HW2

2024-02-07

Given code

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(ggpubr)
# Check whether matrix `x` == `y` for all entries,
# up to the tolerance `tol`.  If not, an error is raised.
AssertMatricesEqual <- function(x, y, tol=1e-9) {
  if (!(all(dim(x) == dim(y)))) {
    stop("The dimensions do not match.")
  }
  err <- max(abs(x - y))
  if (err > tol) {
    stop(sprintf("The error %f is greater than the tolerance.", err))
  }
}


# Generate an `nrow` x `ncol` matrix with random standard normal entries.
GenerateMatrix <- function(nrow, ncol) {
  return(rnorm(nrow * ncol) %>% matrix(nrow, ncol))
}
```

Question 1

Starting from GenerateMatrix(), write a function to generate a random symmetric positive semi-definite matrix of a given size.

```r
GenerateSPSDMatrix <- function(dim) {
  a_mat <- GenerateMatrix(dim, dim)
  s_mat <- 0.5 * (a_mat + t(a_mat))

  eig = eigen(s_mat)
  eig$values[eig$values < 0] <- 0
  psd_mat = eig$vectors %*% diag(eig$values) %*% t(eig$vectors)
  return(psd_mat)
}

CheckSPSDMatrix <- function(a) {
  AssertMatricesEqual(a, t(a))
  eig = eigen(a)
  if (! all(eig$values >= -1e-9)) {
    stop("The matrix is not semi(positive) definite")
  }
  print("The matrix is: ")
  print(a)
  print("The eigen values are: ")
  print(eig$values)
  print("You've passed the test!")
}

psd_mat <- GenerateSPSDMatrix(4)
CheckSPSDMatrix(psd_mat)
```

```
## [1] "The matrix is: "
##            [,1]       [,2]       [,3]       [,4]
## [1,]  0.2250079  0.2582078  0.1971737 -0.2461156
## [2,]  0.2582078  0.3127628  0.2312264 -0.1272211
## [3,]  0.1971737  0.2312264  0.1742775 -0.1688924
## [4,] -0.2461156 -0.1272211 -0.1688924  1.7330515
## [1] "The eigen values are: "
## [1]  1.823227e+00  6.218722e-01  6.661338e-16 -1.934160e-18
## [1] "You've passed the test!"
```

Question 2

Starting from GenerateMatrix(), write a function to generate a random symmetric positive definite matrix of a given size whose smallest eigenvalue is greater than one. (Hint: you can add something to your previous solution.)

```r
GeneratePSDMatrix <- function(dim) {
  a_mat <- GenerateMatrix(dim, dim)
  s_mat <- 0.5 * (a_mat + t(a_mat))

  eig = eigen(s_mat)
  eig$values[eig$values < 1] <- 1
  psd_mat = eig$vectors %*% diag(eig$values) %*% t(eig$vectors)
  return(psd_mat)
}
```

```r
CheckSPDMatrix <- function(a) {
  AssertMatricesEqual(a, t(a))
  eig = eigen(a)
  if (! all(eig$values >= 1 - 1e-9)) {
    stop("The matrix is not semi(positive) definite")
  }
  print("The matrix is: ")
  print(a)
  print("The eigen values are: ")
  print(eig$values)
  print("You've passed the test!")
}

psd_mat <- GeneratePSDMatrix(4)
CheckSPDMatrix(psd_mat)
```

```
## [1] "The matrix is: "
##              [,1]         [,2]         [,3]        [,4]
## [1,]  1.413925244  0.112929609 -0.003794214 -0.40470396
## [2,]  0.112929609  1.051497650  0.008886661 -0.07799538
## [3,] -0.003794214  0.008886661  1.004793330  0.01925771
## [4,] -0.404703962 -0.077995376  0.019257713  1.44648951
## [1] "The eigen values are: "
## [1] 1.857952 1.058754 1.000000 1.000000
## [1] "You've passed the test!"
```

Question 3

Write a function that takes in a symmetric PSD matrix and returns an inverse computed from the eigendecomposition. You may use eigen().

```r
GeneratePSDInverse <- function(s_mat) {
  eig = eigen(s_mat)

  inv_eig_values <-  1 / eig$values

  return(eig$vectors %*% diag(inv_eig_values) %*% t(eig$vectors))
}

CheckInverse <- function(a, b) {
  AssertMatricesEqual(a %*% b, diag(rep(1, nrow(a))))
  print("You've passed the test!")
}

psd_mat <- GeneratePSDMatrix(4)
psd_mat_inv <- GeneratePSDInverse(psd_mat)
CheckInverse(psd_mat, psd_mat_inv)
```

```
## [1] "You've passed the test!"
```

Question 4

Write a function that takes in a symmetric PSD matrix and returns a square root computed from the eigen-decomposition. You may use eigen().

```
GeneratePSDsqrt <- function(s_mat) {
  eig = eigen(s_mat)

  inv_eig_values <-  sqrt(eig$values)

  return(eig$vectors %*% diag(inv_eig_values) %*% t(eig$vectors))
}

CheckSqrt <- function(a, b) {
  AssertMatricesEqual(a %*% a, b)
  print("You've passed the test!")
}

psd_mat <- GeneratePSDMatrix(4)
psd_mat_sqrt <- GeneratePSDsqrt(psd_mat)
CheckSqrt(psd_mat_sqrt, psd_mat)
```

```
## [1] "You've passed the test!"
```

Question 5

Write a function that takes in a symmetric PSD matrix and returns a (possibly non-symmetric) square root computed from the cholesky decomposition. You may use chol().

```
GeneratePSDsqrtCholDecomp <- function(s_mat) {
  cho = chol(s_mat)

  return(t(cho))
}

CheckSqrtTrans <- function(a, b) {
  AssertMatricesEqual(a %*% t(a), b)
  print("You've passed the test!")
}
psd_mat <- GeneratePSDMatrix(4)
psd_mat_sqrt <- GeneratePSDsqrtCholDecomp(psd_mat)
CheckSqrtTrans(psd_mat_sqrt, psd_mat)
```

```
## [1] "You've passed the test!"
```

Question 6

Write a function that takes in a potentially non-square matrix and returns a projection matrix onto the column span.

```r
GenerateProj <- function(X) {
  return(X %*% solve(t(X) %*% X) %*% t(X))
}

CheckProj <- function(P) {
  # it is same to take a random vector and check the projection of the vector is in the column space of
  AssertMatricesEqual(P %*% P, P)
  print("You've passed the test!")
}

X <- GenerateMatrix(5, 3)
G <- GenerateProj(X)
CheckProj(G)
```

```
## [1] "You've passed the test!"
```

Question 7

Write a function that takes in a potentially non-square matrix and returns a projection matrix onto the row span.

```r
GenerateProj_row <- function(X) {
  return(t(X) %*% solve(X %*% t(X)) %*% X)
}

X <- GenerateMatrix(3, 5)
G <- GenerateProj_row(X)
CheckProj(G)
```

```
## [1] "You've passed the test!"
```

Question 8

Write a function that takes in a potentially non-square matrix and returns a projection matrix onto the orthogonal complement of the column span.

```r
GeneratProjCom <- function(X) {
  proj_X <- X %*% solve(t(X) %*% X) %*% t(X)
  proj_X_com <- diag(rep(1, dim(proj_X)[1])) - proj_X

  return(proj_X_com)
}

CheckProjCom <- function(P, PT, y) {
  proj_y = P %*% y
  proj_y_com = PT %*% y
  AssertMatricesEqual(proj_y + proj_y_com, y)
  print("You've passed the test!")
}
X <- GenerateMatrix(5, 3)
P <- GenerateProj(X)
```

```
PC <- GeneratProjCom(X)
y = rnorm(5)
CheckProjCom(P, PC, y)
```

```
## [1] "You've passed the test!"
```

Question 9

Write a function that takes in a potentially non-square matrix and returns an orthonormal basis for its column span.

```
GenerateOrthoBasis <- function(X) {
  X_qr <- qr(X)

  return(qr.Q(X_qr))
}

CheckOrtho <- function(X) {
  AssertMatricesEqual(t(X) %*% X, diag(rep(1,dim(X)[2])))
  print("You've passed the test!")
}
X <- GenerateMatrix(3, 5)
ortho_basis <- GenerateOrthoBasis(X)
CheckOrtho(ortho_basis)
```

```
## [1] "You've passed the test!"
```

Question 10

Write a function that takes in a covariance matrix and returns a draw from the multivariate normal distribution with mean zero and the given covariance matrix. You may use only rnorm(n, mean=0, sd=1) to generate random variables. Verify that you have succeeded using Monte Carlo draws and AssertMatricesEqual() with an appropriate tolerance.

```
GenerateMND <- function(Cov) {
  n <-  dim(Cov)[1]
  A <-  t(chol(Cov))
  u <-  rnorm(n, 0, 1)
  return(A %*% u)
}

CheckMND <- function(trials, Cov) {
  sample_mean <- rep(0, dim(Cov)[1])
  sample_cov <- matrix(0,dim(Cov)[1], dim(Cov)[2])
  for(i in 1:trials) {
    x <- GenerateMND(Cov)
    sample_mean <- sample_mean + x
  }
  sample_mean <-  sample_mean / trials
  # print(sample_mean)
  AssertMatricesEqual(sample_mean, rep(0, dim(Cov)[1]), 1e-2)
```

```r
  for(i in 1:trials) {
    x <- GenerateMND(Cov)
    sample_cov <- sample_cov + x %*% t(x)
  }
  sample_cov <- sample_cov / trials
  # print(sample_cov)
  AssertMatricesEqual(sample_cov, Cov, 1e-2)

  print("You've passed the test!")
}
PSD = GeneratePSDMatrix(5)
CheckMND(500000,PSD)
```

```
## [1] "You've passed the test!"
```