0
## [20,] 134 11.00 12.0
## [21,] 100 10.35 11.7
```

3.1.2 Random Hot-deck Imputation from the Sample of Respondents Using Bootstrapping

A Review of Hot Deck Imputation for Survey Non-response (Andridge and Little 2010):

Hot deck imputation is a method for handling missing data in which each missing value is replaced with an observed response from a "similar" unit. Despite being used extensively in practice, the theory is not as well developed as that of other imputation methods.

```
hot.deck.imputation <-
  function(x){
    imputed <- as.matrix(x)

    for (i in 1:ncol(x)) {
      missing <- is.na(x[, i]) # cases with missing data
      n.missing <- sum(missing) # number of cases with missing data
      x.obs <- x[!missing, i]
      imputed[missing, i] <- sample(x.obs, n.missing, replace = TRUE)
    }
```

```
    return(imputed)
  } # hot.deck.imputation

# save function
dump("hot.deck.imputation", file = "hot.deck.imputation.R")

hot.deck.imputation(A)
```

```
##          X  Y  Z
##  [1,]  78 13 15
##  [2,]  84  9 10
##  [3,]  84 10 12
##  [4,]  85 10 10
##  [5,]  87  3 12
##  [6,]  91  3 12
##  [7,]  92 12 16
##  [8,]  94  3 12
##  [9,]  94 13 10
## [10,]  96 11 14
## [11,]  99  6  7
## [12,] 105 12 10
## [13,] 105 14 11
## [14,] 106 10 15
## [15,] 108 12 10
## [16,] 112 10 10
## [17,] 113 14 12
## [18,] 115 14 14
## [19,] 118 12 16
## [20,] 134 11 12
## [21,]  85 11 10
```

3.1.3 Imputing Missing Values by Draws from a Predictive Distribution

```
hot.deck.imputation <-
  function(x){
    imputed <- as.matrix(x)

    for (i in 1:ncol(x)) {
      missing <- is.na(x[, i]) # cases with missing data
      n.missing <- sum(missing) # number of cases with missing data
      # Normal Distribution
      x.obs <- sample(rnorm(n = 1000,
                            mean = mean(x[,i], na.rm = TRUE),
                            sd = sd(x[,i], na.rm = TRUE)))
      imputed[missing, i] <- sample(x.obs, n.missing, replace = TRUE)
    }

    return(imputed)
  } # hot.deck.imputation

hot.deck.imputation(A)
```

```
##              X       Y       Z
##  [1,]  78.000 13.0000 10.4233
##  [2,]  84.000  9.0000  8.2071
##  [3,]  84.000 10.0000 13.1459
##  [4,]  85.000 10.0000 11.7381
##  [5,]  87.000  9.3445 12.9550
##  [6,]  91.000  3.0000 14.5822
##  [7,]  92.000 12.0000  8.2617
##  [8,]  94.000  3.0000  9.5681
##  [9,]  94.000 13.0000 11.7381
## [10,]  96.000 10.1927  7.9356
## [11,]  99.000  6.0000  7.0000
## [12,] 105.000 12.0000 10.0000
```

```
## [13,] 105.000 14.0000 11.0000
## [14,] 106.000 10.0000 15.0000
## [15,] 108.000  8.9560 10.0000
## [16,] 112.000 10.0000 10.0000
## [17,] 113.000 14.0000 12.0000
## [18,] 115.000 14.0000 14.0000
## [19,] 118.000 12.0000 16.0000
## [20,] 134.000 11.0000 12.0000
## [21,]  81.228 11.8611 12.6250
```

3.1.4 Estimating Parameters by Bootstrapping

```
hot.deck.parameters <-
  function(x, iterations){
    imputed <- as.matrix(x)
    standard.deviations <- matrix(NA, nrow = iterations, ncol = ncol(x))
    means <- matrix(NA, nrow = iterations, ncol = ncol(x))

    for (j in 1:iterations) {
      for (i in 1:ncol(x)) {
        missing <- is.na(x[, i]) # cases with missing data
        n.missing <- sum(missing) # number of cases with missing data
        x.obs <- x[!missing, i]
        imputed[missing, i] <- sample(x.obs, n.missing, replace = TRUE)
      }

      # Column standard deviations (listwise deletion)
      standard.deviations[j,] <- apply(imputed, 2, function(x){sd(x, na.rm = TRUE)})

      # Column means (listwise deletion)
      means[j,] <- colMeans(imputed, na.rm = TRUE)
    }
    parameters <- rbind(mean = colMeans(means), sdev = colMeans(standard.deviations))
    colnames(parameters) <- colnames(x)
```

```
    return(parameters)
  } # hot.deck.parameters

hot.deck.parameters(A, 1e3)
```

```
##           X       Y       Z
## mean 99.996 10.3523 11.7276
## sdev 14.078  3.4319  2.6001
```

3.2 Multiple Imputation Techniques

3.2.1 The Missing Data Patterns

A missing data pattern is the layout of observed and missing values in a data set (Enders 2010, ~3). (2010) defines six data patterns: When the missing values are isolated to a single variable, it is called the univariate pattern (Enders 2010, ~3). In a general missing data pattern, missing values dispersed throughout the data matrix in a haphazard fashion; however, there still may be a relationship between observed values and the propensity for missing data (Enders 2010, ~5). Unit nonresponse pattern occurs when a portion of the unit of analysis in the data is missing (Enders 2010, ~3). Unit nonresponse may occur because of several reasons such as failure to make contact with the sample unit in the sampling frame; refusal of the sample unit to participate; or accidental loss of the questionnaire data. A monotone missing data pattern typically occurs because of attrition in a longitudinal study (Enders 2010, ~4). The planned missing data pattern occurs when a parallel-form questionnaire design is used for the test and retest of the measurement procedure (Enders 2010, ~5). These alternate forms include some items that are common between the forms in addition to some others that are unique to individual forms so that the reliability of the instruments can be calculated. Finally, the latent variable pattern, which is unique to structural equation models, occurs when the values of the latent variables are missing for the entire sample (Enders 2010, ~5). Historically, methodologists have developed analytic tools which deals with a particular missing data pattern: however, modern missing data analysis methods does not distinguish between different missing data patterns (Enders 2010, ~5).

```r
# Number of incomplete cases
sum(!complete.cases(A))
```

```
## [1] 12
```

```r
# Number of missing values
sum(is.na(A))
```

```
## [1] 16
```

```r
# Which cases (row numbers) are incomplete?
which(!complete.cases(A))
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 15 21
```

Missingness Pattern

```r
pattern <-
  function(Y){
    # Missingness pattern
    return(unique(ifelse(!is.na(Y), 1, NA)))
  } # pattern

# save function
dump("pattern", file = "pattern.R")

(S <- pattern(A))
```

```
##       X  Y  Z
## [1,]  1  1 NA
## [2,]  1 NA NA
## [3,]  1  1  1
## [4,]  1 NA  1
## [5,] NA NA NA
```

Extract Data From a Matrix Matching a Particular Missingness Pattern

```
extract <-
  function(Y, s){
    # number of cases
    n <- nrow(Y)

    # extract data matching a particular pattern
    X <- NULL
    for (i in 1:n) {
      if (length(Y[i,]) == length(s) &&
          all((!is.na(Y[i,])) == !is.na(s))) {
        X <- rbind(X, Y[i,])
      } # if
    } # for i

    # drop variables that are all NAs
    return(Filter(function(x)!all(is.na(x)), data.frame(X, row.names = NULL)))
  } # extract

# save function
dump("extract", file = "extract.R")

S <- pattern(A)

for (i in 1:nrow(S)) {
  cat("Pattern:")
  cat("\n")
  print(S[i,])
  cat("\n")
  cat("Matching data:")
  cat("\n")
  print(extract(A, S[i,]))
  cat("\n")
}
```

```
## Pattern:
##  X  Y  Z
##  1  1 NA
##
## Matching data:
##    X  Y
## 1 78 13
## 2 84  9
## 3 84 10
## 4 85 10
## 5 91  3
## 6 92 12
## 7 94  3
## 8 94 13
##
## Pattern:
##  X  Y  Z
##  1 NA NA
##
## Matching data:
##    X
## 1 87
## 2 96
##
## Pattern:
## X Y Z
## 1 1 1
##
## Matching data:
##     X  Y  Z
## 1  99  6  7
## 2 105 12 10
## 3 105 14 11
## 4 106 10 15
```

```
## 5 112 10 10
## 6 113 14 12
## 7 115 14 14
## 8 118 12 16
## 9 134 11 12
##
## Pattern:
##  X  Y  Z
##  1 NA  1
##
## Matching data:
##     X  Z
## 1 108 10
##
## Pattern:
##  X  Y  Z
## NA NA NA
##
## Matching data:
## data frame with 0 columns and 1 row
```

3.2.2 The Missing Data Mechanisms

According to (Rubin 1976) who has established a classification system for missing data mechanisms and assumptions for missing data analyses, there are three types of missing data mechanisms, which describe the relationship between observed variables and the probability of missing data. The missing data mechanisms represent generic mathematical relationships between the data and missingness; however, they do not offer a causal explanation for the missing data (Enders 2010, ~3). For example, there may be a systematic relationship between the propensity for nonresponse and income level.

3.2.2.1 Missing Completely at Random ($MCAR$) If the probability of an observation being missing on a variable X is related to neither measurements of other variables (observed or unobserved) nor to the values of X itself, then the missingness is due to completely random

mechanisms (Rubin 1976). The probability distribution function (pdf) for MCAR data is

$$P(R|X_{obs}, X_{mis}, \phi) = P(R|\phi)$$

where

R is the binary missing data indicator ($R = 1$, if the value of the random variable X is observed; $R = 0$, if the value of the random variable X is missing).

X_{obs} is the observed part of the data

X_{mis} is the missing part of the data

ϕ is a parameter (or set of parameters) that describes the relationship between R and the data

3.2.2.2 Missing at Random (MAR) If the probability of an observation being missing on a variable X is only related to values of X itself but not to other observed measurements, then the missingness is due to random mechanisms (Rubin 1976). The pdf of MAR data is,

$$P(R|X_{obs}, X_{mis}, \phi) = P(R|X_{obs}, \phi)$$

3.2.2.3 Missing not at Random ($MNAR$) If the missing values in a random variable X is due to a specific reason hence are not at random, then the data is missing not at random (MNAR) (Rubin 1976). An example of this is if certain question on a questionnaire such as income or age tend to be skipped deliberately by participants with certain characteristics. The pdf for MNAR data includes all possible associations between the data and missingness,

$$P(R|X_{obs}, X_{mis}, \phi) = P(R|X_{obs}, X_{mis}, \phi)$$

3.2.3 Little's MCAR Test

Many statistical analyses with missing data require that the missing data is not related to the values of any variables in the data set (Little 1988). In other words the missing values in the data are MCAR. However, MLE from a data with missing values require that the data is MAR, if the data also satisfies the multivariate normality assumption (Orchard and

Woodbury 1972). (Little 1988) MCAR test computes mean differences across subgroups of cases with the same missing data pattern. The test statistic is a weighted sum of the standardized differences between the subgroup means and the grand means, as follows:

$$d^2 = \sum_{j=1}^{J} n_j \left(\hat{\mu}_j - \hat{\mu}_j^{(ML)}\right)^T \hat{\Sigma}_j^{-1} \left(\hat{\mu}_j - \hat{\mu}_j^{(ML)}\right)$$

where

n_j is the number of cases in missing data pattern j,

$\hat{\mu}_j$ contains the variable means for the cases in missing data pattern j,

$\hat{\mu}_j^{(ML)}$ contains maximum likelihood estimates of the grand means, and

$\hat{\Sigma}_j$ is the maximum likelihood estimate of the covariance matrix.

d^2 is approximately distributed as a chi-square statistic with $\Sigma k_j - k$ degrees of freedom, where k_j is the number of complete variables for pattern j, and k is the total number of variables. A significant d^2 statistic provides evidence against MCAR.

```
MCAR_Test <-
  function(x, alpha = 0.05, plot = FALSE){
    # default alpha, 0.05: type I error rate
    # (incorrect rejection of a true null hypothesis)

    require(bbmle)
    require(stats)
    source("shadowtext.R") # for pretty font
    source("pattern.R") # for missingness pattern
    source("extract.R") # for data extraction
    source("SIGMA.R") # for variance-covariance matrix
    source("MEANS.R") # for variable means
    source("INV.R") # for matrix inversion

    # remove cases that are all NAs and ensure proper data matrix
    x <- x[apply(x, 1, function(x){!all(is.na(x))}),]

    # missingness pattern
```

```
  S <- pattern(x)

  # initiate test statistic, d-squared
  d2 <- 0.0

  # total number of variables
  sum_n <- 0

  # for each pattern
  for (i in 1:nrow(S)) {
    # extract data matching a particular pattern
    M1 <- data.matrix(extract(x, S[i,]))
    M2 <- subset(x, select = !is.na(S[i,]))

    # matrix of means
    mu1 <- MEANS(M1)
    mu2 <- MEANS(M2)

    # mean difference
    mean.diff <- mu1 - mu2

    # for degrees of freedom computation
    sum_n <- sum_n + ncol(M1)

    # variance-covariance matrix
    Sigma <- SIGMA(M2)

    # test statistic, d^2
    d2 <- as.numeric(d2 + nrow(M1)*t(mean.diff) %*% INV(Sigma) %*% (mean.diff))
    df <- sum_n - ncol(x) # degrees of freedom
  } # for i

  # significance
  p <- 1 - pchisq(d2, df)
```

```
# Plot
#-----------------------------------------------------------------
if (plot) {
  n <- 1e3 # sample size = 1,000

  # x-values
  minx <- qchisq(.00001, df) # shading minimum x-value
  critical <- qchisq(1 - alpha, df) # shading critical x-value
  maxx <- qchisq(.999, df) # shading maximum x-value
  x <- seq(0, maxx, length = n) # x continium

  # chi-square density
  y <- dchisq(x, df, ncp = 0, log = FALSE)

  #data
  data <- cbind(x, y)

  # plot margins
  par(mar = c(6, 6, 6, 0) + .1, mgp = c(4, 1, 0), las = 1)
  # chi-squared distribution plot
  plot(data,
       type = "l",
       lwd = 1,
       col = "black",
       yaxs = "i",
       xaxs = "i",
       ylim = c(0, max(y)*1.05),
       xlab = substitute(paste(chi[df]^{2}), list(df = df)),
       ylab = substitute(paste(P(chi[df]^{2})), list(df = df)),
       cex.lab = 1.5,
       cex.axis = 1.5,
       cex.main = 1.5,
       col.main = "darkgreen",
```

```
          col.lab = "darkgreen",
          main = substitute(atop("Little's MCAR Test",
                                 paste("(", chi[df]^{2}, " - distribution)")),
                            list(df = df)))

    # gridlines
    grid(nx = NULL, ny = NULL,
          col = "lightgray",
          lty = "dotted",
          lwd = par("lwd"),
          equilogs = TRUE)

    box()

    # rejection area shading
    xx <- c(seq(critical, maxx, length = 100), maxx, critical, critical)
    yy <- c(dchisq(seq(critical, maxx, length = 100), df), 0, 0, dchisq(critical, df))

    polygon(xx, yy,
            border = NA,
            col = adjustcolor("red", alpha.f = 0.7))

    # acceptable area shading
    xx <- c(seq(minx, critical, length = 100), critical, minx, minx)
    yy <- c(dchisq(seq(minx, critical, length = 100), df), 0, 0, dchisq(minx, df))

    polygon(xx, yy,
            border = NA,
            col = adjustcolor("green", alpha.f = 0.7))

    # test-statistic marking line
    segments(d2, -0.05, d2, dchisq(d2, df) + 0.0025,
             col = "yellow",
             lty = 1,
```

```
          lwd = 4
  )
  segments(d2, -0.05, d2, dchisq(d2, df) + 0.0025,
          col = "black",
          lty = 1,
          lwd = 2
  )

  # text: statistic, p-value, degrees of freedom
  shadowtext(d2, dchisq(d2, df) + 0.01,
             pos = 4,
             cex = 1.3,
             srt = 90,
             offset = -.6,
             col = "darkgreen",
             substitute(paste(chi[df]^{2} == d2, ", p = ", p),
                        list(d2 = round(d2, 2),
                             df = df,
                             p = round(p, 3))))
  # text: critical value, alpha, degrees of freedom
  shadowtext(d2 + .4, dchisq(d2, df) + 0.01,
             pos = 4,
             cex = .8,
             srt = 90,
             col = "maroon",
             substitute(paste("     (", {chi[df]^{2}}[][-critical] == d2,
                              ", ", alpha == a, ")"),
                        list(d2 = round(critical,2),
                             df = df,
                             a = round(alpha, 2))))

  # significance indicator circle
  if (d2 < critical) {indicator.col <- "green"}
  else if (d2 >= critical){indicator.col <- "red"}
```

```
      symbols(x = maxx - .05*diff(range(x)),
              y = max(y) - .015*diff(range(y)),
              circles = .5,
              inches = FALSE,
              add = TRUE,
              fg = "white",
              bg = indicator.col)
    }

    return(list(d2 = as.numeric(d2),
                df = as.numeric(df),
                p.value = as.numeric(p),
                pattern = data.frame(ifelse(is.na(S), 0, 1), row.names = NULL)))
  } # MCAR_Test

# save function
dump("MCAR_Test", file = "MCAR_Test.R")

MCAR_Test(A, alpha = 0.01, plot = TRUE)
```

```
## Loading required package: bbmle
## Loading required package: stats4


## $d2
## [1] 15.747
##
## $df
## [1] 5
##
## $p.value
## [1] 0.0076052
##
## $pattern
```

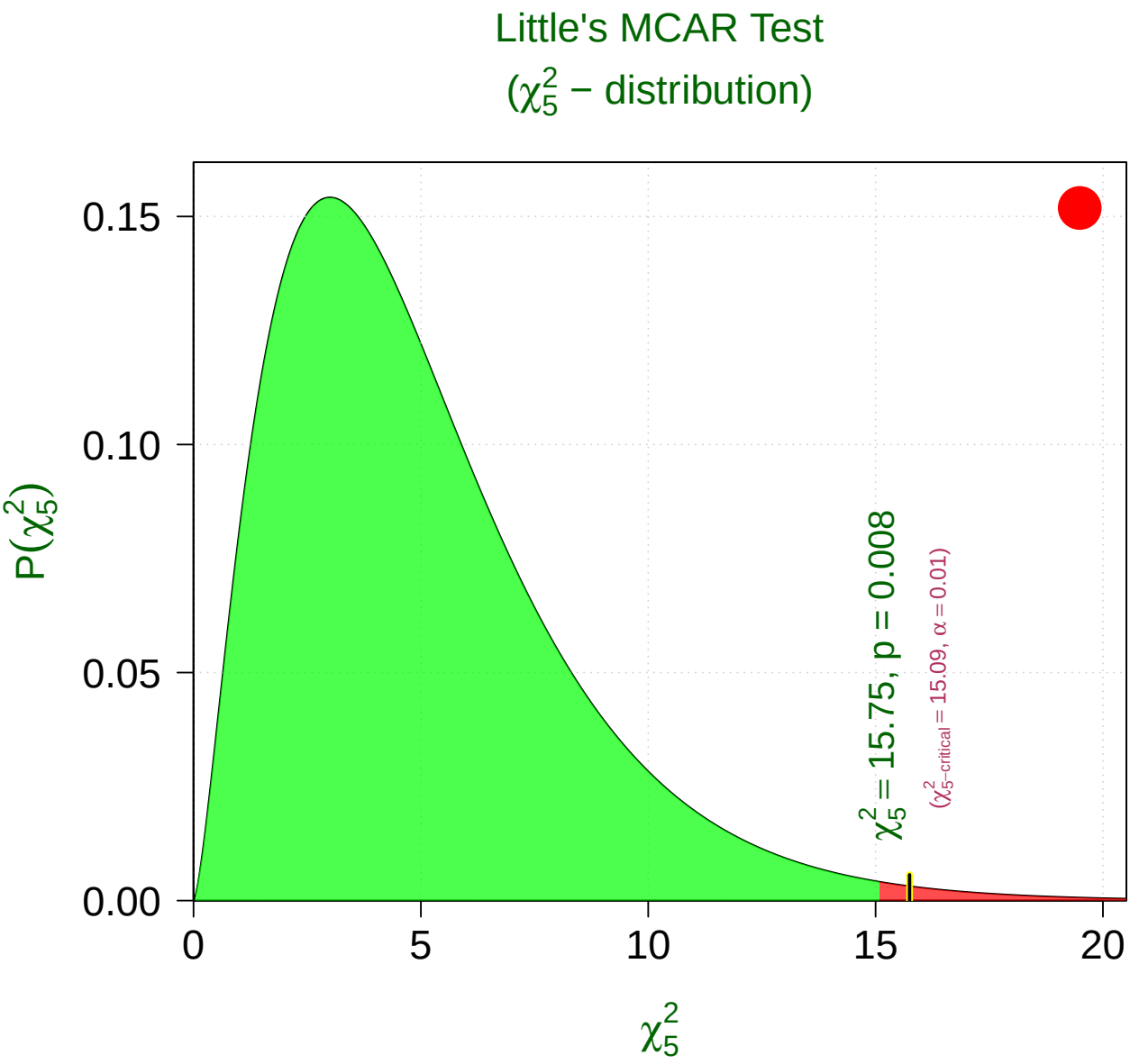
Little's MCAR Test
(χ^2_5 − distribution)
0.15
0.10
0.05
0.00
$P(\chi^2_5)$
0
5
10
15
20
χ^2_5
$\chi^2_5 = 15.75$, p = 0.008
($\chi^2_{5\text{-critical}} = 15.09$, $\alpha = 0.01$)

Figure 1:

```
##   X Y Z
## 1 1 1 0
## 2 1 0 0
## 3 1 1 1
## 4 1 0 1
```

```
# compare
require(BaylorEdPsych)
```

```
## Loading required package: BaylorEdPsych
```

```
LittleMCAR(A2)
```

```
## Loading required package: mvnmle

## this could take a while

## $chi.square
## [1] 1.3686e-11
##
## $df
## [1] 0
##
## $p.value
## [1] 0
##
## $missing.patterns
## [1] 1
##
## $amount.missing
##                 X Y Z
## Number Missing  0 0 0
## Percent Missing 0 0 0
##
## $data
```

```
## $data$DataSet1
##     X  Y  Z
## 1  99  6  7
## 2 105 12 10
## 3 105 14 11
## 4 106 10 15
## 5 112 10 10
## 6 113 14 12
## 7 115 14 14
## 8 118 12 16
## 9 134 11 12
```

```
# the difference is due to mle estimates of means and covariances
SIGMA(A) # vs
```

```
##         X       Y     Z
## X 189.600 11.6912 5.225
## Y  11.691  9.5941 1.565
## Z   5.225  1.5650 3.305
```

```
mlest(A)$sigmahat
```

```
##         [,1]    [,2]    [,3]
## [1,] 189.601 12.2068 22.3064
## [2,]  12.207 11.0363  5.6064
## [3,]  22.306  5.6064  8.6760
```

```
MEANS(A) # vs
```

```
##      [,1]
## X 100.000
## Y  10.353
## Z  11.700
```

```
mlest(A)$muhat
```

```
## [1] 100.000  10.271  10.231
```

Reject the null hypothesis if the test statistic is greater than the critical value. In other words, if the probability ($p = 0.01$) of obtaining the test statistic ($\chi^2 = 11.07$) is less than 5% ($\alpha = 0.05$), reject the null hypothesis (H_0). The null hypothesis for Little's test states that the data are $MCAR$, so a statistically significant test statistic provides evidence against the $MCAR$ mechanism. In other words, missing values are not completely at random and the probability of missing data on a variable is related to other measured variables or to its own values. In the sample above, missing values are $MCAR$ ($\chi^2_5 = 11.07, p = 0.012$).

Second Example

```
MCAR_Test(B2[,1:4], plot = TRUE)
```

```
## $d2
## [1] 9.1088
##
## $df
## [1] 4
##
## $p.value
## [1] 0.058437
##
## $pattern
##   X1 X2 X3 X4
## 1  1  1  1  1
## 2  1  1  1  0
## 3  0  0  1  0
```

Failed to reject the null hypothesis. Missing values are MCAR ($\chi^2_4 = 9.11, p = 0.06$)..

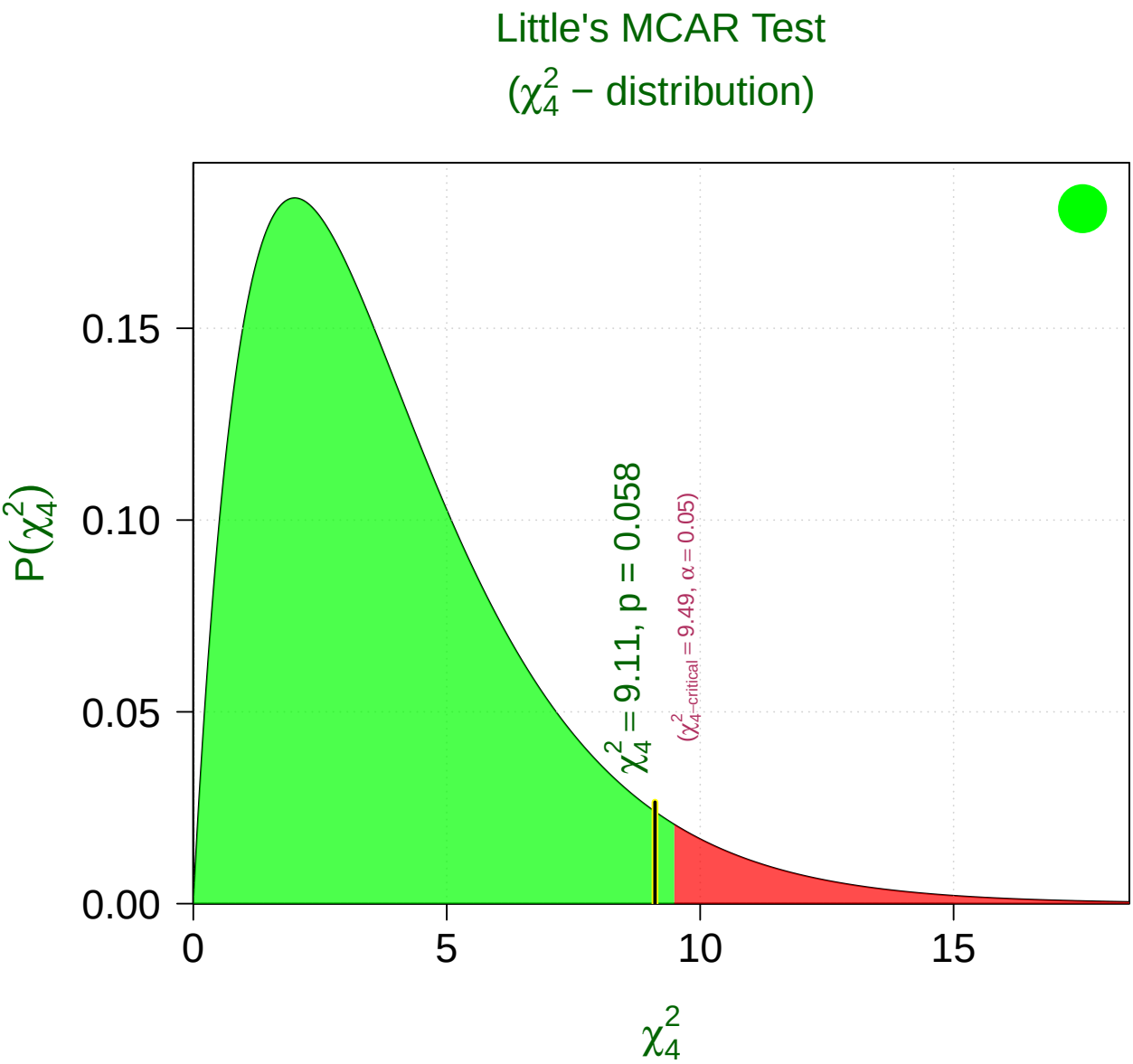
Little's MCAR Test
(χ^2_4 − distribution)
0.15
0.10
0.05
0.00
P(χ^2_4)
0
5
10
15
χ^2_4
χ^2_4 = 9.11, p = 0.058
($\chi^2_{4\text{-critical}}$ = 9.49, α = 0.05)

Figure 2:

3.2.4 An Introduction to Maximum Likelihood Estimation

Maximum likelihood missing data technique yields unbiased parameter estimates under a missing at random (MAR) mechanism. From a practical standpoint, this means that maximum likelihood will produce accurate parameter estimates in situations where traditional approaches fail. Even when the data are missing completely at random (MCAR), maximum likelihood will still be superior to traditional techniques (e.g., deletion methods) because it maximizes statistical power by borrowing information from the observed data. Despite these desirable properties, maximum likelihood estimation is not a perfect solution and will yield biased parameter estimates under a missing not at random (MNAR) mechanism. However, this bias tends to be isolated to a subset of the analysis model parameters, whereas traditional techniques are more apt to propagate bias throughout the entire model.

3.2.4.1 Maximum Likelihood Estimation with Univariate Data The density function describes the relative probability of obtaining a score value from a normally distributed population with a particular mean and variance.

The density function for the univariate normal distribution is:

$$L_i = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-0.5(x_i-\mu)^2}{\sigma^2}}$$

where

x_i is a score value,

μ is the population mean,

σ^2 s the population variance, and

L_i is a likelihood value that describes the height of the normal curve at a particular score value.

```
univariate.likelihood <-
  function(x){

    # remove cases that are all NAs and ensure proper data matrix
    x <- as.matrix(x)
    x <- as.matrix(x[apply(x, 1, function(x){!all(is.na(x))}),])
```

```
    n <- nrow(x)

    # parameters
    mu <- mean(x, na.rm = TRUE) # mean
    var <- var(x, na.rm = TRUE) # variance

    # likelihood intialized
    L <- matrix(NA, n, 1)

    # likelihood computed
    for (i in 1:n) {
      L[i, 1] <- exp((-.5*(x[i, 1] - mu)^2)/(var))/sqrt(2*pi*var)
    }
    return(L)
  } # univariate.likelihood

# save function
dump("univariate.likelihood", file = "univariate.likelihood.R")

# save function
dump("univariate.likelihood", file = "univariate.likelihood.R")

# compare
X1 <- as.matrix(A[,1][!is.na(A[,1])])

data.frame("X" = X1,
           "L1" = round(univariate.likelihood(X1), 5),
           "L2" = round(dnorm(X1, mean(X1), sd(X1)), 5),
           "logL" = round(log(univariate.likelihood(X1)), 5))
```

```
##      X      L1      L2    logL
## 1   78 0.00840 0.00840 -4.7796
## 2   84 0.01487 0.01487 -4.2084
## 3   84 0.01487 0.01487 -4.2084
```

```
## 4   85 0.01607 0.01607 -4.1307
## 5   87 0.01849 0.01849 -3.9904
## 6   91 0.02305 0.02305 -3.7700
## 7   92 0.02406 0.02406 -3.7274
## 8   94 0.02580 0.02580 -3.6572
## 9   94 0.02580 0.02580 -3.6572
## 10  96 0.02713 0.02713 -3.6071
## 11  99 0.02817 0.02817 -3.5696
## 12 105 0.02652 0.02652 -3.6297
## 13 105 0.02652 0.02652 -3.6297
## 14 106 0.02580 0.02580 -3.6572
## 15 108 0.02406 0.02406 -3.7274
## 16 112 0.01969 0.01969 -3.9278
## 17 113 0.01849 0.01849 -3.9904
## 18 115 0.01607 0.01607 -4.1307
## 19 118 0.01254 0.01254 -4.3788
## 20 134 0.00156 0.00156 -6.4631
```

```
likelihood.plot <-
  function(y, var.name = NULL, FUN = NULL, title = NULL){
    require(graphics)
    require(shape)
    source("shadowtext.R")

    # x continuum
    mu <- mean(y, na.rm = TRUE) # mean of data
    var <- var(y, na.rm = TRUE)
    sd <- sqrt(var) # variance of data
    min <- mu - 3.5*sd # minimum value of data
    max <- mu + 3.5*sd # maximum value of data

    x <- seq(from = min, to = max, by = .1)
```

```
  if (is.null(FUN)) {
    # normal likelihood (default) computed
    FUN <- function(x, mu, var){
      L <- rep(NA, length(x)) # intialize L
      for (i in 1:length(x)) {
        L[i] <- exp((-.5*(x[i] - mu)^2)/(var))/sqrt(2*pi*var)
      }
      return(L)
    }
  }

  # plot margins
  par(mar = c(5, 8, 4, 0) + .1, mgp = c(4, 1, 0), las = 1)

  # plot
  plot(x, FUN(x, mu, var),
       axes = FALSE,
       las = 1,
       xlab = "",
       ylab = "",
       type = "l",
       lwd = 5,
       col = "darkgreen",
       xaxs = "i",
       yaxs = "i",
       ylim = c(0, FUN(x = mu, mu, var)*1.05),
       xlim = c(min, max))

  # plot title
  if (is.null(title)) { # default title
    title(substitute(paste(Nu,"~(", mu, "=", m, ", ", sigma, "=", s,")"),
                     list(m = round(mu, 2), s = round(sd, 2))),
          col.main = "darkgreen",
          cex.main = 2)
```

```
  }
  else {# custom title
    title(title,
          col.main = "darkgreen",
          cex.main = 2)
  }

  box()

  # likelihood for mean, mean-sdev and mean+2*sdev
  X0 <- c(mu, mu - 2*sd, mu + sd)
  Y0 <- rep(0, 3)
  X1 <- rep(min, 3)
  Y1 <- c(FUN(x = mu, mu, var),
          FUN(x = mu - 2*sd, mu, var),
          FUN(x = mu + sd, mu, var))

  # x-axis
  #axis(1, pretty(c(min(x), max(x)), 4), cex.axis = 1.2)
  axis(1, at = X0, labels = sprintf("%.2f", X0), cex.axis = 1)
  mtext(var.name,
        col = "darkgreen",
        side = 1,
        cex = 1.5,
        line = 3.5)

  # y-axis
  #axis(2, pretty(c(0, max(d.norm)), 3), cex.axis = 1.2)
  axis(2, at = Y1, labels = sprintf("%.4f", Y1), cex.axis = 1)
  mtext("Likelihood",
        las = 0,
        col = "darkgreen",
        cex = 1.5,
        side = 2,
```

```
          line = 5)

    # tracing lines
    segments(x0 = X0,
             y0 = Y0,
             x1 = X0,
             y1 = Y1,
             col = "black",
             lty = 3,
             lwd = 1)

    Arrows(x0 = X0,
           y0 = Y1,
           x1 = X1,
           y1 = Y1,
           col = "black",
           arr.type = "triangle",
           arr.adj = 1,
           code = 2,
           lty = 3,
           lwd = 1)

    points(X0, Y1,
           col = "white",
           bg = "darkgreen",
           cex = .6,
           pch = 21)
  } # likelihood.plot

likelihood.plot(X1, var.name = "X", title = "normal dist.")
```

3.2.4.2 Probability Density Function (pdf): Normal Distribution

```
## Loading required package: shape
```

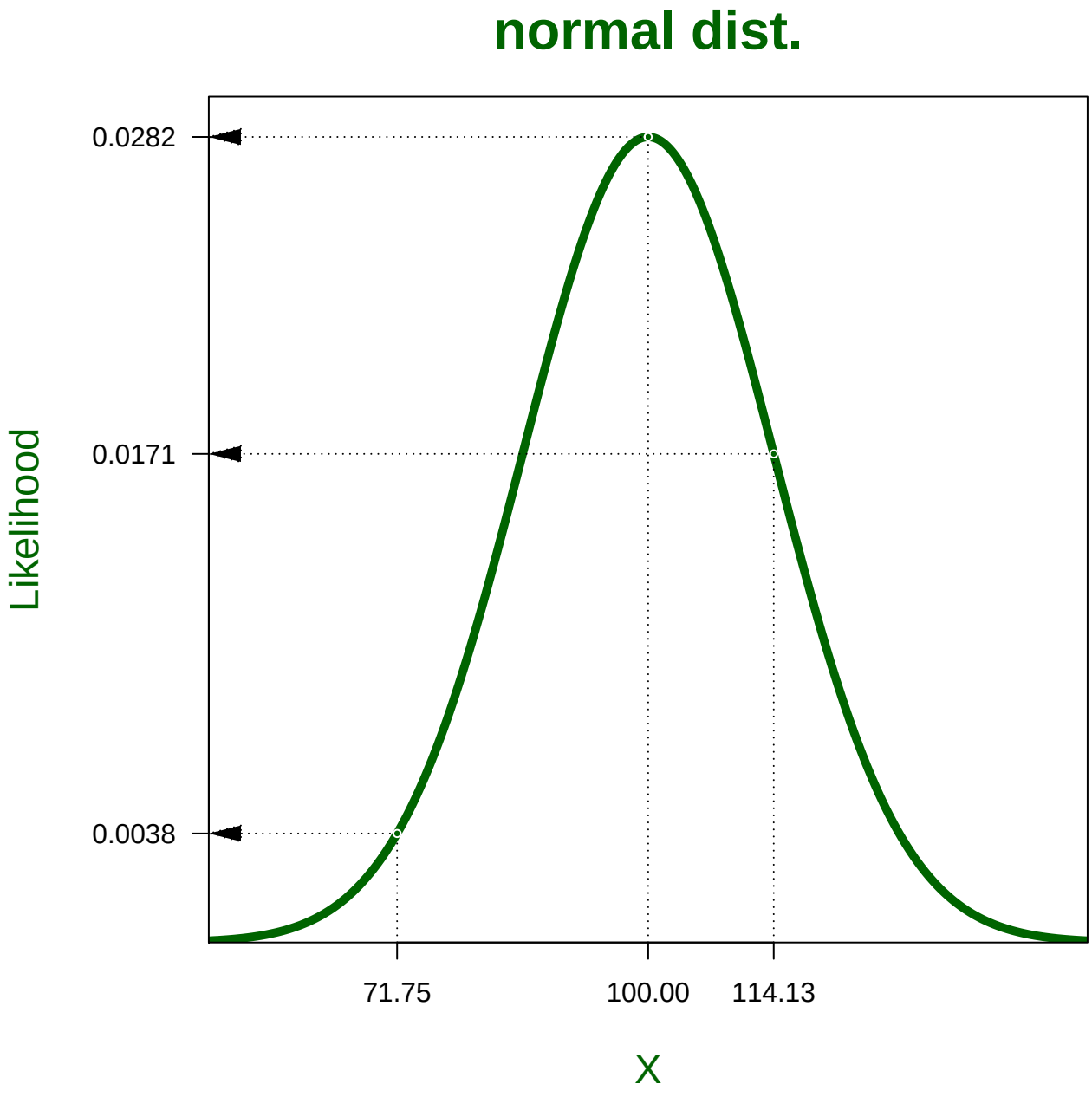
normal dist.
0.0282
0.0171
0.0038
Likelihood
71.75
100.00
114.13
X

Figure 3:

```
# Standard Normal Distribution
x <- rnorm(1e5, 0, 1)

likelihood.plot(x)
```

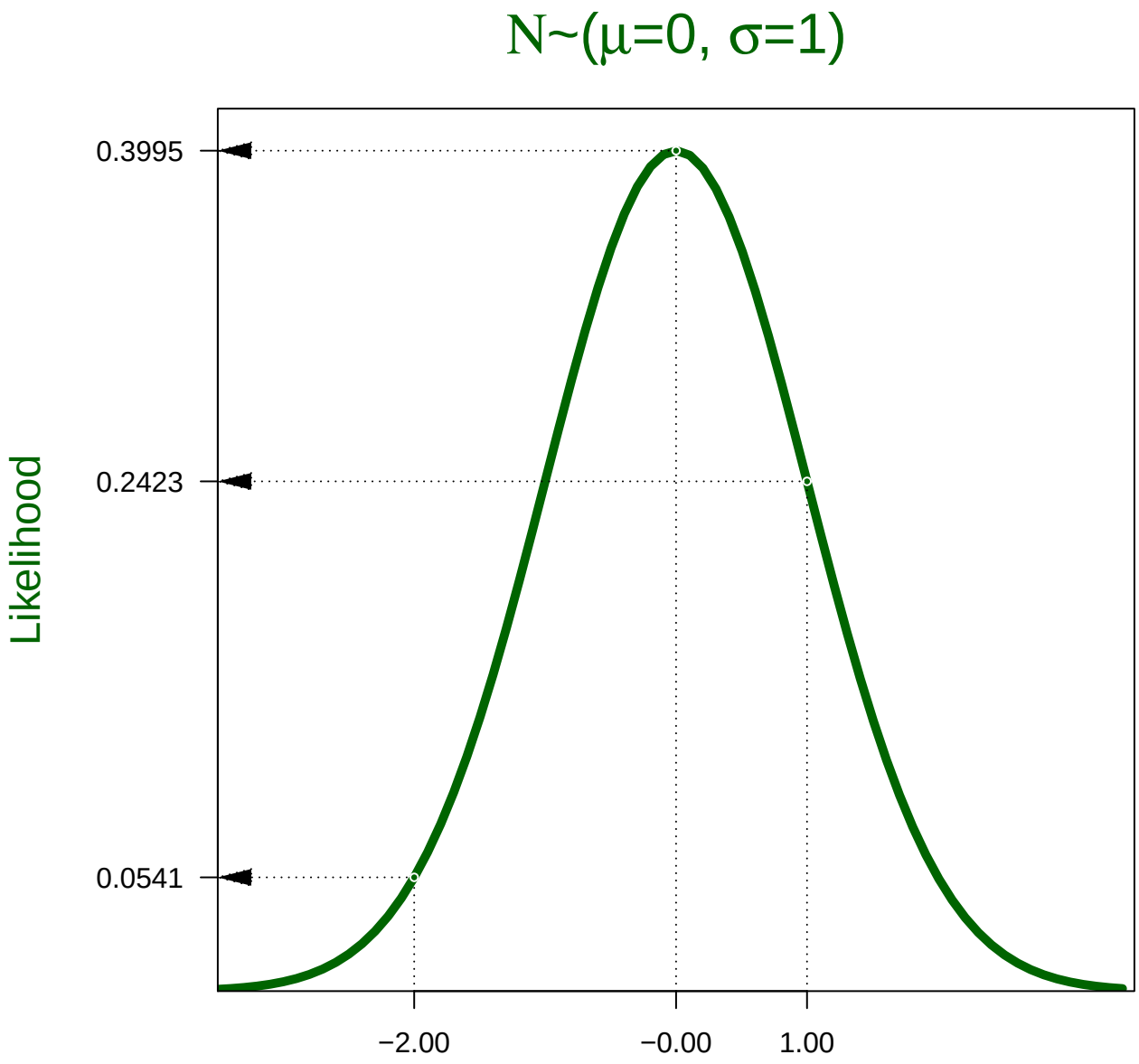

Figure 4:

3.2.4.3 Likelihood of Standard Normal Distribution

3.2.4.4 The Sample Likelihood The goal of maximum likelihood estimation is to identify the population parameter values that have the highest probability of producing a particular sample of data. Identifying the most likely parameter values requires a summary fit measure for the entire sample, not just a single score. In probability theory, the joint probability for a set of independent events is the product of N individual probabilities. The sample likelihood quantifies the joint probability of drawing this collection of N scores from a normal distribution with a mean of μ and a variance of σ^2.

The sample likelihood is:

$$L = \prod_{i=1}^{N} \left\{ \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-0.5(x_i-\mu)^2}{\sigma^2}} \right\}$$

where the braces contain the likelihood of a single score, and $\prod$ is the multiplication operator.

```
prod(univariate.likelihood(X1))
```

```
## [1] 7.785e-36
```

3.2.4.5 The Log Likelihood Because the sample likelihood is such a small number, it is difficult to work with and is prone to rounding error. Computing the natural logarithm of the individual likelihood values solves this problem and converts the likelihood to a more tractable metric. Sample log-likelihood is the sum of the individual log-likelihood values

$$\log(L) = \sum_{i=1}^{N} \log \left\{ \frac{1}{\sqrt{2\pi\sigma^2} e^{\frac{-0.5(x_i-\mu)^2}{\sigma^2}}} \right\}$$

```
# method 1:
sum(log(univariate.likelihood(X1)))
```

```
## [1] -80.841
```

Log identity:

$\log_b(xy) = \log_b(x) + \log_b(y)$

because $b^c \times b^d = b^{c+d}$

```
# method 2:
log(prod(univariate.likelihood(X1)))
```

```
## [1] -80.841
```

3.2.4.6 Estimating Unknown Parameters The estimation procedure is an iterative process that repeats the log-likelihood computations many times, each time with different values of the population parameters until it finds the estimates that maximize the log-likelihood and thus produce the best fit to the data that is the estimates that minimize the standardized distances between the scores and the mean (Enders 2010).

```
logLiM <-
  function(x, title = NULL){
    require(graphics)
    require(shape)

    mu <- mean(x, na.rm = TRUE)
    var <- var(x, na.rm = TRUE)

    # continuum x-values
    minx <- mu - 6*sqrt(var)
    maxx <- mu + 6*sqrt(var)
    mu <- seq(minx, maxx, by = .01)

    s.logL <- cbind(mu, rep(0, length(mu)))
    for (i in 1:length(mu)) {
      logL <- 0
      for (j in 1:length(x)) {
        logL <- logL + log(exp((-.5*(x[j] - mu[i])^2)/(var))/sqrt(2*pi*var))
        #print(logL)
      }
      s.logL[i,2] <- logL
      #print(s.logL[i,])
    }
```

```
plot(s.logL,
     yaxs = "i",
     xaxs = "i",
     ylim = c(min(s.logL[,2]), max(s.logL[,2]) + diff(range(s.logL[,2]))*.2),
     ylab = "log-likelihood",
     xlab = "mean",
     type = "l",
     lwd = 3,
     pch = 20,
     cex = .5,
     col = "darkgreen")

# title
if (is.null(title)) {
  title("Log-Likelihood of the Mean", col = "black")
}
else if (!is.null(title)) {
  title(title, col = "black")
}

# xlab
if (!is.null(colnames(x))) {
  xlab = colnames(x)
}
else if (is.null(colnames(x))) {
  xlab = "mean"
}

# tracing lines
x0 <- subset(s.logL, subset = s.logL[,2] == max(s.logL[,2]))[1]
y0 <- subset(s.logL, subset = s.logL[,2] == max(s.logL[,2]))[2]

Arrows(x0 = x0,
```

```
            y0 = y0,
            x1 = x0,
            y1 = min(s.logL[,2]),
            col = "black",
            arr.type = "triangle",
            arr.adj = 1,
            code = 2,
            lty = 3,
            lwd = 1)

    segments(x0 = x0,
             y0 = y0,
             x1 = minx,
             y1 = y0,
             col = "black",
             lty = 3,
             lwd = 1)

    # maximum point
    points(x0,
           y0,
           col = "white",
           bg = "darkgreen",
           pch = 21,
           cex = .6)

    text(x0,
         y0,
         pos = 3,
         round(x0, 4))
  }

logLiM(X1)
```

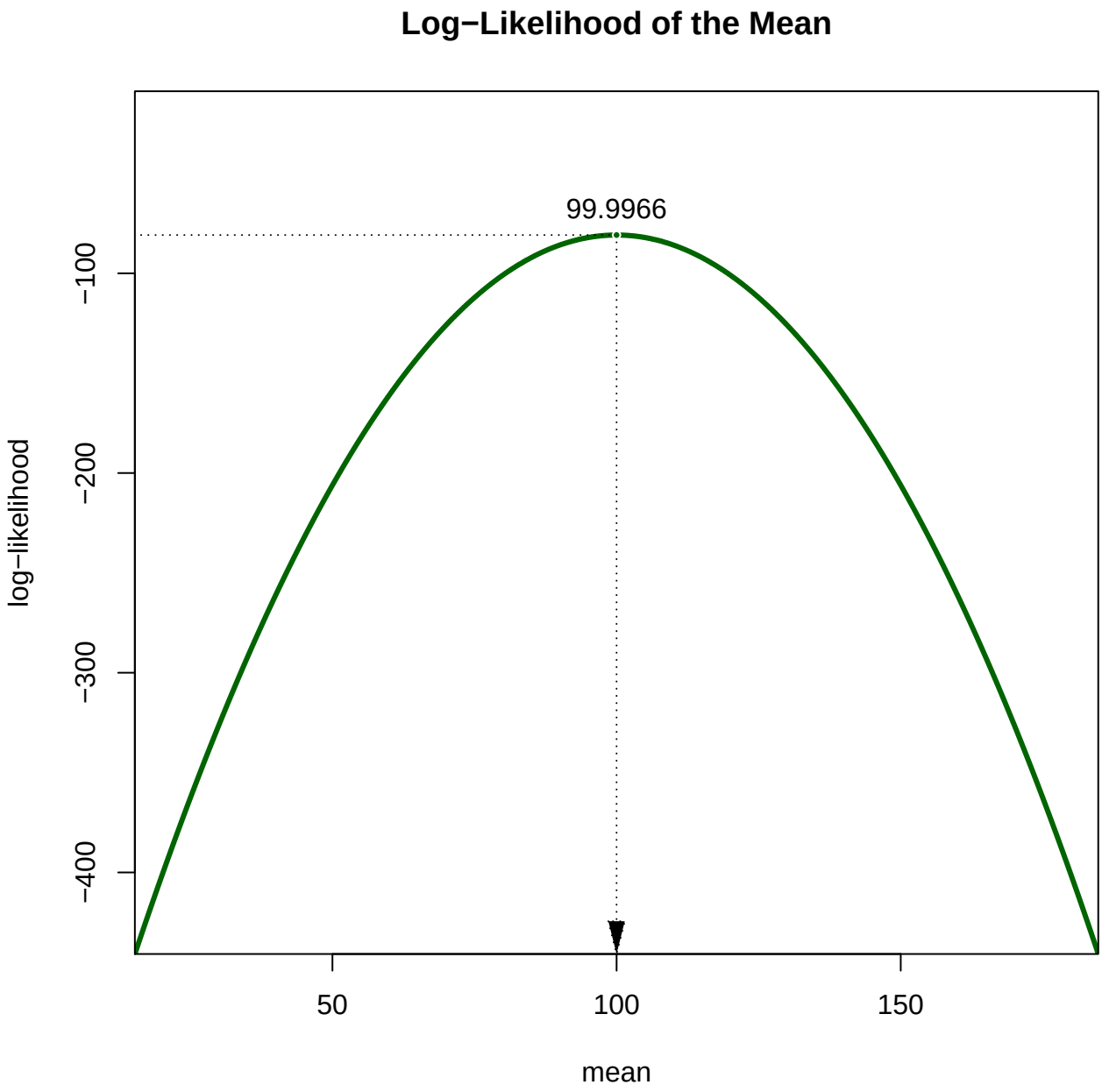
Log−Likelihood of the Mean
99.9966
log−likelihood
−100
−200
−300
−400
50
100
150
mean

Figure 5:

The sample mean is an unbiased estimator of the population mean.

```
logLiV <-
  function(x, title = NULL){
    require(graphics)
    require(shape)

    mu <- mean(x, na.rm = TRUE)
    var <- var(x, na.rm = TRUE)

    # continuum x-values
    minx <- var - .5*var
    maxx <- var + .5*var
    var <- seq(minx, maxx, by = .1)

    s.logL <- cbind(var, rep(0, length(var)))
    for (i in 1:length(var)) {
      logL <- 0
      for (j in 1:length(x)) {
        logL <- logL + log(exp((-.5*(x[j] - mu)^2)/(var[i]))/sqrt(2*pi*var[i]))
        #print(logL)
      }
      s.logL[i,2] <- logL
      #print(s.logL[i,])
    }

    # Plot
    #-----------------------------------------------------
    plot(s.logL,
         yaxs = "i",
         xaxs = "i",
         ylim = c(min(s.logL[,2]), max(s.logL[,2]) + diff(range(s.logL[,2]))*.2),
         ylab = "log-likelihood",
```

```
        xlab = "variance",
        type = "l",
        lwd = 3,
        pch = 20,
        cex = .5,
        col = "darkgreen")

  # title
  if (is.null(title)) {
    title("Log-Likelihood of the Variance", col = "black")
  }
  else if (!is.null(title)) {
    title(title, col = "black")
  }

  # xlab
  if (!is.null(colnames(x))) {
    xlab = colnames(x)
  }
  else if (is.null(colnames(x))) {
    xlab = "variance"
  }

  # tracing lines
  Arrows(x0 = subset(s.logL, subset = s.logL[,2] == max(s.logL[,2]))[1],
         y0 = subset(s.logL, subset = s.logL[,2] == max(s.logL[,2]))[2],
         x1 = subset(s.logL, subset = s.logL[,2] == max(s.logL[,2]))[1],
         y1 = min(s.logL[,2]),
         col = "black",
         arr.type = "triangle",
         arr.adj = 1,
         code = 2,
         lty = 3,
         lwd = 1)
```

```
    segments(x0 = subset(s.logL, subset = s.logL[,2] == max(s.logL[,2]))[1],
             y0 = subset(s.logL, subset = s.logL[,2] == max(s.logL[,2]))[2],
             x1 = minx,
             y1 = subset(s.logL, subset = s.logL[,2] == max(s.logL[,2]))[2],
             col = "black",
             lty = 3,
             lwd = 1)

    points(subset(s.logL, subset = s.logL[,2] == max(s.logL[,2]))[1],
           subset(s.logL, subset = s.logL[,2] == max(s.logL[,2]))[2],
           col = "white",
           bg = "darkgreen",
           pch = 21,
           cex = .6)

    text(subset(s.logL, subset = s.logL[,2] == max(s.logL[,2]))[1],
         subset(s.logL, subset = s.logL[,2] == max(s.logL[,2]))[2],
         pos = 3,
         round(subset(s.logL, subset = s.logL[,2] == max(s.logL[,2]))[1], 4))
  }

logLiV(X1, title = "Log-likelihood of the variance")
```

```
# 2 x 2 pictures on one plot
op <- par(mfrow = c(2, 2), oma = c(0, 0, 2, 0))

# variable names / plot titles
names <- colnames(A)

# produce likelihood plots for each variable in the data frame
for (i in 1:ncol(A)) {
```

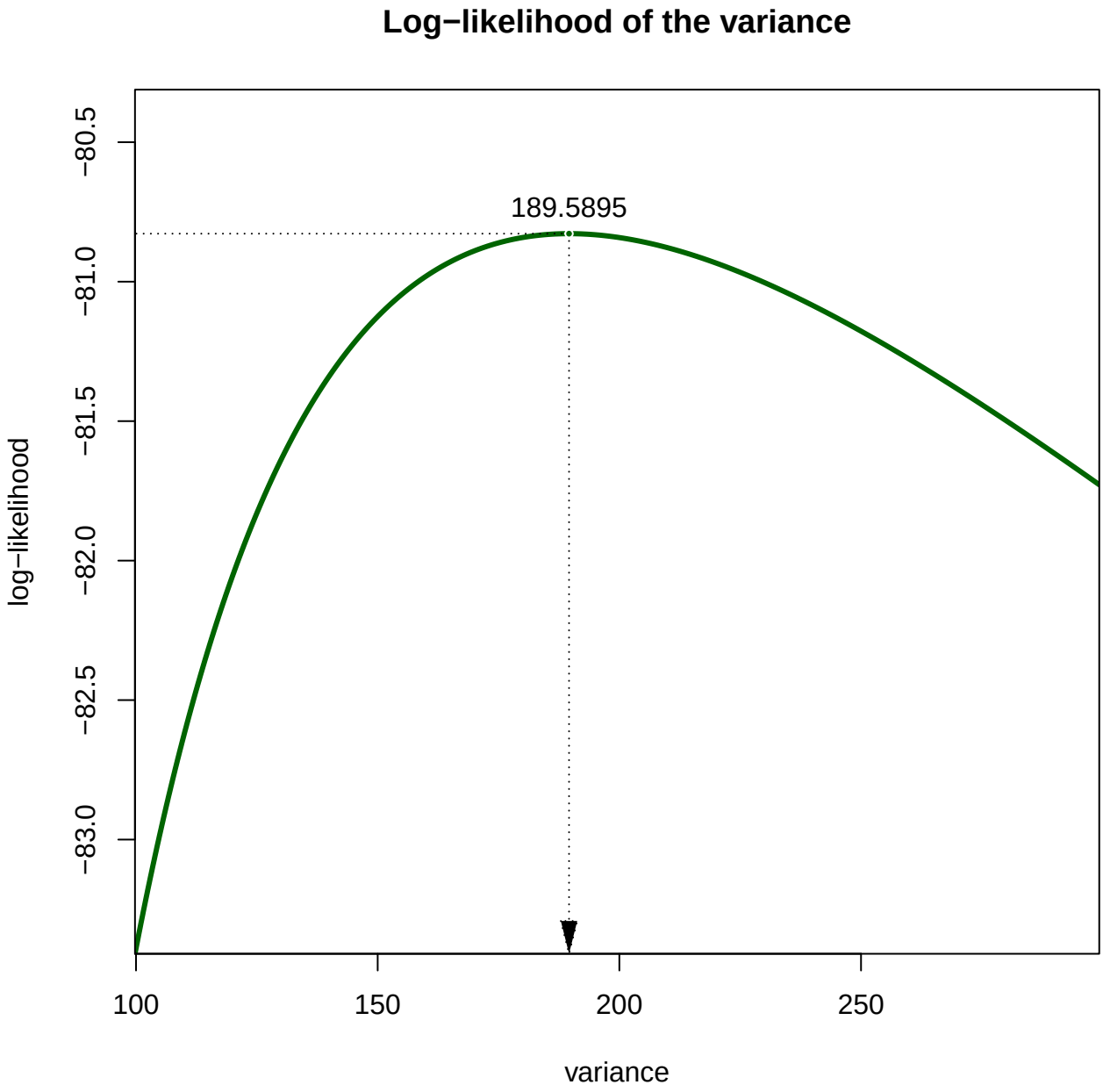
Log−likelihood of the variance
189.5895
log−likelihood
−80.5
−81.0
−81.5
−82.0
−82.5
−83.0
100
150
200
250
variance

Figure 6:

```
  logLiM(A[,i][!is.na(A[,i])], names[i])
  title(main = names[i], col.main = "black")
}

mtext("Likelihood Estimation of Means", cex = 1.5, outer = TRUE, col = "black")

## At end of plotting, reset to previous settings:
par(op)
```

```
# check:
# The sample mean is an unbiased estimator of the population mean.
colMeans(A, na.rm = TRUE)
```

3.2.4.7 Likelihood Estimation of Means

```
##       X       Y       Z
## 100.000  10.353  11.700
```

```
# 2 x 2 pictures on one plot
op <- par(mfrow = c(2, 2), oma = c(0, 0, 2, 0))

# variable names / plot titles
names <- colnames(A)

# produce likelihood plots for each variable in the data frame
for (i in 1:ncol(A)) {
  logLiV(A[,i][!is.na(A[,i])], names[i])
  title(main = names[i], col.main = "black")
}
```

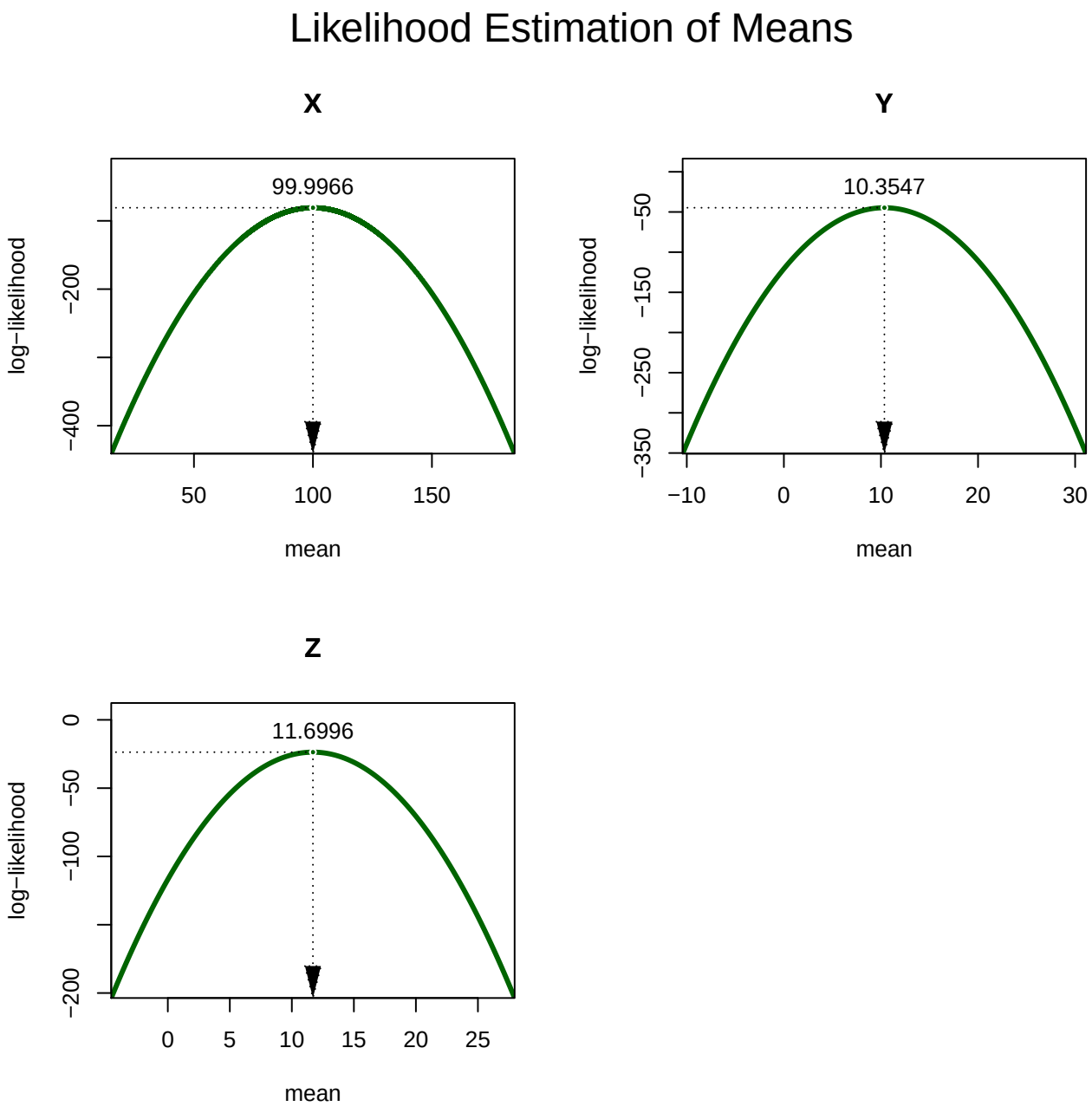
Likelihood Estimation of Means
X
99.9966
log−likelihood
−200
−400
50
100
150
mean
Y
10.3547
log−likelihood
−50
−150
−250
−350
−10
0
10
20
30
mean
Z
11.6996
log−likelihood
0
−50
−100
−200
0
5
10
15
20
25
mean

Figure 7:

```
mtext("Likelihood Estimation of Variances", cex = 1.5, outer = TRUE, col = "black")

## At end of plotting, reset to previous settings:
par(op)
```

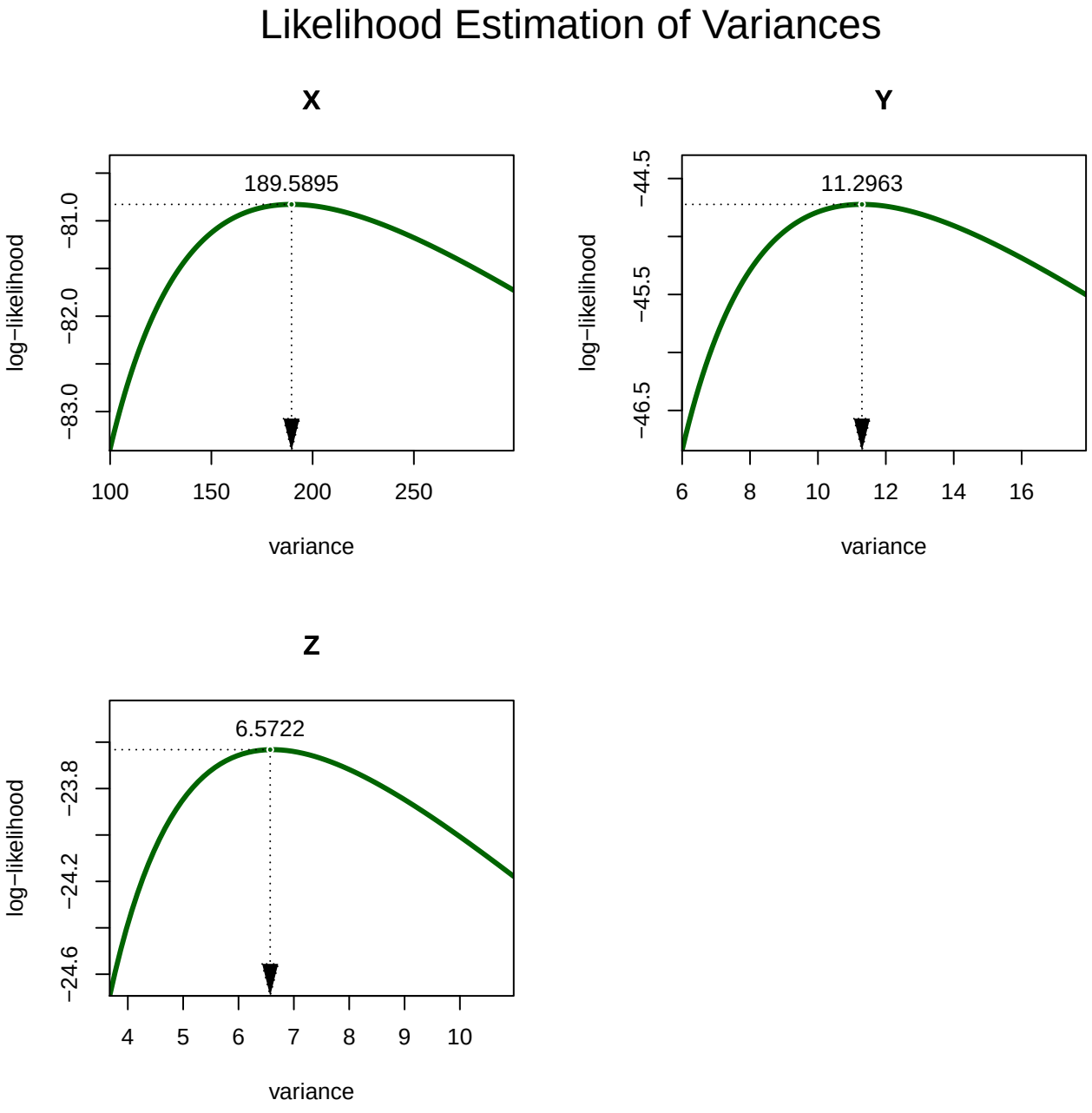

Figure 8:

```
# check:
# method 1 denominator (N-1) is used:
apply(A2, 2, FUN = function(x){x <- na.omit(x); sum((x - mean(x))^2)/(length(x))})
```

3.2.4.8 Likelihood Estimation of Variances

```
##        X        Y        Z
## 92.5432   6.0247   6.9877
```

3.2.4.9 Maximum Likelihood Estimation with Multivariate Data The probability density function defines the shape of the multivariate normal distribution

$$L_i = (2\pi)^{-\frac{k}{2}} \parallel \Sigma \parallel^{-\frac{1}{2}} e^{-\frac{1}{2}(x_i-\mu)'\Sigma^{-1}(x_i-\mu)}$$

where

k is the number of variables in X, so each individual has a set of k scores,

μ_j is the mean vector for the j^{th} variable, and

Σ is the covariance matrix

```
multivariate.likelihood <-
  function(A){
    source("MEANS.R")
    source("COV.R")
    source("INV.R")
    source("DET.R")

    A <- as.matrix(A)
    n <- nrow(A) # number of cases
    k <- ncol(A) # number of variables
    S <- COV(A) # variance-covariance matrix
    mu <- MEANS(A) # vetor of column means
    R <- apply(A, 2, function(x){x - mean(x)}) # residual scores
```

```
    # loglikelihood
    L <- rep(NA, n) # initiate
    for (i in 1:n) {
      L[i] <- (-0.5*k*log(2*pi) - 0.5*log(DET(S)) -
                  # The Mahalanobis distance:
                  0.5*t(R[i,]) %*% INV(S) %*% R[i,])
    }
    return(as.matrix(L))
  } # multivariate.likelihood

# save function
dump("multivariate.likelihood", file = "multivariate.likelihood.R")

Q <- data.frame(apply(mean.imputation(A2),
                      2,
                      function(x){replace(x, is.na(x), mean(x, na.rm = TRUE))}),
           round(multivariate.likelihood(mean.imputation(A2)), 6))
colnames(Q) <- c("X", "Y", "Z", "log.likelihood"); Q
```

```
##     X  Y  Z log.likelihood
## 1  99  6  7        -9.4182
## 2 105 12 10        -7.3676
## 3 105 14 11        -8.0015
## 4 106 10 15        -8.8112
## 5 112 10 10        -7.1365
## 6 113 14 12        -7.4360
## 7 115 14 14        -7.3452
## 8 118 12 16        -8.0760
## 9 134 11 12        -9.7446
```

3.2.4.10 The Missing Data Log-Likelihood With missing data, the log-likelihood for case i is

$$\log L_i = -\frac{k_i}{2}\log(2\pi) - \frac{1}{2}\log \parallel \Sigma_i \parallel -\frac{1}{2}(Y_i - \mu_i)^T \Sigma_i^{-1}(Y_i - \mu_i)$$

where

k_i is the number of complete data points for case i,

Y_i is the score vector for case i,

μ is the population mean vector, and

Σ is the population variance-covariance matrix.

The Mahalanobis distance value is a squared z-score that quantifies the standardized distance between each score and the center of the multivariate normal distribution. Small Mahalanobis distance produce large log-likelihood values, whereas large deviations yield small likelihood values (Enders 2010).

Example 1:

$$\log L_1 = -\frac{k_1}{2}\log(2\pi) - \frac{1}{2}\log\left\|\begin{matrix}\hat{\sigma}_1^2 & \hat{\sigma}_{1,2} \\ \hat{\sigma}_{2,1} & \hat{\sigma}_2^2\end{matrix}\right\| - \frac{1}{2}\left(\begin{bmatrix} y_{(1,1)} \\ y_{(1,2)}\end{bmatrix} - \begin{bmatrix}\hat{\mu}_1 \\ \hat{\mu}_2\end{bmatrix}\right)^T \begin{bmatrix}\hat{\sigma}_1^2 & \hat{\sigma}_{1,2} \\ \hat{\sigma}_{2,1} & \hat{\sigma}_2^2\end{bmatrix}^{-1} \left(\begin{bmatrix} y_{(1,1)} \\ y_{(1,2)}\end{bmatrix} - \begin{bmatrix}\hat{\mu}_1 \\ \hat{\mu}_2\end{bmatrix}\right)$$

$$= -\frac{2}{2}\log(2\pi) - \frac{1}{2}\log\left\|\begin{matrix}189.60 & 11.69 \\ 11.69 & 9.59\end{matrix}\right\| - \frac{1}{2}\left(\begin{bmatrix}78 \\ 13\end{bmatrix} - \begin{bmatrix}100.00 \\ 10.35\end{bmatrix}\right)^T \begin{bmatrix}189.60 & 11.69 \\ 11.69 & 9.59\end{bmatrix}^{-1} \left(\begin{bmatrix}78 \\ 13\end{bmatrix} - \begin{bmatrix}100.0 \\ 10.3\end{bmatrix}\right.$$

$$= -6.64$$

Example 2:

$$\log(L_5) = -\frac{k_5}{2}\log(2\pi) - \frac{1}{2}\log \parallel \hat{\sigma}_1^2 \parallel -\frac{1}{2}(y_{(5,1)} - \hat{\mu}_1)^T(\hat{\sigma}_1^2)^{-1}(y_{(5,1)} - \hat{\mu}_1)$$

$$= -\frac{1}{2}\log(2\pi) - \frac{1}{2}\log \parallel 189.6 \parallel -\frac{1}{2}(87.0 - 100.0)^T(189.6)^{-1}(87.0 - 100)$$

$$= -3.99$$

```
observed.likelihood <-
  function(x){
    source("MEANS.R")
    source("SIGMA.R")
    source("INV.R")
    source("DET.R")

    # remove cases that are all NAs
    x <- as.matrix(x[apply(x, 1, function(x){!all(is.na(x))}),])
    n <- nrow(x)

    # initiate likelelihood vector
    L <- matrix(NA, n, 1)
    index <- 1

    for (i in 1:n) {
      # data
      y <- as.matrix(x[i,][!is.na(x[i,])])
      z <- subset(x, select = !is.na(x[i,]))
      k <- nrow(y) # number of variables
      p <- ncol(y) # number of cases

      # variable means
      M <- MEANS(z)
```

```
      # variance-covariance matrix
      S <- SIGMA(z)

      # Constants
      D <- log(DET(S))
      L2P <- log(2*pi)
      I <- INV(S)

      # compute log-likelelihood
      for (j in 1:p) {
        # Residuals
        R <- as.matrix(y[,j] - M)

        # Log-likelihood
        L[index] <- -(k/2) * L2P - (1/2) * D - (1/2) * t(R) %*% I %*% R

        index <- index + 1
      }
    }
    return(L)
  } # observed.likelihood

# save function
dump("observed.likelihood", file = "observed.likelihood.R")

W <- data.frame(A1, round(observed.likelihood(A), 5))
colnames(W) <- c("X", "Y", "Z", "log.likelihood"); W
```

```
##      X  Y  Z log.likelihood
## 1   78 13 NA        -6.6399
## 2   84  9 NA        -5.1430
## 3   84 10 NA        -5.1581
## 4   85 10 NA        -5.0721
## 5   87 NA NA        -3.9871
```

```
## 6   91  3 NA        -7.2780
## 7   92 12 NA        -4.8873
## 8   94  3 NA        -7.3030
## 9   94 13 NA        -5.0682
## 10  96 NA NA        -3.5836
## 11  99  6  7       -10.8330
## 12 105 12 10        -7.8715
## 13 105 14 11        -7.9973
## 14 106 10 15        -8.8951
## 15 108 NA 10        -5.2125
## 16 112 10 10        -8.0551
## 17 113 14 12        -7.9548
## 18 115 14 14        -8.4081
## 19 118 12 16       -10.2160
## 20 134 11 12       -10.2164
```

Smaller Mahalanobis distance values (i.e., smaller squared z scores) produce larger likelihood values, whereas larger Mahalanobis distance values yield smaller likelihoods. Consequently, a score that yields a high likelihood value also has a good fit because it falls close to the population mean.

3.2.5 The Expectation Maximization (EM) Algorithm

The EM algorithm is a two-step iterative procedure that consists of an E-step and an M-step (E and M stand for expectation and maximization, respectively). The iterative process starts with an initial estimate of the mean vector and the covariance matrix. The E-step uses the elements in the mean vector and the covariance matrix to build a set of regression equations that predict the incomplete variables from the observed variables.

3.2.5.1 Bivariate EM The ultimate goal of the E-step is to fill in the missing components of ΣY, ΣY^2, and ΣXY. Specifically, the predicted values fill in the missing components of ΣY and ΣXY, and $\hat{Y}_i^2 + \hat{\sigma}_{Y|X}^2$ replaces the missing parts of ΣY^2 so that the M-step can use Equations

$$\hat{\mu}_Y = \frac{1}{N} \sum Y$$

$$\hat{\sigma}_Y^2 = \frac{1}{N} \left(\sum Y^2 - \frac{(\Sigma Y)^2}{N} \right)$$

$$\hat{\sigma}_{XY} = \frac{1}{N} \left(\sum XY - \frac{\Sigma X \Sigma Y}{N} \right)$$

More accurately, the E-step fills in each case's contribution to the sufficient statistics (Dempster, Laird, and Rubin 1977). The E-step uses the elements in the mean vector and the covariance matrix to build a set of regression equations that predict the incomplete variables from the observed variables. In a bivariate data set with missing value on Y, the necessary equations are

$$\hat{\beta}_1 = \frac{\hat{\sigma}_{X,Y}}{\hat{\sigma}_X^2}$$

$$\hat{\beta}_0 = \hat{\mu}_Y - \hat{\beta}_1 \hat{\mu}_X$$

$$\hat{\sigma}_{Y|X}^2 = \hat{\sigma}_Y^2 - \hat{\beta}_1^2 \hat{\sigma}_X^2$$

$$\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_i$$

```
EM <-
  function(x){ # Arguments: x = a data matrix
    source("is.identical.R")
    source("COV.R")

    cycle <- 500 # limit number of iterations
    significant.digits <- 4

    # remove cases that are all NAs and ensure proper data matrix
    x <- data.matrix(x[apply(x, 1, function(x){!all(is.na(x))}),])

    # Replace the missing values with their respective variable means
    data <- mean.imputation(x)
    n <- nrow(data)

    # compare covariance matrices between each consecutive EM cycle for optimal soluti
    Sigma1 <- round(COV(data), significant.digits)
    Sigma2 <- matrix(0, nrow = nrow(Sigma1), ncol = ncol(Sigma1))

    Y2 <- x[,2]^2

    counter <- 1 # to keep track of number of EM cycles
    while (!is.identical(Sigma1, Sigma2)){

      # Sufficient Statistics
      sumX <- sum(data[,1], na.rm = TRUE)
      sumY <- sum(data[,2], na.rm = TRUE)
      sumX2 <- sum(data[,1]^2, na.rm = TRUE)
      sumY2 <- sum(Y2)
      sumXY <- sum(data[,1] * data[,2], na.rm = TRUE)

      ### E-step
```

```
# means
meanX <- mean(data[,1], na.rm = TRUE)
meanY <- mean(data[,2], na.rm = TRUE)

# variances
n1 <- length(data[,1])
n2 <- length(data[,2])

varX <- (sumX2 - ((sumX^2)/n1)) / n1
varY <- (sumY2 - ((sumY^2)/n2)) / n2
covXY <- (sumXY - (sumX*sumY / min(n1, n2))) / min(n1, n2)

### M-step

# slope coefficient
B1 <- covXY / varX

# intercept coefficient
B0 <- meanY - B1 * meanX

# residual variance from the regression of Y on X
SigmaYX <- varY - B1^2 * varX

# predicted Y score for a given value of X
Y <- B0 + B1 * data[,1]

# impute missing values
for (g in 1:nrow(data)) {
  if (is.na(x[g, 2])) {
    data[g, 2] <- Y[g]
  }
}
```

```
      # compute Y-squared
      for (s in 1:length(Y2)) {
        if (is.na(x[s,2])) {
          Y2[s] <- data[s,2]^2 + SigmaYX
        }
      }

      # check for optimum solution by comparing the change in
      # consecutive computations of the covariance matrix
      Sigma2 <- Sigma1
      Sigma1 <- round(cov(data, use = "everything"), significant.digits)

      counter <- counter + 1 # number of EM cycles
      if (counter > cycle) {break} # set maximum number of cycles
    }
    # print(sprintf(cbind(meanY, varY, covXY), fmt = '%.3f'))
    # print(sum(Li(z)))
    return(list(convergence = paste(counter, "cycles", sep = " "),
                imputations = round(data, 3)))
  } # EM

# save function
dump("EM", file = "EM.R")

EM1 <- EM(A[,1:2]); EM1
```

```
## $convergence
## [1] "7 cycles"
##
## $imputations
##         X      Y
##  [1,]  78 13.000
##  [2,]  84  9.000
##  [3,]  84 10.000
```

```
##  [4,]  85 10.000
##  [5,]  87  9.459
##  [6,]  91  3.000
##  [7,]  92 12.000
##  [8,]  94  3.000
##  [9,]  94 13.000
## [10,]  96 10.054
## [11,]  99  6.000
## [12,] 105 12.000
## [13,] 105 14.000
## [14,] 106 10.000
## [15,] 108 10.847
## [16,] 112 10.000
## [17,] 113 14.000
## [18,] 115 14.000
## [19,] 118 12.000
## [20,] 134 11.000
```

```
MEANS(EM1[[2]])
```

```
##      [,1]
## X 100.000
## Y  10.318
```

```
SIGMA(EM1[[2]])
```

```
##        X       Y
## X 189.60 12.5296
## Y  12.53  9.6495
```

```
EM2 <- EM(A[,c(1, 3)]); EM2
```

```
## $convergence
## [1] "86 cycles"
```

```
##
## $imputations
##         X      Z
##  [1,]  78  7.565
##  [2,]  84  8.305
##  [3,]  84  8.305
##  [4,]  85  8.429
##  [5,]  87  8.676
##  [6,]  91  9.169
##  [7,]  92  9.293
##  [8,]  94  9.540
##  [9,]  94  9.540
## [10,]  96  9.787
## [11,]  99  7.000
## [12,] 105 10.000
## [13,] 105 11.000
## [14,] 106 15.000
## [15,] 108 10.000
## [16,] 112 10.000
## [17,] 113 12.000
## [18,] 115 14.000
## [19,] 118 16.000
## [20,] 134 12.000
```

```
MEANS(EM2[[2]])
```

```
##     [,1]
## X 100.00
## Z  10.28
```

```
SIGMA(EM2[[2]])
```

```
##         X       Z
## X 189.600 23.4047
## Z  23.405  5.5491
```

3.2.5.2 Applying EM to Multivariate Data

3.2.5.3 Complete-data Sufficient Statistics

$$T = \begin{bmatrix} n & Y^T \\ Y & Y^TY \end{bmatrix} = \begin{bmatrix} n & \sum y_{i1} & \sum y_{i2} & \cdots & \sum y_{ip} \\ & \sum y_{i1}^2 & \sum y_{i1}y_{i2} & \cdots & \sum y_{i1}y_{ip} \\ & & \sum y_{i2}^2 & \cdots & \sum y_{i2}y_{ip} \\ & & & \ddots & \vdots \\ & & & & \sum y_{ip}^2 \end{bmatrix}$$

```
T <-
  function(A){
    # remove cases that are all NAs and ensure proper data matrix
    A <- data.matrix(A[apply(A, 1, function(x){!all(is.na(x))}),])

    # Missing values imputed by variable means
    A <- apply(A, 2, function(x){replace(x, is.na(x), mean(x, na.rm = TRUE))})
    n <- nrow(A) # number of cases

    Y <- cbind(rep(1, n), A)
    return(t(Y) %*% Y)
  } # T

# save function
dump("T", file = "T.R")

T(B1)
```

```
##        [,1]  [,2]  [,3]  [,4]  [,5]    [,6]
## [1,]   13.0    97   626   153   390  1240.5
## [2,]   97.0  1139  4922   769  2620 10032.0
```

```
## [3,]  626.0  4922 33050  7201 15739  62027.8
## [4,]  153.0   769  7201  2293  4628  13981.5
## [5,]  390.0  2620 15739  4628 15062  34733.3
## [6,] 1240.5 10032 62028 13982 34733 121088.1
```

3.2.5.4 The Expectation Step, Θ

$$\Theta = \begin{bmatrix} -1 & \mu^T \\ \mu & \Sigma \end{bmatrix} = \begin{bmatrix} -1 & \mu_1^T & \mu_2^T & \dots & \mu_p^T \\ \mu_1 & \Sigma_{11} & \Sigma_{12} & \dots & \Sigma_{1p} \\ \mu_2 & \Sigma_{21} & \Sigma_{22} & \dots & \Sigma_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mu_p & \Sigma_{p1} & \Sigma_{p2} & \dots & \Sigma_{pp} \end{bmatrix}$$

```
THETA <-
  function(A){
    source("T.R")
    source("SWP.R")

    # remove cases that are all NAs and ensure proper data matrix
    A <- data.matrix(A[apply(A, 1, function(x){!all(is.na(x))}),])

    # Missing values imputed by variable means
    A <- apply(A, 2, function(x){replace(x, is.na(x), mean(x, na.rm = TRUE))})
    n <- nrow(A) # number of cases

    return(SWP(T(A)/n, 1))
  } # THETA

# save function
dump("THETA", file = "THETA.R")
```

```
THETA(B1)
```

```
##          [,1]     [,2]     [,3]     [,4]      [,5]     [,6]
## [1,] -1.0000   7.4615   48.154  11.7692   30.0000   95.423
## [2,]  7.4615  31.9408   19.314 -28.6627  -22.3077   59.689
## [3,] 48.1538  19.3136  223.515 -12.8107 -233.9231  176.381
## [4,] 11.7692 -28.6627  -12.811  37.8698    2.9231  -47.556
## [5,] 30.0000 -22.3077 -233.923   2.9231  258.6154 -190.900
## [6,] 95.4231  59.6893  176.381 -47.5562 -190.9000  208.905
```

```
# Augmented parameter matrix
U <- SWP(T(B1), 1)
round(U, 4)
```

```
##          [,1]      [,2]      [,3]     [,4]    [,5]      [,6]
## [1,] -0.0769    7.4615    48.154   11.769    30.0    95.423
## [2,]  7.4615  415.2308   251.077 -372.615  -290.0   775.962
## [3,] 48.1538  251.0769  2905.692 -166.538 -3041.0  2292.954
## [4,] 11.7692 -372.6154  -166.538  492.308    38.0  -618.231
## [5,] 30.0000 -290.0000 -3041.000   38.000  3362.0 -2481.700
## [6,] 95.4231  775.9615  2292.954 -618.231 -2481.7  2715.763
```

```
# Components:
round(CSSCP(B1), 2) # U[2:r, 2:r]
```

```
##          [,1]     [,2]    [,3]    [,4]     [,5]
## [1,]  415.23   251.08 -372.62  -290.0   775.96
## [2,]  251.08  2905.69 -166.54 -3041.0  2292.95
## [3,] -372.62  -166.54  492.31    38.0  -618.23
## [4,] -290.00 -3041.00   38.00  3362.0 -2481.70
## [5,]  775.96  2292.95 -618.23 -2481.7  2715.76
```

```r
round(MEANS(B1), 2) # U[1, 2:r] & U[2:r, 1]
```

```
##       [,1]
## [1,]  7.46
## [2,] 48.15
## [3,] 11.77
## [4,] 30.00
## [5,] 95.42
```

```r
round(-1/nrow(B1), 2) # U[1,1]
```

```
## [1] -0.08
```

References

Afifi, A. A., and R. M. Elashoff. 1966. "Missing Observations in Multivariate Statistics: I. Review of the Literature." *Journal of the American Statistical Association* 61 (315): 595–604. doi:10.2307/2282773.

Anderson, T. W. 1957. "Maximum Likelihood Estimates for a Multivariate Normal Distribution When Some Observations Are Missing." *Journal of the American Statistical Association* 52 (278): 200–203. doi:10.2307/2280845.

Andridge, R. R., and R. J. A. Little. 2010. "A Review of Hot Deck Imputation for Survey Non-Response." *International Statistical Review* 78 (1): 40–64. doi:http://doi.org/10.1111/j.1751-5823.2010.00103.x.

Beale, E. M. L., and R. J. A. Little. 1975. "Missing Values in Multivariate Analysis." *Journal of the Royal Statistical Society. Series B (Methodological)* 37 (1): 129–45.

Bodner, T. E. 2006. "Missing Data: Prevalence and Reporting Practices." *Psychological Reports* 99 (3): 675–80. doi:doi: 10.2466/PR0.99.3.675-680.

Cochran, W. G. 1977. *Sampling Techniques*. Third Edition. Wiley Series in Probability and Mathematical Statistics: Applied Probability and Statistics. New York, NY: John Wiley; Sons. doi:10.1002/bimj.19650070312.

Dempster, A. P., N. M. Laird, and D. B. Rubin. 1977. "Maximum Likelihood from Incomplete Data via the EM Algorithm." *Journal of the Royal Statistical Society. Series B (Methodological)* 39 (1): 1–38.

Enders, C. K. 2010. *Applied Missing Data Analysis*. Methodology in the Social Sciences. New York, NY: The Guilford Press.

Goodnight, J. H. 1979. "A Tutorial on the SWEEP Operator." *The American Statistician* 33 (3): 149–59.

Hartley, H. O. 1956. "A Plan for Programming Analysis of Variance for General Purpose Computers." *Biometrics* 12 (2): 110–22. doi:10.2307/3001755.

Hartley, H. O., and R. R. Hocking. 1971. "The Analysis of Incomplete Data." *Biometrics* 27 (4): 783–823. doi:10.2307/2528820.

Healy, M. J. R., and M. Westmacott. 1956. "Missing Values in Experiments Analysed on Automatic Computers." *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 5 (3): 203–6. doi:10.2307/2985421.

Hinkelmann, K., and O. Kempthorne. 2007. *Design and Analysis of Experiments*. Second Edition. Vol. 1: Introduction to Experimental Design. Wiley Series in Probability and Statistics. New York, NY: John Wiley; Sons. doi:10.1002/9780470191750.

Hocking, R. R., and W. B. Smith. 1968. "Estimation of Parameters in the Multivariate Normal Distribution with Missing Observations." *Journal of the American Statistical Association* 63 (321): 159–73. doi:10.2307/2283837.

Little, R. J. A. 1988. "A Test of Missing Completely at Random for Multivariate Data with Missing Values." *Journal of the American Statistical Association* 83 (404): 1198–1202. doi:10.2307/2290157.

Little, R. J. A., and D. B. Rubin. 2002. *Statistical Analysis with Missing Data*. Hoboken, N.J: Wiley.

Orchard, T., and M. A. Woodbury. 1972. *A Missing Information Principle: Theory and Applications. Theory of Statistics*. Vol. Volume 1. Berkeley, CA: University of California Press; Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics; Probability.

Peugh, J. L., and C. K. Enders. 2004. "Missing Data in Educational Research: A Review of Reporting Practices and Suggestions for Improvement." *Review of Educational Research* 74 (4): 525–56. doi:10.3102/00346543074004525.

Rubin, D. B. 1972. "A Non-Iterative Algorithm for Least Squares Estimation of Missing Values in Any Analysis of Variance Design." *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 21 (2): 136–41. doi:10.2307/2346485.

———. 1976. "Inference and Missing Data." *Biometrika* 63 (3): 581–92. doi:10.2307/2335739.

———. 1978. "Bayesian Inference for Causal Effects: The Role of Randomization." *The Annals of Statistics* 6 (1): 34–58.

———. 1987. *Multiple Imputation for Nonresponse in Surveys*. Wiley Series in Probability and Statistics. Hoboken, NJ: John Wiley; Sons. doi:10.1002/9780470316696.

Schafer, J. L. 1997. *Analysis of Incomplete Multivariate Data*. Chapman and Hall/CRC Monographs on Statistics and Applied Probability. London, UK: CRC Press.

Schafer, J. L., and J. W. Graham. 2002. "Missing Data: Our View of the State of the Art." *Psychological Methods* 7 (2): 147–77. doi:10.1037/1082-989X.7.2.147.

Wilkinson, G. N. 1958. "Estimation of Missing Values for the Analysis of Incomplete Data." *Biometrics* 14 (2): 257–86. doi:10.2307/2527789.

Wilks, S. S. 1932. “Moments and Distributions of Estimates of Population Parameters from Fragmentary Samples.” *The Annals of Mathematical Statistics* 3 (3): 163–95.