Chapter 1

Regression and Correlation

The examples in this chapter emphasize the use of matrices for statistical calculations.

1.1 Linear regression

Certain statistical models are most naturally represented using matrix notation. Fitting such models is simplified and more efficient when the model is expressed in matrix form. To illustrate, consider the standard multiple regression model

$$y_i \;=\; \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i, \quad i = 1, \ldots, n, \tag{1.1}$$

Where y_i is the response for observation i, $x_{i1}, \ldots, x_{ip}$ are fixed predictors for observation i, and $\beta_0, \beta_1, \ldots, \beta_p$ are unknown regression parameters. It is common to assume $\varepsilon \overset{\text{ind}}{\sim} \text{Normal}(0, \sigma^2)$. In matrix notation, (1.1) can

be rewritten as

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$
$$\underset{\sim}{y} = \mathbf{X}\underset{\sim}{\beta} + \underset{\sim}{\varepsilon},$$

where $\underset{\sim}{y}$ is the n-by-1 response vector, $\mathbf{X}$ is the n-by-$(p+1)$ design matrix, $\underset{\sim}{\beta}$ is the $(p+1)$-by-1 regression parameter vector, and $\underset{\sim}{\varepsilon}$ is the n-by-1 residual vector.

The least squares (LS) estimate of $\underset{\sim}{\beta}$, say

$$\underset{\sim}{\hat{\beta}} = \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_p \end{bmatrix},$$

minimizes

$$\begin{aligned} \text{SSE}(\underset{\sim}{\beta}) &= \sum_{i=1}^{n} \{y_i - (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip})\}^2 \\ &= (\underset{\sim}{y} - \mathbf{X}\underset{\sim}{\beta})^\top (\underset{\sim}{y} - \mathbf{X}\underset{\sim}{\beta}). \end{aligned}$$

That is, $\underset{\sim}{\hat{\beta}}$ minimizes the squared length of $(\underset{\sim}{y} - \mathbf{X}\underset{\sim}{\beta})^\top (\underset{\sim}{y} - \mathbf{X}\underset{\sim}{\beta})$. Assuming the columns of $\mathbf{X}$ are linearly independent, one can show that

$$\underset{\sim}{\hat{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \underset{\sim}{y}.$$

Note that, computationally, it is better to solve $(\mathbf{X}^\top \mathbf{X})\underset{\sim}{\hat{\beta}} = \mathbf{X}^\top \underset{\sim}{y}$ to avoid computing the inverse of $(\mathbf{X}^\top \mathbf{X})$.

Additional summaries The **expected value** of each response is given by

$$\begin{aligned} \mathrm{E}[y_i] \equiv \mu_i &= \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}, \quad i = 1, \ldots, n \\ \mathrm{E}[\underset{\sim}{y}] \equiv \underset{\sim}{\mu} &= \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix} = \mathbf{X}\underset{\sim}{\beta}. \end{aligned}$$

These are estimated by

$$\begin{aligned} \hat{\mu}_i &= \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip}, \quad i = 1, \ldots, n \\ \underset{\sim}{\hat{\mu}} &= \begin{bmatrix} \hat{\mu}_1 \\ \hat{\mu}_2 \\ \vdots \\ \hat{\mu}_n \end{bmatrix} = \mathbf{X}\underset{\sim}{\hat{\beta}}. \end{aligned}$$

The **observed residuals** are

$$\begin{aligned} e_i &= y_i - \hat{\mu}_i \\ &= y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip}), \quad i = 1, \ldots, n, \end{aligned}$$

and can be represented as

$$\begin{aligned} \underset{\sim}{e} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} &= \begin{bmatrix} y_1 - \hat{\mu}_1 \\ y_2 - \hat{\mu}_2 \\ \vdots \\ y_n - \hat{\mu}_n \end{bmatrix} \\ &= \underset{\sim}{y} - \underset{\sim}{\hat{\mu}} \\ &= \underset{\sim}{y} - \mathbf{X}\underset{\sim}{\hat{\beta}}. \end{aligned}$$

The **residual sum of squares (SS)** can be represented in many equivalent forms,

$$\begin{aligned}
\mathrm{SSE}(\hat{\underset{\sim}{\beta}}) &= \sum_{i=1}^{n}\{y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip})\}^2 \\
&= \sum_{i=1}^{n}\{y_i - \hat{\mu}_i\}^2 \\
&= \sum_{i=1}^{n} e_i^2 \\
&= \underset{\sim}{e}^\top \underset{\sim}{e} \\
&= (\underset{\sim}{y} - \hat{\underset{\sim}{\mu}})^\top (\underset{\sim}{y} - \hat{\underset{\sim}{\mu}}) \\
&= (\underset{\sim}{y} - \mathbf{X}\hat{\underset{\sim}{\beta}})^\top (\underset{\sim}{y} - \mathbf{X}\hat{\underset{\sim}{\beta}}).
\end{aligned}$$

Code for computing these summaries (not necessarily in the most numerically sound way) are given here.

Example: Cheddar cheese taste As cheese[1] ages, various chemical processes take place that determine the taste of the final product. The taste of matured cheese is related to the concentration of several chemicals in the final product. In a study of cheddar cheese from the LaTrobe Valley of Victoria, Australia, samples of cheese were analyzed for their chemical composition and were subjected to taste tests. Overall taste scores were obtained by combining the scores from several tasters. The variables "Acetic" and "H2S" are the natural logarithm of the concentration of acetic asid and hydrogen sulfide, respectively. The variable "Lactic" has not been transformed.

[1]The Data and Story Library (DASL, pronounced "dazzle") is an online library of datafiles and stories that illustrate the use of basic statistics methods. The Cheese example is described here with the data `http://lib.stat.cmu.edu/DASL/Datafiles/Cheese.html`.

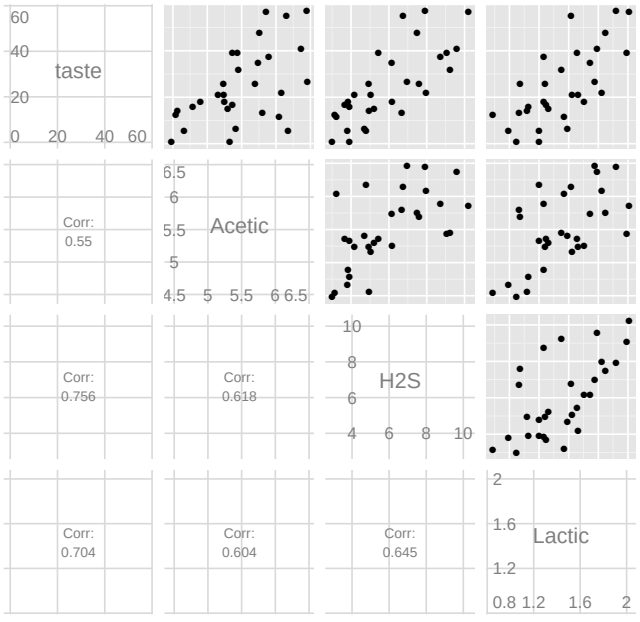
Start with a scatterplot of the data.

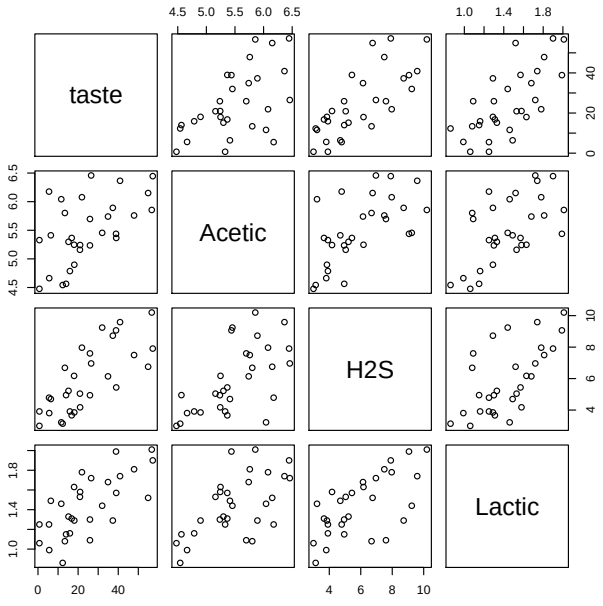
```
# read dataset from online
cheese <- read.csv("http://statacumen.com/teach/SC1/SC1_03_cheese.csv")
# structure of cheese data.frame
str(cheese)


## 'data.frame': 30 obs. of  4 variables:
##  $ taste : num  12.3 20.9 39 47.9 5.6 25.9 37.3 21.9 18.1 21 ...
##  $ Acetic: num  4.54 5.16 5.37 5.76 4.66 ...
##  $ H2S   : num  3.13 5.04 5.44 7.5 3.81 ...
##  $ Lactic: num  0.86 1.53 1.57 1.81 0.99 1.09 1.29 1.78 1.29 1.58 ...

# Plot the data using ggplot with GGally
library(ggplot2)
library(GGally)
p1 <- ggpairs(cheese)
# put scatterplots on top so y axis is vertical
p1 <- ggpairs(cheese, upper = list(continuous = "points")
                    , lower = list(continuous = "cor")
              )
print(p1)

# R base graphics
pairs(cheese)
```





Perform the calculation of the regression model.

```
# assign response variable
y <- as.matrix(cheese$taste)
X <- as.matrix(cheese[, c("Acetic", "H2S", "Lactic")])

n <- nrow(X) # sample size
n

## [1] 30

p <- ncol(X) # number of predictors
p

## [1] 3


# create design matrix, append columns of 1s to left side of X matrix
X.int <- cbind(matrix(rep(1, n), ncol=1), X)
head(X.int, 3) # print the first 3 rows to show the design matrix

##        Acetic   H2S Lactic
## [1,] 1  4.543 3.135   0.86
## [2,] 1  5.159 5.043   1.53
## [3,] 1  5.366 5.438   1.57

colnames(X.int)[1] <- "Intercept" # name the intercept column of 1s
head(X.int, 3) # print the first 3 rows to show the design matrix

##      Intercept Acetic   H2S Lactic
## [1,]         1  4.543 3.135   0.86
## [2,]         1  5.159 5.043   1.53
## [3,]         1  5.366 5.438   1.57


# Regression summaries
# LS estimate, "solve" computes a matrix inverse
beta.hat <- solve( t(X.int) %*% X.int ) %*% t(X.int) %*% y
beta.hat

##               [,1]
## Intercept -28.8768
## Acetic      0.3277
## H2S         3.9118
## Lactic     19.6705
```

```
# fitted values
y.hat <- X.int %*% beta.hat
# residuals
e.hat <- y - y.hat
```

Therefore, the fitted regression equation[2] is

$$\begin{aligned}\hat{\mu} &= \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \hat{\beta}_3 x_3 \\ &= -28.88 + 0.3277 x_1 + 3.912 x_2 + 19.67 x_3 \\ &= -28.88 + 0.3277 \text{ Acetic} + 3.912 \text{ H2S} + 19.67 \text{ Lactic}.\end{aligned}$$

Create a residual plot versus the fitted values.

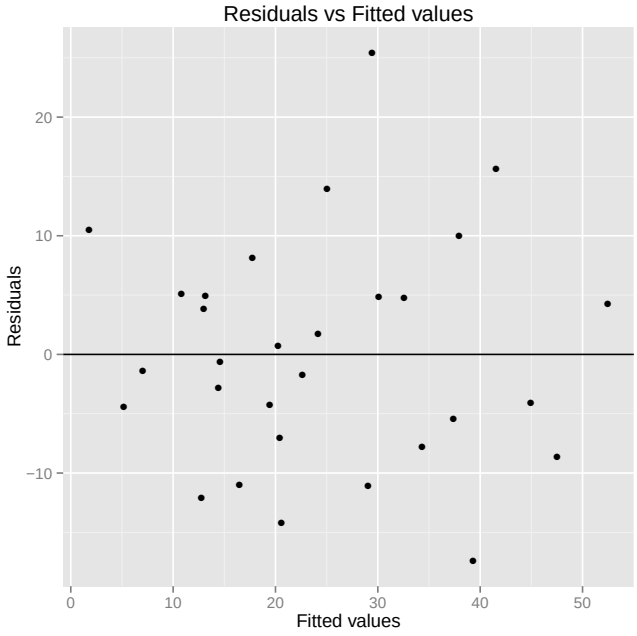
```
library(ggplot2)
# first put fitted values and residuals into a data.frame
resid_df <- data.frame(y.hat, e.hat)
p <- ggplot(resid_df, aes(x = y.hat, y = e.hat))
p <- p + geom_hline(aes(yintercept=0), colour="black")
p <- p + geom_point()
p <- p + labs(title = "Residuals vs Fitted values")
p <- p + xlab("Fitted values")
```

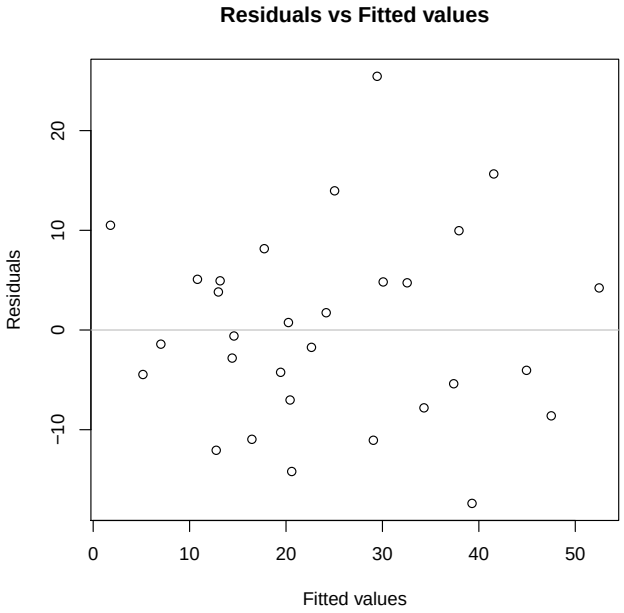
[2]This was typeset by drawing the coefficients and variable names from the data and results:

```
Therefore, the fitted regression equation is
 %===============
\begin{eqnarray}
  \hat{\mu}
    & = &
  \hat{beta}_{0} +
  \hat{beta}_{1} x_{1} +
  \hat{beta}_{2} x_{2} +
  \hat{beta}_{3} x_{3}
\nonumber\\ %===
    & = &
  \Sexpr{signif(beta.hat[1+0],4)} +
  \Sexpr{signif(beta.hat[1+1],4)} x_{1} +
  \Sexpr{signif(beta.hat[1+2],4)} x_{2} +
  \Sexpr{signif(beta.hat[1+3],4)} x_{3}
\nonumber\\ %===
    & = &
  \Sexpr{signif(beta.hat[1+0],4)} +
  \Sexpr{signif(beta.hat[1+1],4)} \textrm{ \Sexpr{colnames(X.int)[1+1]}} +
  \Sexpr{signif(beta.hat[1+2],4)} \textrm{ \Sexpr{colnames(X.int)[1+2]}} +
  \Sexpr{signif(beta.hat[1+3],4)} \textrm{ \Sexpr{colnames(X.int)[1+3]}}
.
\nonumber
\end{eqnarray}
 %===============
```

```
p <- p + ylab("Residuals")
print(p)

# Plot residuals
plot(y.hat, e.hat
      , main = "Residuals vs Fitted values"
      , xlab = "Fitted values"
      , ylab = "Residuals")
  # horizontal reference line at zero
  abline(h = 0, col = "gray75")
```





1.2 Covariance and correlation matrices

Suppose you have data on p variables from n individuals. Let

$$x_{ij} \;=\; \text{response on person } i \text{ for variable } j.$$

The **covariance** between the jth and kth response is defined as

$$\text{Cov}(\underset{\sim}{x}_j, \underset{\sim}{x}_k) \;=\; \frac{1}{n-1}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k),$$

where $\bar{x}_j$ and $\bar{x}_k$ are the average responses for the jth and kth variables.

The covariance matrix is defined as

$$\text{Cov}(\mathbf{X}) \;=\; [\text{Cov}(\underset{\sim}{x}_j, \underset{\sim}{x}_k)]_{p\text{-by-}p},$$

that is, $\text{Cov}(\mathbf{X})$ is a p-by-p matrix with $\text{Cov}(\underset{\sim}{x}_j, \underset{\sim}{x}_k)$ in the jth row and kth column. Note that $\text{Cov}(\mathbf{X})$ is symmetric, that is $\text{Cov}(\mathbf{X}) = \text{Cov}(\mathbf{X})^\top$ because $\text{Cov}(\underset{\sim}{x}_j, \underset{\sim}{x}_k) = \text{Cov}(\underset{\sim}{x}_k, \underset{\sim}{x}_j)$. Also note that the diagonal elements of $\text{Cov}(\mathbf{X})$ are the sample variances,

$$\text{Cov}(\underset{\sim}{x}_j, \underset{\sim}{x}_j) \;=\; \frac{1}{n-1}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)(x_{ij} - \bar{x}_j) \;=\; \frac{1}{n-1}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2.$$

The **correlation** is a function of the covariance and variance terms,

$$\text{Cor}(\underset{\sim}{x}_j, \underset{\sim}{x}_k) \;=\; \frac{\text{Cov}(\underset{\sim}{x}_j, \underset{\sim}{x}_k)}{\sqrt{\text{Var}(\underset{\sim}{x}_j)\text{Var}(\underset{\sim}{x}_k)}}$$

which can be collected in the correlation matrix

$$\text{Cor}(\mathbf{X}) \;=\; [\text{Cor}(\underset{\sim}{x}_j, \underset{\sim}{x}_k)]_{p\text{-by-}p}.$$

Note that $\text{Cor}(\mathbf{X})$ is symmetric because $\text{Cor}(\mathbf{X}) = \text{Cor}(\mathbf{X})^\top$ and that the diagonal elements are 1:

$$\text{Cor}(\underset{\sim}{x}_j, \underset{\sim}{x}_j) \;=\; \frac{\text{Cov}(\underset{\sim}{x}_j, \underset{\sim}{x}_j)}{\sqrt{\text{Var}(\underset{\sim}{x}_j)\text{Var}(\underset{\sim}{x}_j)}} \;=\; \frac{\text{Var}(\underset{\sim}{x}_j)}{\text{Var}(\underset{\sim}{x}_j)} \;=\; 1.$$

Prior to matrix programming, computer programming languages such as FORTRAN allowed matrices, but calculations were performed elementwise. To compute a vector of means and a covariance matrix required looping. Here is the R analog of such calculations using `for` loops, assuming data stored in an n-by-p matrix $\mathbf{X}$ with

$$\text{R indexing: } \texttt{X[i,j]} \;=\; \mathbf{X}_{ij} \text{ notation},$$

that is, rows are individuals and columns are variables. Let

$$\underset{\sim}{\bar{x}} = \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \vdots \\ \bar{x}_p \end{bmatrix}_{p\text{-by-}1}$$

be the vector of means. To get $\underset{\sim}{\bar{x}}$ in R , use this built-in function:

```
# calculate matrix/data.frame column means
m.x <- colMeans(X)
m.x

## Acetic    H2S Lactic
##  5.498  5.942  1.442

# transpose into a column
m.x <- matrix(m.x, ncol = 1)
m.x

##       [,1]
## [1,] 5.498
## [2,] 5.942
## [3,] 1.442

# time to do this is
system.time(m.x <- matrix(colMeans(X), ncol = 1))

##    user  system elapsed
##       0       0       0
```

In terms of loops and elementwise calculations, one strategy is to add each column vector of $\mathbf{X}$ and divide by the sample size.

```
# output: m.x column mean vector
# input:  X design matrix
slow.mean <- function(X) {
  n <- nrow(X) # sample size
  p <- ncol(X) # number of predictors
```

```
  # initialize vector to store means
  m.x <- matrix(0, nrow = p, ncol = 1)

  # for each variable
  for (j in 1:p) {
    # for each individual
    for (i in 1:n) {
      # increment sum for jth variable
      m.x[j] <- m.x[j] + X[i,j]
    }
    # inner loop completed, scale jth sum to mean
    m.x[j] <- m.x[j] / n
  }
  return(m.x)
}

# call the function to compute the mean
m.x <- slow.mean(X)
m.x


##       [,1]
## [1,] 5.498
## [2,] 5.942
## [3,] 1.442


# time to do this is
system.time(m.x <- slow.mean(X))


##    user  system elapsed
##       0       0       0
```

My computer is so fast (and the Cheese dataset is so small) that the time taken shows 0 seconds in both cases. Here's an example with a larger dataset so we see a time difference.

```
XX <- matrix(rnorm(1000*20), nrow=1000, ncol=20)
system.time(matrix(colMeans(XX), ncol = 1))


##    user  system elapsed
##       0       0       0


system.time(slow.mean(XX))
```

```
##    user  system elapsed
##    0.04    0.00    0.05
```

To get the covariance in R only requires using the built-in function

```
c.X <- cov(X)
c.X


##         Acetic    H2S  Lactic
## Acetic 0.3259 0.7503 0.10461
## H2S    0.7503 4.5236 0.41622
## Lactic 0.1046 0.4162 0.09211


system.time(c.X <- cov(X))


##    user  system elapsed
##       0       0       0
```

To calculate the covariance via loops requires 3 `for` loops: one to index the observation number and the other two to index row and column of the covariance matrix. For example:

```
# output: c.x covariance matrix
# input:  X design matrix
slow.cov <- function(X) {
  n <- nrow(X) # sample size
  p <- ncol(X) # number of predictors

  # initialize matrix to store covariances
  c.x <- matrix(0, nrow = p, ncol = p)

  # compute mean vector (the fast way)
  m.x <- matrix(colMeans(X), ncol = p)

  # for each variable
  for (j in 1:p) {
    # for each variable
    for (k in j:p) {
      # for each individual
      # calculate the covariance of the diagonal and upper-off-diagonal
      for (i in 1:n) {
        # increment sum for jth variable
        c.x[j, k] <- c.x[j, k] + (X[i, j] - m.x[j]) * (X[i, k] - m.x[k])
```

```
      }
      # inner loop completed, scale jth sum to mean
      c.x[j, k] <- c.x[j, k] / (n - 1)
      # assign the lower-off-diagonal the symmetric upper value
      if (k > j) {
        c.x[k, j] <- c.x[j, k]
      }
    }
  }
  return(c.x)
}

# call the function to compute the mean
c.x <- slow.cov(X)
c.x

##         [,1]   [,2]    [,3]
## [1,] 0.3259 0.7503 0.10461
## [2,] 0.7503 4.5236 0.41622
## [3,] 0.1046 0.4162 0.09211

# time to do this is
system.time(c.x <- slow.cov(X))

##    user  system elapsed
##       0       0       0
```

Here's an example with a larger dataset so we see a time difference.

```
system.time(cov(XX))

##    user  system elapsed
##       0       0       0

system.time(slow.cov(XX))

##    user  system elapsed
##    1.11    0.00    1.11
```

Avoid coding with excessive loops. The code becomes more difficult to understand and is not computationally efficient. Always search for matrix representations of calculations.

Although R directly computes the mean and covariance, it is useful to learn how to represent the calculation using matrix expressions. To see this, let

$$\underset{\sim}{x}_i \;=\; \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{ip} \end{bmatrix}_{p\text{-by-}1}$$

be the data on individual i, so that

$$\mathbf{X}_{n\text{-by-}p} \;=\; \begin{bmatrix} \underset{\sim}{x}_1^\top \\ \underset{\sim}{x}_2^\top \\ \vdots \\ \underset{\sim}{x}_n^\top \end{bmatrix}.$$

If, as before,

$$\underset{\sim}{\bar{x}} \;=\; \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \vdots \\ \bar{x}_p \end{bmatrix}_{p\text{-by-}1},$$

then one can show

$$\mathrm{Cov}(\mathbf{X}) \;=\; \frac{1}{n-1}\sum_{i=1}^{n}(\underset{\sim}{x}_i - \underset{\sim}{\bar{x}})_{p\text{-by-}1}(\underset{\sim}{x}_i - \underset{\sim}{\bar{x}})^\top_{1\text{-by-}p}.$$

If we define the "centered data matrix" to be

$$\mathbf{X}_{\mathrm{c}} \;=\; \mathbf{X} - \left\{ \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}_{n\text{-by-}1} \times\, \underset{\sim}{\bar{x}}^\top_{1\text{-by-}p} \right\},$$

where the n-by-p matrix in the braces has $\underset{\sim}{\bar{x}}^\top$ for each row. That is,

$$\begin{aligned}\mathbf{X}_{\mathrm{c}} &= \begin{bmatrix} \underset{\sim}{x}_1^\top - \underset{\sim}{\bar{x}}^\top \\ \underset{\sim}{x}_2^\top - \underset{\sim}{\bar{x}}^\top \\ \vdots \\ \underset{\sim}{x}_n^\top - \underset{\sim}{\bar{x}}^\top \end{bmatrix} \\ &= \begin{bmatrix} (\underset{\sim}{x}_1 - \underset{\sim}{\bar{x}})^\top \\ (\underset{\sim}{x}_2 - \underset{\sim}{\bar{x}})^\top \\ \vdots \\ (\underset{\sim}{x}_n - \underset{\sim}{\bar{x}})^\top \end{bmatrix}.\end{aligned}$$

Then,

$$\begin{aligned}\mathrm{Cov}(\mathbf{X}) &= \frac{1}{n-1}\sum_{i=1}^{n} \begin{bmatrix} (\underset{\sim}{x}_1 - \underset{\sim}{\bar{x}}) & (\underset{\sim}{x}_2 - \underset{\sim}{\bar{x}}) & \cdots & (\underset{\sim}{x}_n - \underset{\sim}{\bar{x}}) \end{bmatrix} \begin{bmatrix} (\underset{\sim}{x}_1 - \underset{\sim}{\bar{x}})^\top \\ (\underset{\sim}{x}_2 - \underset{\sim}{\bar{x}})^\top \\ \vdots \\ (\underset{\sim}{x}_n - \underset{\sim}{\bar{x}})^\top \end{bmatrix} \\ &= \frac{1}{n-1}\mathbf{X}_{\mathrm{c}}^\top \mathbf{X}_{\mathrm{c}}.\end{aligned}$$

This is the sum of n p-by-p matrices[3].

The correlation matrix is also easy to compute. Recall that

$$\mathrm{Cor}(\mathbf{X}) = [\mathrm{Cor}(\underset{\sim}{x}_j, \underset{\sim}{x}_k)]_{p\text{-by-}p},$$

where

$$\mathrm{Cor}(\underset{\sim}{x}_j, \underset{\sim}{x}_k) = \frac{\mathrm{Cov}(\underset{\sim}{x}_j, \underset{\sim}{x}_k)}{\sqrt{\mathrm{Var}(\underset{\sim}{x}_j)}\sqrt{\mathrm{Var}(\underset{\sim}{x}_k)}}.$$

[3]Note that

$$\mathrm{Cov}(\mathbf{X}) = \frac{1}{n-1}\left(\mathbf{X}^\top\mathbf{X} - n\underset{\sim}{\bar{x}}\underset{\sim}{\bar{x}}^\top\right)$$

is fast, but can result in negative numbers from round-off.

If we define a diagonal p-by-p matrix with diagonal elements $1/\sqrt{\text{Var}(x_j)}$, that is,

$$\mathbf{D} = \begin{bmatrix} 1/\sqrt{\text{Var}(x_1)} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1/\sqrt{\text{Var}(x_p)} \end{bmatrix},$$

then it is easy to see that $\text{Cor}(\mathbf{X})$ is the matrix product

$$\text{Cor}(\mathbf{X}) = \mathbf{D}\text{Cov}(\mathbf{X})\mathbf{D}.$$

These calculations are illustrated below.

```
# output: out   list of the following summary statistics for a multivariate data set
#         m.x   column mean vector
#         s.x   column standard deviations vector
#         c.x   covariance matrix
#         cor.x correlation matrix
# input:  X design matrix
fast.summ <- function(X) {
  n <- nrow(X) # sample size
  p <- ncol(X) # number of predictors

  # compute mean vector (the fast way)
  m.x <- matrix(colMeans(X), ncol = 1)

  # compute mean vector (the fast way)
  s.x <- apply(X, 2, sd)

  # n-by-p matrix with means in each row
  m.inrows <- matrix(rep(1, n), ncol = 1) %*% t(m.x)
  # centered data matrix
  Xc <- X - m.inrows

  # p-by-p matrix of covariances
  c.x <- t(Xc) %*% Xc / (n - 1)

  # diagonal matrix of inverse standard deviations
  D <- diag(1 / s.x)

  # p-by-p matrix of correlations
```

```
  cor.x <- D %*% c.x %*% D


  ## initialize a list to hold all data/output
  out <- as.list(new.env());
  out$m.x   <- m.x
  out$s.x   <- s.x
  out$c.x   <- c.x
  out$cor.x <- cor.x

  return(out)
}

# call the function
summ <- fast.summ(X)
summ

## $m.x
##       [,1]
## [1,] 5.498
## [2,] 5.942
## [3,] 1.442
##
## $s.x
## Acetic    H2S Lactic
## 0.5709 2.1269 0.3035
##
## $c.x
##        Acetic    H2S  Lactic
## Acetic 0.3259 0.7503 0.10461
## H2S    0.7503 4.5236 0.41622
## Lactic 0.1046 0.4162 0.09211
##
## $cor.x
##        [,1]   [,2]   [,3]
## [1,] 1.0000 0.6180 0.6038
## [2,] 0.6180 1.0000 0.6448
## [3,] 0.6038 0.6448 1.0000


# time the matrix version of the calculations on the big matrix
system.time( fast.summ(XX) )

##    user  system elapsed
##       0       0       0
```

Code summary Below is a list of (most of) the functions used in this chapter.

```
str()

solve()
plot()
  abline()
pairs()

library(ggplot2)
ggplot()
  geom_hline()
  geom_point()
  labs(title=)
  xlab()
  ylab()

library(GGally)
ggpairs()

matrix()
rep()

mean()
sd()
cov()
cor()
colMeans()

apply()
%*%
as.list(new.env())
return()

system.time()
```