Implementing various statistical regression models

Aiden Kenny and Thu Do
MAT338: Computational Mathematics
December 15, 2019

1 Introduction

It is often of interest in many fields to develop some type of model that can make a prediction given a certain observation. For example, if you knew a college student's GPA, you may want to use this information to predict their salary in 10 years time, or if you knew how old a car was, you would like to know how many miles the car has driven. *Statistical learning* involves creating a model based off of a series of observations that have already been made. That is, we use some previously-collected data to develop a model that can make predictions about future observations that have not yet been made. Suppose that we have made several observations on a *predictor*, denoted as X, and a *response*, denoted as Y. We are trying to determine some type of *relationship* between the two, generally taking the form

$$Y = f(X) + \epsilon, \tag{1}$$

where ϵ is some *error* between the predicted response and the true response. The entire purpose of statistical learning is to estimate f, which describes how Y depends on X. If our estimate of f is good enough, one can hope to make accurate future predictions of the response,

$$\hat{Y} = \hat{f}(X), \tag{2}$$

where $\hat{Y}$ represents the predicted response and $\hat{f}$ represents the predicted function.

The scope of statistical learning is immensely vast, in regard to both the types of predictors and responses being observed and the types of models being employed. While developing a high level of sophistication for a multitude of situations takes time and effort, it also shows just how far-reaching and important the field is. One who hopes to be an effective statistician must be skilled in two areas: understanding the theoretical implications of the problem at hand, and having access to efficient tools that allow properer implementation of the theory.

To that end, the major goal of our project was to develop a theoretical understanding of several statistical models, along with the algorithms used to implement them, and to then employ the algorithms ourselves. Along the way, we sought to develop several practical real-world skills, such as learning new coding languages and writing packages that compile all of our functions for easy usage.

The rest of this paper is organized as follows. In section 2 we explore first the theoretical properties of several types of models that we are interested in, and section 3 explain the algorithms used and details about their implementation. Section 4 then gives an example of how to use the `pkg338` package in `R`, our custom package for this project. Section 5 then gives an overall summary of our accomplishments for this project and closes with a reflection on our experience during this process.

2 Theoretical methodology

As mentioned before, there are many types of situations one may encounter in statistical learning, and it is impossible to cover them all. For that reason, we will be limiting our models to those that involve only a *single, continuous predictor* and a *continuous response.* As a motivating example, we will consider the `bonemineral` data set; in this data set, we are hoping to use a patient's age to predict their relative spinal bone mineral density (SBMD), and we can see that both the predictor and the response are continuous.

In general, let N denote the number of observations in the data set. Let x_i denote the ith observed value of the predictor, and let y_i denote the corresponding observed value of the response. We store all of the predictor observations in the vector $\mathbf{x} = (x_1, \ldots, x_N)^T$, and similarly, store all of the response observations in the vector $\mathbf{y} = (y_1, \ldots, y_N)^T$.

In order to estimate f, we must first make some type of assumption about as to what types of models f could embody. This assumption will in turn determine the type of model being fit, and our goal is to determine which *specific* model of that type is the most accurate. We are choosing the optimal model based off of the observed data, so given the general class of functions that f falls under, we will choose the function where the predicted responses are closest to the observed responses. In other words, we are minimizing the difference between the observed and predicted responses, $y_i - f(x_i)$, for each observation $i = 1, \ldots, N$.

2.1 The linear model and least-squares

The simplest model one can assume is a *linear* model that takes the general form

$$f(X) = \beta_0 + \beta_1 X. \tag{3}$$

Here, β_0 and β_1 represent the unknown intercept and slope terms, respectively. A linear model assumes that the relationship between X and Y is linear, meaning that a corresponding change in X will result in the same corresponding change in Y, *regardless* of their values.

As mentioned before, the coefficients β_0 and β_1 are estimated from the data such that the difference between all of the observed and predicted responses are minimized. There are several ways one can go about measuring this difference, but the most common way is to minimize the *residual sum-of-squares* (RSS), which is defined to be

$$\mathrm{RSS}(\beta_0, \beta_1) = \sum_{i=1}^{N} (y_i - \beta_0 - \beta_1 x_i)^2. \tag{4}$$

The idea is to think of RSS as a function where β_0 and β_1 can change freely, and we want to find $\hat{\beta}_0$ and $\hat{\beta}_1$ that *minimize this error function.* It is easiest to solve this optimization problem using linear algebra; let $\boldsymbol{\beta} = (\beta_0, \beta_1)^T$ be a vector containing the two unknown coefficients, and define the matrix $\mathbf{X}$ as

$$\mathbf{X} = \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_N \end{pmatrix}.$$

In other words, the first column of $\mathbf{X}$ is just $\mathbf{1} = (1, \ldots, 1)^T$, a vector of all 1's, and the second column is $\mathbf{x}$. We can now re-write RSS as

$$\mathrm{RSS}(\boldsymbol{\beta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2, \tag{5}$$

where $\|\cdot\|_2$ denotes the ℓ_2-norm. Using the fact that $\|\boldsymbol{v}\|_2^2 = \boldsymbol{v}^T\boldsymbol{v}$ for any vector $\boldsymbol{v}$, we have

$$\begin{aligned} \mathrm{RSS}(\boldsymbol{\beta}) &= (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) = (\mathbf{y}^T - \boldsymbol{\beta}^T\mathbf{X}^T)(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) \\ &= \mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\boldsymbol{\beta} - \boldsymbol{\beta}^T\mathbf{X}^{\mathbf{T}}\mathbf{y} + \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \mathbf{y}^T\mathbf{y} - 2\boldsymbol{\beta}^T\mathbf{X}^{\mathbf{T}}\mathbf{y} + \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}. \end{aligned}$$

We can now take the *gradient* (the multivariate generalization of a derivative) of RSS with respect to $\boldsymbol{\beta}$, which yields

$$\nabla_{\boldsymbol{\beta}}\text{RSS} = -2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\mathbf{y}.$$

The final step is to now set $\nabla_{\boldsymbol{\beta}}\text{RSS} = 0$ and solve for $\boldsymbol{\beta}$; doing so gives us

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}, \tag{6}$$

and so we can now estimate our linear model using the coefficients $\hat{\boldsymbol{\beta}} = (\hat{\beta}_0, \hat{\beta}_1)^T$ as

$$\hat{f}(X) = \hat{\beta}_0 + \hat{\beta}_1 X. \tag{7}$$

Figure 1 shows the fitted linear regression model on the `bonemineral` data set, where the exact estimated model is given by

$$\hat{f}(\texttt{age}) = 11.7367 - 0.4834 \cdot \texttt{age}.$$

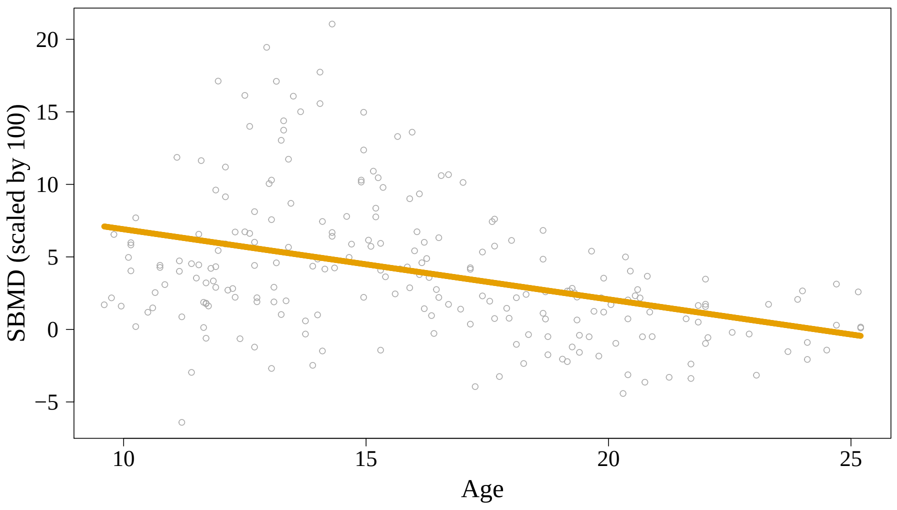

Figure 1: *Fitting a linear regression model to the* `bonemineral` *data set.*

2.2 Polynomial regression and generalizations

While making the assumption that f is linear is often a sufficient approximation of the true relationship f, it is almost never the case in practice that this relationship is linear. In Figure 1 we can see that the linear model may not be *flexible* enough to make very good predictions; around the 10-15 year age range, we can see that the observations deviate quite drastically from our fitted model. Because of this, we would like to estimate a different type of model that can better fit the data.

There are many criterion one could use to define a more "flexible" model, but we will limit the scope of this discussion to *polynomial models*, which take the general form

$$f(X) = \beta_0 + \beta_1 X + \beta_2 X^2 + \cdots + \beta_D X^D = \beta_0 + \sum_{j=1}^{D} \beta_j X^j. \tag{8}$$

Here D is the *degree* of the polynomial being fit, and $D = 1$ gives us the linear model from the previous section. In this case, by letting $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_D)^T$ and

$$\mathbf{X} = \begin{pmatrix} 1 & x_1 & \cdots & x_1^D \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & \cdots & x_N^D \end{pmatrix},$$

we again want to minimize the residual sum-of-squares,

$$\mathrm{RSS}(\boldsymbol{\beta}) = \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{D} \beta_j x_i^j \right)^2 = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2.$$

The derivation of $\hat{\boldsymbol{\beta}}$ is exactly the same as before; we have $\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$, and our estimated model is then given by

$$\hat{f}(X) = \hat{\beta}_0 + \sum_{j=1}^{D} \hat{\beta}_j X^j. \tag{9}$$

In Figure 2, we fit a cubic polynomial ($D = 3$) to the `bonemineral` data set. The estimated model is given by

$$\hat{f}(\texttt{age}) = -101.330749 + 19.916369 \cdot \texttt{age} - 1.174610 \cdot \texttt{age}^2 + 0.021697 \cdot \texttt{age}^3.$$

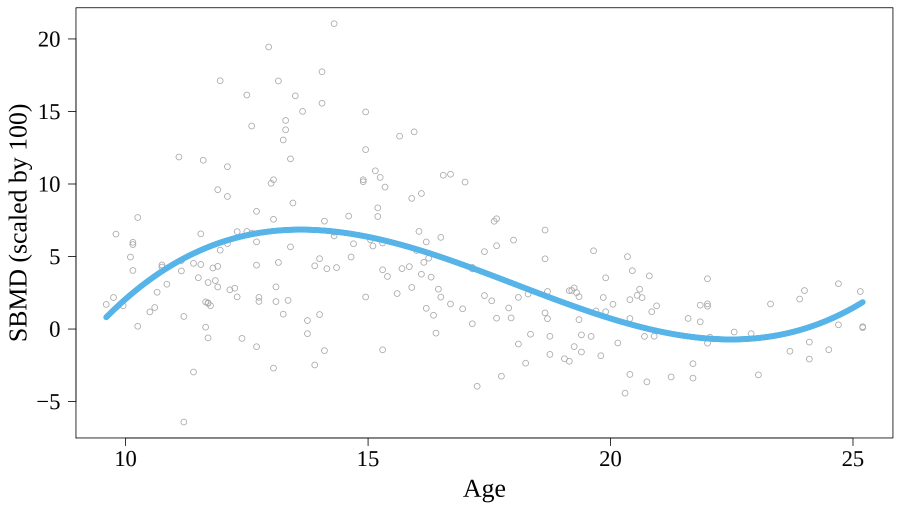

Figure 2: *Fitting a cubic polynomial to the* `bonemineral` *data set.*

Least-squares estimation is an incredibly powerful tool that can be used to fit a surprisingly vast number of non-linear models. This is because even though the models themselves are not linear with respect to X, *they will always be linear with respect to the unknown parameters* $\boldsymbol{\beta}$. Using a technique known as *basis expansion*, we can fit a model

$$f(X) = \sum_{j=0}^{M} \beta_j h_j(X),$$

where $h_j(X)$ is some type of *transformation*, or *expansion*, to the predictor X. It is from these M expansions that the model becomes non-linear with respect to X, but it will always be linear with respect to $\boldsymbol{\beta}$, so least-squares can be used to determine $\hat{f}(X)$. Polynomial regression is a special case of this, where $M = D + 1$ and $h_j(X) = X^j$.

3 Computational methodology

We have seen that estimating linear and polynomial models comes down to performing the matrix calculation

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}.$$

While this solution is sufficient on a theoretical level, in practice more efficient algorithms exist that can be used to compute $\hat{\boldsymbol{\beta}}$ that require fewer operations. Several different topics will first be introduced, and while they may initially seem irrelevant to the problem at hand, we will soon see that when used together they give an effective algorithm for computing $\hat{\boldsymbol{\beta}}$.

3.1 Computing $\hat{\boldsymbol{\beta}}$ for the linear model

In the case of the linear model (3), it can be shown that the coefficient estimates are given by

$$\hat{\beta}_1 = \frac{(\mathbf{x} - \bar{x}\mathbf{1})^T(\mathbf{y} - \bar{y}\mathbf{1})}{(\mathbf{x} - \bar{x}\mathbf{1})^T(\mathbf{x} - \bar{x}\mathbf{1})} \quad \text{and} \quad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x}, \tag{10}$$

where $\bar{x} = \sum_{i=1}^N x_i/N$ and $\bar{y} = \sum_{i=1}^N y_i/N$. That is, in the special case of the linear model, estimating β_0 and β_1 amounts to mainly performing two inner-product calculations, as opposed to a more involved matrix decomposition. The vectors $\bar{x}\mathbf{1}$ and $\bar{y}\mathbf{1}$ correspond to centering the vectors $\mathbf{x}$ and $\mathbf{y}$, respectively, so that they each have a mean of zero.

Algorithm 1 gives a description of how the `linfit338` function computes $\hat{\boldsymbol{\beta}}$. The function then returns the vector $\hat{\boldsymbol{\beta}}$, as well as the estimated linear model (7).

Algorithm 1 (Linear $\hat{\boldsymbol{\beta}}$): Given $\mathbf{x}, \mathbf{y} \in \mathrm{R}^N$, this function, defined as `linfit338`, accepts $\mathbf{x}$ and $\mathbf{y}$ as inputs. It computes $\hat{\boldsymbol{\beta}} = (\hat{\beta}_0, \hat{\beta}_1)^T$ that are to be used for the estimated function $\hat{f}(X)$, and then returns $\hat{\boldsymbol{\beta}}$ and $\hat{f}(X)$ as outputs.

```
function $[\hat{\boldsymbol{\beta}}, \hat{f}(X)] = \texttt{linfit338}(\mathbf{x}, \mathbf{y})$
    $\bar{x} = \sum_{i=1}^N x_i/N$ and $\bar{y} = \sum_{i=1}^N y_i/N$.
    $\mathbf{x}_0 = \mathbf{x} - \bar{x}\mathbf{1}$ and $\mathbf{y}_0 = \mathbf{y} - \bar{y}\mathbf{1}$
    $\hat{\beta}_1 = \mathbf{x}_0^T\mathbf{y}_0/\mathbf{x}_0^T\mathbf{x}_0$
    $\hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x}$
    $\hat{\boldsymbol{\beta}} = (\hat{\beta}_0, \hat{\beta}_1)^T$
    $\hat{f}(X) = \hat{\beta}_0 + \hat{\beta}_1 X$
    return $[\hat{\boldsymbol{\beta}}, \hat{f}(X)]$
```

3.2 Householder matrices

We begin by introducing *Householder matrices*, which are orthogonal and symmetric matrices that have the general form

$$\boldsymbol{\Theta}_j = \mathbf{I} - \theta_j \boldsymbol{v}_j \boldsymbol{v}_j^T, \quad \text{with} \quad \theta_j = \frac{2}{\boldsymbol{v}_j^T \boldsymbol{v}_j}. \tag{11}$$

The vector $\boldsymbol{v}_j$ is the corresponding *Householder vector*, and Householder matrices are defined with respect to a corresponding Householder vector. The Householder vectors are normalized such that $v_{j,1} = 1$, and the remaining $\boldsymbol{v}_{j,(2:N)}$ is known as the *essential part.*

Householder matrices have many applications that are beyond the scope of this paper. For our purposes, we first need to know that for any vector $\boldsymbol{x} \in \mathbb{R}^N$, we can determine a Householder vector $\boldsymbol{v}$ (and the corresponding Householder matrix $\boldsymbol{\Theta}$) such that the vector $\boldsymbol{\Theta x}$ is a multiple of $\boldsymbol{e}_1 = (1, 0, \ldots, 0)^T$. Determining $\boldsymbol{v}$ is quite simple, and it is given by (before normalization)

$$\boldsymbol{v} = \boldsymbol{x} - \|\boldsymbol{x}\|_2 \boldsymbol{e}_1 = \begin{pmatrix} x_1 - \|\boldsymbol{x}\|_2 \\ x_2 \\ \vdots \\ x_N \end{pmatrix};$$

one then normalizes by dividing each entry of $\boldsymbol{v}$ by $v_1 = x_1 - \|\boldsymbol{x}\|_2$. As a result, we have

$$\boldsymbol{\Theta x} = \|\boldsymbol{x}\|_2 \boldsymbol{e}_1 = \begin{pmatrix} \|\boldsymbol{x}\|_2 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

so the vector $\boldsymbol{\Theta x}$ is just $\boldsymbol{e}_1$ scaled by $\|\boldsymbol{x}\|_2$. The extremely important takeaway is that, except for the first entry, *all of the entries of* $\boldsymbol{\Theta x}$ *are zero*; multiplying any vector $\boldsymbol{x}$ by the appropriate Householder matrix will introduce many zeros.

Algorithm 1 below explains the steps of the function `house`. It accepts a vector $\boldsymbol{x}$ as an input and computes an appropriate $\boldsymbol{v}$ (and θ) such that $\boldsymbol{\Theta x}$ is a multiple of $\boldsymbol{e}_1$, where $\boldsymbol{\Theta}$ is the corresponding Householder matrix.

3.3 Householder QR-decomposition

Throughout this report, we will assume that the data matrix $\mathbf{X} \in \mathbb{R}^{N \times M}$ is tall and full rank, i.e. $N > M$ $\text{rank}(\mathbf{X}) = M$. The *QR-decomposition* of the matrix $\mathbf{X}$ is given by

$$\mathbf{X} = \mathbf{QR}. \tag{12}$$

Here, $\mathbf{Q} \in \mathbb{R}^{N \times N}$ is an *orthogonal matrix*, and has several important properties:

- We have $\mathbf{Q}^T\mathbf{Q} = \mathbf{Q}\mathbf{Q}^T = \mathbf{I}$, i.e. $\mathbf{Q}^{-1} = \mathbf{Q}^T$.
- For any vector $\boldsymbol{v} \in \mathbb{R}^N$, we have $\|\boldsymbol{v}\|_p = \|\mathbf{Q}\boldsymbol{v}\|_p = \|\mathbf{Q}^T\boldsymbol{v}\|_p$ for any ℓ_p norm.

Secondly, $\mathbf{R} \in \mathbb{R}^{N \times M}$ is an *upper-triangular* matrix, meaning that all entries below the main diagonal are zero. We can re-write $\mathbf{R}$ as

$$\mathbf{R} = \begin{pmatrix} \mathbf{R}_1 \\ \mathbf{C}_0 \end{pmatrix},$$

where $\mathbf{R}_1 \in \mathbb{R}^{M \times M}$ is now a *square* upper-triangular matrix and $\mathbf{C}_0 \in \mathbf{R}^{(N-M) \times M}$ is a matrix of all zeros. In other words, we are splitting $\mathbf{R}$ into two smaller matrices: the square component $\mathbf{R}_1$, and the appropriate number of zeros underneath.

Householder matrices offer a surprisingly efficient way to determine the QR-decomposition of $\mathbf{X}$, and the fact that Householder matrices induce many zeros in a vector is crucial for determining $\mathbf{R}$. The idea is that we can *iteratively update* the matrix $\mathbf{X}$ by successively multiplying it by appropriate Householder matrices. At the jth step, many zeros are introduced to the jth column of $\mathbf{X}$, and after M steps, we have

$$\boldsymbol{\Theta}_M \cdots \boldsymbol{\Theta}_1 \mathbf{X} = \mathbf{Q}^T\mathbf{X} = \mathbf{R},$$

so we are left with the desired upper-triangular matrix $\mathbf{R}$.

Algorithm 2 (Householder vector): Given $\boldsymbol{x} \in \mathrm{R}^N$, this function, defined as `house`, accepts $\boldsymbol{x}$ as an input. It computes $\boldsymbol{v}$, with $v_1 = 1$, and $\theta \in \mathbb{R}$ such that $\boldsymbol{\Theta} = \mathbf{I} - \theta \boldsymbol{v}\boldsymbol{v}^T$ is orthogonal and $\boldsymbol{\Theta}\boldsymbol{x} = \|\boldsymbol{x}\|_2 \boldsymbol{e}_1$. The function then returns $\boldsymbol{v}$ and θ as outputs.

```
function $[\boldsymbol{v}, \theta] = \texttt{house}(\boldsymbol{x})$
    $N = \texttt{length}(\boldsymbol{x})$, $\sigma = \boldsymbol{x}_{(2:N)}^T \boldsymbol{x}_{(2:N)}$, and $\boldsymbol{v} = (1, \boldsymbol{x}_{(2:N)})^T$
    if $\sigma = 0$ and $x_1 \geq 0$
        $\theta = 0$
    elseif $\sigma = 0$ and $x_1 < 0$
        $\theta = -2$
    else
        $\mu = \sqrt{x_1^2 + \sigma}$
        if $x_1 \leq 0$
            $v_1 = x_1 - \mu$
        else
            $v_1 = -\sigma/(x_1 + \mu)$
        end
        $\theta = 2v_1^2/(\sigma + v_1^2)$
        $\boldsymbol{v} \leftarrow \boldsymbol{v}/v_1$
    return $[\boldsymbol{v}, \theta]$
```

As a motivating example, consider the 5×4 matrix

$$\mathbf{X} = \begin{pmatrix} \mathtt{x} & \mathtt{x} & \mathtt{x} & \mathtt{x} \\ \mathtt{x} & \mathtt{x} & \mathtt{x} & \mathtt{x} \\ \mathtt{x} & \mathtt{x} & \mathtt{x} & \mathtt{x} \\ \mathtt{x} & \mathtt{x} & \mathtt{x} & \mathtt{x} \\ \mathtt{x} & \mathtt{x} & \mathtt{x} & \mathtt{x} \end{pmatrix},$$

where `x` denotes some generic entry. The first step of our algorithm will be to determine a Householder matrix $\boldsymbol{\Theta}_1$ such that

$$\boldsymbol{\Theta}_1 \mathbf{X} = \boldsymbol{\Theta}_1 \begin{pmatrix} \mathtt{x} & \mathtt{x} & \mathtt{x} & \mathtt{x} \\ \mathtt{x} & \mathtt{x} & \mathtt{x} & \mathtt{x} \\ \mathtt{x} & \mathtt{x} & \mathtt{x} & \mathtt{x} \\ \mathtt{x} & \mathtt{x} & \mathtt{x} & \mathtt{x} \\ \mathtt{x} & \mathtt{x} & \mathtt{x} & \mathtt{x} \end{pmatrix} = \begin{pmatrix} \mathtt{x} & \mathtt{x} & \mathtt{x} & \mathtt{x} \\ \mathtt{0} & \mathtt{x} & \mathtt{x} & \mathtt{x} \\ \mathtt{0} & \mathtt{x} & \mathtt{x} & \mathtt{x} \\ \mathtt{0} & \mathtt{x} & \mathtt{x} & \mathtt{x} \\ \mathtt{0} & \mathtt{x} & \mathtt{x} & \mathtt{x} \end{pmatrix}.$$

That is, this first Householder matrix will introduce zeros in the first column. We can see that we have updated the first column to have zeros below the main diagonal. To do this, we first find an augmented Householder matrix $\tilde{\mathbf{\Theta}}_1$ such that

$$\tilde{\mathbf{\Theta}}_1 \begin{pmatrix} \mathtt{x} \\ \mathtt{x} \\ \mathtt{x} \\ \mathtt{x} \\ \mathtt{x} \end{pmatrix} = \begin{pmatrix} \mathtt{x} \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix},$$

which can be easily done by using $\mathtt{house}\big(\mathbf{X}_{(1:5),1}\big)$ to find an appropriate $\tilde{\boldsymbol{v}}_1$ and $\tilde{\theta}_1$. In this first case, we let $\tilde{\mathbf{\Theta}}_1 = \mathbf{\Theta}_1$. For the next step of the algorithm, we want to find $\mathbf{\Theta}_2$ such that

$$\mathbf{\Theta}_2\mathbf{\Theta}_1\mathbf{X} = \mathbf{\Theta}_2 \begin{pmatrix} \mathtt{x} & \mathtt{x} & \mathtt{x} & \mathtt{x} \\ 0 & \mathtt{x} & \mathtt{x} & \mathtt{x} \\ 0 & \mathtt{x} & \mathtt{x} & \mathtt{x} \\ 0 & \mathtt{x} & \mathtt{x} & \mathtt{x} \\ 0 & \mathtt{x} & \mathtt{x} & \mathtt{x} \end{pmatrix} = \begin{pmatrix} \mathtt{x} & \mathtt{x} & \mathtt{x} & \mathtt{x} \\ 0 & \mathtt{x} & \mathtt{x} & \mathtt{x} \\ 0 & 0 & \mathtt{x} & \mathtt{x} \\ 0 & 0 & \mathtt{x} & \mathtt{x} \\ 0 & 0 & \mathtt{x} & \mathtt{x} \end{pmatrix}.$$

As before, we need an augmented Householder matrix $\tilde{\mathbf{\Theta}}_2$ that gives us $\tilde{\mathbf{\Theta}}_2\mathbf{X}_{(2:5),2} = (\mathtt{x}, 0, 0, 0)^T$, and again we can use $\mathtt{house}\big(\mathbf{X}_{(2:5),2}\big)$ to find an appropriate $\tilde{\boldsymbol{v}}_2$ and $\tilde{\theta}_2$. We now set $\mathbf{\Theta}_2 = \operatorname{diag}(1, \tilde{\mathbf{\Theta}}_2)$. This process is to be repeated M times, and after all iterations we will have the upper-triangular matrix $\mathbf{R}$.

Algorithm 2 below describes how to go about implementing the Householder QR-decomposition using the function `qr`. It accepts the basis matrix $\mathbf{X}$ as an input and returns an augmented matrix $\tilde{\mathbf{R}}$, as opposed to only the upper-triangular matrix $\mathbf{R}$. The purpose of this is storage efficiency. The upper-triangular portion of $\tilde{\mathbf{R}}$ is just $\mathbf{R}$, and instead of storing zeros in the entries below the main diagonal, we instead store the essential part of each augmented Household vector $\tilde{\boldsymbol{v}}_j$. For the previous example, if we were to apply $\mathtt{qr}(\mathbf{X})$, the output would be

$$\tilde{\mathbf{R}} = \begin{pmatrix} \mathtt{r} & \mathtt{r} & \mathtt{r} & \mathtt{r} \\ \tilde{v}_{1,2} & \mathtt{r} & \mathtt{r} & \mathtt{r} \\ \tilde{v}_{1,3} & \tilde{v}_{2,2} & \mathtt{r} & \mathtt{r} \\ \tilde{v}_{1,4} & \tilde{v}_{2,3} & \tilde{v}_{3,2} & \mathtt{r} \\ \tilde{v}_{1,5} & \tilde{v}_{2,4} & \tilde{v}_{3,3} & \tilde{v}_{4,2} \end{pmatrix}, \quad \text{where} \quad \mathbf{R} = \begin{pmatrix} \mathtt{r} & \mathtt{r} & \mathtt{r} & \mathtt{r} \\ 0 & \mathtt{r} & \mathtt{r} & \mathtt{r} \\ 0 & 0 & \mathtt{r} & \mathtt{r} \\ 0 & 0 & 0 & \mathtt{r} \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Another difference to note is that in Algorithm 3, we are not replacing some augmented $\tilde{\mathbf{\Theta}}$ with the true $\mathbf{\Theta}$ for each step; we instead only update the appropriate portion of $\mathbf{X}$.

3.3.1 Least-squares using the QR-decomposition

Now that we have developed an efficient algorithm to determine the QR-decomposition of $\mathbf{X}$, it remains to determine exactly *how* such a decomposition can be used to efficiently implement linear regression. We saw before that for a general expansion, the solution to the estimated coefficient vector minimizes

$$\operatorname{RSS}(\boldsymbol{\beta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2,$$

where $\mathbf{y} \in \mathbb{R}^N$ is the response vector. From the QR decomposition of $\mathbf{X}$, we have $\mathbf{Q}^T\mathbf{X} = \mathbf{R}$, and using the second property of $\mathbf{Q}$, RSS can be re-written as

$$\operatorname{RSS}(\boldsymbol{\beta}) = \|\mathbf{Q}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})\|_2^2 = \|\mathbf{Q}^T\mathbf{y} - \mathbf{Q}^T\mathbf{X}\boldsymbol{\beta}\|_2^2 = \|\mathbf{Q}^T\mathbf{y} - \mathbf{R}\boldsymbol{\beta}\|_2^2.$$

We can partition the vector $\mathbf{Q}^T\mathbf{y}$ as

$$\mathbf{Q}^T\mathbf{y} = \begin{pmatrix} \boldsymbol{c} \\ \boldsymbol{d} \end{pmatrix},$$

Algorithm 3 (Householder QR): Given $\mathbf{X} \in \mathrm{R}^{N\times M}$, this function, defined as `qr`, accepts $\mathbf{X}$ as an input. It computes the matrix $\tilde{\mathbf{R}}$, where the entries in and above the main diagonal give the matrix $\mathbf{R}$ from the QR-decomposition $\mathbf{Q}^T\mathbf{X} = \mathbf{R}$. The entries below the main diagonal in the jth column are the essential part of the augmented Householder vector $\tilde{\boldsymbol{v}}_j$ for the corresponding augmented householder matrix $\tilde{\boldsymbol{\Theta}}_j$. The function then returns $\tilde{\mathbf{R}}$ as an output.

```
function $[\tilde{\mathbf{R}}] = \mathtt{qr}(\mathbf{X})$
    for $j = 1 : M$
        $[\boldsymbol{v}, \theta] = \mathtt{house}\big(\mathbf{X}_{(j:M),j}\big)$
        $\boldsymbol{\Theta} = \mathbf{I} - \theta \boldsymbol{v}\boldsymbol{v}^T$
        $\mathbf{X}_{(j:N),(j:M)} \leftarrow \boldsymbol{\Theta}\mathbf{X}_{(j:N),(j:M)}$
        if $j < M$
            $\mathbf{X}_{(j+1:N),j} \leftarrow \boldsymbol{v}_{(2:M-j+1)}$
        end
    end
    return $\mathbf{X} \triangleq \tilde{\mathbf{R}}$
```

Algorithm 4 (Householder LS solution): Given $\mathbf{X} \in \mathrm{R}^{N\times M}$ and $\mathbf{y} \in \mathbb{R}^N$, this function, defined as `hls`, accepts $\mathbf{X}$ and $\mathbf{y}$ as inputs. It first determines $\tilde{\mathbf{R}}$ from the QR-decomposition of $\mathbf{X}$, then updates the response vector $\mathbf{y}$ according to the essential part of each augmented Household vector $\tilde{\boldsymbol{v}}_j$, and finally computes $\hat{\boldsymbol{\beta}}$ via back-substitution. The function then returns $\hat{\boldsymbol{\beta}}$ as an output.

```
function $[\hat{\boldsymbol{\beta}}] = \mathtt{hls}(\mathbf{X}, \mathbf{y})$
    $\tilde{\mathbf{R}} = \mathtt{qr}(\mathbf{X})$
    for $j = 1 : M$
        $\boldsymbol{v} = (1, \tilde{\mathbf{R}}_{(j+1:N),j})^T$
        $\theta = 2/\boldsymbol{v}^T\boldsymbol{v}$
        $\mathbf{y}_{(j:N)} \leftarrow \mathbf{y}_{(j:N)} - \theta(\boldsymbol{v}^T\mathbf{y}_{(j:N)})\boldsymbol{v}$
    end
    $\boldsymbol{c} = \mathbf{y}_{(1:M)}$
    solve $\mathbf{R}\hat{\boldsymbol{\beta}} = \boldsymbol{c}$ via back-substitution
    return $\hat{\boldsymbol{\beta}}$
```

where $\boldsymbol{c} \in \mathbb{R}^M$ and $\boldsymbol{d} \in \mathbb{R}^{(N-M)}$. The RSS then further reduces to

$$\mathrm{RSS}(\boldsymbol{\beta}) = \left\| \begin{pmatrix} \boldsymbol{c} \\ \boldsymbol{d} \end{pmatrix} - \begin{pmatrix} \mathbf{R}_1 \\ \mathbf{C}_0 \end{pmatrix} \boldsymbol{\beta} \right\|_2^2 = \left\| \begin{pmatrix} \boldsymbol{c} - \mathbf{R}_1\boldsymbol{\beta} \\ \boldsymbol{d} \end{pmatrix} \right\|_2^2 = \|\boldsymbol{c} - \mathbf{R}_1\boldsymbol{\beta}\|_2^2 + \|\boldsymbol{d}\|_2^2.$$

Minimizing RSS($\boldsymbol{\beta}$) then amounts to simply determining $\hat{\boldsymbol{\beta}}$ such that $\mathbf{R}_1\hat{\boldsymbol{\beta}} = \boldsymbol{c}$, as this will give us $\|\boldsymbol{c} - \mathbf{R}_1\hat{\boldsymbol{\beta}}\|_2^2 = 0$. Because $\mathbf{R}_1$ is upper-triangular, this can easily be solved by using back-substitution.

Algorithm 4 explains this process. The first step is to determine the corresponding $\tilde{\mathbf{R}}$ from the data matrix $\mathbf{X}$, which is done using qr($\mathbf{X}$). Next, we need to determine $\mathbf{Q}^T\mathbf{y}$ in order to get $\boldsymbol{c}$, and this is why the essential part of each Householder matrix from the QR-decomposition was stored in $\tilde{\mathbf{R}}$ to begin with. Just like with Algorithm 3, instead of directly computing $\mathbf{Q}^T\mathbf{y}$, we can iteratively update $\mathbf{y}$ using each Householder vector. Finally, back-substitution can be used to solve $\mathbf{R}_1\hat{\boldsymbol{\beta}} = \boldsymbol{c}$, which is a system of M equations with M unknowns.

4 The pkg338 package

As stated in the introduction, while it is important to understand the theory behind the models being used, it is just as important to be able to easily and effectively implement the models. We saw that actually understanding what linear regression was did not take much explanation, and one could argue that someone using the method should not have to necessarily fully understand *how* it is implemented. That is, they would only need to know the theory behind linear regression, and would not need to know the Householder matrices and QR-decompositions being performed under the hood.

As an aside, the programming language R is an extremely popular and powerful tool for doing statistics and data science, by far more popular than MATLAB. Because R is open-source (and free), in addition to the commands installed in base R, a user has access to over 10,000 *packages* written by others that, once installed, provide a simple interface to use commands offered by the package, including fitting different types of functions and performing more efficient computation and data-mining.

For this project, we decided to *write our own package* that would gives users an easy way to fit linear and polynomial models to a data set with a continuous predictor and response. We wrote the algorithms for implementing the models using C++, and then built the package using the algorithms in R. The point was to make implementing the models extremely easy with the use of a single command, meaning despite all of the linear algebra and computation going on in the background, *the unfamiliar user would not need to know.*

To motivate the effectiveness of our package, we will give a simple example of using a car's horsepower (hp) to predict miles per gallon (mpg). We assume the user has R, and preferably RStudio as well, installed and ready for use. The user must then manually download the pkg338 package and put it in the win-library directory. That is, their directory path will be as follows:

```
> Documents > R > win-library > 3.5 > pkg338.
```

For Mac users, the equivalent path would be

```
Library > Frameworks > R.framework > Resources > library.
```

The data set we are using is actually a subset of the Auto data set. To install this data set and get our predictor and response vector, one would type the following code into their console:

```
library(ISLR)
x = Auto$horsepower
y = Auto$mpg
```

One of the packages two main functions, linfit338, was designed to easily implement a linear model. With the predictor x and the response y, one types the following code into the console:

```
lmod = linfit338(x, y)
```

With just a single line of code, we have fit a linear model to the data! To see the coefficients of the regression line, type the following:

```
lmod$coef
[1] 39.9358610 -0.1578447
```

This output tells us that our estimated model is given by

$$\hat{f}(\texttt{hp}) = 39.9358610 - 0.1578447 \cdot \texttt{hp}.$$

The function also nicely outputs vectors that store finely spaced `x` values and their predicted values `y` that we can use to plot the regression line. To make such a plot, type the following:

```
plot(x, y, col = "darkgrey", las = 1, xlab = "Horsepower", ylab = "MPG")
lines(lmod$X, lmod$Y, col = "#CC79A7", lwd = 3)
```

Similarly, we can fit a polynomial using the `polyfit338` command, the other major function of the `pkg338` package. This function takes in three inputs: the predictor vector `x`, the response vector `y`, and an integer `d` indicating the degree of the polynomial being fit. If one wanted to fit a *quadratic polynomial* to the data, they would use the following code:

```
pmod = polyfit338(x, y, 2)
pmod$coef
[1] 56.900099702 -0.466189630  0.001230536
```

This output tells us that our estimated model is given by

$$\hat{f}(\texttt{hp}) = 56.900099702 - 0.466189630 \cdot \texttt{hp} + 0.001230536 \cdot \texttt{hp}^2.$$

As before, we can also plot the regression line using the vectors outputted by the function:

```
plot(x, y, col = "darkgrey", las = 1, xlab = "Horsepower", ylab = "MPG")
lines(pmod$X, pmod$Y, col = "#F0E442", lwd = 3)
```

Figure 3 shows the resulting plots from our code. We should also note that the models that our functions output are *identical* to the models that one would get using the `lm` function in base `R`, confirming that our package has indeed correctly implemented the algorithms.

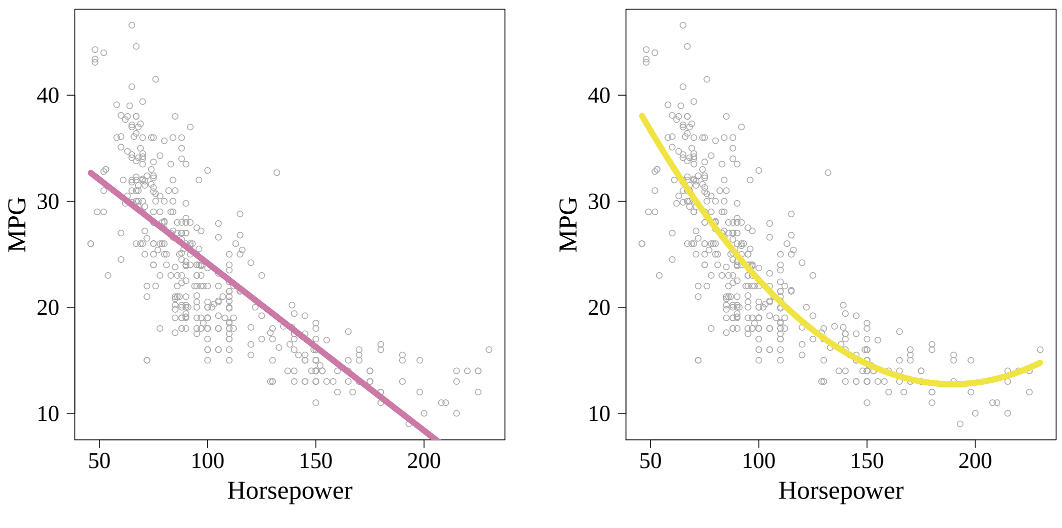

Figure 3: *Fitting a linear and quadratic model to the* `Auto` *data set.*

5 Results and discussion

By the end of the project, we have effectively implemented least-squares using the Householder QR-decomposition, as well as created a user-friendly interface for implementing linear and polynomial models. Throughout the project, we gained valuable practical experience in implementing and exporting complicated algorithms first-hand. It is one thing to understand how to use functions in order to fit a certain type of model, but to be able to implement our own from scratch was an entirely new process for both of us.

While we both found the experience extremely worthwhile, we admit that it was difficult to carry out this project on our own. As mentioned before, while understanding the theory about the models themselves is not too difficult, actually understanding how they are effectively implemented requires much more work, and involves more advanced topics that one would not even think as relevant. More specifically, writing an `R` package that can work in tandem with the algorithms written in `C++` was a daunting hurdle; Aiden had never written in `C++` before, and neither of us had any clue as to how one would get the two languages to work with each other. Being our first time, the package we build is surely not *optimal* in the sense of functions being called and exported, but it is functional, which is what we wanted. The only way to get better at this is to practice more, which we will surely continue to do.

It is interesting to see how much work goes into effectively implementing our models, which are both rather simple compared to other groups. However, we again stress that linear regression is powerful tool that can be used to fit a wide class of basis expansion models. This is one of the "shortcomings" of our project: using the idea of linear regression, we wanted to fit an even more flexible model known as a *cubic spline*. Unfortunately we were unable to effectively implement that model in our package, but we view this less as a failure and more of a reality check. Given the time that we had to work on this project and our inexperience with implementing algorithms, we are very satisfied with our accomplishments.

In conclusion, we found this project to be extremely fun and worthwhile. We both gained valuable practical experience implementing algorithms and building packages in `R`, two extremely useful skills for a career in statistics and data science.

Thank you for a great semester!

6 Bibliography

Information about the theoretical implications for linear and polynomial regression can be found in chapter 3 of Hastie, Tibshirani, and Friedman (2009), and more information about basis expansions and cubic splines are presented in chapter 5. Chapter 5 of Golub and van Loan (2013) presents all of the necessary information about Householder matrices, Householder QR-decomposition, and linear regression one would need to implement the model as we did. Finally, Wickham (2019) gives a good overview of using the `Rcpp` package in `R` to correctly use `C++` code within `R`. We would also like to thank Dirk Eddelbuettel, author of the `Rcpp` package, for his helpful comments on building packages with `Rcpp`.

References

Golub, G. H., & van Loan, C. F. (2013). *Matrix computations* (Fourth ed.). JHU Press. Retrieved from `http://www.cs.cornell.edu/cv/GVL4/golubandvanloan.htm`

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning* (Second ed.). Springer, New York. Retrieved from `https://doi.org/10.1007/978-0-387-84858-7` (Data mining, inference, and prediction) doi: 10.1007/978-0-387-84858-7

Wickham, H. (2019). *Advanced r, second edition.* CRC Press. Retrieved from `https://books.google.com/books?id=5PycDwAAQBAJ`