STAT–611 Final Assignment

Due Thursday May 8, 2025 **AT NOON**

This is the final assignment for this class. It serves as the final examination, and is worth x3 a regular homework (see syllabus).

Special instructions:

i. You should turn in this assignment exclusively in **electronic form**.

ii. Please upload your report and computer code to Canvas.

iii. Your report should be in pdf format with the name: `yourname_611_final_HW.pdf`.

iv. Please read the questions carefully. Students who don't answer exactly the questions asked, with the methods asked, will receive deductions even if the answers are correct.

v. You **cannot** collaborate in this assignment, but are free to ask clarifying questions to the instructor or the TA.

Background

A matrix $\mathbf{Q}_{p\times q}$ with $p \geq q$ is called orthonormal if $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}_{q\times q}$. Given a matrix $\mathbf{A}_{p\times q}$ with p linearly independent columns, the *Gram-Schmidt orthogonalization algorithm* allows us to compute $\mathbf{Q}_{p\times q}$ and $\mathbf{R}_{q\times q}$ such that $\mathbf{A}_{p\times q} = \mathbf{QR}$ where $\mathbf{Q}$ is orthonormal and $\mathbf{R}$ is an upper-triangular full-rank matrix.

Let's consider the linear model

$$E[\mathbf{y}] = \mathbf{X}\boldsymbol{\beta},$$

where $\mathbf{y} = (y_1, ..., y_n)^T$ is a column vector of n response observations, $\mathbf{X}$ is a $n \times p$ design matrix with rows given by $\mathbf{x}_i = (x_{i1}, x_{i2}, .., x_{ip})$, and $\boldsymbol{\beta} = (\beta_1, ..., \beta_p)^T$ is a column vector of regression coefficients. According to the theory of linear models, the OLS estimator of $\boldsymbol{\beta}$, $\hat{\boldsymbol{\beta}}$, is given by the solution of

$$\mathbf{X}^T\mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{X}^T\mathbf{y}. \tag{1}$$

This leads to the well known canonical formula $\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$. Unfortunately this solution, while mathematically correct, is known to be numerically unstable, and therefore inadequate for practical use.

A better way of calculating $\hat{\boldsymbol{\beta}}$ is by first calculating the orthonormalization of $\mathbf{X} = \mathbf{QR}$ using the Gram-Schmidt procedure. Replacing in (1):

$$\begin{aligned}
(\mathbf{QR})^T\mathbf{QR}\hat{\boldsymbol{\beta}} &= (\mathbf{QR})^T\mathbf{y} \\
\mathbf{R}^T\cancel{\mathbf{Q}^T\mathbf{Q}}\mathbf{R}\hat{\boldsymbol{\beta}} &= \mathbf{R}^T\mathbf{Q}^T\mathbf{y} \\
\mathbf{R}^T\mathbf{R}\hat{\boldsymbol{\beta}} &= \mathbf{R}^T\mathbf{Q}^T\mathbf{y} \\
\mathbf{R}\hat{\boldsymbol{\beta}} &= \mathbf{Q}^T\mathbf{y} = \mathbf{z}
\end{aligned} \tag{2}$$

where $\mathbf{z}$ is a *known* p-row column vector. The last equality holds because $\mathbf{R}$ is full-rank. Noting that $\mathbf{R}$ is an upper triangular matrix, it is evident that $\hat{\beta}_p = z_p / r_{pp}$. From here, again exploiting the fact that $\mathbf{R}$ is upper triangular, we can back-solve all the coefficients. For $j = p-1, p-2, ..., 1$ we have that:

$$\hat{\beta}_j = \frac{1}{r_{jj}} \left(z_j - \sum_{k=j+1}^{p} r_{jk} \hat{\beta}_k \right)$$

Your Task

In this homework you will implement a linear regression solver using the QR decomposition method. For this, I will provide you with C code that implements the QR decomposition using the modified Gram-Schmidt algorithm that you'll have to call from R. You will then write the rest of the algorithm in R.

The files "QR.cpp" and "QR.h" (available in Canvas) contain an implementation of the QR decomposition via the modified Gram-Schmidt procedure. The prototype of this function, in the file QR.h, is:

```
void QR(double *At_raw, double *Rt_raw, int nrow_A, int ncol_A);
```

Where the arguments are

- `At_raw`: The data of the matrix A, of dimensions $I \times J$, to decompose. These data are represented as a flat array in column-major-order (or equivalently, a flat array containing the transpose of such matrix in row-major-order).
- `Rt_raw`: space allocated *externally* for the R matrix, as a flat array (of length J^2).
- `nrow_A`: the number of rows of A (I).
- `ncol_A`: the number of columns of A (J).

The resulting matrix $\mathbf{Q}$ substitutes the data in `At_raw` and is returned in column-major-order (or as a flat representation of $\mathbf{Q}^T$ in row-major-order). The matrix $\mathbf{R}$ is returned in the space pointed by `Rt_raw`, again as a flat array in column-major-order.

You have to:

1. Write a C function callable from R that invokes the QR() function, so that we can use it from R (note that `QR` cannot be called from R using the `.C()` interface because not all its arguments are pointers). The prototype should be

   ```
   extern"C" void QR_R(double *At_raw, double *Rt_raw, int *I, int *J);
   ```

   Write your function at the bottom of the QR.cpp file, and the prototype in the file QR.h. Note that by including `extern"C"` in the prototype of the function you don't need to include the "#ifdef cplusplus..." clause, but compilation could fail if you try to compile it with a C compiler (not C++). Compile the source code so that the binary file is callable from R.

   Load your module into R and write a wrapper function of the form

```
my_QR <- function(X){
  #your code here
}
```

where `X` is an R matrix. The function should return a named list with the matrices $\mathbf{Q}$ and $\mathbf{R}$.

Test your interface by decomposing a matrix $\mathbf{X}$ of your choosing, and testing that it is the case that $\mathbf{X} = \mathbf{QR}$, and that $\mathbf{Q}^t\mathbf{Q} = \mathbf{I}$. Comment your results. Do you note anything strange in the product $\mathbf{Q}^T\mathbf{Q}$? Explain. (hint: think of the floating point concepts we discussed at the beginning of the course).

The following R functions/operators might be useful:

- `t(a)`: Compute the transpose of the matrix a.
- `a %*% b`: multiply matrices a and b.
- `ncol(a)`, `nrow(a)`: get the number of columns and rows of matrix a, respectively.

Check the help files of those functions for more details.

2. Write an R function for estimating the coefficients of a multiple linear regression

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

using your `my_QR` R function. Your function must have the form:

```
lin_regr <- function(y, X){
....
}
```

where `X` is an $n \times p$ R design matrix, and `y` is a vector containing n response observations. Your function must return a vector containing the estimated regression coefficients. Do some basic input validation in your code (at least check if the arguments have the correct dimensions).

Test your work with the following code:

```
> set.seed(1)
> X <- matrix(rnorm(3*7),ncol = 3)
> y <- X %*% 1:3 + rnorm(7)
> lin_regr(y, X)
```

and compare your output to R's implementation of least squares regression (the "-1" is to avoid fitting an intercept term):

```
> lm(y~X - 1)
```

NOTE: Don't worry about writing efficient R code for this task—this is the type of calculation that *should* be written directly in C. If you're interested in how a solution written in C would look like, I have included an implementation for you to inspect (look at the function `my_lm()` in the file `QR.cpp`).

3. Create a minimal R package called "myLinearModel" that bundles all the code you have worked out so far. Your package should only export the function `lin_regr()` to the user space. Some guidelines, reminders, and hints:

   (a) Please upload your final package as a distribution file created by "`R CMD build`".

   (b) **Make sure that your code works before creating the package**.

(c) For the purpose of this evaluation you're not required to write documentation nor to register the native routines with R. Use `R CMD check` to test your work but, as long as your package works, don't worry about messages requiring you to write documentation, nor about missing entries in `DESCRIPTION`.

(d) You need to place your C code into the sub-directory `src/`.

(e) You need to tell R to load the compiled code by including the directive `useDynLib(myLinearModel)` into the `NAMESPACE` file.

(f) Don't forget to make `lin_regr()` visible to the user by including the `export(lin_regr)` directive in `NAMESPACE`.

(g) Remember that you need to unload your package from R, using `detach()` with the option "`unload=TRUE`", if you want to rebuild it.