Problem Set 1: Solutions

Stat 154, Spring 2018, Prof. Sanchez

Due date: Tue Feb-06 (before midnight)

General Self-Grading Instructions

- Please see the solutions posted on bCourses (in the *Files* folder).
- You will have to enter your score and comments in the *Assingments Comments* section of the correspoind assignment HW01.
- Enter your own scores and comments for (every part) of every problem in the homework on a simple coarse scale:
  - **0** = Didn't attempt or very very wrong,
  - **2** = Got started and made some progress, but went off in the wrong direction or with no clear direction,
  - **5** = Right direction and got half-way there,
  - **8** = Mostly right but a minor thing missing or wrong,
  - **10** = 100% correct.
- Also, enter your total score (out of an overall score of 130 points).
- If there is a cascading error, use a single deduction (don't double or triple or multiple penalize).
- Note: You must justify every self-grade score with a comment. If you are really confused about how to grade a particular problem, you should post on Piazza. This is not supposed to be a stressful process.
- Your self-grades will be due four days after the homework deadline at 11:59 PM sharp (i.e Tue Feb-06).
- We will accept late self-grades up to a week after the original homework deadline for half credit on the associated homework assignment.
- If you don't enter a proper grade by this deadline, you are giving yourself a zero on that assignment.
- Merely doing the homework is not enough, you must do the homework; turn it in on time; read the solutions; do the self-grade; and turn it in on time. Unless all of these steps are done, you will get a zero for that assignment.

Problem 1 (10 pts)

Create the following matrices in R (and display them).

$$\mathbf{X} = \begin{bmatrix} 2 & 3 \\ -1 & 4 \end{bmatrix}; \quad \mathbf{Y} = \begin{bmatrix} 2 & 0 & 1 \\ 1 & -2 & 3 \end{bmatrix}; \quad \mathbf{Z} = \begin{bmatrix} 1 & 1 \\ -1 & 1 \\ 0 & 2 \end{bmatrix}; \quad \mathbf{W} = \begin{bmatrix} 1 & 0 \\ 8 & 3 \end{bmatrix}; \quad \mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

It is possible to create matrices $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$, $\mathbf{W}$, and $\mathbf{I}$ in other ways

```
X <- matrix(c(2, -1, 3, 4), nrow = 2)
X
```

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

```
Y <- matrix(c(2, 1, 0, -2, 1, 3), nrow = 2)
Y
```

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

```
Z <- matrix(c(1, -1, 0, 1, 1, 2), nrow = 3)
Z
```

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

```
W <- matrix(c(1, 8, 0, 3), nrow = 2)
W
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    8    3
```

```
I <- diag(nrow = 2)
I
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

Problem 2 (10 pts)

Use the matrices created in Problem 1 to perform each of the following operations in R. If the indicated operation cannot be performed, explain why.

a. $\mathbf{X} + \mathbf{Y}$

Cannot be performed because $\mathbf{X}$ and $\mathbf{Y}$ have different number of columns

```
X + Y
```

```
## Error in X + Y: non-conformable arrays
```

b. $\mathbf{X} + \mathbf{W}$

```
X + W
```

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

c. $\mathbf{X} - \mathbf{I}$

```
X - I
```

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

d. $\mathbf{XY}$

```
X %*% Y
```

```
##      [,1] [,2] [,3]
## [1,]    7   -6   11
## [2,]    2   -8   11
```

e. $\mathbf{XI}$

```
X %*% I
```

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

f. $\mathbf{X} + (\mathbf{Y} + \mathbf{Z})$

Cannot be performed because $\mathbf{Y}$ and $\mathbf{Z}$ have different dimensions; likewise $\mathbf{X}$ and $\mathbf{Y}$ have different number of columns.

```
X + (Y + Z)
```

```
## Error in Y + Z: non-conformable arrays
```

g. $\mathbf{Y}(\mathbf{I} + \mathbf{W})$

Cannot be performed because $\mathbf{Y}$ and $(\mathbf{I} + \mathbf{W})$ have different number of columns

```
Y %*% (I + W)
```

```
## Error in Y %*% (I + W): non-conformable arguments
```

Problem 3 (10 pts)

Determine whether the following statements are True or False.

a. Every orthogonal matrix is nonsingular.

TRUE

b. Every nonsingular matrix is orthogonal.

FALSE

c. Every matrix of full rank is square.

FALSE

d. Every square matrix is of full rank.

FALSE

e. Every nonsingular matrix is of full rank.

TRUE

Problem 4 (10 pts)

Let $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$ be conformable. Using the properties of transposes, prove that:

$$(\mathbf{XYZ})^\top = \mathbf{Z}^\top\mathbf{Y}^\top\mathbf{X}^\top$$

Note that $(\mathbf{XYZ})^\top = [(\mathbf{XY})\mathbf{Z}]^\top$

By property of transposes, $[(\mathbf{XY})\mathbf{Z}]^\top = \mathbf{Z}^\top(\mathbf{XY})^\top$

Likewise, $(\mathbf{XY})^\top = \mathbf{Y}^\top\mathbf{X}^\top$

Thus: $(\mathbf{XYZ})^\top = \mathbf{Z}^\top\mathbf{Y}^\top\mathbf{X}^\top$

Problem 5 (10 pts)

Consider the eigenvalue decomposition of a symmetric matrix $\mathbf{A}$. Prove that two eigenvectors $\mathbf{v_i}$ and $\mathbf{v_j}$ associated with two distinct eigenvalues λ_i and λ_j of $\mathbf{A}$ are mutually orthogonal; that is, $\mathbf{v_i}^\top\mathbf{v_j} = 0$

By hypothesis, we have:

$$\mathbf{Av_i} = \lambda_i\mathbf{v_i} \tag{1}$$

and

$$\mathbf{Av_j} = \lambda_j\mathbf{v_j} \tag{2}$$

with $\lambda_i \neq \lambda_j$

Taking the transpose of the first equation we have:

$$\mathbf{v_i}^\mathsf{T}\mathbf{A} = \lambda_i \mathbf{v_i}^\mathsf{T}$$

Premultiplying both sides of equation (2) by $\mathbf{v_i}^\mathsf{T}$ yields

$$\mathbf{v_i}^\mathsf{T}\mathbf{A}\mathbf{v_j} = \lambda_j \mathbf{v_i}^\mathsf{T}\mathbf{v_j}$$

Substituting $\mathbf{v_i}^\mathsf{T}\mathbf{A}$ by $\lambda_i \mathbf{v_i}^\mathsf{T}$ we have:

$$\lambda_i \mathbf{v_i}^\mathsf{T}\mathbf{v_j} = \lambda_j \mathbf{v_i}^\mathsf{T}\mathbf{v_j}$$

or

$$(\lambda_i - \lambda_j)(\mathbf{v_i}^\mathsf{T}\mathbf{v_j}) = 0$$

Since $\lambda_i - \lambda_j \neq 0$ by hypothesis, it follows that $\mathbf{v_i}^\mathsf{T}\mathbf{v_j} = 0$

Problem 6

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.

The *projection operator* is 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 $\mathbf{u}$.

6.1 Function `inner_product` (10 pts)

Write an R function `inner_product()` that calculates the inner product $<\mathbf{u}, \mathbf{v}>$ of two vectors (of the same length) $\mathbf{u}$ and $\mathbf{v}$.

Given two vectors `v` and `u`, you should be able to invoke your function like:

```
inner_product(v, u)
```

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

```
# function: inner product
# notice that the function checks that 'u' and 'v'
# have equal lengths (this is not mandatory)
inner_product <- function(v, u) {
  if (length(v) != length(u)) {
    stop("\narguments have different lengths")
  }
  vu <- sum(v * u)
  return(vu)
}

v <- c(1, 3, 5)
u <- c(1, 2, 3)

inner_product(v, u)
```

```
## [1] 22
```

6.2 Function `projection()` (10 pts)

Use your `inner_product()` function to write an R function `projection()` for the projection operator.

Given two vectors `u` and `v`, you should be able to call your function like:

```
projection(v, u)
```

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

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

```
# function: projection operator "v onto u"
projection <- function(v, u) {
  uv <- inner_product(u, v)
  uu <- inner_product(u, u)
  proj <- (uv / uu) * u
  return(proj)
}

v <- c(1, 3, 5)
```

```r
u <- c(1, 2, 3)
projection(v, u)
```

```
## [1] 1.571429 3.142857 4.714286
```

Problem 7 (10 pts)

Refer to the same wikipedia entry of the previous question. Once you have your function `projection()`, write R code to apply the Gram-Schmidt orthonormalization procedure to the following sets of vectors:

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

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 $k = 1, 2, 3$.

```r
# a)
x <- c(3, 2, 1)
y <- c(1, 0, 1)
z <- c(1, 5, 10)

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

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

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

# vectors u
u1; u2; u3
```

```
## [1] 3 2 1

## [1]  0.1428571 -0.5714286  0.7142857

## [1] -4.666667  4.666667  4.666667
```

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

```
## [1] 0.8017837 0.5345225 0.2672612

## [1]  0.1543033 -0.6172134  0.7715167

## [1] -0.5773503  0.5773503  0.5773503
```

Problem 8 (10 pts)

The length of a vector $\mathbf{x} = (x_1, x_2, \dots, x_n)$ 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:

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

Here's one possible way to implement `lp_norm()`

```
# 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))
  }
}
```

Problem 9 (10 pts)

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

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

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

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

b. `ones <- rep(1, 5)` and `p = 2`

```r
# b)
ones <- rep(1, 5)
lp_norm(ones, p = 2)
```

```
## [1] 2.236068
```

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

```r
# c)
u <- c(0.1825742, 0.3651484, 0.5477226, 0.7302967)
lp_norm(u, p = 2)
```

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

d. `u <- 1:500` and `p = 100`

```r
# d)
u <- 1:500
lp_norm(u, p = 100)
```

```
## [1] 508.5663
```

e. `u <- 1:500` and `p = "max"`

```r
# e)
u <- 1:500
lp_norm(u, p = "max")
```

```
## [1] 500
```

Problem 10 (10 pts)

Consider the eigendecomposition of a square matrix $\mathbf{A}$.

a. Prove that the matrix $b\mathbf{A}$, where b is an arbitrary scalar, has $b\lambda$ as an eigenvalue, with $\mathbf{v}$ as the associated eigenvector.

Multiplying both members of $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ by b, we have:

$$(b\mathbf{A})\mathbf{v} = b(\lambda\mathbf{v}) = (b\lambda)\mathbf{v}$$

b. Prove that the matrix $\mathbf{A} + c\mathbf{I}$, where c is an arbitrary scalar, has $(\lambda + c)$ as an eigenvalue, with $\mathbf{v}$ as the associated eigenvector.

The proof is straightforward:

$$\begin{aligned}(\mathbf{A} + c\mathbf{I})\mathbf{v} &= \mathbf{A}\mathbf{v} + c\mathbf{v} \\ &= \lambda\mathbf{v} + c\mathbf{v} \\ &= (\lambda + c)\mathbf{v}\end{aligned}$$

Problem 11 (20 pts)

For this problem, use the data set `state.x77` that comes in R.

a. Select the first five columns of `state.x77` and convert them as a matrix; this will be the data matrix $\mathbf{X}$. Let n be the number of rows of $\mathbf{X}$, and p the number of columns of $\mathbf{X}$

```
# a)
X <- as.matrix(state.x77[ ,1:5])
n <- nrow(X)
p <- ncol(X)
```

b. 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))`.

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

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

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

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

```
##                  [,1]
## Population 4246.4200
## Income     4435.8000
## Illiteracy    1.1700
## Life Exp     70.8786
## Murder        7.3780
```

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

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

```
##    Population        Income    Illiteracy      Life Exp        Murder
## -3.637979e-14  7.275958e-13 -5.950795e-16 -5.968559e-15  7.283063e-16
```

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

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

```
##              Population      Income  Illiteracy    Life Exp      Murder
## Population 19533050.0836 559805.1840  287.010600 -399.685612 5550.253240
## Income       559805.1840 370021.8400 -160.428000  275.049920 -511.456400
## Illiteracy      287.0106    -160.4280    0.364100   -0.471882    1.550140
## Life Exp       -399.6856     275.0499   -0.471882    1.765980   -3.792091
## Murder         5550.2532    -511.4564    1.550140   -3.792091   13.354916
```

f. 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 $\mathbf{D}_{1/S}$

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

```
##  Population      Income  Illiteracy    Life Exp      Murder
## 4419.621034  608.294205    0.603407    1.328902    3.654438
```

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

Ideally, you should obtain the standard deviation dividing by $\sqrt{n}$; if you used `sd()` keep in mind that R calculates it dividing by $\sqrt{n-1}$ and thus your solutions will slightly vary from the answer key.

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

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

```
## [1] -1.755540e-17  1.191824e-15 -9.642287e-16 -4.497301e-15  1.920686e-16
```

```
apply(Z, 2, var)
```

```
## [1] 1.020408 1.020408 1.020408 1.020408 1.020408
```

Again, recall that R calculates variance dividing by $n-1$.

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

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

```
##             [,1]       [,2]       [,3]        [,4]       [,5]
## [1,]  1.00000000  0.2082276  0.1076224 -0.06805195  0.3436428
## [2,]  0.20822756  1.0000000 -0.4370752  0.34025534 -0.2300776
## [3,]  0.10762237 -0.4370752  1.0000000 -0.58847793  0.7029752
## [4,] -0.06805195  0.3402553 -0.5884779  1.00000000 -0.7808458
## [5,]  0.34364275 -0.2300776  0.7029752 -0.78084575  1.0000000
```

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

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

```
##             [,1]       [,2]       [,3]        [,4]       [,5]
## [1,]  1.00000000  0.2082276  0.1076224 -0.06805195  0.3436428
## [2,]  0.20822756  1.0000000 -0.4370752  0.34025534 -0.2300776
## [3,]  0.10762237 -0.4370752  1.0000000 -0.58847793  0.7029752
## [4,] -0.06805195  0.3402553 -0.5884779  1.00000000 -0.7808458
## [5,]  0.34364275 -0.2300776  0.7029752 -0.78084575  1.0000000
```