R Notes for Multivariate Analysis

Yongfu, Liao

2018-04-05

Contents

About

This is a very simplified book about Multivariate Analysis in R. It is written as a note to facilitate my learning of Multivariate Analysis at NTU, Spring, 2018.

Feel free to share, modify, and reshare the work. For more information, see the License section below.

License

Chapter 1

Multivariate Normal Distribution & Covariance Matrix

```
library(dplyr)
library(latex2exp)
library(ggplot2)
theme <- theme(axis.text.x = element_text(size = 7, face = "plain", angle = 30),
               axis.text.y = element_text(size = 7, face = "plain"),
               axis.title.x = element_text(size = 9, face = "bold"),
    axis.title.y = element_text(size = 9, face = "bold"))
```

1.1 Bivariate Normal Contour Map

1.1.1 `ellipse` function

```
ellipse(x, scale, centre, level, npoints = 1000)
```

- `x`: a single number, correlation of the two variables.
- `scale`: vector, **standard deviation** of the two variables.
- `centre`: vector, center of the ellipse, i.e. the mean vector of the bivariate normal distribution.
- `level`: a single number, the contour probability.
- `npoints`: number of points used to draw the contour.

`ellipse` returns a **matrix** with dimension (`npoints` $\times$ 2), which can be used to plot contour.

1.1.2 Data Generation

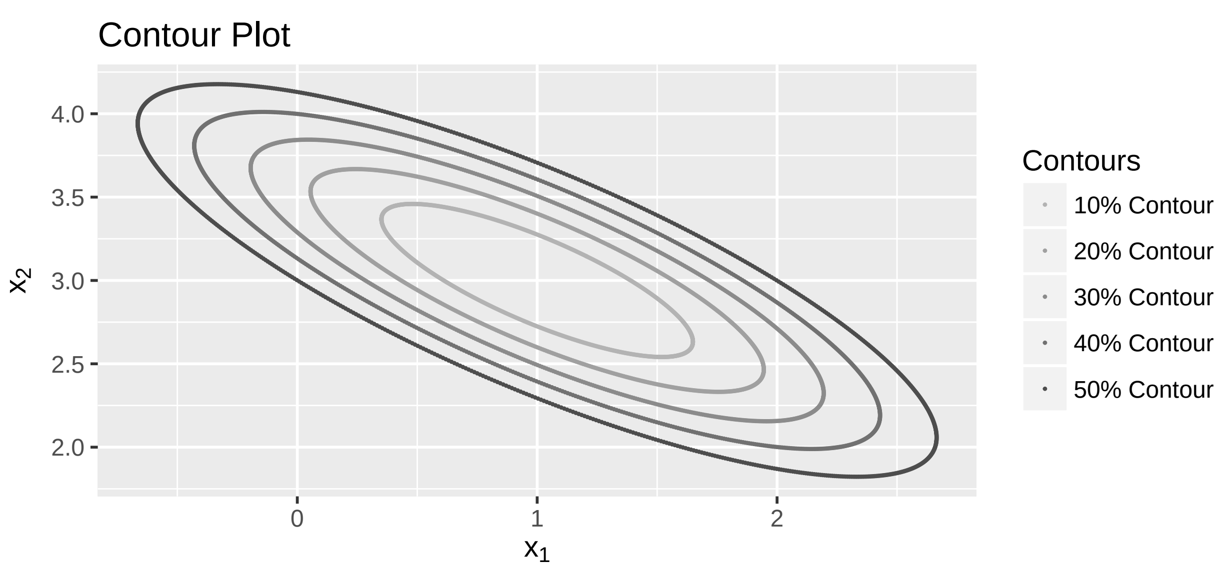
The `for` loop below is used to generate a data frame with 3 columns(variables): - Column 1: First variable of bivariate normal function (x_1) - Column 2: Second variable of bivariate normal function (x_2) - Column 3: The contour that x_1 & x_2 on the same row belongs to.

```
library(ellipse)
```

```
All_contours <- c(NA, NA, NA)
    ## Set empty start for appending ##

for (i in 1:5) {
    level <- 0.1*i
        ## Set Contour prob., prob. of obs within contour ##
    ell_data <-ellipse(-0.8, c(sqrt(2), 1), centre = c(1, 3), level = level, npoints = 800+(i-1)^3)
        ## npoints: bigger contours with more points ##
    class <- rep(paste(level*100, "% Contour", sep=""), nrow(ell_data))
        ## Assign contour class ##
    ell_data <- as.data.frame(ell_data)
        ## Change to data.frame BEFORE cbind, ##
        ## or coersion happens ##
    ell_data <- cbind(ell_data, class)

    All_contours <- rbind(All_contours, ell_data)
}

All_contours <- All_contours[-1,]
    ## Remove the empty start ##
```

1.1.3 Plotting

```
ggplot(data = All_contours) +
    geom_point(aes(x = x, y = y, color = class),
               size = 0.1) +
    scale_colour_grey(start = 0.7, end = 0.3) +
        ## Use gray scales instead of colored default ##
    labs(color = "Contours",
         title = "Contour Plot",
         x = TeX("$x_1$"), y = TeX("$x_2$")
    )
```



1.2 Multivariate Normal Functions

1.2.1 Generate density f(x)

```
library(mvtnorm)

mu <- c(1, 3) # mean vector
Sigma <- matrix(c(2, -0.8*sqrt(2), -0.8*sqrt(2), 1),
                nrow = 2) # covariance matrix

dmvnorm(x = c(2, 5), mean = mu, sigma = Sigma)
```

```
[1] 1.562995e-05
```

- `x`: Vector x in f(x), all variables of the multivariate normal distribution.
- `mean`: Mean vector(center of ellipse) of the multivariate normal distribution.
- `sigma`: Covariance matrix of the multivariate normal distribution.

`dmvnorm` returns f(x), the range of the multivariate normal function. For example, `dmvnorm(x = c(2, 5), mean = mu, sigma = Sigma)` returns the value $f(x_1 = 2, x_2 = 5)$ of the multivariate normal distribution specified by mean vector, `mu`, and covariance matrix, `Sigma`.

1.2.1.1 Example: Densities of a Contour

```
data <- All_contours %>%
    filter(class == "50% Contour")

dmvnorm(x = data[1, 1:2], mean = mu, sigma = Sigma)[[1]]
```

```
[1] 0.09378295
```

```
dmvnorm(x = data[4, 1:2], mean = mu, sigma = Sigma)[[1]]
```

```
[1] 0.09378295
```

The retured values are the same(very close), since they are on the same contour. See the section above for more details.

1.2.2 Covariance Matrix

Generater covariance and correlation Matricies:

```
library(mat2tex)
cov.mt <- cov(iris[,1:4]) ## Cov Matrix of variable 1~4
cor.mt <- cor(iris[,1:4]) ## Cor Matrix of variable 1~4
```

$$\text{Covariance matrix} = \begin{pmatrix} 0.69 & -0.04 & 1.27 & 0.52 \\ -0.04 & 0.19 & -0.33 & -0.12 \\ 1.27 & -0.33 & 3.12 & 1.30 \\ 0.52 & -0.12 & 1.30 & 0.58 \end{pmatrix}$$

$$\text{Correlation matrix} = \begin{pmatrix} 1.00 & -0.12 & 0.87 & 0.82 \\ -0.12 & 1.00 & -0.43 & -0.37 \\ 0.87 & -0.43 & 1.00 & 0.96 \\ 0.82 & -0.37 & 0.96 & 1.00 \end{pmatrix}$$

Chapter 2

Principle Component Analysis

2.1 Workflow of PCA

2.1.1 Conceptual

2.1.2 Computational (with R)

- Note: `sdev` of `prcomp()` are **Standard Deviations**. To get the eigenvalues of the covariance (correlation) matrix, or equivalently, variances of the principle components, you need to **square `sdev`**.

2.2 Conversion Between Correlation & Covaraince Matrices

The function `prcomp()` in base R `stats` package performs principle component analysis to input `data.frame`(with observations as rows and variables as columns), but it returns neither covariance nor correlation matrix. You can compute them directly by passing `data.frame` to `cor()` and `cov()` directly in R without any additional package.

Sometimes there is no raw data but only covariance or correlation matrix, and you may want to convert one to another. This can be done by using simple matrix multiplication, based on the fact that

$$\mathbf{R} = diag(\mathbf{S})^{\frac{-1}{2}} \ \mathbf{S} \ diag(\mathbf{S})^{\frac{-1}{2}}$$

, where $\mathbf{R}$ is the correlation matrix, $\mathbf{S}$ is the covariance matrix, and $diag(\mathbf{S})$ is the diagonal matrix composed of diagonal elements of $\mathbf{S}$.

2.2.1 `eigen()`

After obtaining the covariance or correlation matrix, direct computation of eigenvalue and eigenvectors is straightforward: pass the matrix to base R `eigen()` function.

```
cov(iris[,1:3]) %>% eigen()
```

```
eigen() decomposition
$values
[1] 3.69111979 0.24137727 0.05945372
```

```
$vectors
            [,1]       [,2]       [,3]
[1,]  0.38983343  0.6392233 -0.6628903
[2,] -0.09100801  0.7430587  0.6630093
[3,]  0.91637735 -0.1981349  0.3478435
```

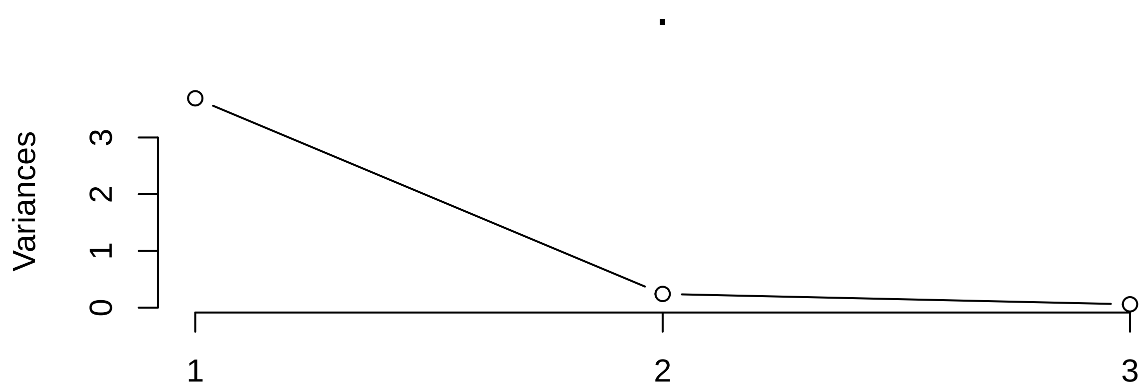
2.3 Scree Plot

Scree plot is an important tool for determining the importance of principle components. Although the logic of plotting scree plots is easy, it may be quite annoying for repeating the code every time.

2.3.1 `screeplot()` from Base R

There is a ready-written function for scree plot in `stats` package, but the output is terrible:

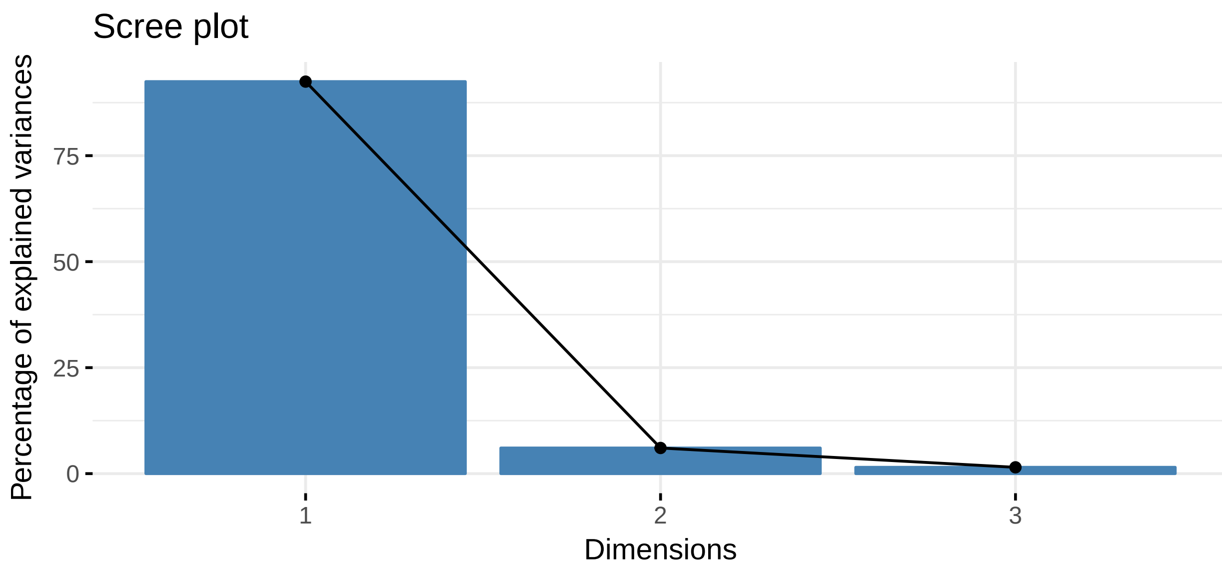
```
prcomp(iris[,1:3]) %>% screeplot(type="lines")
```



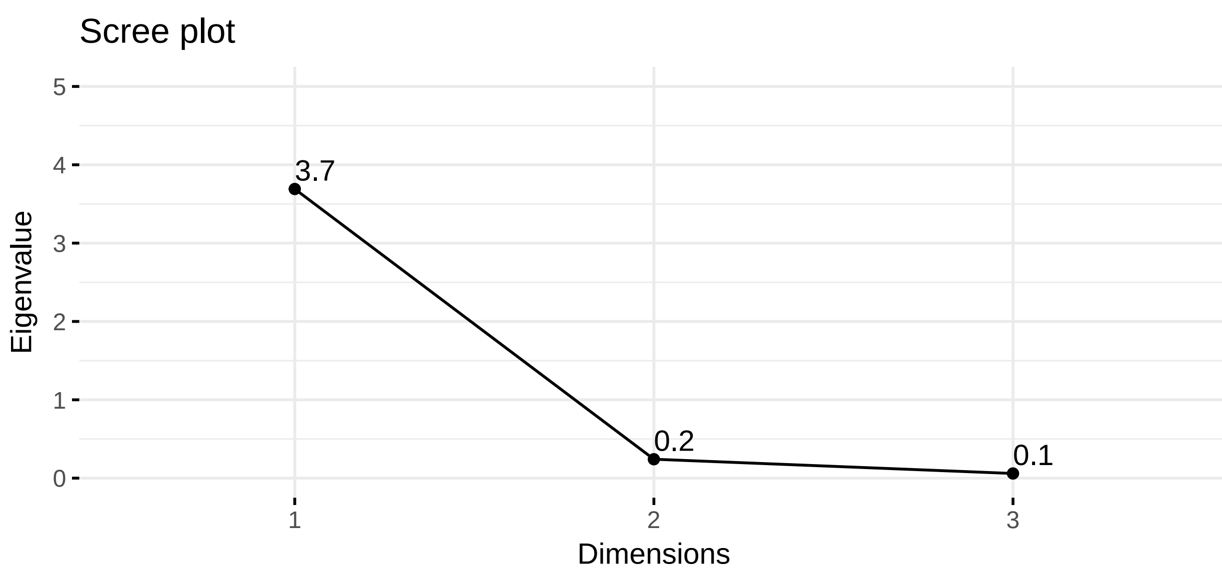
2.3.2 `fviz_eig()` from `factoextra`

For a better looking scree plot function, I recommend `fviz_eig()` from `factoextra` package. `fviz_eig()` has better looking outputs and more customizable plotting parameters, and since it is based on `ggplot2`, you can actually enhance it with the `ggplot2` syntax: `+`.

```
library(factoextra)
prcomp(iris[,1:3]) %>% fviz_eig()
```




```
prcomp(iris[,1:3]) %>%
    fviz_eig(choice = "eigenvalue", # y as eigenvalue
             geom = "line",
             addlabels = T) +
    scale_y_continuous(limits = c(0, 5))
```



2.3.3 Customized Function

I have OCD with plotting, so not completely satisfied with `factoextra::fviz_eig()`. So I created my own `scree_plot()` by building on `fviz_eig()`[1], which supports **double y-axis**: one showing eigenvalue, the other proportion of total variance explained.

[1] Check `multivariate_fc.R` starting at line 46.

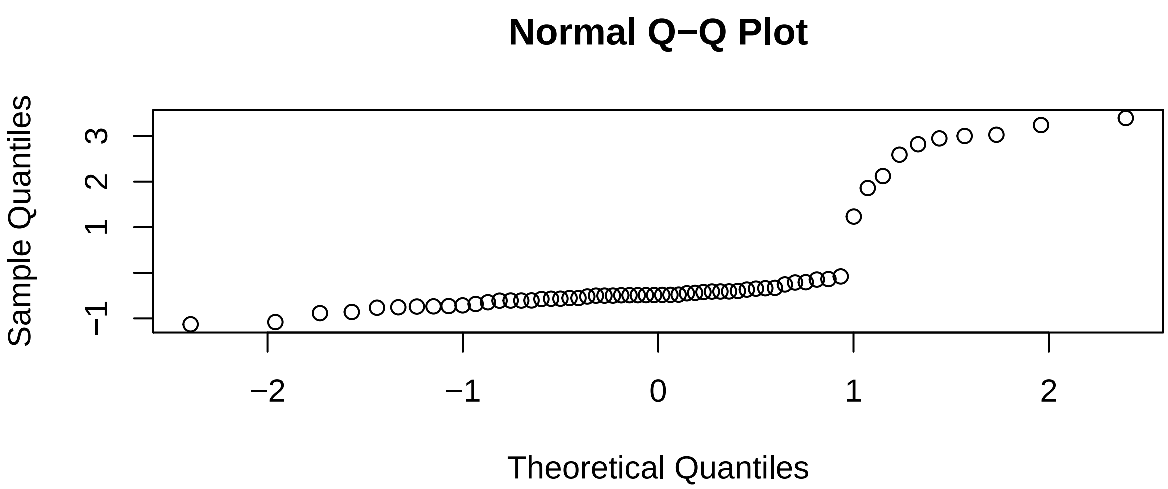
2.4 Q-Q Plot

Q-Q plots are for checking the normality assumuption and are also useful for detecting outlyers. Principle components are linear combinations of the original variables, so if the original variables come from a multivariate normal distribution, principle components are expected to have normal distributions.

2.4.1 `qqnorm()` from Base R

There is also a base R `qqnorm()` function, which plots sample quantiles against theoretical quantiles obtain from the standard normal distribution.

```
prcomp(iris[1:60, 1:3])[["x"]][,1] %>%
    qqnorm()
```



`prcomp(data.frame)[["x"]]` returns the principle component scores, i.e. data that are **rotated** or **weighted** by the elements of the eigenvectors.

`prcomp(data.frame)[["x"]][,1]` subsets the first column of the principle component scores, which is the scores of the First principle component, i.e. data weighted according to elements of the first (corresponding to the largest eigenvalue) eigenvector.

2.4.2 Self-defined Function

`qqnorm()` is pretty good but lacking one important features: **labeling points on the Q-Q plot so that identification of the points is possible**.

So I wrote my own function `QQplot`, which labels every point on the graph:

```
QQplot <- function(x, ID="none",
                   theme=NULL, color="red", text=TRUE,
                   text_adj=
                       c(hjust=-0.1, vjust=0, size=3)) {
    library(dplyr)
    library(ggplot2)
```

```
    x <- as_data_frame(x)
    n <- nrow(x)
    quantiles <- qnorm(p=seq(0.5/n, 1-0.5/n, 1/n))

    if (ID == "none") { # assign ID if not passed
        ID <- as.character(1:n)
    } else {
        ID <- as_data_frame(ID)
        ID <- as.character(ID[[colnames(ID)]])
        }

    if (text == TRUE) {
        text <- geom_text(aes(label=ID),
                          hjust=text_adj[1],
                          vjust=text_adj[2],
                          size = text_adj[3])
    } else {text <- NULL}

    data <- cbind(ID, x)
    colnames(data) <- c("ID", "x")
    data <- data %>% arrange(x) %>% mutate(quantile=quantiles)

    pl <- ggplot(data, aes(x=quantiles, y=x))+
        geom_point(color=color)+
        text + theme +
        labs(x="Theoretical Quantile",
             y="x",
             title="Q-Q Plot")
    pl
}
```

```
prcomp(iris[1:60, 1:3])[["x"]][,1] %>%
    QQplot()
```