MAT4110 Compulsory assignment 1

Jan Ole Åkerholm

September 24, 2019

Introduction

The goal of this assignment is to approximate two data sets with a polynomial curve using the least squares method, with two different approaches. This is done by setting up and minimizing

$$S = ||A\mathbf{x} - \mathbf{b}||_2^2$$

where A is the Vandermonde matrix, $\mathbf{x}$ is a matrix containing the coefficients of the polynomial that will be used to fit to the data, and $\mathbf{b}$ is a matrix containing the data points. S will then be minimized using both a QR factorization, and a Cholesky factorization. All programming is done in Python 3.

Exercise 1: QR factorization

Setting $A\mathbf{x} - \mathbf{b} = 0$ is an obvious way to minimize S, giving us the problem $A\mathbf{x} = \mathbf{b}$, where the only unknown is $\mathbf{x}$.

The QR factorization of A gives $A = QR$ where Q is an orthogonal matrix, and R is an $m \times m$ upper triangular matrix, where m is the number of coefficients in the polynomial to be fitted to the data.

This gives us the problem: $QR\mathbf{x} = \mathbf{b}$. Because Q is orthogonal, its inverse Q^{-1} is the same as Q^T, so the problem can be reduced to:

$$\begin{aligned} Q^T QR\mathbf{x} &= Q^T\mathbf{b} \\ R\mathbf{x} &= Q^T\mathbf{b} \end{aligned}$$

The right side is known, so now the next step is to find $\mathbf{x}$ by using back substitution on R. The algorithm is as follows (with $\mathbf{c} = Q^T\mathbf{b}$):

$$x_i = \frac{\mathbf{c}_i - \sum_{j=i+1}^{m} R_{i,j}x_j}{R_{i,i}}, \qquad i = m, m-1, ..., 1$$

(Note: in Python the indices run from 0 to $m-1$, instead of 1 to m, so the indices are shifted in the code loops.)

The code for the back substitution is as follows:

```
def ex1(x, y, m):
    """ Solution to exercise 1 of the obligatory exercise """
    A = make_A(x, y, m) # sets up the Vandermonde matrix given data y,
                        # x-values x and the size of the polynomial to fit m
    n = len(y)

    x_coeffs = np.zeros(m) # polynomial coefficients to find

    Q, R = np.linalg.qr(A)

    # right hand side
    RHS = np.dot(Q.transpose(), y)[0]

    # to solve: Q*R*x = y
    # R*x = Q^T*y
    # apply back substitution to R to find x

    for i in range(0, m)[::-1]:
        sum_ = 0
        for j in range(i+1, m):
            sum_ += R[i, j]*x_coeffs[j]
        x_coeffs[i] = (RHS[0,i]-sum_)/R[i,i]

    x_model = np.linspace(x[0], x[-1], 1000)
    model = np.zeros(1000)
    for i, c in enumerate(x_coeffs):
        model += c*x_model**i

    return x_model, model
```

I have also included the code used to set up the Vandermonde matrix in the end of the report.

Exercise 2: Cholesky factorization

Another way to minimize S is to solve the equation:

$$A^T A\mathbf{x} = A^T\mathbf{b}$$

In order to do this, we set

$$B = A^T A$$

and find the Cholesky factorization $RR^T = B$ of this, giving:

$$RR^T\mathbf{x} = A^T\mathbf{b}$$

The algorithm for finding the Cholesky factorization is included in the code snippet below.

With the problem reduced to $RR^T\mathbf{x} = A^T\mathbf{b}$, first define $R^T\mathbf{x} = \mathbf{w}$. Now the first step is to solve

$$R\mathbf{w} = A^T\mathbf{b} \tag{1}$$

in order to find $\mathbf{w}$.

When $\mathbf{w}$ is known, the next step is to solve

$$R^T\mathbf{x} = \mathbf{w} \tag{2}$$

in order to find $\mathbf{x}$.

R is lower triangular, so in order to solve equation 1, forward substitution must be implemented. This is quite similar to back substitution, and the algorithm is as follows (setting $\mathbf{c} = A^T\mathbf{b}$):

$$\mathbf{w}_i = \frac{\mathbf{c_i} - \sum_{j=1}^{i-1} R_{i,j}\mathbf{w}_j}{R_{i,i}}$$

With $\mathbf{w}$ known, the problem is now to solve equation 2, which can be done using the back substitution method as explained in the previous section.

The code for both the Cholesky decomposition and the forward and back substitution follows:

```
def cholesky(A):
    """ Returns the Cholesky factorization R of the matrix A where
    A = R*R^T

    Requires A to be a symmetric and positive definite matrix
    """
    n = len(A)
    A = np.matrix(A)

    L = np.matrix(np.zeros((n,n)))
    D = np.matrix(np.zeros((n,n)))

    for k in range(n):
        D[k,k] = A[k,k]
        L[:,k] = A[:,k]/D[k,k]

        A = A - D[k,k]*np.dot(L[:,k], L[:,k].transpose())

    D_root = np.power(D, 0.5)

    R = np.dot(L, D_root)
    return R

def ex2(x, y, m):
    """ Solution to exercise 2 of the obligatory exercise """
    A = make_A(x, y, m) # sets up the Vandermonde matrix given data y,
                        # x-values x and the size of the polynomial to fit m
    n = len(y)

    x_coeffs = np.zeros(m) # polynomial coefficients to find
    w = np.zeros(m) # temp variable, w = R^T*x

    B = np.dot(A.transpose(), A)
    R = cholesky(B)

    RHS = np.dot(A.transpose(), y)

    for i in range(0, m):
        sum_ = 0
        for j in range(0, i):
            sum_ += R[i, j]*w[j]
        w[i] = (RHS[0,i] - sum_)/R[i,i]

    RT = R.transpose()

    for i in range(0, m)[::-1]:
        sum_ = 0
        for j in range(i+1, m):
```

```
            sum_ += RT[i, j]*x_coeffs[j]
        x_coeffs[i] = (w[i]-sum_)/RT[i,i]

    x_model = np.linspace(x[0], x[-1], 1000)
    model = np.zeros(1000)
    for i, c in enumerate(x_coeffs):
        model += c*x_model**i

    return x_model, model
```

Exercise 3: discuss the difference

From the plots we can see that (at least to the eye) there is a very small difference between results from the QR and Cholesky methods, as the lines appear to completely overlap. Additionally, in the case of the second dataset with $m = 8$ (figure 4), we see that the fitted function almost fits entirely perfectly over the data. Because the data set is based on a polynomial of degree 5 (with some random noise), it should be expected that a model with an 8th degree polynomial fits almost perfectly, and this is indeed what we see.

For $m = 3$, none of the fits appear to be particularly good.

The methods do vary in terms of the conditioning. The condition number of the matrix A^TA is the square of the condition number of A, so when we solve $A^TA\mathbf{x} = A^T\mathbf{b}$ (Cholesky), we get a matrix on the left side with a higher condition number than when we solve $A\mathbf{x} = \mathbf{b}$ (QR). For this reason, we can assume fairly safely that the QR method provides a more stable numerical method.

Plots

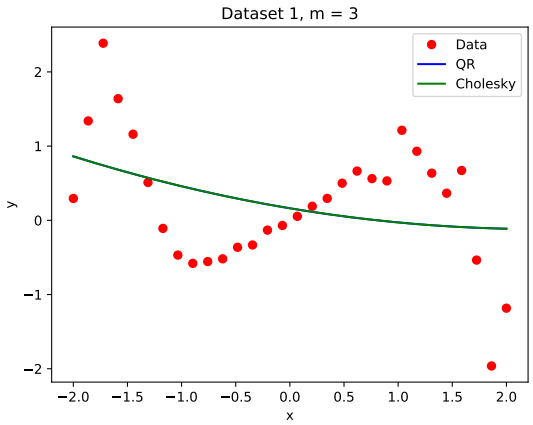

Figure 1: Dataset one with m = 3

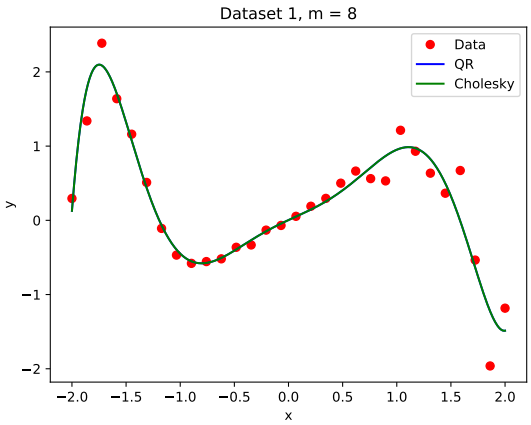

Figure 2: Dataset one with m = 8

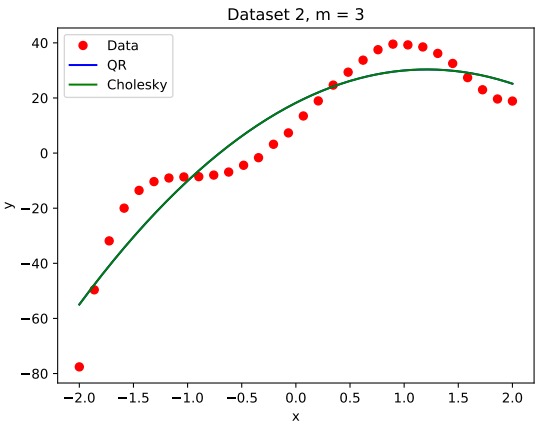

Figure 3: Dataset two with m = 3

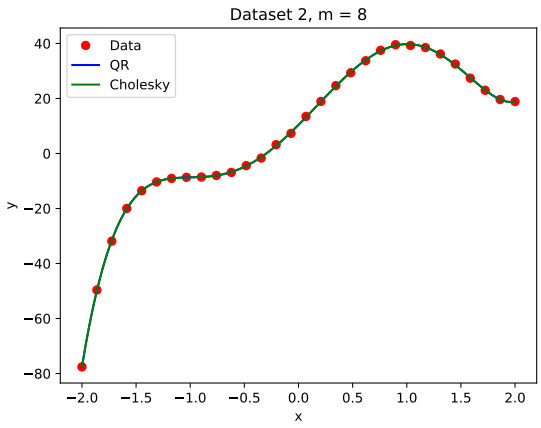

Figure 4: Dataset two with m = 8

Setting up the Vandermonde matrix

The following code sets up the Vandermonde matrix used in the program:

```
1  def make_A(x, y, m):
2      """ Sets up and returns the Vandermonde matrix A for the minimization of
3      ||Ax - b||^2
4      """
5      n = len(y)
6
7      # setting up Vandermonde matrix
8      A = np.matrix(np.zeros((n, m)))
9      A[:,0] = np.full((n,1), 1) # first column is 1 everywhere
10
11     # set up x-values as a column to fill the columns of A
12     x_col = x[:]
13     x_col.resize((n,1))
14
15     # loop over columns
16     for j in range(1,m):
17         A[:,j] = x_col**j
18
19     return A
```