NumCSE

Autumn Semester 2017

Prof. Rima Alaifari

Exercise sheet 4

Constrained least squares, singular value decomposition, principal components analysis

A. Dabrowski

Problem 4.1: Matrix least squares in Frobenius norm

We use two different approaches (Lagrange multipliers and augmented normal equations) to solve a constrained minimization problem.

Fix non-zero vectors $\mathbf{z} \in \mathbb{R}^n$ and $\mathbf{g} \in \mathbb{R}^n$. We want to find the matrix $\mathbf{M}^*$ which minimizes the Frobenius norm among all matrices $\mathbf{M}$ which belong to the set

$$\{\mathbf{M} \in \mathbb{R}^{n,n},\ \mathbf{Mz} = \mathbf{g}\}. \tag{4.1}$$

Recall that the Frobenius norm is defined as

$$\|\mathbf{M}\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} |\mathbf{M}_{i,j}|^2}.$$

First, we look at the method of Lagrange multipliers to solve Eq. (4.1) analytically.

(a) Let $L : \mathbb{R}^{n,n} \times \mathbb{R}^n \longrightarrow \mathbb{R}$ be the functional defined as

$$L(\mathbf{M}, \boldsymbol{\lambda}) = \|\mathbf{M}\|_F^2 + \boldsymbol{\lambda}^\top(\mathbf{Mz} - \mathbf{g}).$$

Derive simbolically the critical points of L, that is find analytically $\mathbf{M}$ and $\boldsymbol{\lambda}$ which solve $\nabla L(\mathbf{M}, \boldsymbol{\lambda}) = 0$.

Another approach, which we now explore, is to recast Eq. (4.1) in a linear least squares form and use the augmented normal equations method to compute the solution.

(b) Reformulate the problem as an equivalent standard linearly constrained least squares problem. That is find suitable matrices $\mathbf{A}$, $\mathbf{C}$ and vectors $\mathbf{b}$ and $\mathbf{d}$, which you should specify based on $\mathbf{z}$ and $\mathbf{g}$, so that the minimization of

$$\|\mathbf{Ax} - \mathbf{b}\|_2$$

among all $\mathbf{x} \in \mathbb{R}^{n^2}$ such that

$$\mathbf{Cx} = \mathbf{d}$$

will be equivalent (after an appropriate reshaping of $\mathbf{x}$) to the minimization of (4.1).

Hint: to define $\mathbf{C}$ you may use the Kronecker product.

(c) State the *augmented normal equations* corresponding to the constrained linear least squares problem of Point (b) and give necessary and sufficient conditions on $\mathbf{g}$ and $\mathbf{z}$ that ensure existence and uniqueness of solutions.

(d) Implement a C++ function

```
MatrixXd min_frob(const VectorXd &z, const VectorXd &g)
```

that computes the solution $\mathbf{M}^*$ of the minimization of Eq. (4.1) given the vectors $\mathbf{z}, \mathbf{g} \in \mathbb{R}^n$. Use the augmented normal equations approach.

Hint: You may use

```
#include <unsupported/Eigen/KroneckerProduct>
```

(e) Using random vectors $\mathbf{z}$ and $\mathbf{g}$, check with a numerical experiment that the matrix given by

$$\frac{\mathbf{g}\mathbf{z}^\top}{\|\mathbf{z}\|_2^2} \tag{4.2}$$

minimizes Eq. (4.1).

Problem 4.2: Recompression of the sum of low rank approximations of matrices

Large matrices of low rank can be stored more efficiently using their singular value decomposition. However, adding two low rank matrices usually leads to an increase of the rank, requiring further "recompression" by computing a low-rank best approximation of the sum. We want to implement an efficient approach to recompression.

Template: `lowRankRecompression.cpp`.

(a) Show that for a matrix $\mathbf{X} \in \mathbb{R}^{m,n}$ the following statements are equivalent:

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

(ii) $\mathbf{X} = \mathbf{A}\mathbf{B}^\top$ for $\mathbf{A} \in \mathbb{R}^{m,k},\ \mathbf{B} \in \mathbb{R}^{n,k},\ k \leq \min\{m,n\}$, both full rank.

Hint: for (i) $\Rightarrow$ (ii) use SVD, for (ii) $\Rightarrow$ (i) the definition of rank and its relation with the dimension of the null-space.

(b) Implement a C++ function

```
void factorize(const MatrixXd &X, int k, MatrixXd &A,
    MatrixXd &B)
```

that factorizes the matrix $\mathbf{X}$ with $\operatorname{rank}(\mathbf{X}) = k$ into $\mathbf{A}\mathbf{B}^\top$, as in (a).

Hint: Use function **Eigen::svd** and pay attention on how to return $\mathbf{U}$, $\mathbf{\Sigma}$ and $\mathbf{V}$ correctly.

(c) Let $\mathbf{A} \in \mathbb{R}^{m,k}$, $\mathbf{B} \in \mathbb{R}^{n,k}$, and assume that k is much smaller than m and n. Write an efficient C++ function that calculates a singular value decomposition of the product $\mathbf{A}\mathbf{B}^\top = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, where orthogonal $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{n,k}$ and $\mathbf{\Sigma} \in \mathbb{R}^{k,k}$:

```
void svd_AB(const MatrixXd & A, const MatrixXd & B, MatrixXd & U,
    MatrixXd & S, MatrixXd & V);
```

Hint: If you directly compute an SVD of $\mathbf{A}\mathbf{B}^\top$, you will always have to deal with dimensions m and n, regardless of the smallness of k. Exploit the *economical QR decomposition* of both $\mathbf{A}$ and $\mathbf{B}$ separately to reduce the problem to the computation of the SVD of a small matrix.

(d) What is the asymptotic computational cost of the function `svd_AB` for a small k and $m = n \to \infty$? Discuss the effort required by the different steps of your algorithm.

(e) If $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{m,n}$ satisfy $\operatorname{rank}(\mathbf{X}) = \operatorname{rank}(\mathbf{Y}) = k$, show that $\operatorname{rank}(\mathbf{X} + \mathbf{Y}) \leq 2k$.

(f) Consider $\mathbf{A}_X, \mathbf{A}_Y \in \mathbb{R}^{m,k}$, $\mathbf{B}_X, \mathbf{B}_Y \in \mathbb{R}^{n,k}$, $\mathbf{X} = \mathbf{A}_X\mathbf{B}_X^\top$ and $\mathbf{Y} = \mathbf{A}_Y\mathbf{B}_Y^\top$. Find a factorization of sum $\mathbf{X} + \mathbf{Y}$ as $\mathbf{X} + \mathbf{Y} = \mathbf{A}\mathbf{B}^\top$, with $\mathbf{A} \in \mathbb{R}^{m,2k}$ and $\mathbf{B} \in \mathbb{R}^{n,2k}$.

(g) Implement the formula derived in the previous subproblems in an efficient C++ function

```
void lowRankApprox(const MatrixXd &Ax, const MatrixXd &Ay,
    const MatrixXd &Bx, const MatrixXd &By, MatrixXd &Az,
    MatrixXd &Bz),
```

where $\mathbf{A}_Z \in \mathbb{R}^{m,k}$ and $\mathbf{B}_Z \in \mathbb{R}^{n,k}$ are the two terms of the decomposition of $\mathbf{Z} = \mathbf{A}_Z\mathbf{B}_Z^\top$, the rank-$k$ best approximation of the sum $\mathbf{X} + \mathbf{Y} = \mathbf{A}_X\mathbf{B}_X^\top + \mathbf{A}_Y\mathbf{B}_Y^\top$:

$$\mathbf{Z} = \operatorname*{argmin}_{\substack{\mathbf{M}\in\mathbb{R}^{m,n} \\ \operatorname{rank}(\mathbf{M})\leq k}} \left\| \mathbf{A}_X\mathbf{B}_X^\top + \mathbf{A}_Y\mathbf{B}_Y^\top - \mathbf{M} \right\|_F$$

Here $\mathbf{A}_X, \mathbf{A}_Y \in \mathbb{R}^{m,k}$ and $\mathbf{B}_X, \mathbf{B}_Y \in \mathbb{R}^{n,k}$.

(h) What is the asymptotic computational cost of the function `rank_k_approx` for a small k and $m = n \to \infty$?

Problem 4.3: Face recognition by PCA

We apply principal component analysis to develop a recognition/classification technique for pictures of faces.

Template: `eigenfaces.cpp`

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) 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 $hw \times M$.

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).

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

(c) 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?

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

(e) Given a new face y, implement C++ code which computes its projection on the space spanned by the eigenfaces, that is 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.

(f) Implement 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.

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