

ETH Lectures 401-0663-00L Numerical Methods for Computer Science
401-2673-00L Numerical Methods for CSE

Homework Problems and Projects

Prof. R. Hiptmair, SAM, ETH Zurich

Autumn Term 2021, Version of October 2, 2022
(C) Seminar für Angewandte Mathematik, ETH Zürich

[Link](#) to the current version of this homework collection

0.1 General Information

0.1.1 Weekly Homework Assignments

All problems will be published in this single “.pdf” file. Every week, we publish a list of problem numbers on the lecture Moodle page. These are the current assignments for that week. Details are given on the lecture Moodle page.

0.1.2 Importance of Homework

Homework assignments are **not** mandatory. However, it is **very important** that you constantly exercise with the material you learn. “Solving” the homework assignments one week before the main exam by looking at the solutions will likely result in failure.

We provide hints and solutions from the beginning. It is your responsibility to look at the solutions only if you are stuck with a difficult problem and you tried to find a solution for a sufficient amount of time.

Most homework problems come with an estimate for the time required for its solution. These times are what we expect a bright and well-prepared student to require for that problem during an exam.

0.1.3 Corrections and Grading of Assignments

Homework assignments will not be graded.

However, you may submit your handwritten solutions by handing them to your tutor directly or uploading them via the Moodle upload interface, see the lecture Moodle page for instructions. The assistants will

Problem 3-8: Low rank approximation of matrices

As explained in the course, large $m \times n$ matrices of low rank $k \ll \min\{m, n\}$ can be stored using only $k(m + n)$ real numbers when using their SVD factorization/representation as sum of tensor products [Lecture → Eq. (3.4.1.9)]. Thus, low rank matrices are of considerable interest for *matrix compression* (see [Lecture → Ex. 3.4.4.24]). Unfortunately, adding two low rank matrices usually leads to an increase of the rank and entails “**recompression**” by computing a low-rank best approximation of the sum. This problem demonstrates an efficient approach to recompression.

Study [Lecture → Section 3.4.4.2] to prepare for this exercise. The algorithms have to be implemented based on EIGEN and they rely on the SVD [Lecture → Section 3.4.2] and QR factorization [Lecture → Section 3.3.3.4].

(3-8.a) ☐ (10 min.) Show that for a matrix $\mathbf{X} \in \mathbb{R}^{m,n}$ the following statements are equivalent:

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

HINT 1 for (3-8.a): Recall [Lecture → Thm. 3.4.1.1] and [Lecture → Lemma 3.4.1.13].

SOLUTION of (3-8.a):

See also [Lecture → Lemma 3.4.4.15].

❶ Denote by σ_i the **singular values** of the matrix $\mathbf{X}$ [Lecture → Def. 3.4.1.3]. According to [Lecture → Lemma 3.4.1.13] the rank of a matrix $\mathbf{X}$ is given by the number of its nonzero singular values, $k = \underset{i}{\operatorname{argmax}}\{\sigma_i \neq 0\}$, or, equivalently, by the dimension of its range (the subspace of $\mathbb{R}^m$ generated by the columns of $\mathbf{X}$), see [Lecture → Def. 2.2.1.3].

(ii)⇒(i): If $\text{rank}(\mathbf{X}) = k$, then $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, with $\mathbf{\Sigma} = \text{diag}(\sigma_1, \dots, \sigma_k, 0, \dots, 0)$, see [Lecture → Eq. (3.4.1.14)].

Thus, we can omit the last rows/columns of the matrices in the decomposition: $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \mathbf{U}_{:,1:k}\mathbf{\Sigma}_{1:k,1:k}(\mathbf{V}_{:,1:k})^\top$.

By defining $\left\{ \begin{array}{l} \mathbf{A} := \mathbf{U}_{:,1:k}\mathbf{\Sigma}_{1:k,1:k} \\ \mathbf{B} := \mathbf{V}_{:,1:k} \end{array} \right\}$, the claim follows.

(ii)⇒(i): The matrix $\mathbf{B}^\top$ corresponds to a linear mapping from $\mathbb{R}^n$ to $\mathbb{R}^k$, with $k \leq n$. Since it has rank k , the range of $\mathbf{B}^\top$ is the whole space $\mathbb{R}^k$.

The matrix $\mathbf{A}$ corresponds to a linear mapping from $\mathbb{R}^k$ to $\mathbb{R}^m$, with $k \leq m$. Since it has rank k , it is injective, i.e. $\dim[\ker(\mathbf{A})] = k - \text{rank}(\mathbf{A}) = 0$, therefore the dimension of its range is k .

Finally, the rank of $\mathbf{X} = \mathbf{A}\mathbf{B}^\top : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is equal to the dimension of its range, which is then k .

❷ An alternate argument for (i)⇒(ii): As $\text{rank}(\mathbf{X}) = \dim \mathcal{R}(\mathbf{X}) = k$, we can find a basis of $\mathcal{R}(\mathbf{X}) \subset \mathbb{R}^m$ composed of k linearly independent vectors and collect them into the columns of a matrix $\mathbf{A} \in \mathbb{R}^{m,k}$. Thus we can write

$$\mathbf{X}\mathbf{z} = \sum_{\ell=1}^k f_\ell(\mathbf{z})(\mathbf{A})_{:, \ell} \quad \forall \mathbf{z} \in \mathbb{R}^n,$$

with *linear functionals* $f_\ell : \mathbb{R}^n \rightarrow \mathbb{R}$. Every f_ℓ has a matrix representation through a row vector:

$$\exists \mathbf{b}_\ell \in \mathbb{R}^n: f_\ell(\mathbf{z}) = \mathbf{b}_\ell^\top \mathbf{z} \quad \forall \mathbf{z} \in \mathbb{R}^n.$$

$$\blacktriangleright \quad \mathbf{X}\mathbf{z} = \sum_{\ell=1}^k (\mathbf{A})_{:, \ell} \mathbf{v}_\ell^\top \mathbf{z} = \mathbf{A}\mathbf{B}^\top \mathbf{z} \quad \forall \mathbf{z} \in \mathbb{R}^n \quad \text{with} \quad \mathbf{B} := [\mathbf{b}_1, \dots, \mathbf{b}_k] \in \mathbb{R}^{n,k}.$$

Of course, there are many more ways to prove the asserted equivalence.



(3-8.b) 🕒 (30 min.) [depends on Sub-problem (3-8.a)]

In the file `lowrankrep.hpp` write an EIGEN-based C++ function

```
std::pair<Eigen::MatrixXd, Eigen::MatrixXd>
factorize_X_AB(const Eigen::MatrixXd & X, unsigned int k);
```

that factorizes the matrix $\mathbf{X} \in \mathbb{R}^{m,n}$ with $\text{rank}(\mathbf{X}) = k$ into $\mathbf{A}\mathbf{B}^\top$, as in Sub-problem (3-8.a), and returns the two matrices $\mathbf{A} \in \mathbb{R}^{m,k}$ and $\mathbf{B} \in \mathbb{R}^{n,k}$ in this order.

The function should issue a warning in case $\text{rank}(\mathbf{X}) = k$ is in doubt.

HINT 1 for (3-8.b): Study [Lecture → Code 3.4.2.1], use EIGEN's facilities for *economical SVD* [Lecture → Section 3.4.2] and pay attention on how to deal with the SVD-factors $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}$ properly.



SOLUTION of (3-8.b):

The following code uses the economical SVD of $\mathbf{X} \in \mathbb{R}^{m,n}$ [Lecture → § 3.4.1.4]:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top, \quad \mathbf{U} \in \mathbb{R}^{m, \min\{m,n\}}, \quad \mathbf{V} \in \mathbb{R}^{n, \min\{m,n\}}, \quad \mathbf{\Sigma} \in \mathbb{R}^{\min\{m,n\}, \min\{m,n\}}.$$

If $\text{rank}(\mathbf{X}) = k \leq \min\{m, n\}$, then only the upper left $k \times k$ -block of $\mathbf{\Sigma}$ is different from zero. Therefore, in this case

$$\mathbf{X} = \mathbf{A}\mathbf{B}^\top \quad \text{with} \quad \mathbf{A} := (\mathbf{U})_{:,1:k} (\mathbf{\Sigma})_{1:k,1:k} \in \mathbb{R}^{m,k}, \quad \mathbf{B} := (\mathbf{V})_{:,1:k} \in \mathbb{R}^{n,k}. \quad (3.8.1)$$

Note that testing from $\text{rank}(\mathbf{X}) = k$ has to rely on the relative size of its singular values, compare [Lecture → Code 3.4.2.5].

The following code implements (3.8.1). Also consult [Lecture → Code 3.4.2.1] as regards the use of EIGEN's built-in functions.

C++11-code 3.8.2: Solution of (3-8.b)

```
2 std::pair<Eigen::MatrixXd, Eigen::MatrixXd> factorize_X_AB(
3     const Eigen::MatrixXd& X, const unsigned int k) {
4     std::size_t m = X.rows(), n = X.cols();
5     Eigen::MatrixXd A, B;
6     constexpr double tol =
7         1e-6; // Tolerance for numerical rank: \ref{ex:svdrank}
8     // TODO: (3-8.b) Define A,B with k columns such that X = A*B^T
9     // Hint: warnings can be displayed with std::cerr, error messages with assert
10    // START
```

```

11  assert(k <= std::min(m, n) && "Rank k cannot be larger than dimensions of X");
12
13  Eigen::JacobiSVD<Eigen::MatrixXd> svd(
14      X, Eigen::ComputeThinU | Eigen::ComputeThinV);
15  // With Eigen::svd you can ask for thin U or V to be computed.
16  // In case of a rectangular m x n matrix,
17  // with j the smaller value among m and n,
18  // there can only be at most j singular values.
19  // The remaining columns of U and V do not correspond
20  // to actual singular vectors and are not computed in thin format.
21
22  Eigen::VectorXd s = svd.singularValues();
23
24  if (k + 1 <= std::min(m, n) && s(k) > tol * s(0)) {
25      // 1.\ condition checks if there is a singular value after k.
26      // (S cannot be larger than the dimensions of X.)
27      // 2.\ condition checks if such singular value
28      // (by definition non-negative) is "numerically" different from 0.
29      std::cerr << "Rank of matrix X is greater than required rank k"
30                  << std::endl;
31  }
32  // We know that k <= std::min(m,n) by the assert
33  A = svd.matrixU().leftCols(k) * s.asDiagonal();
34  B = svd.matrixV().leftCols(k);
35  // END
36  return std::make_pair(A, B);
37 }

```

Get it on  GitLab (lowrankrep.hpp).

To understand the test for $\text{rank}(X) > k$ please revisit [Lecture → Ex. 3.4.2.3], which explains how to check, if a singular value is “relatively close to zero”.

(3-8.c)  (30 min.) Let $A \in \mathbb{R}^{m,k}$, $B \in \mathbb{R}^{n,k}$, with $k \leq \min\{m, n\}$. Based on the **economical QR-factorization** [Lecture → Thm. 3.3.3.4] of both A and B

$$\begin{aligned}
 A &= Q_A R_A, \quad Q_A \in \mathbb{R}^{m,k} \text{ with } Q_A^\top Q_A = I, \quad R_A \in \mathbb{R}^{k,k} \text{ upper triangular,} \\
 B &= Q_B R_B, \quad Q_B \in \mathbb{R}^{n,k} \text{ with } Q_B^\top Q_B = I, \quad R_B \in \mathbb{R}^{k,k} \text{ upper triangular,}
 \end{aligned}$$

devise a way, how the **economical singular value decomposition** $AB^\top = U \Sigma V$ [Lecture → Eq. (3.4.1.5)] of $AB^\top$ can be computed using only the singular value decomposition of a $k \times k$ -matrix.

SOLUTION of (3-8.c):

We insert both QR-factorizations:

$$AB^\top = Q_A R_A R_B^\top Q_B^\top = Q_A S Q_B^\top \quad \text{with} \quad S := R_A R_B^\top \in \mathbb{R}^{k,k}. \quad (3.8.3)$$

Next, we rely on the singular value decomposition of the (small) square matrix S :

$$S = \tilde{U} \Sigma \tilde{V}^\top, \quad \tilde{U}, \Sigma, \tilde{V} \in \mathbb{R}^{k,k}, \quad \tilde{U}, \tilde{V} \text{ orthogonal, } \Sigma \text{ diagonal.} \quad (3.8.4)$$

Then, insert (3.8.4) into (3.8.3):

$$AB^\top = Q_A \underbrace{R_A R_B^\top}_{=\tilde{U} \Sigma \tilde{V}^\top} Q_B^\top = \underbrace{(Q_A \tilde{U})}_{=:U} \Sigma \underbrace{(Q_B \tilde{V})^\top}_{=:V}. \quad (3.8.5)$$

Note that $\mathbf{U} \in \mathbb{R}^{m,k}$, $\mathbf{V} \in \mathbb{R}^{n,k}$, and

$$\begin{aligned}\mathbf{U}^\top \mathbf{U} &= (\mathbf{Q}_A \tilde{\mathbf{U}})^\top \mathbf{Q}_A \tilde{\mathbf{U}} = \tilde{\mathbf{U}}^\top \underbrace{\mathbf{Q}_A^\top \mathbf{Q}_A}_{=\mathbf{I}} \tilde{\mathbf{U}} = \tilde{\mathbf{U}}^\top \tilde{\mathbf{U}} = \mathbf{I}, \\ \mathbf{V}^\top \mathbf{V} &= (\mathbf{Q}_B \tilde{\mathbf{V}})^\top \mathbf{Q}_B \tilde{\mathbf{V}} = \tilde{\mathbf{V}}^\top \underbrace{\mathbf{Q}_B^\top \mathbf{Q}_B}_{=\mathbf{I}} \tilde{\mathbf{V}} = \tilde{\mathbf{V}}^\top \tilde{\mathbf{V}} = \mathbf{I}.\end{aligned}$$

This shows that (3.8.5) really provides an economical SVD of $\mathbf{AB}^\top$.



(3-8.d) 🕒 (30 min.) [depends on Sub-problem (3-8.c)]

Given $\mathbf{A} \in \mathbb{R}^{m,k}$, $\mathbf{B} \in \mathbb{R}^{n,k}$, with $k \ll m, n$, write an efficient C++ function

```
std::tuple<Eigen::MatrixX<double>, Eigen::MatrixX<double>, Eigen::MatrixX<double>>
svd_AB(const Eigen::MatrixX<double> & A, const Eigen::MatrixX<double> & B);
```

that calculates a singular value decomposition (SVD) [Lecture → Thm. 3.4.1.1] of the product $\mathbf{AB}^\top = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, where $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{n,k}$ have orthogonal columns and $\mathbf{\Sigma} \in \mathbb{R}^{k,k}$ is a diagonal matrix. The SVD-factors are returned as a triple $(\mathbf{U}, \mathbf{\Sigma}, \mathbf{V})$.

HINT 1 for (3-8.d): If you directly compute an SVD of $\mathbf{AB}^\top$, you will always have to deal with dimensions m and n , regardless of $k \ll m, n$. As elaborated in Sub-problem (3-8.c) 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!

SOLUTION of (3-8.d):

Of course, (3.8.5) is cheaper than the direct SVD of the $m \times n$ matrix $\mathbf{AB}^\top$, because the SVD has to be computed only for the $k \times k$ matrix $\mathbf{R}_A \mathbf{R}_B^\top$, with $k \ll m, n$.

C++11-code 3.8.6: Solution of (3-8.d)

```
2 std::tuple<Eigen::MatrixX<double>, Eigen::MatrixX<double>, Eigen::MatrixX<double>> svd_AB(
3     const Eigen::MatrixX<double> & A, const Eigen::MatrixX<double> & B) {
4     assert(A.cols() == B.cols() &&
5           "Matrices A and B should have the same column number");
6     assert(A.cols() < A.rows() && A.cols() < B.rows() &&
7           "Matrices A and B should have less columns than rows");
8     std::size_t m = A.rows();
9     std::size_t n = B.rows();
10    std::size_t k = A.cols();
11    Eigen::MatrixX<double> U, S, V;
12    // TODO: (3-8.d) Define the tuple of
13    // matrices U, S, V of the svd decomposition of A*B^T
14    // Hint: You can define a tuple of objects using std::make_tuple
15    // START
16    // QA: m x k; QB: n x k; RA, RB k x k
17    Eigen::HouseholderQR<Eigen::MatrixX<double>> QRA = A.householderQr();
18    Eigen::MatrixX<double> QA =
19        QRA.householderQ() * Eigen::MatrixX<double>::Identity(m, std::min(m, k));
20    Eigen::MatrixX<double> RA = Eigen::MatrixX<double>::Identity(std::min(m, k), m) *
21        QRA.matrixQR().triangularView<Eigen::Upper>();
22    Eigen::HouseholderQR<Eigen::MatrixX<double>> QRB = B.householderQr();
23    Eigen::MatrixX<double> QB =
```

```

24     QRB.householderQ() * Eigen::MatrixXd::Identity(n, std::min(n, k));
25     Eigen::MatrixXd RB = Eigen::MatrixXd::Identity(std::min(n, k), n) *
26         QRB.matrixQR().triangularView<Eigen::Upper>();
27
28     // U,V: k x k
29     Eigen::JacobiSVD<Eigen::MatrixXd> svd(
30         RA * RB.transpose(), Eigen::ComputeFullU | Eigen::ComputeFullV);
31     // Thin matrices are unnecessary here as RA * RB' is
32     // a square k x k matrix!
33     // Moreover, if you had computed the thin matrices U and V
34     // of the direct SVD of AB',
35     // you would still have dealt with the minimum between m and n,
36     // without exploiting k << m, n!
37
38     S = svd.singularValues().asDiagonal();
39     U = svd.matrixU();
40     V = svd.matrixV();
41     // U: m x k; V: n x k
42     U = QA * U;
43     V = QB * V;
44     // END
45     return std::make_tuple(U, S, V);
46 }

```

Get it on  GitLab (lowrankrep.hpp).

Remark. If the matrices **A** and **B** came from the decomposition of a matrix **X** as in the previous subproblems, their columns would be orthogonal (i.e. $\mathbf{A}^\top \mathbf{A}$ and $\mathbf{B}^\top \mathbf{B}$ would be diagonal). Therefore, the triangular part of their QR decompositions would be diagonal, which can be used to speed up Code 3.8.6. However, our code works for every choice of matrices **A**, **B** with the same number of columns k .



(3-8.e) (10 min.) [depends on Sub-problem (3-8.d)]

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

SOLUTION of (3-8.e):

The algorithm can be divided into the following steps:

1. $(\mathbf{A}, \mathbf{B}) \mapsto (\mathbf{Q}_A, \mathbf{R}_A, \mathbf{Q}_B, \mathbf{R}_B)$, i.e. two economical QR decompositions (k Householder transformations of length $n-1, n-2, \dots, n-k+1$): each is worth $O(nk^2)$ operations (see [Lecture $\rightarrow$ Exp. 3.3.3.39]).
2. $(\mathbf{R}_A, \mathbf{R}_B) \mapsto \mathbf{R}_A \mathbf{R}_B^\top$, i.e. a matrix multiplication $O(k^3)$, independent of n .
3. $\mathbf{R}_A \mathbf{R}_B^\top \mapsto (\tilde{\mathbf{U}}, \tilde{\Sigma}, \tilde{\mathbf{V}})$, i.e. an SVD for the $k \times k$ matrix, independent of n .
4. $(\tilde{\mathbf{U}}, \tilde{\mathbf{V}}, \mathbf{Q}_A, \mathbf{Q}_B) \mapsto (\mathbf{Q}_A \tilde{\mathbf{U}}, \mathbf{Q}_B \tilde{\mathbf{V}})$, i.e. two $(n, k) \times (k, k)$ multiplications $O(nk^2)$.

The total computational effort is linear in n .



(3-8.f) (10 min.) If $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{m,n}$ satisfy $\text{rank}(\mathbf{Y}) = \text{rank}(\mathbf{X}) = k$, show that $\text{rank}(\mathbf{Y} + \mathbf{X}) \leq 2k$.

SOLUTION of (3-8.f):

From $\text{rank}(\mathbf{X}) = k$ it follows that there is a basis for the image of $\mathbf{X}$ with k basis vectors, i.e. each column of $\mathbf{X}$ is a linear combination of these k vectors. The same holds for $\mathbf{Y}$ with other k basis vectors, where some can be obtainable from linear combinations of the basis vectors of $\mathbf{X}$. Hence, each column of $\mathbf{X} + \mathbf{Y}$ is a linear combination of *at most* $2k$ vectors. Thus, $\text{rank}(\mathbf{X} + \mathbf{Y}) \leq 2k$.

▲

(3-8.g) (10 min.) 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 the sum $\mathbf{X} + \mathbf{Y}$ as $\mathbf{X} + \mathbf{Y} = \mathbf{A} \mathbf{B}^\top$, with $\mathbf{A} \in \mathbb{R}^{m,2k}$ and $\mathbf{B} \in \mathbb{R}^{n,2k}$.

SOLUTION of (3-8.g):

Solving this problem requires deeper understanding of matrix multiplication.

We can define $\mathbf{A} := [\mathbf{A}_X \ \mathbf{A}_Y] \in \mathbb{R}^{m,2k}$ and $\mathbf{B} := [\mathbf{B}_X \ \mathbf{B}_Y] \in \mathbb{R}^{n,2k}$ by column-wise concatenation of matrices.

Hence, we have that

$$\mathbf{A} \mathbf{B}^\top = [\mathbf{A}_X \ \mathbf{A}_Y] \begin{bmatrix} \mathbf{B}_X^\top \\ \mathbf{B}_Y^\top \end{bmatrix} = \mathbf{A}_X \mathbf{B}_X^\top + \mathbf{A}_Y \mathbf{B}_Y^\top = \mathbf{X} + \mathbf{Y}.$$

▲

(3-8.h) (30 min.) [depends on Sub-problem (3-8.d), Sub-problem (3-8.g)]

Rely on the previous subproblems to write an *efficient* C++ function

```
std::pair<Eigen::MatrixXd, Eigen::MatrixXd>
rank_k_approx(const Eigen::MatrixXd & Ax, const Eigen::MatrixXd &
    Ay,
               const Eigen::MatrixXd & Bx, const Eigen::MatrixXd &
    By);
```

that returns two matrix factors $\mathbf{A}_Z \in \mathbb{R}^{m,k}$ and $\mathbf{B}_Z \in \mathbb{R}^{n,k}$ whose product $\mathbf{Z} = \mathbf{A}_Z \mathbf{B}_Z^\top$ yields the *rank- k best approximation* of the matrix sum $\mathbf{X} + \mathbf{Y} = \mathbf{A}_X \mathbf{B}_X^\top + \mathbf{A}_Y \mathbf{B}_Y^\top$:

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

Here the matrices $\mathbf{A}_X, \mathbf{A}_Y \in \mathbb{R}^{m,k}$ and $\mathbf{B}_X, \mathbf{B}_Y \in \mathbb{R}^{n,k}$ are given and passed to the function through the arguments $\mathbf{A}_X, \mathbf{A}_Y, \mathbf{B}_X$, and $\mathbf{B}_Y$. You may only consider the case where $2k < m, n$.

HINT 1 for (3-8.h): $\|\cdot\|_F$ stands for the Frobenius norm of a matrix (see [Lecture → Def. 3.4.4.17]). Recall [Lecture → Thm. 3.4.4.19] and apply the function implemented in Sub-problem (3-8.d) to the product representation of $\mathbf{X} + \mathbf{Y}$ derived in Sub-problem (3-8.g).

┘

SOLUTION of (3-8.h):

First we appeal to the result of Sub-problem (3-8.g) to write $\mathbf{X} + \mathbf{Y}$ as a product $\mathbf{AB}^\top$ of “tall” matrices. To this product we apply `svd_AB()` from Sub-problem (3-8.d) to compute its economical SVD.

According to [Lecture → Thm. 3.4.4.19], the best low rank approximation of a matrix can be obtained by first calculating the SVD of the matrix and then extracting the first k columns of $\mathbf{U}$ and $\mathbf{V}$ and the upper left $k \times k$ submatrix of Σ .

C++11-code 3.8.7: Solution of (3-8.h)

```

2  std::pair<Eigen::MatrixXd, Eigen::MatrixXd> rank_k_approx(
3      const Eigen::MatrixXd& Ax, const Eigen::MatrixXd& Ay,
4      const Eigen::MatrixXd& Bx, const Eigen::MatrixXd& By) {
5      assert(Ax.rows() == Ay.rows() && Ax.cols() == Ay.cols() &&
6          "Matrices Ax and Ay should have the same dimensions");
7      assert(Bx.rows() == By.rows() && Bx.cols() == By.cols() &&
8          "Matrices Bx and By should have the same dimensions");
9      assert(Ax.cols() == Bx.cols() &&
10         "Matrices Ax, Ay, Bx and By should have the same amount of columns");
11     assert(Ax.cols() * 2 < Ax.rows() && Bx.cols() * 2 < Bx.rows() &&
12         "2*k should not be bigger equal min(m, n)");
13
14     Eigen::MatrixXd Az, Bz;
15     // TODO: (3-8.h) Use the function svd_AB with the appropriate A,B and
16     // return the pair Az, Bz of matrices of rank k with Z = Az*Bz
17     // Hint: you can define pairs of objects using std::make_pair
18     // Hint: use std::get<i>(my_tuple) to extract the i-th element of a tuple
19     // START
20     Eigen::MatrixXd A(Ax.rows(), Ax.cols() + Ay.cols());
21     A << Ax, Ay;
22     Eigen::MatrixXd B(Bx.rows(), Bx.cols() + By.cols());
23     B << Bx, By;
24     // U: m x 2k; S: 2k x 2k; V: n x 2k
25     Eigen::MatrixXd U, S, V;
26     std::tuple<Eigen::MatrixXd, Eigen::MatrixXd, Eigen::MatrixXd> svd_tuple =
27         svd_AB(A, B);
28     U = std::get<0>(svd_tuple);
29     S = std::get<1>(svd_tuple);
30     V = std::get<2>(svd_tuple);
31     std::size_t k = Ax.cols();
32     Az = U.leftCols(k) * S.topLeftCorner(k, k);
33     Bz = V.leftCols(k);
34     // END
35     return std::make_pair(Az, Bz);
36 }

```

Get it on  [GitLab \(lowrankrep.hpp\)](#).

(3-8.i)  (10 min.) [depends on Sub-problem (3-8.h)]

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

SOLUTION of (3-8.i):

`svd_AB()` has linear complexity in n and m , as derived in (3-8.e). The multiplication of the truncated matrices $\mathbf{U}$ and $\mathbf{S}$ requires $O(mk)$ operations (since $\mathbf{S}$ is diagonal). Hence, the total computational effort is $O(n + m)$ for $m, n \rightarrow \infty$.



End Problem 3-8, 170 min.