Installing R and RStudio

We solve the computer exercises of this course using the statistical software R. R is a widely used and freely distributed programming language that is particularly suitable for statistical analysis. You should be able to find R from every computer located in the Undergraduate Centre (Otakaari 1) or Maarintalo. The computer exercises of this course can be solved by using the basic R software, that you can download free of charge to your personal computer. There exists many different integrated development environments (IDE) for the R programming language. We recommend that you use the one called RStudio. RStudio is also free of charge. Note that, in order to use RStudio, you need the basic R software installed. Below are the links for installing R and RStudio, respectively.

- Install R – https://cran.r-project.org/
- Install Rstudio – https://posit.co/download/rstudio-desktop/

Demo Problem 1: Introduction to R

a) Change your working directory. Try the commands `help(c)` and `help(matrix)`.

b) Calculate the affine transformation $\boldsymbol{y} = \boldsymbol{x}\boldsymbol{A}^{-1} + \boldsymbol{b}$, where

$$\boldsymbol{A} = \begin{pmatrix} 2 & 1 & 5 \\ -2 & 7 & 0 \\ 5 & -8 & -1 \end{pmatrix}, \qquad \boldsymbol{x}^\mathsf{T} = \begin{pmatrix} 8 \\ -4 \\ 2 \end{pmatrix}, \qquad \boldsymbol{b}^\mathsf{T} = \begin{pmatrix} 3 \\ 10 \\ -19 \end{pmatrix}.$$

c) Install the package `mvtnorm` and load the corresponding functions to your workspace. Set the seed to 123 using the command `set.seed(123)`. Generate 100 observations from a two-dimensional normal distribution with the expected value $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$, where

$$\boldsymbol{\mu} = \begin{pmatrix} 3 \\ 1 \end{pmatrix} \qquad \text{and} \qquad \boldsymbol{\Sigma} = \begin{pmatrix} 4 & 1 \\ 1 & 2 \end{pmatrix}.$$

Visualize the observations.

d) Use the data from part c) and compute the sample mean $\bar{\boldsymbol{x}}$ and the sample covariance matrix $\boldsymbol{S}_x$. Compute the eigenvalues and eigenvectors of the matrix $\boldsymbol{S}_x$. Verify from the data, that the following equations hold,

$$\begin{aligned} &\mathrm{Tr}\,(\boldsymbol{S}_x) = \lambda_1 + \lambda_2 + \cdots + \lambda_p \text{ and} \\ &\mathrm{Det}\,(\boldsymbol{S}_x) = \lambda_1 \lambda_2 \cdots \lambda_p, \end{aligned}$$

where λ_i are the eigenvalues of $\boldsymbol{S}_x$.

e) Let $\boldsymbol{x}_i$, $i \in \{1, \ldots, 100\}$, denote the ith observation of the sample generated in part c). Compute the affine transformations $\boldsymbol{y}_i = \boldsymbol{A}\boldsymbol{x}_i + \boldsymbol{b}$, where

$$\boldsymbol{b} = \begin{pmatrix} 3 \\ 1 \end{pmatrix} \qquad \text{and} \qquad \boldsymbol{A} = \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix},$$

verify that $\bar{\boldsymbol{y}} = \boldsymbol{A}\bar{\boldsymbol{x}} + \boldsymbol{b}$ and $\boldsymbol{S}_y = \boldsymbol{A}\boldsymbol{S}_x\boldsymbol{A}^\mathsf{T}$. What does affine equivariance mean in practice?

f) Upload the data from the file `data.txt` into your workspace. Create a function that centers your data (removes the mean) and pairwise scatterplots the variables. Compute the sample covariance and correlation matrices and the corresponding eigenvalues and eigenvectors.

Solution

a) Whenever you refer to a file with a relative path, you have to type the path relative to your working directory.

Path of the current working directory can be seen with the function `getwd`. The working directory can be set with the function `setwd`. Alternatively, one can use the RStudio interface for setting the working directory: `Session` → `Set Working Directory` → `Choose Directory` (`Ctrl + Shift + H`).

```
setwd("~/teaching/multivariate/01week/markdown/")
getwd()
```

Manual pages for functions, classes etc. can be found with the command `help` or `?`. You can comment code by starting a line with `#`.

```
# How do I create vectors?
help(c)
?c

# How do I create matrices?
help(matrix)
?matrix
```

b) First, the matrix $\boldsymbol{A}$ and vectors $\boldsymbol{x}$ and $\boldsymbol{b}$ are created. Values to variables can be assigned with either `=` or `<-`. Choose one and stick with it. For more information on the differences between `=` and `->` see the answer to the following stackoverflow question.

We create vectors $\boldsymbol{x}$ and $\boldsymbol{b}$ in two ways. First, with command `c` and then as a row vector with command `matrix`.

```
a <- matrix(c(2, 1, 5, -2, 7, 0, 5, -8, -1), nrow = 3, byrow = TRUE)
x1 <- c(8, -4, 2)
b1 <- c(3, 10, -19)

x2 <- matrix(x1, nrow = 1, byrow = TRUE)
b2 <- matrix(b1, nrow = 1, byrow = TRUE)
```

Matrix multiplication can be performed with the operator `%*%` and inverse matrix can be computed with the function `solve`. Both `y1` and `y2` give the right result. This is because according to the documentation of `%*%`, vectors are interpreted as row or column vectors such that the arguments are conformable. If you want to be explicit about dimensions of vectors, use the function `matrix`.

```
y1 <- x1 %*% solve(a) + b1
y2 <- x2 %*% solve(a) + b2

all(y1 == y2)
```

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

```
y1
```

```
##           [,1]     [,2]      [,3]
## [1,] 3.774775 11.45946 -17.12613
```

For example, the following code gives an error.

```
x3 <- matrix(x1, ncol = 1, byrow = FALSE)
b3 <- matrix(b1, ncol = 1, byrow = FALSE)

x3 %*% solve(a) + b3
```

c) If there is some functionality that is not implemented in base R there is most probably a package for it. You can install packages with the function `install.packages`. Note that the package name has to be given as a character string for the function `install.packages`. For example, the package `mvtnorm` that is required for this exercise session can be installed with the following line of code.

```
install.packages("mvtnorm")
```

Once the package is installed you can use functionality inside the package by specifying the correct namespace and using double colon `::` between the namespace and the function. Below we use the function `rmvnorm` from the package `mvtnorm`.

```
mvtnorm::rmvnorm(3, rep(0, 2))
```

Namespaces are a useful concept since there can be functions with the same name in different packages. For example function `lag` can be found at least in two different packages.

```
?stats::lag
?dplyr::lag
```

Instead of specifying the namespace one can attach the package with the function `library`. For an example, see following lines of code.

```
library(mvtnorm)
rmvnorm(3, rep(0, 2))
```

If you decide to attach packages it is a good practice to put `library` commands at the top of your script, instead of scattering them all around the script.

Next, we create a sample of size $n = 100$ from a multivariate normal distribution with the location vector $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$ specified in the exercise description. With function `head` you can see the first few observations.

```
n <- 100
mu <- c(3, 1)
sigma <- matrix(c(4, 1, 1, 2), byrow = TRUE, ncol = 2)

set.seed(123)
x <- rmvnorm(n, mu, sigma)
head(x)
```

```
##          [,1]       [,2]
## [1,] 1.823028  0.5149745
## [2,] 6.103696  1.5613434
## [3,] 3.766090  3.4096342
## [4,] 3.535096 -0.6118415
## [5,] 1.508960  0.1794479
## [6,] 5.527988  1.8617391
```

Resulting figure from `plot(x)` should resemble Figure 1 if you set the seed to 123 before creating the sample.

```
plot(x, pch = 20, xlab = expression("X"[1]), ylab = expression("X"[2]),
     main = expression(paste("Sample from ", "N(", mu, ", ", Sigma, ")")))
```

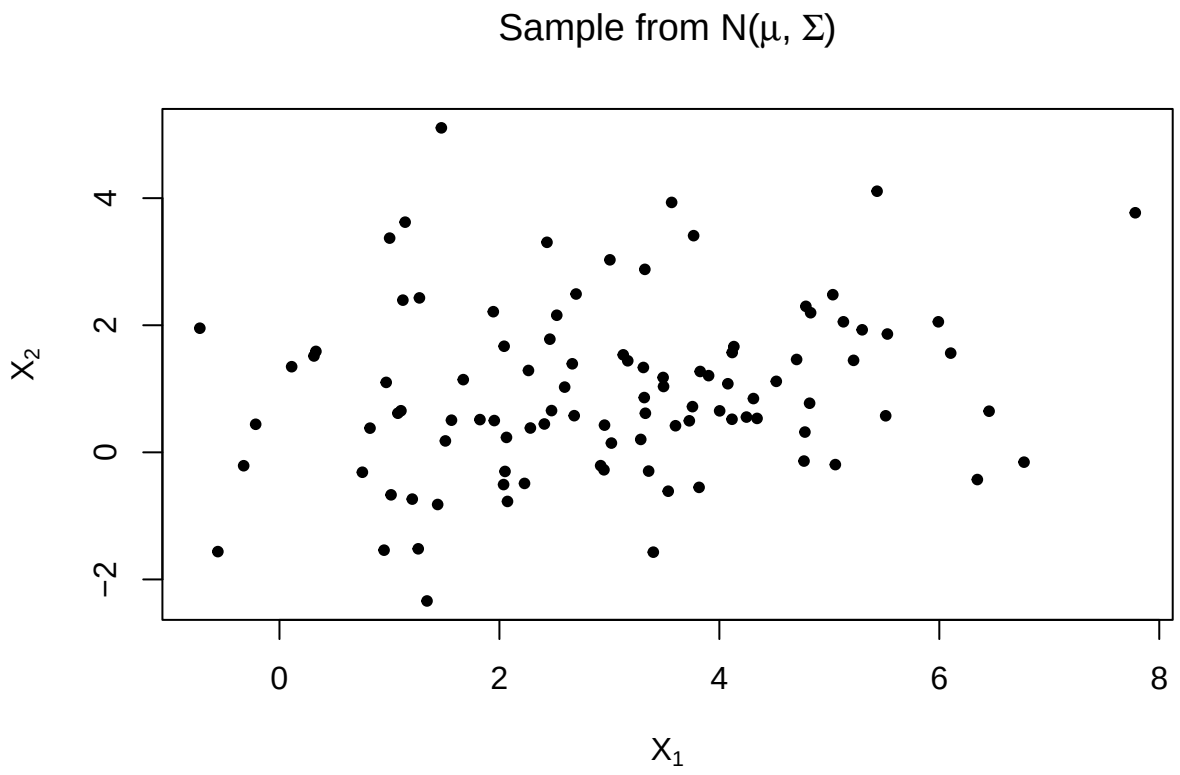

Figure 1: Scatter plot of a sample from a bivariate normal distribution.

d) There are a couple of ways to compute the sample mean vector.

```
x_mean1 <- apply(x, 2, mean) # Apply function "mean" to every column
x_mean2 <- colMeans(x)
all(x_mean1 == x_mean2)
```

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

```
x_mean1
```

```
## [1] 3.004655 0.970002
```

One can use the function `cov` to compute the sample covariance.

```
x_cov <- cov(x)
x_cov
```

```
##           [,1]      [,2]
## [1,] 3.0744739 0.4545397
## [2,] 0.4545397 1.8184535
```

Next, with the function `eigen` we can calculate the eigenvalues and eigenvectors of the sample covariance matrix $\boldsymbol{S}_x$. Columns of the variable `eigvec` contain the eigenvectors.

```
eig <- eigen(x_cov)
eigval <- eig$values
```

```
eigvec <- eig$vectors
eigval
```

```
## [1] 3.221708 1.671220
```

```
eigvec
```

```
##            [,1]       [,2]
## [1,] -0.9513361  0.3081552
## [2,] -0.3081552 -0.9513361
```

Lastly, let us verify that

$$\mathrm{Tr}(\boldsymbol{S}_x) = \lambda_1 + \lambda_2$$

and that

$$\mathrm{Det}(\boldsymbol{S}_x) = \lambda_1 \lambda_2.$$

```
# Remember that trace of a square matrix is defined as the sum of the
# diagonal elements.
sum(diag(x_cov)) - sum(eigval)
```

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

```
det(x_cov) - prod(eigval)
```

```
## [1] -1.776357e-15
```

e) First, let us create the matrix $\boldsymbol{A}$ and the vector $\boldsymbol{b}$.

```
b <- c(3, 1)
a <- matrix(c(1, 2, 3, 1), byrow = TRUE, ncol = 2)
```

For loops can be avoided by performing matrix operations and using the function `sweep`. Differences between the functions `sweep` and `apply` are subtle. Below we give two examples that illustrate the differences.

i) Sum vector $\boldsymbol{b}$ to each row of $\boldsymbol{A}$.

```
test1 <- sweep(a, 2, b, "+")
test2 <- t(apply(a, 1, "+", b))
all(test1 == test2)
```

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

```
test1
```

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

Notice that in this example you need to use `t` (transpose) after `apply` in order to acquire the correct result.

ii) Sum vector $\boldsymbol{b}$ to each column of $\boldsymbol{A}$.

```
test1 <- sweep(a, 1, b, "+")
test2 <- apply(a, 2, "+", b)
all(test1 == test2)
```

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

```
test1
```

```
##      [,1] [,2]
## [1,]    4    5
## [2,]    4    2
```

Now that we have introduced the function `sweep`, let us perform the affine transformations. Notice that

$$\boldsymbol{X}\boldsymbol{A}^\intercal + \mathbf{1}_{100}\boldsymbol{b}^\intercal = \begin{pmatrix} (\boldsymbol{A}\boldsymbol{x}_1 + \boldsymbol{b})^\intercal \\ (\boldsymbol{A}\boldsymbol{x}_2 + \boldsymbol{b})^\intercal \\ \vdots \\ (\boldsymbol{A}\boldsymbol{x}_{100} + \boldsymbol{b})^\intercal \end{pmatrix} = \begin{pmatrix} \boldsymbol{y}_1^\intercal \\ \boldsymbol{y}_2^\intercal \\ \vdots \\ \boldsymbol{y}_{100}^\intercal \end{pmatrix},$$

where $\mathbf{1}_{100} \in \mathbb{R}^{100}$ is a column vector of ones.

```
# First way
y1 <- sweep(x %*% t(a), 2, b, "+")

# Second way
ones <- rep(1, n)
y2 <- x %*% t(a) + ones %*% t(b)

all(y1 == y2)
```

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

Let us check that $\bar{\boldsymbol{y}} = \boldsymbol{A}\bar{\boldsymbol{x}} + \boldsymbol{b}$ and $\boldsymbol{S}_y = \boldsymbol{A}\boldsymbol{S}_x\boldsymbol{A}^\intercal$. We can use any matrix norm $\|\cdot\|$ to check that two matrices $\boldsymbol{X}, \boldsymbol{Y} \in \mathbb{R}^{n\times m}$ are equal since

$$\boldsymbol{X} = \boldsymbol{Y} \iff \|\boldsymbol{X} - \boldsymbol{Y}\| = 0.$$

We can choose to use, e.g., Frobenius norm.

```
norm(colMeans(y1) - (a %*% colMeans(x) + b), type = "F")
```

```
## [1] 8.881784e-16
```

```
norm(cov(y1) - (a %*% cov(x) %*% t(a)), type = "F")
```

```
## [1] 3.552714e-15
```

- *What does affine equivariance mean in practice?*

  Affine equivariant location and scatter functionals behave as expected under coordinate transformations and changes of units. Notice that, for example, componentwise median is **not** an affine equivariant location functional.

f) Be sure that you know the path to the data with respect to your working directory. For example, in this case `data.txt` is located in a different directory than the R script.

```
getwd()
```

```
## [1] "/home/perej/teaching/multivariate/01week"
```

```
data <- read.table("data/data.txt", sep = "\t", header = FALSE)
head(data)
```

```
##           V1         V2        V3        V4
## 1 0.04301325 0.39287882 0.3078980 0.8659341
## 2 0.07597216 0.16503982 0.9537341 0.6011355
```

```
## 3 0.82452548 0.60148149 0.7152086 0.9866722
## 4 0.56774748 0.81009764 0.7085845 0.8704875
## 5 0.57517488 0.12338534 0.3329567 0.9171741
## 6 0.81746640 0.08863711 0.6017773 0.2063420
```

Here we create the function that plots the pairwise scatter plots and centers the data. Additionally, Figure 2 shows the resulting plot.

```
center <- function(x) {
  pairs(x, pch = 19, col = "midnightblue", gap = 0, upper.panel = NULL,
        cex.labels = 1)
  sweep(x, 2, colMeans(x), "-")
}

data_center <- center(data)
```

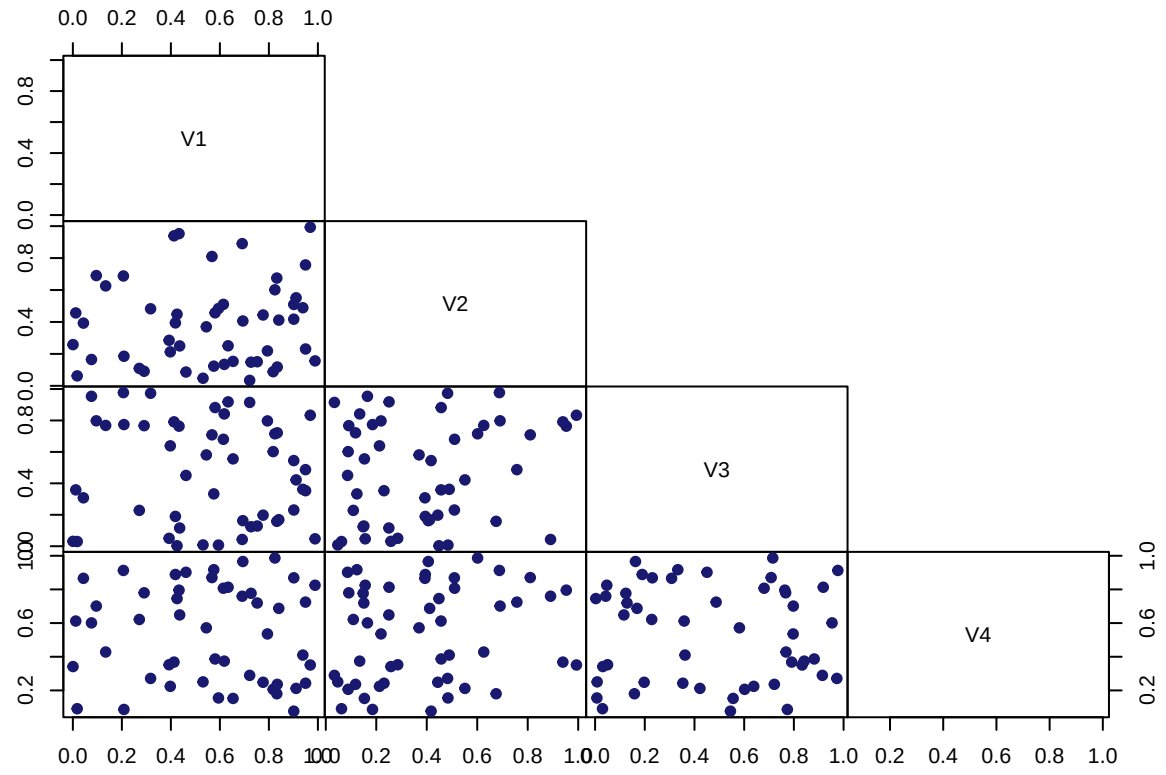

Figure 2: Pairwise scatter plots of variables.

```
colMeans(data_center) # Indeed, the data is centered now
```

```
##            V1            V2            V3            V4
## -1.887379e-17  1.221245e-17 -1.665335e-18 -2.109424e-17
```

Lastly, we calculate sample covariance, sample correlation and the corresponding eigenvalues and eigenvectors.

```
center_cov <- cov(data_center)
center_cor <- cor(data_center)
eigen(center_cov)$values
```

```
## [1] 0.11163469 0.08903608 0.08697829 0.05218629
```

```
eigen(center_cov)$vectors
```

```
##             [,1]        [,2]        [,3]       [,4]
## [1,] -0.24232756  0.51237112  0.76290091  0.3110231
## [2,]  0.30797566  0.55314104  0.04145492 -0.7729602
## [3,]  0.91739205 -0.09949153  0.23313028  0.3068282
## [4,]  0.06942745  0.64931676 -0.60159285  0.4600583
```

```
eigen(center_cor)$values
```

```
## [1] 1.2367328 1.0762947 1.0211537 0.6658188
```

```
eigen(center_cor)$vectors
```

```
##             [,1]       [,2]       [,3]       [,4]
## [1,] -0.02338207  0.7717167 -0.5016233  0.3902315
## [2,]  0.72821948  0.2244426 -0.1739016 -0.6237629
## [3,]  0.47737240 -0.5465891 -0.4787554  0.4941145
## [4,]  0.49118760  0.2352002  0.6992321  0.4631307
```

Demo Problem 2: The Eigenvalues of a Symmetric Matrix

Show that the eigenvalues of a real valued symmetric matrix are always real valued.

Solution

Let A be a symmetric real valued $p \times p$ matrix $\left(A = A^T\right)$. Note that, if the symmetry condition is dropped, A can have complex valued eigenvalues and -vectors. Let λ_i be the ith eigenvalue and v_i the corresponding eigenvector of A.

Definition 1 (Eigenvalues and eigenvectors)**.** A scalar λ_i is called an eigenvalue of $p \times p$ matrix A if there is a nontrivial solution v_i to

$$Av_i = \lambda_i v_i,$$

where v_i is called an eigenvector corresponding to the eigenvalue λ_i.

Here, trivial solutions are obtained if $v_i = 0$ (zero vector) since every scalar λ_i would then satisfy the equation above. First, we take the complex conjugate from both sides,

$$\begin{aligned} \overline{(Av_i)} &= \overline{(\lambda_i v_i)} \\ \Rightarrow A\bar{v}_i &= \bar{\lambda}_i \bar{v}_i, \end{aligned}$$

since A is real valued. Then, we multiply the above with $v_i^\intercal$ from the left side

$$\begin{aligned} v_i^\intercal A\bar{v}_i &= v_i^\intercal \bar{\lambda}_i \bar{v}_i \\ v_i^\intercal A^\intercal \bar{v}_i &= v_i^\intercal \bar{\lambda}_i \bar{v}_i \\ (Av_i)^\intercal \bar{v}_i &= \bar{\lambda}_i v_i^\intercal \bar{v}_i \\ \lambda_i v_i^\intercal \bar{v}_i &= \bar{\lambda}_i v_i^\intercal \bar{v}_i \\ \Rightarrow \left(\lambda_i - \bar{\lambda}_i\right) v_i^\intercal \bar{v}_i &= 0. \end{aligned}$$

Thus we must have that $\lambda_i - \bar{\lambda}_i = 0$ or $v_i^\intercal \bar{v}_i = 0$. However, note that $v_i^\intercal \bar{v}_i = \langle v_i, v_i \rangle$ is the canonical Hermitian inner product. By properties of the inner product we have

- $\langle v_i, v_i \rangle \geq 0$ and

- $\langle v_i, v_i \rangle = 0$ if and only if $v_i = 0$.

Remember that by definition of eigenvectors we assume that $v_i \neq 0$. Thus the option $v_i^\intercal \bar{v}_i = 0$ is not possible. That is, we must have $\lambda_i = \bar{\lambda}_i$.

Remark. Real *nonsymmetric* matrices can have complex eigenvalues. For example, consider the matrix

$$A = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}.$$

```
a <- matrix(c(1, -1, 1, 1), ncol = 2, byrow = TRUE)
eigen(a)$values
```

```
## [1] 1+1i 1-1i
```

Thus the assumption of symmetricity cannot be dropped from the claim of the exercise.

Homework Problem 1: Functions

In this exercise do not use the built-in functions `cov`, `cor`, `cov2cor` or any additional R packages.

a) Create an R function that takes a data matrix $\boldsymbol{X} \in \mathbb{R}^{n \times p}$, $n > p$, as an argument and returns the unbiased estimator of the covariance matrix.

b) Create an R function that takes a full-rank covariance matrix $\boldsymbol{A} \in \mathbb{R}^{p \times p}$ as an argument and returns the square root of the inverse matrix such that $\boldsymbol{A}^{-\frac{1}{2}} \boldsymbol{A}^{-\frac{1}{2}} = \boldsymbol{A}^{-1}$.

c) Create an R function that takes a full-rank covariance matrix $\boldsymbol{A}$ as an argument and returns the corresponding correlation matrix.