

1 Low-rank data algorithms

This material is inspired by the course *Open Topics in Applied Mathematics: Fast Algorithms for Big Data* given at Univ. Texas. http://users.ices.utexas.edu/~pgm/Teaching/APP5720_2016s/index.html

Further motivation for the low-rank data algorithms can be found on wikipedia https://en.wikipedia.org/wiki/Low-rank_approximation and http://users.ices.utexas.edu/~pgm/Teaching/APP5720_2016s/lecture01c.pdf

The abbreviation PGM will be used to refer to the PDF-file lecture notes of the “Fast algorithms for big data”: http://users.ices.utexas.edu/~pgm/Teaching/APP5720_2016s/notes15.pdf

1.1 Matrix decompositions

Rather than directly computing a solution to a matrix problem, we will here work with a decomposition of the matrix. Given a matrix $A \in \mathbb{R}^{n \times m}$, a matrix decomposition (equivalently matrix factorization) consists of decomposing the matrix into a product of other matrices, for instance

$$A = BCD.$$

Such decompositions can be useful, since they can simplify the solution of certain matrix problems.

The matrices B, C, D have properties such that one can efficiently solve matrix problems associated with A , or sometimes even directly read-off the solution.

- **Eigval decomposition.** If $A \in \mathbb{R}^{n \times n}$ is a square matrix, the decomposition

$$A = QDQ^{-1} \quad (1.1)$$

where D is diagonal. We can directly *read-off* the eigenvalues and eigenvectors: The eigenvalues are diagonal elements of D , and the columns of Q are eigenvectors.

- **LU decomposition.** If $A \in \mathbb{R}^{n \times n}$ is invertible the Gaussian elimination process gives a decomposition

$$A = LU \quad (1.2)$$

Matrix decompositions are important in all technical and scientific computing, not only in data analysis. The society of industrial and applied mathematics listed the use of decompositions as the most important discovery in computing in the 20th century <https://archive.siam.org/pdf/news/637.pdf>. The FFT-algorithm is also in the list. We will learn about FFT in block 3. The matrix decomposition is one of the ways numerical linear algebra appears in machine learning: <https://machinelearningmastery.com/introduction-to-matrix-decompositions-for-machine-learning/>

where L is upper triangular with ones on the diagonal, and U is upper triangular. We can cheaply compute the solution of the system $Ax = b$ with the algorithm backward substitution.

In the examples above, we intentionally avoided the discussion of existence of such factorizations. In fact, those factorizations do not always exist. It can be shown that (1.1) exists if A is symmetric, since it is a special case of the Jordan decomposition. It can be shown that (1.2) exists if Gaussian elimination does not have division by zero, or equivalently all leading submatrices of A are non-singular.

1.2 QR factorization

1.2.1 Definition and basic application

Our first tool for analyzing data is the QR-factorization, which exists in two versions.

Definition 1.2.1 (QR-factorization (full)). *The full QR-factorization is a matrix decomposition of a rectangular matrix $A \in \mathbb{R}^{n \times m}$*

$$A = QR$$

where

- $Q \in \mathbb{R}^{n \times n}$ is an orthogonal matrix (see background material)
- $R \in \mathbb{R}^{n \times m}$ is an upper triangular matrix

If the matrix A is tall and skinny ($n > m$), we usually relax the condition that Q should be square and instead have a square R -matrix. This called the skinny QR-factorization.

Definition 1.2.2 (QR-factorization (skinny)). *The skinny QR-factorization is a matrix decomposition of a rectangular matrix $A \in \mathbb{R}^{n \times m}$ with $n > m$*

$$A = QR$$

where

- $Q \in \mathbb{R}^{n \times m}$ is an orthogonal matrix (see background material)
- $R \in \mathbb{R}^{m \times m}$ is an upper triangular matrix

Example: QR-factorization

The full QR-factorization of the matrix

$$A = \begin{bmatrix} \sqrt{2} & 0 \\ 1 & -1 \\ 1 & 1 \end{bmatrix}, \quad (1.3)$$

The backslash operator (in MATLAB as well as Julia) for a rectangular matrix, is based on computing a QR-factorization

A rectangular upper triangular matrix is referring to a matrix with the non-zero structure

$$R = \begin{bmatrix} \times & \times & \times & \times \\ & \times & \times & \times \\ & & \times & \times \\ & & & \times \end{bmatrix}.$$

The skinny QR-factorization can be computed in matlab with the command `[Q,R]=qr(A,0)`

is given by

$$Q = \frac{1}{2\sqrt{2}} \begin{bmatrix} 2 & 0 & 2 \\ \sqrt{2} & -2 & -\sqrt{2} \\ \sqrt{2} & 2 & -\sqrt{2} \end{bmatrix}, R = \begin{bmatrix} 2 & 0 \\ 0 & \sqrt{2} \\ 0 & 0 \end{bmatrix}$$

The corresponding skinny QR-factorization is obtained from the full QR-factorization by removing the last columns of Q and the last rows of R :

$$Q = \frac{1}{2\sqrt{2}} \begin{bmatrix} 2 & 0 \\ \sqrt{2} & -2 \\ \sqrt{2} & 2 \end{bmatrix}, R = \begin{bmatrix} 2 & 0 \\ 0 & \sqrt{2} \end{bmatrix}$$



The most important application of the QR-factorization is its use for solving linear least squares problems (also known as linear overdetermined systems). We are used to analyzing the solution

$$\min_{x \in \mathbb{R}^m} \|Ax - b\|_2 \quad (1.4)$$

with the normal equations

$$A^T Ax = A^T b.$$

This can be avoided with the QR-factorization. Since we can simplify $\|Ax - b\| = \|QRx - QQ^T b\| = \|Q(Rx - Q^T b)\| = \|Rx - Q^T b\|$, the argmin expression can also be simplified:

$$x_* = \operatorname{argmin}_{x \in \mathbb{R}^m} \|Ax - b\| = \operatorname{argmin}_{x \in \mathbb{R}^m} \|Rx - Q^T b\|$$

The right-hand side least squares system corresponds to minimization of the vector

$$\begin{bmatrix} \times & \times & \times & \times \\ & \times & \times & \times \\ & & \times & \times \\ & & & \times \end{bmatrix} x - \begin{bmatrix} \times \\ \times \\ \times \\ \times \\ \times \\ \times \end{bmatrix}.$$

Note that the last elements are $n - m$ elements of this vector do not depend on x . Therefore, the minimizer is unchanged if we only minimize the first m elements. This can be formalized as follows. Let $R_1 \in \mathbb{R}^{m \times m}$ be the top block of R and Q_1 the first m columns of Q :

$$R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}, Q = [Q_1 \quad Q_2]. \quad (1.5)$$

Note: if Q is an orthogonal

$$\|Qx\|_2^2 = (Qx)^T(Qx) = x^T Q^T Q x = x^T x = \|x\|^2.$$

Note: The 0 in (1.5) refers to a zero matrix (slight abuse of notation).

We can now expand the minimization problem and observe that the minimized is the same as the reduced problem.

$$\begin{aligned}
 \operatorname{argmin}_{x \in \mathbb{R}^m} \|Rx - Q^T b\|_2 &= \operatorname{argmin}_{x \in \mathbb{R}^m} \|Rx - Q^T b\|_2^2 \\
 &= \operatorname{argmin}_{x \in \mathbb{R}^m} (Rx - Q^T b)^T (Rx - Q^T b) \\
 &= \operatorname{argmin}_{x \in \mathbb{R}^m} (x^T R^T R x - b^T Q R x - x^T R^T Q^T b + b^T Q Q^T b) \\
 &= \operatorname{argmin}_{x \in \mathbb{R}^m} (x^T R_1^T R_1 x - b^T Q_1 R_1 x - x^T R_1^T Q_1^T b + b^T Q_1 Q_1^T b + b^T Q_2 Q_2^T b) \\
 &= \operatorname{argmin}_{x \in \mathbb{R}^m} \|R_1 x - Q_1^T b\|_2^2
 \end{aligned}$$

The matrix R_1 is a square matrix, and in general invertible. If it is invertible, the minimizer of $\|R_1 x - Q_1^T b\|$ is zero and the solution is given explicitly.

Theorem 1.2.3 (Least squares solution with QR-factorization). *Given $A \in \mathbb{R}^{n \times m}$, let $A = QR$ be a skinny QR-factorization of A , where Q and R are partitioned as in (1.5). Suppose R_1 is invertible, then, the linear least squares problem (1.4) has a solution given by the linear system*

$$\operatorname{argmin}_{x \in \mathbb{R}^m} \|Ax - b\|_2 = R_1^{-1} Q_1^T b.$$

Note that the right-hand side corresponds to a linear system with a triangular matrix. Triangular linear systems can be done efficiently with backward substitution.

QR-factorization (cont)

A linear least squares problem $\min \|Ax - b\|$ with A defined in (1.3) and $b = [1 \ 2 \ 3]^T$ can be directly solved with the above technique. The linear system (1.6) becomes

$$x = R_1^{-1} Q_1^T b = \frac{1}{2\sqrt{2}} \begin{bmatrix} 2 & 0 \\ 0 & \sqrt{2} \end{bmatrix}^{-1} \begin{bmatrix} 2 & 0 \\ \sqrt{2} & -2 \\ \sqrt{2} & 2 \end{bmatrix}^T \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}+5}{4} \\ \frac{1}{2} \end{bmatrix} \approx \begin{bmatrix} 1.60 \\ 0.5 \end{bmatrix}$$

This can be compared with the code:

```
julia> A=[sqrt(2) 0; 1 -1; 1 1]
3x2 Array{Float64,2}:
 1.41421  0.0
 1.0      -1.0
 1.0      1.0
julia> b=[1;2;3]
julia> A\b
```

```
2-element Array{Float64,1}:
 1.603553390593273
 0.5
```



1.2.2 Computing the QR-factorization with Gram-Schmidt method

The Gram-Schmidt method (usually covered in basic linear algebra course) can be used to compute a QR-factorization. The Gram-Schmidt method is usually stated in this way: We want to compute an orthogonal basis of the space span by the vectors $v_1, \dots, v_p \in \mathbb{C}^n$. It can be done by the constructive relations

$$w_1 = v_1 \quad (1.6a)$$

$$w_2 = v_2 - \frac{w_1^T v_2}{w_1^T w_1} w_1 \quad (1.6b)$$

$$w_3 = v_3 - \frac{w_1^T v_3}{w_1^T w_1} w_1 - \frac{w_2^T v_3}{w_2^T w_2} w_2 \quad (1.6c)$$

$$\vdots \quad (1.6d)$$

$$w_p = v_p - \frac{w_1^T v_p}{w_1^T w_1} w_1 - \dots - \frac{w_{p-1}^T v_p}{w_{p-1}^T w_{p-1}} w_{p-1} \quad (1.6e)$$

The vectors w_1, \dots, w_p form an orthogonal basis. An orthonormal basis is obtained by normalizing the vectors

$$q_i = w_i / \|w_i\|, \quad i = 1, \dots, p.$$

By inserting $w_i = q_i \|w_i\|$ and using that $\|q_i\| = 1$ we can see that

$$\frac{w_i^T v_j}{w_i^T w_i} w_i = (q_i^T v_j) q_i$$

and we can change the orthogonalization to involve q -vectors:

$$\|w_1\| q_1 = v_1 \quad (1.7a)$$

$$\|w_2\| q_2 = v_2 - (q_1^T v_2) q_1 \quad (1.7b)$$

$$\|w_3\| q_3 = v_3 - (q_1^T v_3) q_1 - (q_2^T v_3) q_2 \quad (1.7c)$$

$$\vdots \quad (1.7d)$$

$$\|w_p\| q_p = v_p - (q_1^T v_p) q_1 - \dots - (q_{p-1}^T v_p) q_{p-1} \quad (1.7e)$$

We now see that the relation (1.7) can be expressed as a matrix equation

$$V = QR \quad (1.8)$$

We here learn how to compute the QR-factorization with the Gram-Schmidt method. In other courses SF2524 you can learn more advanced techniques based on Householder reflectors.

Throughout this course we will let the capital letter denote the matrix consisting of the corresponding vectors with small letters:

Lecture notes - Elias Jarlebring - Spring 2020 $V = [v_1 \dots v_p]$ and $Q = [q_1 \dots q_p]$

where R is an upper triangular matrix with elements given by

$$r_{i,j} = \begin{cases} 0 & \text{if } j < i \\ q_i^T v_j & \text{if } j > i \\ \|w_i\| & \text{if } i = j. \end{cases}$$

Since Q is an orthogonal matrix, (1.8) is QR-factorization.

A practical procedure: GS by column elimination

In practice the procedure (1.7) is not carried out one row at a time, but instead one column at a time and updating a matrix A which contains not yet orthogonalized elements.

We derive the procedure by constructing a sequence of matrices that satisfy

$$V = Q_j R_j + A_j \quad (1.9)$$

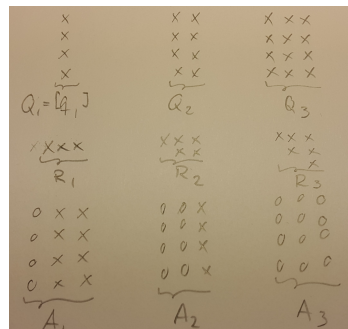
where $R_j \in \mathbb{C}^{j \times p}$, $A_j \in \mathbb{C}^{n \times p}$ and $Q_j \in \mathbb{C}^{n \times j}$ is orthogonal. Moreover, we say that Q_j is orthogonal to A_j , i.e. $Q_j^T A_j = 0$. Clearly, (1.9) is a QR-factorization if $\|A_j\| = 0$.

The matrices are assumed to be expansions of each other

$$Q_j = [Q_{j-1} \quad q_j], \quad R_j = \begin{bmatrix} R_{j-1} \\ r_j^T \end{bmatrix}$$

Elimination

Typical structure of elimination (alt 1 below):



From these relations it follows by induction that

$$\begin{aligned} V &= [Q_{j-1} \quad q_j] \begin{bmatrix} R_{j-1} \\ r_j^T \end{bmatrix} + A_j \\ &= \underbrace{Q_{j-1} R_{j-1} + A_j}_{=V-A_{j-1}} + q_j r_j^T \end{aligned}$$

Hence,

$$A_j = A_{j-1} - q_j r_j^T. \quad (1.10)$$

Since, A_j should be orthogonal to Q_j it is also orthogonal to q_j :

$$0 = q_j^T A_j = q_j^T A_{j-1} - q_j^T q_j r_j^T$$

such that

$$r_j^T = q_j^T A_{j-1} \quad (1.11)$$

In order form a constructive algorithm it remains to select a q_j vector.

We will select it such that

$$q_j = A_{i-1}(:, i) / \|A_{i-1}(:, i)\| \quad (1.12)$$

where i is selected either as

- Alt 1 (standard): Leads to a standard QR-factorization

$$i = j \quad (1.13)$$

- Alt 2 (greedy): Leads to “morally” triangular R -matrix (PGM Figure 1.1):

$$i = \underset{i}{\operatorname{argmax}} \|A_{i-1}(:, i)\|. \quad (1.14)$$

Both algorithms terminate after at most p iterations. However, it turns out the greedy algorithm is likely to make $\|A_j\|$ smaller faster. On a computer (in finite floating point arithmetic) we may want to terminate the algorithm earlier than after p steps.

```

Initialize:  $Q_0 = [], R_j = [], A_0 = V, p = \min(m, n)$ 
for  $j = 1, 2, \dots, p$  do
    Compute  $q_j$  according to (1.12) with either (1.13) or (1.14).
    Set  $r_j^T = q_j^T A_{j-1}$ 
    Set  $Q_j = [Q_{j-1}, q_j]$ 
    Set  $R_j = \begin{bmatrix} R_{j-1} \\ r_j^T \end{bmatrix}$ 
    Set  $A_j = A_{j-1} - q_j r_j^T$ 
end
  
```

Algorithm 1: GS by column elimination

The Algorithm 1 has several drawbacks which are not difficult to circumvent by reordering and condensing the operations. Rather than updating two matrices A and Q we can use only one matrix, if we rearrange the rows. The matrix R can be computed in a direct way rather than being expanded. The way the columns are reordered can also be directly obtained. We present this in Algorithm 2, where we also introduce a truncation, based on the parameter tol .

Continue reading PGM 1.5.1-1.5.3. PGM Figure 1.1 is Algorithm 1 with (1.14) and Algorithm 2 is the algorithm in PGM Figure 1.3 for the special case (1.14) but with an additional reorthogonalization step. (Be-ware: Typo/Error in PGM Figure 1.1-1.3: argmin should be argmax .)

```

[Q,R,J]=CPQR_via_GS(V,tol)
Initialize: Q = V, R=zeros(p,n), p = min(m,n), s = n
for j = 1,2,...,n do
    Compute i with either (1.13) or (1.14) with A replaced by Q.
    Switch J(j) ↔ J(i), R(:,j) ↔ R(:,i) and Q(:,j) ↔ Q(:,i)
    Set ρ = ‖Q(:,j)‖
    Set R(j,j) = ρ and q = Q(:,j)/ρ
    Update Q(:,j) = q
    Compute rT = qTQ(:,(j+1):n) and R(j,(j+1):n) = rT
    Update Q(:,(j+1):n) = Q(:,(j+1):n) - qrT
    Compute indicator t = ‖Q(:,(j+1):n)‖fro
    if t < tol then
        Set s = j
        break
    end
end
Return (Q(:,1:s), R(1:s,:), J(1:s))

```

Algorithm 2: CPQR - Column pivoted QR - via GS. The algorithm returns a factorization such that $AP \approx QR$ where P is a permutation matrix. If tol is set to zero, the approximation is equality in exact arithmetic. In the notation of index vectors (as we will study in more detail in Section 1.6.1) we can write $A(:,J) = QR$.

1.2.3 Low-rank approximation via Algorithm 1 truncation

The procedure above can be used to compute approximate low-rank factorizations. Suppose we truncate Algorithm 1 or Algorithm 2 at step $s < p$. By construction we have

$$V = Q_s R_s + A_s.$$

If the matrix A_s is small, $Q_s R_s$ is an approximation of V , and $Q_s R_s$ has

$$\text{rank}(Q_s R_s) = s \quad (1.15)$$

if R_s is non-singular. Hence,

$$V \approx Q_s R_s$$

is a rank- s approximation of V , with an approximation error given by

$$\|V - Q_s R_s\| = \|A_s\|.$$

If $\|A_s\|$ goes to zero fast, this will be a good approximation. The greedy column selection strategy (defined by (1.14)) in general makes $\|A_s\|$ small faster than the standard version (defined by (1.13)) and can therefore be much better in its truncated form.

Equation (1.15) only holds if R_s is non-singular. If R_s is singular, $\text{rank}(Q_s R_s) < s$, which is even better.

1.3 Singular value decomposition (SVD)

1.3.1 SVD and best low-rank approximation

Although the approximation in Section 1.2.3 can be used to construct a low-rank approximation, it is by no means guaranteed that this is the best low rank approximation. We will now learn techniques to obtain the best low rank approximation.

Problem: Given a matrix $V \in \mathbb{R}^{n \times m}$ compute the best rank k approximation:

$$\min_{\substack{X \in \mathbb{R}^{n \times m} \\ \text{rank}(X)=k}} \|V - X\| \quad (1.16)$$

To study this, we need the matrix factorization known as the singular value decomposition, defined by relation (1.17) in the following theorem.

Theorem 1.3.1 (Singular value decomposition). *If $A \in \mathbb{R}^{n \times m}$, then there exist orthogonal matrices $U \in \mathbb{R}^{n \times n}$ and $V \in \mathbb{R}^{m \times m}$ such that*

$$A = U \Sigma V^T \quad (1.17)$$

where

$$\Sigma = \text{diag}(\sigma_1, \dots, \sigma_p) \in \mathbb{R}^{n \times m} \quad (1.18)$$

where $p = \min(n, m)$ and the singular values are non-negative and ordered

$$\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_p \geq 0.$$

The minimization in (1.16) is for the matrix spectral norm. The rest of the study is heavily based on this.

In the definition of Σ in (1.18) we mean a rectangular diagonal matrix, which can be either

$$\begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_p \end{bmatrix}$$

or

$$\begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_p \end{bmatrix}.$$

1.3.2 Eckart-Young theorem

The relation of best approximation and singular value decomposition is characterized with the Eckart-Young theorem, which basically states that the best approximation is given by selecting parts of the SVD

$$X = U_1 \Sigma_1 V_1^T$$

where

$$U = [U_1, U_2], \quad \Sigma = \text{diag}(\sigma_1, \dots, \sigma_k), \quad V = [V_1, V_2].$$

Theorem 1.3.2 (Eckart-Young theorem). *Let $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_p)$, $V = [v_1, \dots, v_p]$ and $U = [u_1, \dots, u_p]$ be a SVD of the matrix V . Then, the minimizer of (1.16) is given by*

$$X = \sum_{i=1}^k u_i v_i^T \sigma_i.$$

Reading material: Golub and Van Loan (GVL), *Matrix computations* (4th edition) section 2.4.1, theorem 2.4.8 and section 2.4.3. Pages available in CANVAS (where slides PDF-files are located).

Moreover, the error is given by

$$\|V - X\| = \sigma_{k+1}.$$

1.3.3 SVD properties

Some properties of the SVD:

- The eigenvalues of

$$A^T A$$

are the singular values squared: $\sigma_i = \sqrt{\lambda_i(A^T A)}$.

- The singular values are the positive eigenvalues of

$$\begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}. \quad (1.19)$$

- The spectral norm is given by the largest singular value

$$\|A\| = \sigma_1$$

where σ_1 is the largest singular value of A .

- There are round-off errors in all numerical computations. The standard definition of the rank of a matrix is not robust with respect to small changes (round-off errors) in the matrix element. In numerical computations we instead use the *numerical rank*, which is a generalization of rank, where we use the singular values. The matrix A has numerical rank k with tolerance ε if

$$\sigma_1 \geq \dots \geq \sigma_k \geq \varepsilon > \sigma_{k+1} \geq \dots \geq \sigma_p \geq 0.$$

In other words, k singular values are smaller than ε . Note that ε is a (user-defined) tolerance and the numerical rank will depend on it.

The standard choice is $\varepsilon = \max(m, n)\varepsilon_{\text{mach}}$.

Notes on classical methods for the SVD (not a part of the course):

Dense problems: structured QR-method (an eigenvalue algorithm) for the matrix (1.19) is implemented in `svd`. Large and sparse: Krylov method applied to (1.19) is implemented in `svds`.

1.4 SVD via Algorithm 1

The greedy Algorithm 1, can be used to compute a partial SVD as follows. If we apply the algorithm to $W \in \mathbb{R}^{n \times m}$ we know that for every step in the iteration the relation is satisfied:

$$W = Q_j R_j + A_j.$$

Section 1.4 is further described in PGM 1.5.4, where Algorithm 1 is viewed as a column pivoted QR-factorization.

Note that $R_j \in \mathbb{R}^{m \times m}$ is small matrix if W is tall and skinny. Therefore, we can use a SVD-method for small problems for that matrix

$$R_j = \hat{U} \Sigma V^T.$$

Clearly,

$$W = U \Sigma V^T + A_j$$

if we define

$$U = Q_j \hat{U}$$

Note that U is orthogonal, since it is a product of orthogonal matrices. If A_j is small $U \Sigma V^T$ will be an approximate (rank j) SVD-factorization of W .

1.5 Randomized SVD

We have seen that the SVD can be used to obtain the best low-rank approximation

$$A \approx U D V^T. \quad (1.20)$$

where $A \in \mathbb{R}^{m \times n}$, $U \in \mathbb{R}^{m \times k}$, $V \in \mathbb{R}^{n \times k}$ and $D \in \mathbb{R}^{k \times k}$. In this section we will see an algorithm to compute the right-hand side of (1.20) (or a partial SVD-factorization) based on first on obtaining an approximate range of the matrix. We proceed in two stages. Recall: range a matrix is the span of its columns.

Stage A. We will try to find an orthogonal matrix $Q \in \mathbb{R}^{m \times s}$ such that

$$A \approx Q Q^T A.$$

Stage B. Given the orthogonal matrix we form the factoris in the right-hand side of (1.20) similar to how we modified Algorithm 1 in section 1.4:

1. Compute $B = Q^T A$
2. Compute the SVD of (the small matrix) B : $B = \hat{U} D V^T$.
3. Compute $U = Q \hat{U}$.

The approximation of the procedure is only due to stage A, due to the fact that:

$$\|A - Q Q^T A\| = \|A - Q B\| = \|A - Q \hat{U} D V^T\| = \|A - U D V^T\|.$$

If the matrix A is tall and skinny, and if $k \ll \min(n, m)$ then the main computational effort lies in Stage A. Therefore, we now consider an approach for Stage A.

The section 1.5 is a general description of PGM Chapter 2.

We already learned that Algorithm 1 can be used to compute a skinny QR-factorization. A better QR-factorization method is implemented in MATLAB command `qr(Y, 0)`.

The idea can be viewed as follows. The range of A is clearly a superset of the range of AG , for any matrix G . If $G \in \mathbb{R}^{n \times s}$, where $s < m$, the matrix AG will have less columns than A . We compute the matrix AG and carry compute an orthogonal basis of the range using the QR-factorization. This will be cheaper than computing a QR-factorization of A if $s \ll m$.

This leads to “Algorithm: Basic randomized SVD” in PGM.

Also part of the course

- PGM 2.4: Single pass and symmetric matrices
- PGM 2.6.1: Theoretical convergence bounds

1.6 CID and CUR

The need for other types of factorizations can be illustrated with an example (modified from...)

PGM Chapter 4. (The contents of PGM chapter 3 is not a part of the course, but it is treated in the course SF2524.)

1.6.1 Notation and interpolatory decomposition

Application: DNA

Suppose you are contacted by a life-science researcher who has data corresponding to individuals, say encoding of DNA-strings of persons with a particular type of disease. We store the data of each individual as columns in a matrix:

```
>> A=[ 32    27    31    25    26
      15    30    18    33    32
      32    35    34    34    34
      35    24    35    26    23
      26    26    26    27    28
      31     7    30     7     7
      15    17    13    18    17
      29    22    28    21    23
      29    30    28    29    30
      0     9    2     9    31];
```



The double helix information can be encoded into (huge) vectors with positive integers.

Since you have completed this course (and therefore an expert on numerical algorithms for data science), your life-science collaborator has asked you to analyze the data by extracting important characteristics. As we learned earlier, the SVD and the dominant singular vectors correspond to important directions in the data. It turns out that this is not very informative to your collaborator since the data can only be interpreted if it is discrete and positive, which in general the singular vectors are not. A DNA sequence encoding only makes sense if it is positive and discrete.

Instead, we will now learn a decomposition which identifies important columns (corresponding to individuals in this example). In the data above, columns one and three are similar and can be approximated by only one column from the matrix, and correspondingly columns two, four and five are similar such that the entire set of data can be represented by only two columns (individuals). The individuals can then be further studied to identify further properties.



We consider first the exact relation and later discuss approximations. The following defines the interpolatory decomposition, which we sometimes call column interpolatory decomposition (CID) to distinguish from the corresponding algorithm for rows.

Definition 1.6.1 (Interpolatory decomposition (ID)). *Given a matrix $A \in \mathbb{R}^{m \times n}$, a decomposition of A as product*

$$A = CZ \quad (1.21)$$

is called an interpolatory decomposition (ID) if the columns of $C \in \mathbb{R}^{m \times k}$ are copies of columns of A (also referred to as a selection of the columns of A).

Application: DNA (cont.)

This shows that the ID is easier to interpret in applied fields. There are other advantages of the ID. ID preserves properties, for instance sparsity patterns. The decomposition is also more memory efficient, since the C vectors do not have to be stored, but instead pointers to columns in A , which we will call index vectors.



To manipulate IDs and related factorizations, we will use the notation (similar to the syntax in MATLAB and Julia), involving permutation matrices and index vectors

Definition 1.6.2 (Index vector). *The vector $J \in \mathbb{Z}^k$ is called an index vector, when we use it to select columns of a matrix. If $A = [a_1, \dots, a_n] \in \mathbb{R}^{m \times n}$, then a corresponding index vector $J \in \{1, \dots, m\}^k$ satisfies*

$$A(:, J) = [a_{j_1}, \dots, a_{j_k}].$$

A complement index vector is denoted \bar{J} and denotes indices which are not a part of J , such that J and \bar{J} form a disjoint partitioning of $\{1, \dots, n\}$:

$$\{1, \dots, n\} = J \cup \bar{J}.$$

Note that an index vector corresponds to a selection of columns, so the C -matrix in the ID definition (1.21) must satisfy

$$C = A(:, J)$$

for some index vector $J \in \{1, \dots, n\}^k$.

Definition 1.6.3 (Permutation matrix). *A matrix formed by exchanging columns of the identity matrix is called a permutation matrix.*

There is a useful relation between index vectors and permutation matrices. The action of an index vector (of length k) can be expressed in terms of a permutation matrix:

$$A(:, J) = AP(:, 1:k) \quad (1.22)$$

where P is the permutation matrix

$$P = [I(:, 1:J), I(:, \bar{J})] \quad (1.23)$$

and \bar{J} is a complement index vector.

————— Example: Index vectors —————

Let A be the matrix

$$A = \begin{bmatrix} 1 & 2 & -1 \\ 3 & 4 & -2 \\ 5 & 6 & -3 \\ 8 & 9 & -4 \end{bmatrix}$$

The index vector $J = [3 \ 1]$ corresponds to selecting columns 3 and 1:

$$A(:, J) = \begin{bmatrix} -1 & 1 \\ -2 & 3 \\ -3 & 5 \\ -4 & 8 \end{bmatrix} \quad (1.24)$$

The complement index vector is $\bar{J} = [2]$. Hence, the permutation matrix (1.23) corresponds to selecting columns 3, 1, 2 of the identity matrix

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

This MATLAB code illustrates relation (1.22) and verifies (1.24).

```
>> A= [1 2 -1 ; 3 4 -2 ; 5 6 -3 ; 8 9 -4];
>> I=eye(3);
>> J=[3 1]; Jbar=2;
>> P=[I(:,J), I(:,Jbar)];
>> A*P(:,1:2)
```

ans =

-1	1
-2	3
-3	5
-4	8



1.6.2 The standard ID

The Z matrix can in general be assumed to have a structure. First note that the index vector notation allows us to denote $C = A(:, J)$ and therefore write (1.21) as

$$A = A(:, J)Z$$

for an index vector Z^k and

$$A = AP(:, 1:k)Z$$

for a permutation matrix P . The index vector J and the permutation matrix P are related by (1.23).

Suppose j is the first selected column (first column of C), such that

$$C(:, 1) = A(:, j) \quad (1.25)$$

and consider the j th column of the equation (1.21)

$$A(:, j) = CZ(:, j). \quad (1.26)$$

By combining (1.25) and (1.26) we see that a natural choice of the j th column of Z is

$$Z(:, j) = e_1 \quad (1.27)$$

where e_1 is the first unit vector. The above reasoning can be repeated for all selected columns. We select some columns of Z as

$$Z(:, J) = I \in \mathbb{R}^{k \times k} \quad (1.28)$$

where I is the identity matrix.

Hence, in order to uniquely determine Z we only need to also find the other column coefficients which we denote T :

$$Z(:, \bar{J}) = T \in \mathbb{R}^{k \times (m-k)}$$

The values T are usually referred to as the *expansion coefficients*, since they say what linear combination (expansion) of the selected columns $A(:, J)$ we should use to get the nonselected columns $A(:, \bar{J})$.

The choice (1.27) is not the only choice, but it can always be done. See other choices in wiki 2019: Problem 1-15.

An interpolatory decomposition where Z is selected as (1.28) is called a standard ID.

Another way of writing Z is

$$\begin{aligned} Z(:, [J, \bar{J}]) &= [Z(J), Z(\bar{J})] = [I \quad T] \\ ZP &= [I \quad T] \\ Z &= [I \quad T] P^T \end{aligned}$$

where P is the permutation matrix corresponding to the index vector $[J, \bar{J}]$.

1.6.3 Computing ID from Algorithm 1

A deeper analysis of Algorithm 1 reveals that it can be viewed as a decomposition of the type

$$AP = QR$$

where P is a permutation matrix and R is upper triangular (also in the greedy choice). The iterates of the algorithms satisfy

$$AP_k = Q_k R_k + E_k$$

for some permutation matrix P_k , and $\|E_k\|$ is hopefully small. Moreover, the first columns of the permutation matrix are given by the selections in (1.14). We separate the equation into two blocks, the first k columns, and the $n - k$ columns:

$$\begin{aligned} A(:, J_k) &= Q_k R_k(:, 1:k) + E_k(:, 1:k) \\ A(:, \bar{J}_k) &= Q_k R_k(:, k+1:n) + E_k(:, k+1:n) \end{aligned}$$

We now solve the first equation for Q_k

$$Q_k = A(:, J_k) R_k^{-1} - E_k(:, 1:k) R_k^{-1}$$

and insert into the second equation

$$\begin{aligned} A(:, \bar{J}_k) &= A(:, J_k) R_k^{-1} R_k(:, k+1:n) - E_k(:, 1:k) R_k^{-1} R_k(:, k+1:n) + E_k(:, k+1:n) \\ &= A(:, J_k) R_k^{-1} R_k(:, k+1:n) + \mathcal{O}(\|E_k\|) \end{aligned}$$

If $\|E_k\| \ll 1$ we can approximate

$$A(:, [J_k, \bar{J}_k]) = [A(:, J_k), A(:, \bar{J}_k)] \approx A(:, J_k) [I \quad R_k^{-1} R_k(:, k+1:n)]$$

Since $A(:, [J_k, \bar{J}_k]) = AP$ we have

$$A \approx A(:, J_k) [I \quad R_k^{-1} R_k(:, k+1:n)] P^T.$$

This directly leads us to an algorithm to compute the ID factorization from the CPQR-factorization. This can be done either as exactly or

with a truncation, if we apply Algorithm 2 with a tolerance. In Algorithm 3 the CPQR-computation step can either be a call to Algorithm 2 with a tolerance, or in MATLAB the command `qr(A,0)`. A call to the truncated version (with a greedy column selection) can lead to very accurate ID factorizations.

Input: The matrix $A \in \mathbb{R}^{m \times n}$ and either a tolerance or target rank k
 Compute Q, R, J as a CPQR-factorization of A
 Let k be the number of columns of Q
 $T = R(1:k, k:k)^{-1} R(1:k, (k+1):n)$
 $Z = \text{zeros}(k, n)$
 $Z(:, J) = [I, T]$ where $I \in \mathbb{R}^{k \times k}$ is an identity matrix
 $J_s = J(1:k)$
 Return J_s and Z .

Algorithm 3: Column ID via CPQR

Input: The matrix $A \in \mathbb{R}^{m \times n}$ and either a tolerance or target rank k
 Compute Q, R, J as a CPQR-factorization of A^T
 Let k be the number of columns of Q
 $T = R(1:k, k:k)^{-1} R(1:k, (k+1):n)$
 $X = \text{zeros}(m, k)$
 $X(:, J) = [I, T]^T$ where $I \in \mathbb{R}^{k \times k}$ is an identity matrix
 $I_s = J(1:k)$
 Return I_s and X .

Algorithm 4: Row ID via CPQR

$[J_s, Z] = \text{id_col}(A, k)$
 $[I_s, X] = \text{id_row}(A(:, J_s), k)$
 Return I_s, J_s, X, Z

Algorithm 5: Double sided ID via column ID and row ID

1.6.4 Row ID and double sided ID

The objective with the above procedure was to express all the *columns* with a reduced set of the columns. First, we note that the same procedure can be applied to express the *rows* with a reduced set of the rows. The procedure (summarized in Algorithm 4) is completely analogous.

It turns out you can also apply both procedures and express. We

derive the factorization constructively. First we carry out a column ID:

$$A \approx A(:, J_s)Z.$$

We now carry out *row ID on the already selected columns*

$$C = A(:, J_s) = XA(I_s, J_s).$$

Note that is in fact an exact relation (no approximation), due to the fact that $A(I_s, J_s)$ is $k \times k$ -matrix and if it is invertible (which it is A has full rank) X is given uniquely by $X = A(:, J_s)A(I_s, J_s)^{-1}$.

If we combine the above steps we see that

$$A \approx XA(I_s, J_s)Z \quad (1.29)$$

which is what we call the *double sided interpolatory decomposition*. It can be computed with Algorithm 5.

The double sided ID is further described in PGM 4.3.3.

1.6.5 Computing ID via randomized Sampling

PGM section 4.4

1.6.6 CUR

The double sided ID made a selection of both rows and columns and constructed a factorization where the original data is in the middle matrix in (1.29). We will now study another factorization which also involves a selection of rows of columns, but where the original data are in the first and last matrix:

$$A \approx CUR \quad (1.30)$$

where $C = A(:, J_s)$ and $R = A(I_s, :)$, for some index vectors J_s and I_s . To compute this factorization we need to find J_s , I_s and a matrix $U \in \mathbb{R}^{k \times k}$.

The CUR is considerably more useful than the double sided ID from an application perspective. Note that it contains more of the original data. Although we will now illustrate how one can compute a CUR decomposition from a double sided ID, it involves an approximation which will introduce further approximation errors.

The canonical way to construct a CUR from a double-side ID is to use the same row and column selection. That is, we compute a double-sided ID which leads to X , J_s and I_s . We use I_s and J_s in the approximation (1.30) and do not use the matrix X . In order to compute (1.30) it remains to determine $U \in \mathbb{R}^{k \times k}$. By viewing the determination of U as a least squares problem, it is possible to express it in terms of

The variables in (1.30), stem from what it represents. The matrix C is a selection of Columns, the matrix R is a selection of Rows.

pseudoinverses. Unfortunately, pseudoinverses are in general expensive to compute and we will use other less accurate approaches, which are more efficient.

Note due to the way we constructed the double-sided ID by first computing a column ID, we have that

$$A \approx CZ, \quad (1.31)$$

where C is the C -matrix in the CUR in (1.30). In order to obtain (1.30) we now need to factorize Z such that

$$Z = UR \quad (1.32)$$

where $R = A(I_s, :)$ is given. Note that the combination of the equations (1.31) and (1.32) corresponds to a CUR factorization (1.30). The equation (1.32) is an overdetermined equation (more equations than unknowns) and we can only hope to satisfy it approximately. We solve this equation in a least square sense to obtain U :

$$U = \underset{U}{\operatorname{argmin}} \|Z - UR\|_2.$$

Formally, the solution can be written as $U = ZR^\dagger$, where $(\cdot)^\dagger$ denotes the pseudoinverse.

CUR is described in PGM 4.5 and the computation of CUR from ID is described in PGM 4.6.1

1.7 Other useful factorizations

Non-negative matrix factorizations. Information will be added later.