MAT 4110 Obligatory Assignment 1

Bruce Chappell

September 25, 2019

1 Introduction

In this assignment I will study the implementation of QR Factorization and Cholesky factorization as methods to solve the system:

$$\boldsymbol{A\beta} = \boldsymbol{y} \tag{1}$$

Where $\boldsymbol{A}$ is a matrix of form:

$$\boldsymbol{A} = \begin{bmatrix} 1 & x_0^1 & x_0^2 & \dots & x_0^{m-1} \\ 1 & x_1^1 & x_1^2 & \dots & x_1^{m-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{m-1} \end{bmatrix} \tag{2}$$

The vector $\boldsymbol{x}$ contains even spaced elements and the vector $\boldsymbol{y}$ is generated by a polynomial function using $\boldsymbol{x}$ as an argument and is possibly noisy.

2 QR Factorization

To begin, I used the QR factorization function from `numpy` to factor $\boldsymbol{A}$ into the orthogonal matrix $\boldsymbol{Q} \in \mathbb{R}^{n\times m}$ and the upper triangular matrix $\boldsymbol{R} \in \mathbb{R}^{m\times m}$ resulting in:

$$\boldsymbol{QR\beta} = \boldsymbol{y} \tag{3}$$

Due to orthogonality, $\boldsymbol{Q}^T\boldsymbol{Q} = \mathbb{1}$. Multiplying by $\boldsymbol{Q}^T$ gives:

$$\boldsymbol{R\beta} = \boldsymbol{Q^T y} = \boldsymbol{y^*} \tag{4}$$

Which can be easily solved by performing back substitution on $\boldsymbol{R}$ shown by the pseudo code below:

```
for $i \in \{n-1, \dots, 0\}$ do
  for $j \in \{n-1, \dots, i-1\}$ do
    $y^*[i] = y^*[i] - \beta[j]\boldsymbol{R}[i,j]$
  end for
  $\beta[i] = y^*[i]/\boldsymbol{R}[i,i]$
end for
```

$\boldsymbol{\beta}$ can then be multiplied to the design matrix to obtain the linear regression predicted values of $\boldsymbol{y}$:

$$\boldsymbol{A\beta} = \boldsymbol{y_{predict}} \tag{5}$$

3 Cholesky Factorization

This method solves equation 1 by factoring a symmetric positive definite matrix $\boldsymbol{B} \in \mathbb{R}^{n\times n}$ into $\boldsymbol{LL}^T$ where $\boldsymbol{L}$ is a lower triangular matrix. Since our design matrix $\boldsymbol{A}$ is in $\mathbb{R}^{n\times m}$ equation 1 becomes:

$$\boldsymbol{A^T A\beta = A^T y} \tag{6}$$

$$\boldsymbol{A^*\beta = A^T y} \tag{7}$$

We now perform cholesky factorization on the matrix $\boldsymbol{A}*$ where the elements of $\boldsymbol{L}$ are computed by the following equations:

$$l_{kk} = \sqrt{a^*_{kk} - \sum_{j=1}^{k-1} l^2_{kj}} \tag{8}$$

$$l_{ik} = \frac{1}{l_{kk}}(a^*_{ik} - \sum_{j=1}^{k-1} l_{ij}l_{kj}) \tag{9}$$

The system now becomes:

$$\boldsymbol{LL^T\beta = A^T y = y^*} \tag{10}$$

We can define $\boldsymbol{L}^T\boldsymbol{\beta}$ as a new unknown vector $\boldsymbol{g}$ and use forward substitution to solve equation 11 for $\boldsymbol{g}$:

$$\boldsymbol{Lg = y^*} \tag{11}$$

And back substitution to solve for $\boldsymbol{\beta}$:

$$\boldsymbol{L^T\beta = g} \tag{12}$$

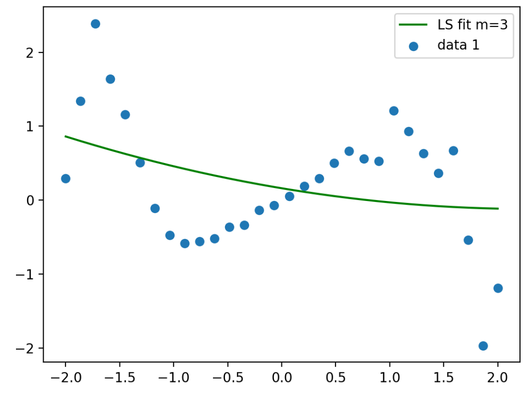
For forward substitution I used to following pseudo code:

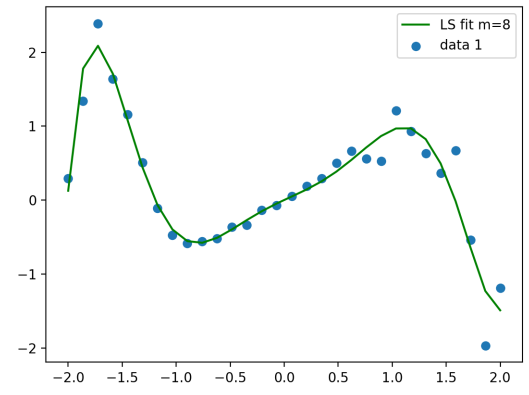
```
for $i \in \{0, \dots, n-1\}$ do
  for $j \in \{0, \dots, i-1\}$ do
    $y^*[i] = y^*[i] - g[j]\boldsymbol{L}[i,j]$
  end for
  $g[i] = y^*[i]/\boldsymbol{L}[i,i]$
end for
```

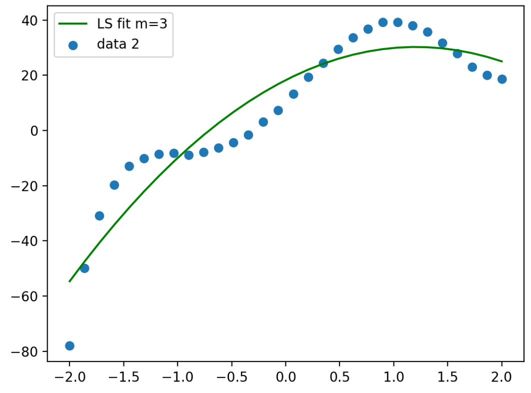
$\boldsymbol{y}_{predict}$ can now be created in the same way as it was with QR factorization.

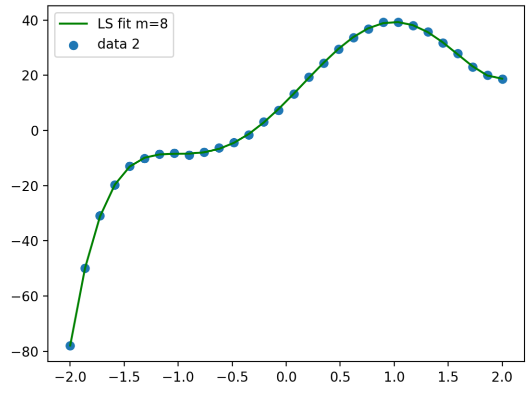
4 Plots

Using two sets of noisy data generated by two different polynomials, both QR and Cholesky factorization were used to to produce fits to the data. Their results for these data sets were identical. Fits were made with degree $m = 3$ and $m = 8$.









For this exercise, I found that QR and Cholesky factorization provided the same results for fitting the data. However, in cases where the data results in a poorly conditioned design matrix QR factorization is a better choice. Cholesky factorization requires taking the square root of a diagonal element of $\boldsymbol{A}^*$ which results in a loss of precision greater than normal arithmetic operations. Additionally, Cholesky factorization requires both forward and backward substitution. If an element on the diagonal of $\boldsymbol{L}$ is close to zero this can create stability problems since $\boldsymbol{L}[i,i]$ is used in the denominator in the algorithm. This would happen in the Cholesky algorithm twice, and only once in the QR algorithm. Below I have attached the code I wrote to generate the data and solve the linear systems.

```python
import numpy as np
np.random.seed(1)

def data_set_one(n, start, stop, eps):

    x = np.linspace(start, stop, n)
    r = np.random.rand(n)*eps
    y = x * (np.cos(r + 0.5*x**3) + np.sin(0.5*x**3))
    return x, y

def data_set_two(n, start, stop, eps):
    x = np.linspace(start, stop, n)
    r = np.random.rand(n)*eps
    y = 4*x**5 - 5*x**4 - 20*x**3 + 10*x**2 + 40*x + 10 + r
    return x, y

```

```python
def designmatrix(x, order):
    mat = np.zeros((len(x),order))
    mat[:,0] = 1
    for i in range(1,order):
        mat[:,i] = x**i
    return mat

def QRsolve(X, y):

    Q, R = np.linalg.qr(X)
    ystar = Q.T @ y

    n = len(ystar)
    beta = np.zeros(n)
    for i in range(n-1, -1, -1):
        tmp = ystar[i]
        for j in range(n-1, i, -1):
            tmp -= beta[j]*R[i,j]
        beta[i] = tmp/R[i,i]
    ypredict = X @ beta
    return ypredict

def cholesky(A):

    A = np.array(A)
    n = A.shape[0]
    L = np.zeros((n,n))

    for row in range(n):
        for col in range(row+1):
            tmp_sum = np.dot(L[row,:col], L[col,:col])
            if (row == col):
                L[row, col] = np.sqrt(A[row,row] - tmp_sum)
            else:
                L[row,col] = (1.0 / L[col,col]) * (A[row,col] - tmp_sum)
    return L

def cholesky_solve(X, y):

    L = cholesky(X.T @ X)
    n = L.shape[0]
    y_1 = X.T @ y

    g = np.zeros(n)
    for i in range(0, n):
         s = y_1[i]
         for j in range(0,i):
              s = s - L[i,j]*g[j]
         g[i] = s / L[i,i]
    beta = np.zeros(n)
    for i in range(n-1, -1, -1):
        tmp = g[i]
        for j in range(n-1, i, -1):
            tmp -= beta[j]*L.T[i,j]
        beta[i] = tmp/L.T[i,i]
    fit = X @ beta
    return fit
```