ABE6933 SML HW3

Directions

Please submit **one PDF** file including all your reports (answer + code + figures + comments; must be easily readable and have file size under a few megabytes) and **one R code script**. The R script is supplementary material to ensure that your code runs correctly. If you are using RMarkdown, please also include your `.Rmd` file.

Place these two (or three) files in a folder, make a zip or rar archive, and submit the archive electronically via Dropbox file request at `tinyurl.com/nbliznyuk-submit-files` (on the landing page, enter your name so that we know it is you and email so that you get a confirmation).

For the full list of rules/policies/expectations, please visit "hw.rules.pdf" document.

Deadline: 08-Oct-2020, 11:59 PM EST.

Practice/Optional Problems (do not submit)

1. Use your solution for the required Typed Problem 1 below to devise an iterative procedure to compute the rank of an $n \times p$ matrix A. Verify your answer using the SVD decomposition (count the number of singular values numerically different from 0; `svd` function in R) or by QR decomposition (`qr` function in R).

2. Explore the use of multiple linear regression/`lm()` function to "solve" general linear systems $Ax = b$, where A is a general $n \times p$ matrix.

3. Suppose one is given a matrix A of size $n \times p$, $n > p$. Use `lm()` function to write your own routine to perform carry out the Gram-Schmidt orthogonalization process thereby obtaining a crude way to the QR decomposition of A.

4. Solving square linear systems $Ax = b$ using LU and QR factorizations when A is invertible:

   (a) The LU factorization/decomposition is available as a part of the `Matrix` library in `R`. Mathematically, $A = P \cdot L \cdot U$, where P is a permutation matrix, L is a lower-triangular matrix and U is an upper-triangular matrix; all matrices are square.

   Given the LU factorization of A, show how to solve the original linear system $Ax = b$.

   (b) The QR factorization/decomposition is available as a part of the base library in `R`. For a square A of full rank, $A = Q \cdot R$, where Q is a matrix with orthonormal columns (i.e., $Q'Q = QQ' = I$) and R is upper-triangular; both matrices are square.

   Given the QR factorization of A, show how to solve the original linear system $Ax = b$.

   Hint to both subproblems: Determine how to solve lower- and upper-triangular square systems by hand, i.e., $Ly = c$ and $Uz = d$. Start with a 2×2 case, then generalize.

Required Problems (for submission)

Typed Problem 1. Let X be an $n \times p$ design matrix with $n > p$ (predictors are in the columns); assume that the column of ones is the first column of X. Argue/show that, if columns of X are not linearly independent, then regressing the ith column of X on all other columns (for $i = 2, ..., p$) will identify this.

Hint: if $R^2 = 1$, what is the residual sum of squares (RSS)? Now relate this to the definition of linear dependence of columns of a matrix. It may help you build intuition if you generate such a matrix and carry out the proposed procedure before providing a conceptual answer.

Typed Problem 2. Verify or refute the following statement: if numerical predictors/covariates (i.e., columns of the design matrix X) are linearly independent, then they are uncorrelated.

Hint 1: make sure you entirely understand the two definitions.

Hint 2: this problem does NOT assume you know how to "prove" mathematical statements.

Typed Problem 3. Multiple linear regression (and its flavors) implemented "by hand" in R.

3.1. Implement by hand an R function myOLS following the interface below.

You are not allowed to call other functions that do statistics (e.g., no lm() calls); you should use linear algebra operations (such as solve(A) to find the inverse of A).

```
myOLS <- function(Y, X, is1 = TRUE) {
# Inputs:
# * Y is the vector of length n of response variables
# * X is an n-by-p matrix of numerical covariates (in columns); p < n
# **  assume the columns of X are linearly independent and
# **  do not include the column for the intercept as a part of the X matrix
# * is1 is a logical "flag" whether the intercept is included; is1 = TRUE by default
# Output:
# the function must return a list L with two elements:
# L[1] will contain the vector of OLS/MLE coefficients, betahat
# L[2] will contain standard errors (i.e., estimated standard deviations) for betahat
}
```

Compare the results with those produced by lm() on the following simulated data:

```
n = 30; set.seed(0); p = 3;
X = matrix(runif(n*p),nrow=n)*2-1;
b = seq(1,p,by=1);
Y = X%*%b + rnorm(n);
fit1 = lm(Y ~ X); summary(fit1); # regression with an intercept
fit0 = lm(Y ~ -1 + X); summary(fit0)  # regression without an intercept
```

3.2. Use your implementation of myOLS to implement polynomial regression with one covariate, i.e., $Y = b_0 + b_1 x + ... + b_k x^k + \epsilon$. Intercept is always included. The interface is below.

```
myPolyReg1 <- function(Y, X1, deg=1) {
# Inputs: same as for myOLS, except
# * X1 is a vector of length n that contain the covariate values (numerical)
# * deg is the degree k (i.e., largest power) of the polynomial fit; k < n ; deg=1 by default.
# Outputs: same as for myOLS
}
```

Compare the results with those produced by lm() on the following simulated data:

```
n = 30; set.seed(0);
X = runif(n)*4-2; # X is uniformly distributed on [-2,2]
Y = 1 + 3*X  -2*X^2 + 1*X^3 + rnorm(n);
fit0 = lm(Y ~ X + I(X^2) + I(X^3)); summary(fit0)
```

3.3. Use your implementation of myOLS to implement a one-way ANOVA model, i.e., regression with a single categorical covariate. The interface for the function is below.

```
myAnova1 <- function(Y, XF, is1=TRUE) {
# Inputs: same as for myOLS, except
# * XF is a vector of length n that contain the covariate values (categorical or "factor")
# Outputs: same as for myOLS
}
```

Compare the results with those produced by lm() on the following simulated data:

```
n = 30; set.seed(0);
XF = rep(c("A","B","C"),each=10)
Y = rnorm(n) + rep(c(1,2,3),each=10)
fit1 = lm(Y ~ XF); summary(fit1) # with an intercept
fit0 = lm(Y ~ -1 + XF); summary(fit0) # without an intercept
```

Typed Problem 4. Exploring the equivalence of OLS and MLE in linear regression model with iid $Normal(0, \sigma^2)$ errors.

Suppose the Nature generates the true data (pairs of (x_i, y_i)) as shown below and then passes the vectors of Y and X values to the statistician who will then fit the simple linear regression model by ordinary least squares (OLS).

```
n = 30; set.seed(0);
X = runif(n)*4-2;
Y = 1 + 3*X + rnorm(n);
fit1 = lm(Y ~ X); summary(fit1) # beta0_hat = 1.0161; beta1_hat = 2.9304, obtained by OLS
```

4.1. Implement by hand the negative log-likelihood for the observed data.

The statistical model is $Y_i = b_0 + b_1 x_i + \epsilon_i$, where ϵ_i are iid $Normal(0, \sigma^2)$ errors; σ^2 is the variance. The interface of the function is as follows:

```
myFullObj <- function(b,sig) {
# Inputs:
# * b is the vector of regression coefficients, b=[b0,b1];
# * sig is the standard dev of errors; sig > 0
# Output: the negative log-likelihood of the observed data (Y given X) evaluated at b and sig
}
```

Your implementation may use the built-in R function dnorm, but this is not required. Make sure that you implement the log-likelihood directly, rather than implementing the likelihood and then applying the log (without explicit mathematical simplifications on your end).

4.2. Assume a fixed known value for sig and minimize the objective function with respect to b only. E.g.,

```
sigKnown = 2; myObj1 <- function(b) {myFullObj(b,sigKnown)};
```

For optimization, specify unconstrained gradient-based search, e.g., method="BFGS".

Carry out the numerical optimization of myObj1 for several different values of sigKnown, examine the solutions and discuss.

Compare your minimizer(s) of myObj1 with the beta_OLS solution produced by lm() and discuss.

4.3. Now, do not assume a known value for sig. Minimize myFullObj jointly with respect to b and sig.

Make sure that you enforce the lower bound constraint on sig (e.g., sig > 10^(-5)) and use the gradient-based optimization routine that can handle such constraints, e.g., "L-BFGS-B". To this end, follow the example from class when we studied the MLE (for iid data).

Compare your solution with that produced by lm() and discuss.

(Optional:) Would you expect the MLE for sig to be equal to the estimate produced by lm ("Residual standard error")? Briefly discuss.