NumCSE exercise sheet 3
least squares, QR decomposition, SVD, PCA

oliver.rietmann@sam.math.ethz.ch
alexander.dabrowski@sam.math.ethz.ch

November 5, 2018

Exercise 3.1. *least squares.*

To determine the least squares solution of an overdetermined linear system of equations $\mathbf{A}\mathbf{x} = \mathbf{b}$, we minimize the residual norm $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2$ w.r.t. $\mathbf{x}$. In this exercise we consider a different least squares problem that arises when minimizing the residual norm w.r.t. the entries of $\mathbf{A}$. Fix an integer $n > 2$ and let $\mathbf{z}, \mathbf{c} \in \mathbb{R}^n$. Define α^* and β^* as

$$(\alpha^*, \beta^*) = \operatorname{argmin}_{\alpha,\beta\in\mathbb{R}} \|\mathbf{T}_{\alpha,\beta}\mathbf{z} - \mathbf{c}\|_2, \tag{1}$$

where $\mathbf{T}_{\alpha,\beta} \in \mathbb{R}^{n\times n}$ is the tridiagonal matrix

$$\mathbf{T}_{\alpha,\beta} = \begin{pmatrix} \alpha & \beta & 0 & \dots & 0 \\ \beta & \alpha & \beta & \ddots & \vdots \\ 0 & \beta & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \alpha & \beta \\ 0 & \dots & 0 & \beta & \alpha \end{pmatrix}.$$

(a) Reformulate (1) as a linear least squares problem

$$\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}\in\mathbb{R}^k} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2,$$

where $\mathbf{A} \in \mathbb{R}^{m\times k}$ and $\mathbf{b} \in \mathbb{R}^m$ for some $m, k \in \mathbb{N}$.

Hint: For $\mathbf{x} = (\alpha, \beta)^\top$, find $\mathbf{A}$ such that $\mathbf{T}_{\alpha,\beta}\mathbf{z} = \mathbf{A}\mathbf{x}$.

Solution: The vector $\mathbf{T}_{\alpha,\beta}\mathbf{z} - \mathbf{c}$ whose norm has to be minimized in (1) can be written as

$$\mathbf{T}_{\alpha,\beta}\mathbf{z} - \mathbf{c} = \begin{pmatrix} \alpha z_1 + \beta z_2 \\ \alpha z_2 + \beta(z_1 + z_3) \\ \dots \\ \alpha z_{n-1} + \beta(z_{n-2} + z_n) \\ \alpha z_n + \beta z_{n-1} \end{pmatrix} - \mathbf{c} = \begin{pmatrix} z_1 & z_2 \\ z_2 & z_1 + z_3 \\ \vdots & \vdots \\ z_{n-1} & z_{n-2} + z_n \\ z_n & z_{n-1} \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} - \mathbf{c} =: \mathbf{A}\mathbf{x} - \mathbf{b}$$

In this form, we have moved the unknown α and β into the vector $\mathbf{x}$, while the data $\mathbf{z}$ and $\mathbf{c}$ are moved into the matrix $\mathbf{A}$ and the right-hand side vector $\mathbf{b} \equiv \mathbf{c}$ (which is not affected by the transformation).

(b) Implement a `C++` function

```
Vector2d lsqEst(const VectorXd &z, const VectorXd &c);
```

which solves the linear least squares problem of (1) using the normal equation method and returns the optimal parameters α^* and β^*.

Solution:

```
10 Eigen::Vector2d lsqEst(const Eigen::VectorXd &z, const Eigen::VectorXd &c) {
11     Eigen::Vector2d x;
12 
13     int n = z.size();
14     Eigen::MatrixXd A(n,2);
15     A.col(0) = z;
16     A(0,1) = z(1);
17     for(int i = 1; i < n - 1; ++i) {
18         A(i, 1) = z(i - 1) + z(i + 1);
19     }
20     A(n - 1, 1) = z(n - 2);
21 
22     // Normal equations
23     Eigen::MatrixXd lhs = A.transpose() * A;
24     Eigen::VectorXd rhs = A.transpose() * c;
25     x = lhs.fullPivLu().solve(rhs);
26 
27     return x;
28 }
```

tridiag_least_squares.cpp

Exercise 3.2. *QR and Cholesky decomposition, Householder reflections.*

Fix $m, n \in \mathbb{N}$ with $m \geq n$ and let $\mathbf{A} \in \mathbb{R}^{m \times n}$. We say that $\mathbf{A}$ is of *full rank* if $\operatorname{rank}(\mathbf{A}) = n$. In this case, $\mathbf{A}$ is injective. Moreover, for every symmetric, positive definite matrix $\mathbf{B} \in \mathbb{R}^{n \times n}$, there exists a lower triangular matrix $\mathbf{L} \in \mathbb{R}^{n \times n}$ such that

$$\mathbf{B} = \mathbf{L}\mathbf{L}^\top. \tag{2}$$

A decomposition of the form (2) is called *Cholesky decomposition* of $\mathbf{B}$.

(a) Prove: If $\mathbf{A}$ is of full rank, then $\mathbf{A}^\top\mathbf{A}$ admits a Cholesky decomposition.

Hint: Verify the conditions on $\mathbf{B}$ from above.

Solution: We have to show that $\mathbf{A}^\top\mathbf{A}$ is symmetric and positive definite. The symmetry follows from

$$(\mathbf{A}^\top\mathbf{A})^\top = \mathbf{A}^\top(\mathbf{A}^\top)^\top = \mathbf{A}^\top\mathbf{A}.$$

It remains to show positive definiteness. Since $\mathbf{A}$ has full rank, it is injective. Hence it holds for all $\mathbf{v} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ that $\mathbf{A}\mathbf{v} \neq \mathbf{0}$ and thus

$$0 < \|\mathbf{A}\mathbf{v}\|_2^2 = \mathbf{v}^\top\mathbf{A}^\top\mathbf{A}\mathbf{v}.$$

We conclude that $\mathbf{A}^\top\mathbf{A}$ is positive definite.

(b) Suppose that $\mathbf{A}$ is of full rank. From (a) we know that there exists a Cholesky decomposition

$$\mathbf{A}^\top\mathbf{A} = \mathbf{L}\mathbf{L}^\top, \tag{3}$$

where $\mathbf{L} \in \mathbb{R}^{n \times n}$ is lower triangular. Prove that $\mathbf{L}$ is invertible and that

$$\mathbf{Q} := \mathbf{A}(\mathbf{L}^{-1})^\top \quad \text{and} \quad \mathbf{R} := \mathbf{L}^\top \tag{4}$$

yield a *reduced/thin* QR decomposition of $\mathbf{A}$.

Solution: Since $\mathbf{A}^\top\mathbf{A}$ is positive definite, it is invertible and so are the factors in its Cholesky decomposition (3). It remains to verify three properties:

(i) $\mathbf{R}$ is upper triangular

(ii) $\mathbf{Q}^\top\mathbf{Q} = \mathbf{I}$ (i.e. the columns of $\mathbf{Q}$ are orthogonal)

(iii) $\mathbf{A} = \mathbf{Q}\mathbf{R}$

This is done as follows:

(i) Since $\mathbf{L}$ is lower triangular, its transpose $\mathbf{R}$ is upper triangular.

(ii) Using the definition of $\mathbf{Q}$ in the first step and (3) in the second step yields

$$\mathbf{Q}^\top\mathbf{Q} = \mathbf{L}^{-1}\mathbf{A}^\top\mathbf{A}(\mathbf{L}^{-1})^\top = \mathbf{I}.$$

(iii) We use (4) to compute

$$\mathbf{Q}\mathbf{R} = \mathbf{A}(\mathbf{L}^{-1})^\top\mathbf{L}^\top = \mathbf{A}.$$

(c) Use Task (b) to implement an `Eigen` based `C++` function

```
void CholeskyQR(const MatrixXd &A, MatrixXd &Q, MatrixXd &R);
```

which computes a *reduced/thin* QR decomposition of a given matrix `A` of full rank.

Solution:

```
11 void CholeskyQR(const Eigen::MatrixXd &A, Eigen::MatrixXd &R, Eigen::MatrixXd &Q) {
12     Eigen::MatrixXd AtA = A.transpose() * A;
13     Eigen::LLT<Eigen::MatrixXd> chol = AtA.llt();
14     Eigen::MatrixXd L = chol.matrixL();
15     R = L.transpose();
16     Q = L.triangularView<Eigen::Lower>().solve(A.transpose()).transpose();
17 }
```

cholesky_qr.cpp

(d) Implement an `Eigen` based `C++` function

```
void DirectQR(const MatrixXd &A, MatrixXd &Q, MatrixXd &R);
```

which computes a *reduced/thin* QR decomposition of `A` using the `HouseholderQR` class from `Eigen`.

Solution:

```
24 void DirectQR(const Eigen::MatrixXd &A, Eigen::MatrixXd &R, Eigen::MatrixXd &Q) {
25     int m = A.rows();
26     int n = A.cols();
27
28     Eigen::HouseholderQR<Eigen::MatrixXd> QR = A.householderQr();
29     Q = QR.householderQ() * Eigen::MatrixXd::Identity(m, std::min(m, n));
30     R = Eigen::MatrixXd::Identity(std::min(m, n), m) * QR.matrixQR().triangularView<
       Eigen::Upper>();
31 }
```

cholesky_qr.cpp

In the code above, we could replace `std::min(m, n)` by `n` since we assume $m \geq n$.

(e) Let $\varepsilon := 10^{-8}$ and define

$$\mathbf{A} := \begin{pmatrix} 1 & 1 \\ \varepsilon & 0 \\ 0 & \varepsilon \end{pmatrix}.$$

Compare the QR decompositions of A obtained from `CholeskyQR` and `DirectQR`.

Solution: The matrix $\mathbf{Q}$ obtained by `CholeskyQR` does *not* have orthogonal columns. However, this does *not* contradict the results from (a) and (b): Note that

$$\mathbf{A}^\top \mathbf{A} = \begin{pmatrix} 1+\epsilon^2 & 1 \\ 1 & 1+\epsilon^2 \end{pmatrix}$$

collapses because of cancellation to

$$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix},$$

which is *not* of full rank (although $\mathbf{A}$ is). Thus the hypothesis of the Cholesky decomposition is violated.

Exercise 3.3. *Face recognition by PCA.*

We apply principal component analysis to develop a simple recognition/classification technique for pictures of faces. In the folder `basePictures`, you can find $M = 15$ photos encoded with the PGM (Portable GrayMap) ASCII format. They are essentially represented by an $h \times w$ matrix of integers between 0 and 255 (in our case $h = 231$, $w = 195$).

a) Prove that for an $m \times n$ matrix X the following statements are equivalent, for any integer $k \in [0, \min(m, n)]$:

(i) $\operatorname{rank}(X) = k$,

(ii) $X = AB^\top$ for A an $m \times k$ matrix, and B an $n \times k$ matrix, both of full rank.

Hint: for (i) $\Rightarrow$ (ii) use SVD, for (ii) $\Rightarrow$ (i) the rank-nullity theorem[1].

Solution:

(i) $\Rightarrow$ (ii): The rank of a matrix is given by the number of its nonzero singular values. Let $X = U\Sigma V^\top$ be the singular value decomposition of X. Then we can take, for instance, as A the first k columns of $U\Sigma$, and as B the first k columns of V.

(ii)$\Rightarrow$(i): Applying the rank-nullity theorem to A we obtain $\dim\ker A = 0$, so A is injective. Therefore $\operatorname{rank}(AB^\top) = \operatorname{rank}(B^\top) = \operatorname{rank}(B) = k$.

b) Through the `load_png` function, the main function of `eigenfaces.cpp` loads every picture as a flattened `Eigen` vector of length hw. Compute the mean of all these vectors, and insert each vector minus the mean as a column of a matrix A of size $hw \times M$.

Solution:

```
56      for (int i=0; i<M; i++) {
57          string filename = "./basePictures/subject"+to_string(i+1)+".pgm";
58          VectorXd flatPic = load_pgm(filename);
59          faces.col(i) = flatPic;
60          meanFace += flatPic;
61      }
62
63      // compute mean face and subtract it from all faces
64      meanFace /= M;
65      faces.colwise() -= meanFace;
```

eigenfaces.cpp

c) The covariance matrix of A is defined as $AA^\top$. The eigenvector of $AA^\top$ corresponding to the largest eigenvalue represents the direction along which the dataset has the maximum variance. Therefore it encodes the features which differ the most among faces. For this reason, since our interest is in recognizing faces from their salient features, we want to compute the eigenvectors of $AA^\top$. We rename such eigenvectors as *eigenfaces* (usually they are known as principal components).

What is the size of $AA^\top$? How many non-zero eigenvalues can the matrix $AA^\top$ have at most?

Hint: use Point (a).

Solution: The size of $AA^\top$ is $hw \times hw$. By Point (a) the rank of A is at most M, therefore $AA^\top$ can have at most M non-zero eigenvalues (counted with their multiplicities).

[1]The rank-nullity theorem can be reformulate in terms of matrices as: for any matrix Z with c columns it holds $\dim\ker(Z) + \operatorname{rank}(Z) = c$.

d) What is the size of $A^\top A$? How are the eigenvalues and eigenvectors of the matrix $AA^\top$ related respectively to the eigenvalues and eigenvectors of $A^\top A$, and to the singular values and singular vectors of A?

Solution: The size of $A^\top A$ is $M \times M$. The eigenvalues of $AA^\top$ and $A^\top A$ are the same and they are the squared singular values of A. The eigenvectors of $AA^\top$ are the left singular vectors of A, while the eigenvectors of $A^\top A$ are the right singular vectors of A.

e) Use the characterization of the previous point to compute with a `C++` code the eigenvectors of $AA^\top$ using the SVD of A.

Solution:

```
67      JacobiSVD<MatrixXd> svd(faces, ComputeThinU | ComputeThinV);
68      MatrixXd U = svd.matrixU();
```

eigenfaces.cpp

f) Given a new face y, implement a code which computes its projection on the space spanned by the eigenfaces, that is, which finds x such that $Ux = (y-$ mean face), where U is the matrix of singular vectors of A.

Hint: instead of solving the linear system, use the properties of the matrix U.

Solution:

```
75      VectorXd projNewFace = U.transpose() * (newFace - meanFace);
```

eigenfaces.cpp

g) Implement a `C++` code which computes the distance between the projection of the new face and the projection of a column k for a generic k, and print the k which minimizes the distance.

Solution:

```
76      int indexMinNorm;
77      (projFaces.colwise()-projNewFace).colwise().norm().minCoeff(&indexMinNorm);
78      cout << testPicName << " is identified as subject "
79           << indexMinNorm+1 << endl;
80  }
```

eigenfaces.cpp

h) Test your code from the previous steps to try to recognize the pictures in the folder `testPictures`.