Computational Linear Algebra
Project 1

Tudor Trita Trita
CID 01199397

November 5, 2019

This is my own work unless stated otherwise.

1 QR Factorization by Householder Reflections

1.1 QR Function

The Python function implementation is inside of the python script 'question1.py' and it is called 'householder'.

1.2 Tests for QR Accuracy

The following are the tests implemented. They are located inside of the python script 'question1.py' and they are 'check_qr', 'check_unitary', 'check_upper_triangular', 'check_determinants'.

- **Checking QR:** In this test we check the accuracy of the implementation by calculating the result of $A - QR$. The test outputs the largest single error attained as well as the sum of the absolute errors attained, giving a sense of 'total lost precision'.
- **Checking that Q is unitary:** By definition, the resulting Q matrix from a QR-factorisation is unitary. That is, if $Q \in \mathbb{C}^{n,m}$, with $m \leq n$ the equations $Q^*Q = I, QQ^* = I$ hold, where * is the conjugate transposition operation (tranpose ^{T} for real matrices) and $I \in \mathbb{C}^{m,m}$ is the identity matrix. The implemented test computes Q^*Q, QQ^* and compares the resulting matrices with the identity. If the differences $I - Q^*Q, I - QQ^*$ are within a certain tolerance level, the test will pass.
- **Checking that R is upper-triangular:** By definition, the resulting R matrix from a QR-factorisation is upper-triangular. The implemented test passes if the absolute value of all the entries below the diagonal are within a tolerance level.
- **Checking determinants:** The following results are true: $\det(A) = \det(QR), |\det(A)| = |\det(R)|, |\det(Q)| = 1$. The test computes these quantities and prints True if the first quantity is true.

If all these tests pass within the set tolerance limit, then we can conclude that we have computed the correct QR-factorisation of the matrix A.

2 Polynomial Fitting by Least Squares

2.1 Least-Squares Polynomial-Fitting Process

Suppose we are given some data $x, b \in \mathbb{C}^n$. We wish to fit a polynomial of degree $m-1$ to this data, for some $m \leq n$. The polynomial will be of the form

$$p(x) = c_0 + c_1 x + \ldots + c_{m-1} x^{m-1}.$$

This polynomial is a least-squares fit to this data if it minimises the sum of squares

$$\sum_{i=1}^{n} |p(x_i) - b_i|^2 .$$

This problem may be expressed as the Vandermonde system $Ac = b$ (1), $c \in \mathbb{C}^m$, where $c = [c_0, c_1, \ldots, c_{m-1}]^T$, $b = [b_0, b_1, \ldots, b_n]^T$ and A is known as the Vandermonde matrix

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{m-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{m-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{m-1} \end{bmatrix}.$$

The solution $c = A^{-1}b$ of the system is the vector of coefficients corresponding to the polynomial of least-squares. Therefore, to compute this polynomial, we need to solve for c.

2.2 LS Polynomial Fitting Implementation

In this section we explore fitting some polynomials to data $x, b \in \mathbb{C}^n$ using the QR factorisation via Householder Reflections. We take the system (1) described in section 2.1 with the appropriate Vandermonde matrix A made up of powers of x as its columns.

To avoid computing the inverse of A, we can use the fact that Q is unitary and that R is upper-triangular.

The procedure goes as follows:

1. Compute the QR factorisation $A = QR$, and reduce Q, R into $\hat{Q}, \hat{R}$. The equation now becomes $\hat{Q}\hat{R}c = b$ (2).(Alternatively, compute reduced QR factorisation directly)
2. Compute the vector $\hat{Q}^* b$. This is because we can multiply the equation (2) on the left by $\hat{Q}^* b$ to be left with the upper-triangular system $\hat{R}c = \hat{Q}^* b$.
3. We can now solve the upper-triangular system for c using back-substitution.

The implementation is found in the file 'question2.py' in the function polynomial_fit.

2.3 Accuracy of polynomial fit

After examining the accuracy of the polynomial fit for degrees $d = \{1, \ldots, 20\}$, we have the following results, taken from running the script 'question2.py':

```
- Poly. d = 1. Error = 0.34154117. Improvement = nan percent.
- Poly. d = 2. Error = 0.30694974. Improvement = 10.128 percent.
- Poly. d = 3. Error = 0.04252878. Improvement = 86.145 percent.
- Poly. d = 4. Error = 0.00397634. Improvement = 90.65 percent.
- Poly. d = 5. Error = 0.00328366. Improvement = 17.42 percent.
- Poly. d = 6. Error = 0.0032734. Improvement = 0.312 percent.
- Poly. d = 7. Error = 0.00326759. Improvement = 0.178 percent.
- Poly. d = 8. Error = 0.00326727. Improvement = 0.01 percent.
- Poly. d = 9. Error = 0.00322675. Improvement = 1.24 percent.
- Poly. d = 10. Error = 0.00311006. Improvement = 3.616 percent.
- Poly. d = 11. Error = 0.00294764. Improvement = 5.222 percent.
- Poly. d = 12. Error = 0.00292681. Improvement = 0.707 percent.
- Poly. d = 13. Error = 0.00291134. Improvement = 0.528 percent.
- Poly. d = 14. Error = 0.00290257. Improvement = 0.301 percent.
- Poly. d = 15. Error = 0.00290125. Improvement = 0.045 percent.
- Poly. d = 16. Error = 0.00290111. Improvement = 0.005 percent.
- Poly. d = 17. Error = 0.00290073. Improvement = 0.013 percent.
- Poly. d = 18. Error = 0.00289587. Improvement = 0.168 percent.
- Poly. d = 19. Error = 0.00289584. Improvement = 0.001 percent.
- Poly. d = 20. Error = 0.00266949. Improvement = 7.817 percent.
```

From the results, we can see that the fit always strictly improves as we increase the degree of the polynomial, i.e. the error decreases as we increase d. This makes sense, as we increase d, we interpolate to the data more and more closely.

The Improvement parameter computes the percentage decrease in error when we increase the degree by 1 at each step.

In theory, we can achieve a 'perfect fit' by taking a polynomial of degree d equal to the number of data points - 1 in our dataset, however, such a polynomial would be badly susceptible to perturbations. This is also known as over-fitting to the data.

We can see that for $d > 5$, the improvement in fit is low, compared to values of d less than 6. Because of this, the most appropriate choice of polynomial degree to model the data is

$$d = 5.$$

3 QR Analysis of Dataset

The script apporpriate for this question is 'question3.py'. The matrix reduced R produced after performing the QR factorisation is:

```
Reduced matrix R:

[[-1.98e+00 -2.01e+00 -2.06e+00 -2.14e+00 -2.24e+00 -2.36e+00 -2.51e+00
  -2.68e+00 -2.87e+00 -3.09e+00]
 [-5.55e-17  1.16e-01  3.43e-01  6.80e-01  1.13e+00  1.69e+00  2.36e+00
   3.14e+00  4.03e+00  5.03e+00]
 [ 6.94e-18 -3.47e-18 -1.02e-14 -2.09e-14 -3.50e-14 -5.31e-14 -7.44e-14
  -1.00e-13 -1.28e-13 -1.62e-13]
 [-6.94e-18 -8.67e-19 -2.47e-32  6.67e-15  9.85e-15  1.57e-14  2.16e-14
   2.95e-14  3.83e-14  4.79e-14]
 [-2.78e-17  4.34e-19  9.86e-32 -1.97e-31  5.46e-15  7.43e-15  1.05e-14
   1.41e-14  1.86e-14  2.27e-14]
 [-2.78e-17 -4.34e-19  2.47e-32 -1.85e-32 -2.47e-32  5.36e-15  5.37e-15
   7.29e-15  9.29e-15  1.27e-14]
 [-6.94e-18 -1.73e-18 -2.47e-32  1.48e-31 -9.86e-32 -9.86e-32 -5.09e-15
  -4.83e-15 -5.40e-15 -7.11e-15]
 [ 0.00e+00 -1.73e-18  4.93e-32 -4.93e-32 -1.97e-31 -9.86e-32 -9.86e-32
   4.03e-15  2.67e-15  4.63e-15]
 [-5.55e-17 -1.73e-18  1.97e-31  1.97e-31  4.93e-32 -4.93e-32  2.47e-32
   1.48e-31  4.11e-15  3.26e-15]
 [-5.55e-17  8.67e-19 -1.97e-31 -9.86e-32  4.93e-32 -4.93e-32  1.97e-31
   9.86e-32  0.00e+00 -9.40e-15]]
```

If we take entries less than 10^{-12} as zero due to rounding we end up with the reduced matrix R:

```
Reduced matrix R with small entries removed:

[[-1.98 -2.01 -2.06 -2.14 -2.24 -2.36 -2.51 -2.68 -2.87 -3.09]
 [ 0.    0.12  0.34  0.68  1.13  1.69  2.36  3.14  4.03  5.03]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]]
```

3.1 What do you notice about the matrix R?

The matrix R has two rows that contain non-zero elements. Otherwise, every other element in it is zero.

3.2 What does this tell you about the matrix A?

This tells us that the matrix A is of Rank 2, and therefore there are only 2 linearly independent columns in the matrix. A consequence of this is that there exist two linearly independent columns c_i, c_j from which every column can be expressed as a linear combination $c_k = ac_i + bc_j$ for some scalars $a, b \in \mathbb{C}$.

3.3 What does this tell you about the dataset?

This tells us that there is a major factor affecting the data globally. In terms of the actual data, we can deduce that each species of yeast changes temperature at different rates as the light intensity changes, however, as the light is applied equally for all the species, we can say that the temperature across the different yeast species changes over time proportionally, i.e. if we let $c_i(t)$ be the temperature of yeast species i, and we let r_i be the 'sensitivity to light' for the yeast species i, and $l(t)$ the ultraviolet light intensity globally at time t, then we have the relationship

$$c_i(t) = r_i l(t).$$