Implementing various statistical regressions

Aiden Kenny and Thu Do
MAT338: Computational Mathematics
December 11, 2019

Introduction

In statistics, we often want to find some type of relationship between two different *variables*, where one variable depends on the other.

- e.g. A doctor wants to find a relationship between a person's age and the bone mineral density in their spine (SBMD).
- Let X be the independent variable, known as a *predictor*, and let Y be the dependent variable, known as a *response*.

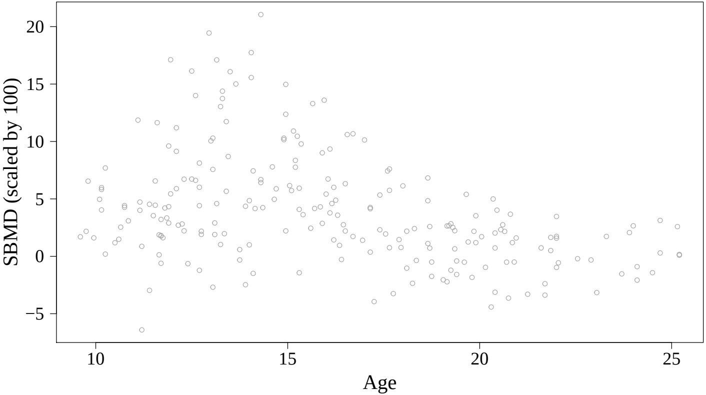
The relationship is generally given by

$$Y = f(X) + \epsilon,$$

where f is the true relationship and ϵ is some error.

- Our goal is to estimate f with some $\hat{f}$.
- We can then make future predictions $\hat{Y} = \hat{f}(X)$.

Example: bone mineral density



$X =$ `age` and $Y =$ `SBMD`. We want to find some kind of relationship between them.

Goals

Throughout this project, we hoped to accomplish the following:

- Understand the theory behind the models being implemented.
- Understand the algorithms used to implement them.
- Gain practical experience coding and implementing algorithms.

Estimating f

In order to estimate f, we need to make some assumptions about its general form.

- e.g. We can assume that f is a linear function.
- The resulting models will be different depending on our assumptions.

Once we decide what kind of model we want, we use collected data to fit the model to the data.

- We make our model so that it is a good estimate of the data we already have.
- This will make the model accurate for future unseen data.

For this project, we limited ourselves to a continuous response and a continuous predictor.

The linear model

The easiest assumption to make is that the relationship between X and Y is *linear*.

A linear model takes the general form

$$f(X) = \beta_0 + \beta_1 X, \tag{1}$$

where β_0 and β_1 are unknown parameters.

Aside: *degrees of freedom* (df) are the number of parameters that can vary without violating any imposed constraints.

- For a linear model, df $= 2$.

Fitting this model is known as *linear regression*.

Estimating β_0 and β_1

We estimate the coefficients with $\hat{\beta}_0$ and $\hat{\beta}_1$. Our estimated function is then

$$\hat{f}(X) = \hat{\beta}_0 + \hat{\beta}_1 \cdot X. \tag{2}$$

The optimal values of $\hat{\beta}_0$ and $\hat{\beta}_1$ will make the difference between the actual values, Y, and the predicted values, $\hat{Y} = \hat{f}(X)$, as small as possible.

- We want to minimize the error between Y and $f(X)$ for each observation.
- Let $\mathbf{x} = (x_1, \ldots, x_N)^T$ be the observed predictor and let $\mathbf{y} = (y_1, \ldots, y_N)^T$ be the observed response, with N total observations.

Residual sum-of-squares

The standard way to measure error is the *residual sum-of-squares* (RSS), given by

$$\mathrm{RSS}(\beta_0, \beta_1) = \sum_{i=1}^{N} (y_i - \beta_0 - \beta_1 x_i)^2 = \|\mathbf{y} - \beta_0 \mathbf{1} - \beta_1 \mathbf{x}\|_2^2. \tag{3}$$

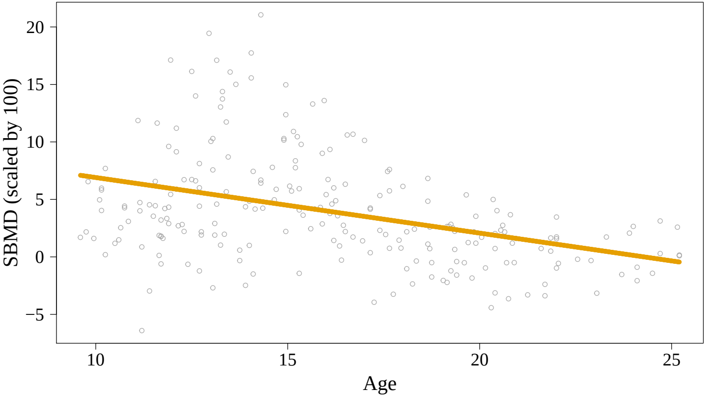
We want to find β_0 and β_1 that minimizes RSS.

It can be shown that

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

where $\bar{x} = \sum x_i / N$ and $\bar{y} = \sum y_i / N$.

Example: bone mineral density



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

Polynomial regression

We can fit a degree D polynomial to the data in a similar fashion, which takes the general form

$$f(X) = \beta_0 + \beta_1 X + \beta_2 X^2 + \cdots + \beta_D X^D = \beta_0 + \sum_{j=1}^{D} \beta_j X^j. \tag{4}$$

We have $\text{df} = D + 1$ unknown parameters.

Let $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_D)^T$ and let

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

New residual sum-of-squares

The residual sum-of-squares for a polynomial is given by

$$\mathrm{RSS}(\boldsymbol{\beta}) = \sum_{i=1}^{N}\left(y_i - \beta_0 - \sum_{j=1}^{D}\beta_j x_i^j\right)^2 = \|\mathbf{y} - \mathbf{H}\boldsymbol{\beta}\|_2^2. \tag{5}$$

We now want to find $\boldsymbol{\beta}$ that minimizes RSS.

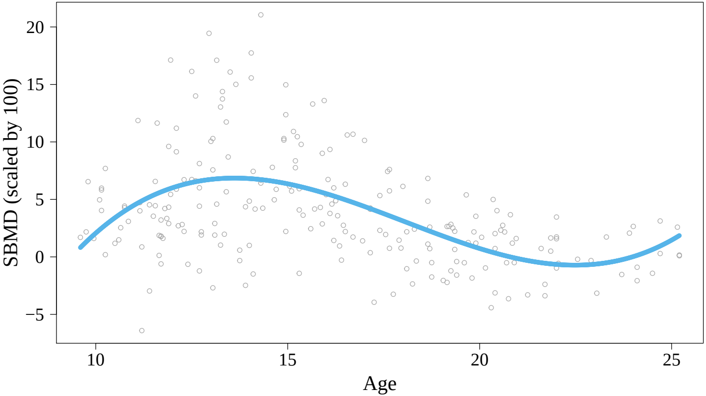
It can be shown that

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{y},$$

and our estimated model is then given by

$$\hat{f}(X) = \hat{\beta}_0 + \sum_{j=1}^{D}\hat{\beta}_j X^j. \tag{6}$$

Example: bone mineral density



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

QR-decomposition

Implementing linear regression is very straightforward. But finding $\hat{\boldsymbol{\beta}}$ for polynomial regression is more involved.

The matrix $\mathbf{H} \in \mathbb{R}^{N \times M}$ (where $M = \mathrm{df}$) can be expressed as

$$\mathbf{H} = \mathbf{QR}, \tag{7}$$

where

- $\mathbf{Q} \in \mathbb{R}^{N \times N}$ is an *orthogonal* matrix.
- $\mathbf{R} \in \mathbb{R}^{N \times M}$ is an *upper-triangular* matrix.

Because $\mathbf{Q}$ is orthogonal, we have $\mathbf{Q}^{-1} = \mathbf{Q}^T$, and so

$$\mathbf{Q}^T \mathbf{H} = \mathbf{R}.$$

Note: we are assuming that $\mathbf{H}$ is full rank, i.e. $\mathrm{rank}(\mathbf{H}) = M$.

Householder matrices

A *Householder matrix* takes the general form

$$\boldsymbol{\Theta}_j = \mathbf{I} - \theta_j \boldsymbol{v}_j \boldsymbol{v}_j^T, \quad \text{where} \quad \theta_j = \frac{2}{\boldsymbol{v}_j^T \boldsymbol{v}_j}. \tag{8}$$

The vector $\boldsymbol{v}_j$ is what defines $\boldsymbol{\Theta}_j$, and is normalized such that $v_1 = 1$.

The idea is that for some vector $\boldsymbol{x}$, we want to find $\boldsymbol{v}_j$ such that

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

so multiplying a vector by a Householder matrix introduces many zeros.

The vector before normalization is given by $\boldsymbol{v}_j = \boldsymbol{x} - \|\boldsymbol{x}\|_2 \boldsymbol{e}_1$.

Using Householder matrices for QR-decomposition

We can use appropriate Householder matrices $\boldsymbol{\Theta}_1, \ldots, \boldsymbol{\Theta}_M$ to *update* $\mathbf{H}$ until it it upper triangular. Multiplying by $\boldsymbol{\Theta}_j$ will introduce zeros to the jth column.

Example: if $\mathbf{H} \in \mathbb{R}^{(5\times4)}$, the first two steps are as follows:

$$\mathbf{H} \leftarrow \boldsymbol{\Theta}_1\mathbf{H} = \boldsymbol{\Theta}_1 \begin{pmatrix} \boxed{\begin{matrix} \mathtt{x} \\ \mathtt{x} \\ \mathtt{x} \\ \mathtt{x} \\ \mathtt{x} \end{matrix}} & \begin{matrix} \mathtt{x} & \mathtt{x} & \mathtt{x} \\ \mathtt{x} & \mathtt{x} & \mathtt{x} \\ \mathtt{x} & \mathtt{x} & \mathtt{x} \\ \mathtt{x} & \mathtt{x} & \mathtt{x} \\ \mathtt{x} & \mathtt{x} & \mathtt{x} \end{matrix} \end{pmatrix} = \begin{pmatrix} \boxed{\begin{matrix} \mathtt{x} \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}} & \begin{matrix} \mathtt{x} & \mathtt{x} & \mathtt{x} \\ \mathtt{x} & \mathtt{x} & \mathtt{x} \\ \mathtt{x} & \mathtt{x} & \mathtt{x} \\ \mathtt{x} & \mathtt{x} & \mathtt{x} \\ \mathtt{x} & \mathtt{x} & \mathtt{x} \end{matrix} \end{pmatrix}.$$

$$\mathbf{H} \leftarrow \boldsymbol{\Theta}_2\mathbf{H} = \boldsymbol{\Theta}_2 \begin{pmatrix} \mathtt{x} & \mathtt{x} & \mathtt{x} & \mathtt{x} \\ 0 & \boxed{\begin{matrix} \mathtt{x} \\ \mathtt{x} \\ \mathtt{x} \\ \mathtt{x} \end{matrix}} & \begin{matrix} \mathtt{x} & \mathtt{x} \\ \mathtt{x} & \mathtt{x} \\ \mathtt{x} & \mathtt{x} \\ \mathtt{x} & \mathtt{x} \end{matrix} \\ 0 & & \\ 0 & & \\ 0 & & \end{pmatrix} = \begin{pmatrix} \mathtt{x} & \mathtt{x} & \mathtt{x} & \mathtt{x} \\ 0 & \boxed{\begin{matrix} \mathtt{x} \\ 0 \\ 0 \\ 0 \end{matrix}} & \begin{matrix} \mathtt{x} & \mathtt{x} \\ \mathtt{x} & \mathtt{x} \\ \mathtt{x} & \mathtt{x} \\ \mathtt{x} & \mathtt{x} \end{matrix} \\ 0 & & \\ 0 & & \\ 0 & & \end{pmatrix}.$$

We can store the essential values of $\boldsymbol{v}_j$ in the zeros of the jth column of $\mathbf{R}$.

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

We could compute

$$\mathbf{Q} = \boldsymbol{\Theta}_1 \cdots \boldsymbol{\Theta}_M,$$

but we do not need to. We only need $\boldsymbol{v}_j$ and $\mathbf{R}$ going forward.

Using QR-decomposition for least-squares

Let

$$\mathbf{R} = \begin{pmatrix} \mathbf{R}_1 \\ \mathbf{0} \end{pmatrix} \quad \text{and} \quad \mathbf{Q}^T \mathbf{y} = \begin{pmatrix} \boldsymbol{c} \\ \boldsymbol{d} \end{pmatrix}.$$

$\mathbf{R}_1$ is the square $(M \times M)$ part of $\mathbf{R}$ and $\boldsymbol{c}$ is the first M elements of $\mathbf{Q}^T \mathbf{y}$. We can get $\boldsymbol{c}$ by updating $\mathbf{y}$ using each $\boldsymbol{v}_j$ from $\tilde{\mathbf{R}}$.

It can be shown that RSS can be re-written as

$$\mathrm{RSS}(\boldsymbol{\beta}) = \|\boldsymbol{c} - \mathbf{R}_1 \boldsymbol{\beta}\|_2^2 + \|\boldsymbol{d}\|_2^2. \tag{9}$$

We find $\hat{\boldsymbol{\beta}}$ such that $\mathbf{R}_1 \hat{\boldsymbol{\beta}} = \boldsymbol{c}$, which is done via back-substitution.

Implementing the algorithms

For this project, we did the following:

1. We first wrote the algorithms used to implement linear and polynomial regression using `C++`.
2. We then created a package in `R`, called `pkg338`, that contained all of the functions and sample data sets for easy use.

The package allows the user to fit a linear function or a degree D polynomial to any data set with a continuous predictor and a continuous response.

- Let `x` and `y` be the vectors containing the observations for the predictor and response.
- `linfit338(x,y)` returns the coefficients of the linear model and a function that can be used to make predictions.
- `polfit338(x,y,d)` returns the coefficients of the polynomial model and a function that can be used to make predictions.

In the Figures presented before, the predictions were made using the `linfit338` and `polfit338` functions, respectively, and the results were plotted using base `R`.

Example: algorithm for Household vectors in C++

This function **house** takes a vector $\boldsymbol{x}$ as an input and outputs a vector $\boldsymbol{v}$ and a scalar θ, which define the corresponding Householder matrix $\boldsymbol{\Theta}$.

```
List house(NumericVector x) {
  int n = x.size();
  double sig = inprod(x,x) - pow(x[1], 2.0);
  NumericVector v(n);
  v[0] = 1;
  for (int i = 1; i < n; i++) {
    v[i] = x[i];
  }
  double theta = 0;
  if ((sig == 0) && (x[0] >= 0)) {
    theta = 0;
  } else if ((sig == 0) && (x[0] < 0)) {
    theta = -2;
  } else {
    double mu = sqrt(pow(x[0], 2.0) + sig);
    if (x[0] <= 0) {
      v[0] = x[0] - mu;
    } else {
      v[0] = (-1 * sig) / (x[0] + mu);
    }
    theta = 2 * pow(v[0], 2.0) / (sig + pow(v[0], 2.0));
    for (int i = 1; i < n; i++) {
      v[i] /= v[0];
    }
    v[0] = 1;
  }
  List vt;
  vt["v"] = v;
  vt["theta"] = theta;
  return vt;
}
```

Obstacles

During the project, we had some general problems that we had to overcome:

- Aiden had to learn some `C++`.
- We both had to learn how to write a package in `R` that worked with `C++`.
- Understanding and implementing the algorithms was difficult.

Because of these issues, there were some topics we wanted to get to that we have not been able to:

1. Implementing cubic spline regression.
2. The case where $\text{rank}(\mathbf{H}) < M$.

As time permits, we will continue to work on these problems until Sunday.

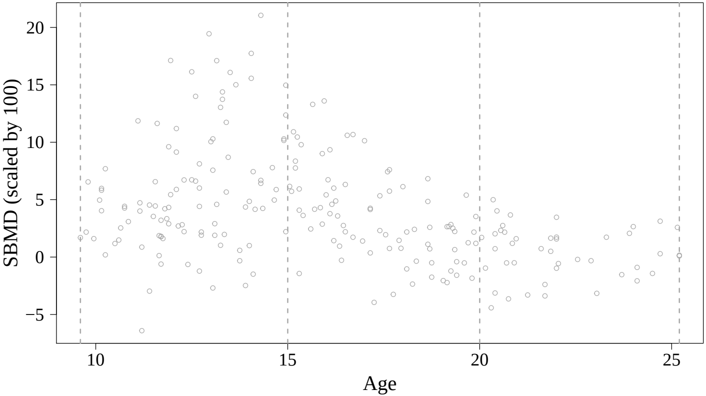
Cubic spline regression

If the relationship between X and Y is extremely non-linear, one might try to fit a high-degree polynomial to the data.

- This is dangerous because of the behavior at the boundaries – will blow up.
- Idea: split the data into several intervals and fit a cubic polynomial *within each region*.
- f is then a *piecewise* cubic polynomial.

The edges of these regions are defined by K different *knots*.

- Given by $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_K)^T$.
- With K knots there will be $K + 1$ regions, so $K + 1$ models to fit.
- We choose how many knots and their placement.



With $\boldsymbol{\xi} = (15, 20)^T$, there are 2 knots and 3 cubic polynomials to be fit.

Conditions on splines

In order to make the function smooth, we impose three conditions at each knot ξ_j:

- $f(\xi_j^-) = f(\xi_j^+)$, function from the left and the right is equal.
- $f'(\xi_j^-) = f'(\xi_j^+)$, first derivative from the left and the right is equal.
- $f''(\xi_j^-) = f''(\xi_j^+)$, second derivative from the left and the right is equal.

These restrictions make f a *cubic spline*.

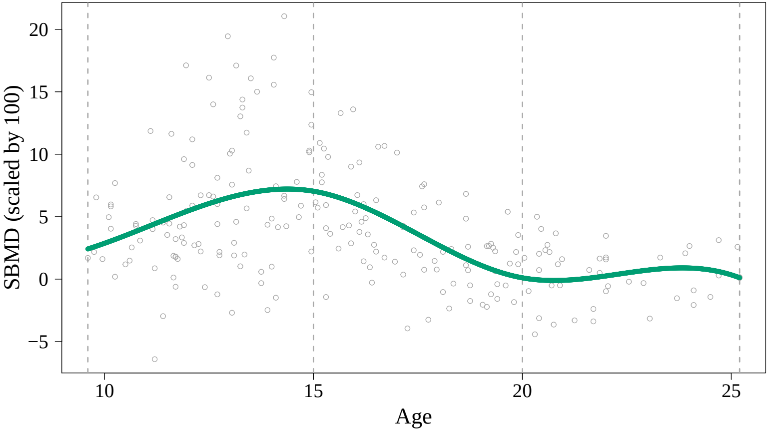
Then the total number of free parameters to be estimated is given by

$$\mathrm{df} = 4(K+1) - 3K = K + 4.$$

While easy to understand, efficiently implementing cubic splines is very difficult.

- Requires creating a basis expansion of $K+4$ different *B-splines*, which are a special type of Bézier curves.
- Once this is done, we can fit the model using least-squares.

Example: bone mineral density



Note: this was fit with the `splines` library from `R`.

A final example

Suppose we want to use a car's horsepower to predict its miles per gallon (MPG). We can obtain the data in `R` as follows:

```
library(ISLR)
x = Auto$horsepower
y = Auto$mpg
mrange = seq(range(x)[1], range(x)[2], 0.1)
```

Our goal is to fit a linear model using `x` and `y`, and then plot the model for each value of `mrange`.

We will fit a linear model and a quadratic polynomial to the data using our functions.

Fitting the linear model

We can fit a linear model using the `linfit338` function in our `pkg338` package.

```
library(pkg338)
lmod = linfit338(x, y);
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.

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

Fitting the quadratic model

Similarly, we can fit a quadratic polynomial using the **polyfit338** function.

```
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

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

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)
```

Plotted models