Problem Set 1: Matrix Algebra Review

Stat 154, Fall 2017, Prof. Sanchez

Due date: Tu Sep-12 (before midnight)

The purpose of this assignment is to apply in R some of the introductory material, giving you the opportunity to do some work with matrices and vectors.

Use an R markdown (`.Rmd`) file to write your code and answers. You can *knit* the `Rmd` file as html or pdf. Please submit both your `Rmd` and knitted file to bCourses. Make sure to include your name, and your lab section. No late assignments will be accepted.

Problem 1 (10 pts)

Create the following matrices in R and compute the operations in parts (a) to (e)

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & -3 \\ 4 & 0 & 1 \end{bmatrix}; \qquad \mathbf{B} = \begin{bmatrix} 2 & 3 & 4 \\ -1 & 2 & 0 \end{bmatrix}; \qquad \mathbf{C} = \begin{bmatrix} 0 & 1 & 0 \\ 4 & -1 & -2 \end{bmatrix}$$

a. $\mathbf{A} + \mathbf{B}$

b. $(\mathbf{A} + \mathbf{C}) + \mathbf{B}$

c. $\mathbf{A} - (\mathbf{C} + \mathbf{B})$

d. $-(\mathbf{A} + \mathbf{B})$

e. $(\mathbf{A} - \mathbf{B}) + \mathbf{C}$

```
# it is possible to create matrices A, B, C in other ways
A <- matrix(c(1, 4, 2, 0, -3, 1), nrow = 2)
B <- matrix(c(2, -1, 3, 2, 4, 0), nrow = 2)
C <- matrix(c(0, 4, 1, -1, 0, -2), nrow = 2)

# a)
A + B
```

```
##      [,1] [,2] [,3]
## [1,]    3    5    1
## [2,]    3    2    1
```

```
# b)
(A + C) + B
```

```
##      [,1] [,2] [,3]
## [1,]    3    6    1
## [2,]    7    1   -1
```

```
# c)
A - (C + B)
```

```
##      [,1] [,2] [,3]
## [1,]   -1   -2   -7
## [2,]    1   -1    3
```

```
# d)
- (A + B)
```

```
##      [,1] [,2] [,3]
## [1,]   -3   -5   -1
## [2,]   -3   -2   -1
```

```
# e)
(A - B) + C
```

```
##      [,1] [,2] [,3]
## [1,]   -1    0   -7
## [2,]    9   -3   -1
```

Problem 2 (20 pts)

Assume the following matrix $\mathbf{X}$

$$\mathbf{X} = \begin{array}{c|cccc} & Y & X_1 & X_2 & X_3 \\ a & 2 & 1 & 0 & 9 \\ b & 4 & 2 & 3 & 8 \\ c & 3 & 5 & 2 & 4 \\ d & 7 & 3 & 4 & 5 \\ e & 8 & 7 & 7 & 2 \\ f & 9 & 8 & 7 & 1 \end{array}$$

Note that $\mathbf{X}$ has row-names and column-names.

Create the matrix $\mathbf{X}$ in R (with row and column names), and find, via vector-matrix operations:

a. $\sum Y$

b. $\bar{X_1}$

c. $\sum Y X_2$

d. $\sum X_3^2 - (\sum X_3)^2/6$

e. the mean-centered matrix

f. the (sample) covariance matrix $\mathbf{S}$

You are not allowed to use `sum()`, `apply()`, `sweep()`, nor loops. You can only use these operators: `+`, `-`, `*`, `/` and `%*%`.

It is very likely that you won't have the exact same operations that the answer key shows. The important thing is to check that the obtained values match those displayed below, and that you didn't use `sum()`, `apply()`, `sweep()`, `scale()`, or loops.

```
# it is possible to create matrices X in other ways
X <- rbind(
  c(2, 1, 0, 9),
  c(4, 2, 3, 8),
  c(3, 5, 2, 4),
  c(7, 3, 4, 5),
  c(8, 7, 7, 2),
  c(9, 8, 7, 1)
)
rownames(X) <- letters[1:6]
colnames(X) <- c('Y', 'X1', 'X2', 'X3')
X
```

```
##   Y X1 X2 X3
## a 2  1  0  9
## b 4  2  3  8
## c 3  5  2  4
## d 7  3  4  5
## e 8  7  7  2
## f 9  8  7  1
```

```
# vector of ones
ones <- rep(1, 6)

# a
t(X[ ,'Y']) %*% ones
```

```
##      [,1]
## [1,]   33
```

```
# b
(1/6) * t(X[ ,'X1']) %*% ones
```

```
##          [,1]
## [1,] 4.333333
```

```r
# c
X[ ,'Y'] %*% X[ ,'X2']
```

```
##      [,1]
## [1,]  165
```

```r
# d
(X[ ,'X3'] %*% X[ ,'X3']) - ((t(X[ ,'X3']) %*% ones)^2 / 6)
```

```
##          [,1]
## [1,] 50.83333
```

```r
# e (mean-centered)
I <- diag(1, 6)
Xc <- (I - (1/6) * ones %*% t(ones)) %*% X
Xc
```

```
##         Y         X1         X2         X3
## [1,] -3.5 -3.3333333 -3.8333333  4.1666667
## [2,] -1.5 -2.3333333 -0.8333333  3.1666667
## [3,] -2.5  0.6666667 -1.8333333 -0.8333333
## [4,]  1.5 -1.3333333  0.1666667  0.1666667
## [5,]  2.5  2.6666667  3.1666667 -2.8333333
## [6,]  3.5  3.6666667  3.1666667 -3.8333333
```

```r
# f) sample covariance matrix
S <- (1/5) * t(Xc) %*% Xc
S
```

```
##       Y        X1        X2        X3
## Y   8.3  6.200000  7.700000 -7.500000
## X1  6.2  7.866667  6.666667 -8.733333
## X2  7.7  6.666667  7.766667 -7.633333
## X3 -7.5 -8.733333 -7.633333 10.166667
```

Problem 3 (10 pts)

Let $\mathbf{a}$ and $\mathbf{b}$ be vectors with given lengths and angle θ. Use R to compute their scalar product under the conditions:

a. $\|\mathbf{a}\| = 0.5; \quad \|\mathbf{b}\| = 4; \quad \theta = 45^o$

b. $\|\mathbf{a}\| = 4; \quad \|\mathbf{b}\| = 1; \quad \theta = 90^o$

c. $\|\mathbf{a}\| = 1; \quad \|\mathbf{b}\| = 1; \quad \theta = 120^o$

The scalar or inner product is obtained as:

$$< a, b >= \|a\| \ \|b\| \ cos(\theta)$$

Note that angles in degrees must be converted to radians.

```
# a)
theta <- pi/4
norm_a <- 0.5
norm_b <- 4
norm_a * norm_b * cos(theta)
```

```
## [1] 1.414214
```

```
# b)
theta <- pi/2
norm_a <- 4
norm_b <- 1
norm_a * norm_b * cos(theta)
```

```
## [1] 2.449294e-16
```

```
# c)
theta <- pi/2 + pi/6
norm_a <- 1
norm_b <- 1
norm_a * norm_b * cos(theta)
```

```
## [1] -0.5
```

Problem 4 (10 pts)

Refer to the Gram-Schmidt orthonormalization process described in the following wikipedia entry:

https://en.wikipedia.org/wiki/Gram%E2%80%93Schmidt_process

This procedure is a method for orthonormalizing a set of vectors in an inner product space. In other words, it allows you to find an orthogonal basis for a set of vectors.

Write an R function `proj()` for the *projection operator* given by:

$$proj_{\mathbf{u}}(\mathbf{v}) = \frac{< \mathbf{u}, \mathbf{v} >}{< \mathbf{u}, \mathbf{u} >}\mathbf{u}$$

This projector operator projects the vector $\mathbf{v}$ orthogonally onto the line spanned by vector $\mathbf{u}$. Given two vectors `u` and `v`, you should be able to call your function like:

```
proj(v, u)
```

Test `proj(v, u)` with $\mathbf{u} = (1, 3, 5)$ and $\mathbf{v} = (2, 4, 6)$

Once again, your function `proj()` may not be exactly as mine. But the output must match the one in this answer key.

```
# function: projection operator "v onto u"
proj <- function(v, u) {
  if (length(u) != length(v)) {
    stop("\narguments have different lengths")
  }
  (sum(u * v) / sum(u * u)) * u
}

u <- c(1, 3, 5)
v <- c(2, 4, 6)
proj(v, u)
```

```
## [1] 1.257143 3.771429 6.285714
```

Problem 5 (20 pts)

Once you have your function `proj()`, write R code to apply the Gram-Schmidt orthonormalization procedure to the following sets of vectors:

Start by setting $\mathbf{u_1} = \mathbf{x}$, and report the set of vectors $\mathbf{u_k}$ and the orthonormalized vectors $\mathbf{e_k}$, for $k = 1, 2, 3$.

a. $\mathbf{x} = (1, 2, 3)$; $\mathbf{y} = (3, 0, 2)$; $\mathbf{z} = (3, 1, 1)$

```
# a)
x <- c(1, 2, 3)
y <- c(3, 0, 2)
z <- c(3, 1, 1)

u1 <- x
e1 <- u1 / sqrt(sum(u1 * u1))

u2 <- y - proj(y, u1)
e2 <- u2 / sqrt(sum(u2 * u2))

u3 <- z - proj(z, u1) - proj(z, u2)
e3 <- u3 / sqrt(sum(u3 * u3))
```

```r
# vectors u
u1; u2; u3
```

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

```
## [1]  2.35714286 -1.28571429  0.07142857
```

```
## [1]  0.5148515  0.9009901 -0.7722772
```

```r
# vectors e
e1; e2; e3
```

```
## [1] 0.2672612 0.5345225 0.8017837
```

```
## [1]  0.87758509 -0.47868278  0.02659349
```

```
## [1]  0.3980149  0.6965260 -0.5970223
```

b. $\mathbf{x} = (2, 1); \quad \mathbf{y} = (1, 2); \quad \mathbf{z} = (1, 1)$

```r
# a)
x <- c(2, 1)
y <- c(1, 2)
z <- c(1, 1)

# function: projection operator "v onto u"
proj <- function(v, u) {
  (sum(u * v) / sum(u * u)) * u
}

u1 <- x
e1 <- u1 / sqrt(sum(u1 * u1))

u2 <- y - proj(y, u1)
e2 <- u2 / sqrt(sum(u2 * u2))

u3 <- z - proj(z, u1) - proj(z, u2)
e3 <- u3 / sqrt(sum(u3 * u3))

# vectors u
u1; u2; u3
```

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

```
## [1] -0.6  1.2
```

```
## [1] 2.775558e-17 1.110223e-16
```

```r
# vectors e
e1; e2; e3
```

```
## [1] 0.8944272 0.4472136
```

```
## [1] -0.4472136  0.8944272
```

```
## [1] 0.2425356 0.9701425
```

Problem 6 (20 pts)

The length of a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ in the n-dimensional real vector space $\mathbb{R}^n$ is usually given by the Euclidean norm:

$$\|\mathbf{x}\| = (x_1^2 + x_2^2 + \cdots + x_n^2)^{1/2}$$

In many situations, the Euclidean distance is insufficient for capturing the actual distances in a given space. The class of p-norms generalizes the notion of *length* of a vector and it is defined by:

$$\|\mathbf{x}\|_p = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{\frac{1}{p}}$$

where p is a real number ≥ 1.

Write a function `lp_norm()` that computes the L_p-norm of a vector. This function should take two arguments:

- `x` the input vector
- `p` the value for p
- Give `p` a default value of 1
- Allow the user to specify `p = "max"` to compute the L_∞-norm

You should be able to call `lp_norm()` like this:

```r
lp_norm(x)              # default p = 1
lp_norm(x, p = 2)
lp_norm(x, p = "max")   # L-max norm
```

```r
# function to compute Lp-norms
lp_norm <- function(x, p = 1) {
  if (p == "max") {
    return(max(abs(x)))
  } else {
    return((sum(abs(x)^p))^(1/p))
```

```
  }
}
```

Test your function `lp_norm()` with the following vectors and values for `p`:

a. `zero <- rep(0, 10)` and `p = 1`

b. `ones <- rep(1, 5)` and `p = 3`

c. `u <- rep(0.4472136, 5)` and `p = 2`

d. `u <- -40:0` and `p = 100`

e. `u <- 1:1000` and `p = "max"`

```
# a)
zero <- rep(0, 10)
lp_norm(zero)
```

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

```
# b)
ones <- rep(1, 5)
lp_norm(ones, p = 3)
```

```
## [1] 1.709976
```

```
# c)
u <- rep(0.4472136, 5)
lp_norm(u, p = 2)
```

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

```
# d)
u <- -40:0
lp_norm(u, p = 100)
```

```
## [1] 40.03297
```

```
# e)
u <- 1:1000
lp_norm(u, p = "max")
```

```
## [1] 1000
```

Problem 7 (10 pts)

Show that the set $\{\mathbf{u_1}, \mathbf{u_2}, \mathbf{u_3}\}$ is orthonormal, with:

- $\mathbf{u_1} = \frac{1}{\sqrt{11}}(3, 1, 1)$
- $\mathbf{u_2} = \frac{1}{\sqrt{6}}(-1, 2, 1)$
- $\mathbf{u_3} = \frac{1}{\sqrt{66}}(-1, -4, 7)$

Hint: You need to check that they are orthogonal, and of unit norm.

```
u1 <- (1/sqrt(11)) * c(3, 1, 1)
u2 <- (1/sqrt(6)) * c(-1, 2, 1)
u3 <- (1/sqrt(66)) * c(-1, -4, 7)
```

Check vectors $\mathbf{u}$ are orthogonal: the inner product among them should be zero

```
sum(u1 * u2)
```

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

```
sum(u1 * u3)
```

```
## [1] 4.163336e-17
```

```
sum(u2 * u3)
```

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

Check vectors $\mathbf{u}$ are of unit norm: they should have euclidean norm 1

```
lp_norm(u1, p = 2)
```

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

```
lp_norm(u2, p = 2)
```

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

```
lp_norm(u3, p = 2)
```

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

Problem 8 (20 pts)

For this problem, use the data set `USArrests` that comes in R.

a. Convert `USArrests` as a matrix; this will be the data matrix $\mathbf{X}$. Confirm that `X` is an object of class `"matrix"`

b. Let n be the number of rows of $\mathbf{X}$, and p the number of columns of $\mathbf{X}$

c. Create a diagonal matrix $\mathbf{D} = \frac{1}{n}\mathbf{I}$ where $\mathbf{I}$ is the $n \times n$ identity matrix. Display the output of `sum(diag(D))`.

d. Compute $\mathbf{g} = \mathbf{X}^\mathsf{T}\mathbf{D1}$ where $\mathbf{1}$ is a vector of 1's of length n. Display (i.e. print) $\mathbf{g}$.

e. Calculate the mean-centered matrix $\mathbf{X_c} = \mathbf{X} - \mathbf{1g}^\mathsf{T}$. Display the output of `colMeans(Xc)`.

f. Compute the (population) variance-covariance matrix $\mathbf{V} = \mathbf{X}^\mathsf{T}\mathbf{DX} - \mathbf{gg}^\mathsf{T}$. Display the output of `V`.

g. Let $\mathbf{D}_{1/S}$ be a $p \times p$ diagonal matrix with elements on the diagonal equal to $1/S_j$, where S_j is the standard deviation for the j-th variable. Display only the elements in the diagonal of $\mathbf{D}_{1/S}$

h. Compute the matrix of standardized data $\mathbf{Z} = \mathbf{X_c}\mathbf{D}_{1/S}$ Display the output of `colMeans(Z)` and `apply(Z, 2, sd)`

i. Compute the (population) correlation matrix $\mathbf{R} = \mathbf{D}_{1/S}\mathbf{V}\mathbf{D}_{1/S}$. Display the matrix `R`

j. Confirm that $\mathbf{R}$ can also be obtained as $\mathbf{R} = \mathbf{Z}^\mathsf{T}\mathbf{DZ}$

```r
# a)
X <- as.matrix(USArrests)
is.matrix(X)
```

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

```r
# b)
n <- nrow(X)
p <- ncol(X)

# c)
D <- diag(1/n, n)
sum(diag(D))
```

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

```r
# d)
ones <- rep(1, n)
g <- t(X) %*% D %*% ones
g
```

```
##             [,1]
## Murder     7.788
## Assault  170.760
## UrbanPop  65.540
## Rape      21.232
```

```r
# e)
Xc <- X - ones %*% t(g)
colMeans(Xc)
```

```
##        Murder       Assault      UrbanPop          Rape
##  2.469136e-15 -7.617018e-14 -6.252776e-15 -2.700062e-15
```

```
# f)
V <- t(X) %*% D %*% X - (g %*% t(g))
V
```

```
##             Murder   Assault  UrbanPop      Rape
## Murder    18.59106  285.2411   4.29848  22.53158
## Assault  285.24112 6806.2624 306.02960 508.88368
## UrbanPop   4.29848  306.0296 205.32840  54.65272
## Rape      22.53158  508.8837  54.65272  85.97458
```

```
# g)
stdevs <- apply(X, 2, function(x) sqrt((n-1)/n) * sd(x))
stdevs
```

```
##    Murder   Assault  UrbanPop      Rape
##  4.311735 82.500075 14.329285  9.272248
```

```
Ds <- diag(1/stdevs)

# h)
Z <- Xc %*% Ds
colMeans(Z)
```

```
## [1]  5.652423e-16 -9.328649e-16 -4.256318e-16 -2.917718e-16
```

```
apply(Z, 2, sd)
```

```
## [1] 1.010153 1.010153 1.010153 1.010153
```

```
# i)
R <- Ds %*% V %*% Ds
R
```

```
##            [,1]      [,2]       [,3]      [,4]
## [1,] 1.00000000 0.8018733 0.06957262 0.5635788
## [2,] 0.80187331 1.0000000 0.25887170 0.6652412
## [3,] 0.06957262 0.2588717 1.00000000 0.4113412
## [4,] 0.56357883 0.6652412 0.41134124 1.0000000
```

```
# j)
R2 <- t(Z) %*% D %*% Z
R2
```

```
##            [,1]      [,2]       [,3]      [,4]
## [1,] 1.00000000 0.8018733 0.06957262 0.5635788
## [2,] 0.80187331 1.0000000 0.25887170 0.6652412
## [3,] 0.06957262 0.2588717 1.00000000 0.4113412
```

```
## [4,] 0.56357883 0.6652412 0.41134124 1.0000000
```