5 Orthogonalization and the Singular Value Decomposition

5.1 Orthogonalization using Gram Schmidt and QR factiorization

Assume that we have a set of vectors $\{\mathbf{a}_i\}_{i=1}^n \in \mathbb{R}^m$, $m \geq n$. We know that if the vectors $\mathbf{a}_i$ are linearly independent, then they span the whole space $\mathbb{R}^n$. Suppose that we have all these vectors as columns of a matrix $A = [\mathbf{a}_1|\mathbf{a}_2|...|\mathbf{a}_n]$. If $\mathbf{a}_i$ are linearly independent, then A is full rank. However, if some of the vectors $\mathbf{a}_i$ are almost linearly dependent, then A is almost singular, and performing operations like inverting A may involve high numerical errors.

Back to our least squares problem, we noticed that the solution is achieved when $A^\top(A\mathbf{x}-\mathbf{b}) = 0$. That is, when the residual $A\mathbf{x}-\mathbf{b}$ is orthogonal to the columns of A. In fact, this also means that for the solution $\mathbf{x}$ the residual vector $A\mathbf{x}-\mathbf{b}$ is not in the span of the subspace defined by A's columns. This means that the least squares approximation is all about the span of A's columns rather than A itself. Again, if the columns of A are close to being linearly dependent, then we may have large numerical errors (that will be introduced when we solve the linear system with the matrix $A^\top A$). We will now see how to solve a LS problem using an orthogonalization process, that will make the calculation more numerically stable.

5.1.1 Gram Schmidt orthogonalization

Assume that we are given with a set of linearly independent vectors $\{\mathbf{a}_i\}_{i=1}^n$, and we wish to build a set of orthogonal vectors $\{\mathbf{q}_i\}_{i=1}^n$, based on linear combinations of $\mathbf{a}_i$'s. The orthogonality is based on some inner product $\langle\cdot,\cdot\rangle$. One process for doing this is called the Gram Schmidt algorithm.

- Step 1: $\mathbf{q}_1 = \mathbf{a}_1$.
- Step 2: $\mathbf{q}_2 = \mathbf{a}_2 - \frac{\langle\mathbf{a}_2,\mathbf{q}_1\rangle}{\langle\mathbf{q}_1,\mathbf{q}_1\rangle}\mathbf{q}_1$.
  We indeed see that $\langle\mathbf{q}_2,\mathbf{q}_1\rangle = \langle\mathbf{a}_2,\mathbf{q}_1\rangle - \frac{\langle\mathbf{a}_2,\mathbf{q}_1\rangle}{\langle\mathbf{q}_1,\mathbf{q}_1\rangle}\langle\mathbf{q}_1,\mathbf{q}_1\rangle = 0$
- Step 3: $\mathbf{q}_3 = \mathbf{a}_3 - \frac{\langle\mathbf{a}_3,\mathbf{q}_1\rangle}{\langle\mathbf{q}_1,\mathbf{q}_1\rangle}\mathbf{q}_1 - \frac{\langle\mathbf{a}_3,\mathbf{q}_2\rangle}{\langle\mathbf{q}_2,\mathbf{q}_2\rangle}\mathbf{q}_2$.
  We indeed see that $\langle\mathbf{q}_3,\mathbf{q}_1\rangle = \langle\mathbf{a}_3,\mathbf{q}_1\rangle - \frac{\langle\mathbf{a}_3,\mathbf{q}_1\rangle}{\langle\mathbf{q}_1,\mathbf{q}_1\rangle}\langle\mathbf{q}_1,\mathbf{q}_1\rangle - \frac{\langle\mathbf{a}_3,\mathbf{q}_2\rangle}{\langle\mathbf{q}_2,\mathbf{q}_2\rangle}\langle\mathbf{q}_2,\mathbf{q}_1\rangle = 0$
  and we also see that $\langle\mathbf{q}_3,\mathbf{q}_2\rangle = \langle\mathbf{a}_3,\mathbf{q}_2\rangle - \frac{\langle\mathbf{a}_3,\mathbf{q}_1\rangle}{\langle\mathbf{q}_1,\mathbf{q}_1\rangle}\langle\mathbf{q}_1,\mathbf{q}_2\rangle - \frac{\langle\mathbf{a}_3,\mathbf{q}_2\rangle}{\langle\mathbf{q}_2,\mathbf{q}_2\rangle}\langle\mathbf{q}_2,\mathbf{q}_2\rangle = 0$

- Step i: $\mathbf{q}_i = \mathbf{a}_i - \sum_{j=1,..,i-1} \frac{\langle \mathbf{a}_i, \mathbf{q}_j \rangle}{\langle \mathbf{q}_j, \mathbf{q}_j \rangle} \mathbf{q}_j$.
  Similarly to before we will have that $\langle \mathbf{q}_i, \mathbf{q}_j \rangle = 0$ for $j = 1, ..., i-1$.

Let us now rewrite Steps 1 to 3 above in a slightly different way $\|\mathbf{q}_i\| = \sqrt{\langle \mathbf{q}_i, \mathbf{q}_i \rangle}$:

- Step 1: $\mathbf{q}_1 = \mathbf{a}_1$.
- Step 2: $\mathbf{q}_2 = \mathbf{a}_2 - \frac{\langle \mathbf{a}_2, \mathbf{q}_1 \rangle}{\|\mathbf{q}_1\|^2} \mathbf{q}_1$.
- Step 3: $\mathbf{q}_3 = \mathbf{a}_3 - \frac{\langle \mathbf{a}_3, \mathbf{q}_1 \rangle}{\|\mathbf{q}_1\|^2} \mathbf{q}_1 - \frac{\langle \mathbf{a}_3, \mathbf{q}_2 \rangle}{\|\mathbf{q}_2\|^2} \mathbf{q}_2$.

And now, with some simple arithmetics we can get:

$$\begin{aligned} \mathbf{a}_1 &= \|\mathbf{q}_1\| \frac{\mathbf{q}_1}{\|\mathbf{q}_1\|} \\ \mathbf{a}_2 &= \|\mathbf{q}_2\| \frac{\mathbf{q}_2}{\|\mathbf{q}_2\|} + \langle \mathbf{a}_2, \frac{\mathbf{q}_1}{\|\mathbf{q}_1\|} \rangle \frac{\mathbf{q}_1}{\|\mathbf{q}_1\|} \\ \mathbf{a}_3 &= \|\mathbf{q}_3\| \frac{\mathbf{q}_3}{\|\mathbf{q}_3\|} + \langle \mathbf{a}_3, \frac{\mathbf{q}_2}{\|\mathbf{q}_2\|} \rangle \frac{\mathbf{q}_2}{\|\mathbf{q}_2\|} + \langle \mathbf{a}_3, \frac{\mathbf{q}_1}{\|\mathbf{q}_1\|} \rangle \frac{\mathbf{q}_1}{\|\mathbf{q}_1\|} \end{aligned} \tag{26}$$

The triangular shape in Eq. (26) above gives rise to a matrix factorization that is based on orthogonalization with respect to the standard inner product $\langle \mathbf{u}, \mathbf{v} \rangle = \mathbf{u}^\top \mathbf{v}$.

5.1.2 The QR factorization

The process above is used to transform a set of vector arranged in a matrix $A = [\mathbf{a}_1 | \mathbf{a}_2 | ... | \mathbf{a}_n]$ into a set of ortho-normal vectors (orthogonal and normalized), arranged in a matrix $Q = [\mathbf{q}_1 | \mathbf{q}_2 | ... | \mathbf{q}_n]$. With minimal effort, we can set this process into a factorization

$$A = QR,$$

where Q is an orthogonal matrix ($Q^\top Q = I$), and R is an upper triangular matrix, defined by the calculated constants in the orthogonalization process. This factorization is called a *QR factorization* and is a very useful tool, especially in the context of least squares. Algorithm 4 is the “economic” QR factorization using Gram-Schmidt for a matrix $A \in \mathbb{R}^{m \times n}$ with $m \geq n$.

It turns out, however, that the Gram-Schmidt process above can be numerically unstable and introduce high numerical errors. This phenomenon can be fixed with a small almost unnoticeable algorithmic change. A key difference between the two algorithms is that the

```
Algorithm: Gram Schmidt QR
# $A = [\mathbf{a}_1|\mathbf{a}_2|...|\mathbf{a}_n] \in \mathbb{R}^{m\times n}$
Initialize: $R = 0^{n\times n}$, $R_{1,1} = \|\mathbf{a}_1\|_2$, $\mathbf{q}_1 = \frac{\mathbf{a}_1}{R_{1,1}}$
for $i = 2, ..., n$ do
    $\mathbf{q}_i \leftarrow \mathbf{a}_i$
    for $j = 1 : i-1$ do
        $R_{j,i} = \mathbf{q}_j^\top \mathbf{a}_i$
        $\mathbf{q}_i \leftarrow \mathbf{q}_i - R_{j,i}\mathbf{q}_j$
    end
    $R_{i,i} = \|\mathbf{q}_i\|_2$
    $\mathbf{q}_i \leftarrow \frac{\mathbf{q}_i}{R_{i,i}}$
end
```

Algorithm 4: QR factorization using Gram Schmidt.

$R_{j,i}$'s must be computed sequentially in MGS because $R_{j,i}$ depends on $\mathbf{q}_i$ which is changing, while in GS they can be computed in parallel because $R_{j,i}$ do not depend on $\mathbf{q}_i$.

```
Algorithm: Modified Gram Schmidt QR
# $A = [\mathbf{a}_1|\mathbf{a}_2|...|\mathbf{a}_n] \in \mathbb{R}^{m\times n}$
Initialize: $R = 0^{n\times n}$, $R_{1,1} = \|\mathbf{a}_1\|_2$, $\mathbf{q}_1 = \frac{\mathbf{a}_1}{R_{1,1}}$
for $i = 2, ..., n$ do
    $\mathbf{q}_i \leftarrow \mathbf{a}_i$
    for $j = 1 : i-1$ do
        # the next line in the only difference from regular GS
        $R_{j,i} = \mathbf{q}_j^\top \mathbf{q}_i$
        $\mathbf{q}_i \leftarrow \mathbf{q}_i - R_{j,i}\mathbf{q}_j$
    end
    $R_{i,i} = \|\mathbf{q}_i\|_2$
    $\mathbf{q}_i \leftarrow \frac{\mathbf{q}_i}{R_{i,i}}$
end
```

Algorithm 5: QR factorization using Modified Gram Schmidt.

Example 16 (Economic QR factorization)**.** *Assume we have the following vectors:*

$$\{\mathbf{a}_i\}_{i=1}^3 = \left\{ \begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 3 \\ -1 \\ 3 \end{bmatrix}, \begin{bmatrix} 1 \\ 3 \\ 5 \\ 7 \end{bmatrix} \right\},$$

or the equivalently, the following matrix

$$A = \begin{bmatrix} -1 & -1 & 1 \\ 1 & 3 & 3 \\ -1 & -1 & 5 \\ 1 & 3 & 7 \end{bmatrix}.$$

We will now decompose the A into a QR factorization.
Initialization:

$$\|\mathbf{a}_1\|_2 = 2 \Rightarrow \mathbf{q}_1 = (-1/2, 1/2, -1/2, 1/2)^T, \quad R_{1,1} = 2.$$

Iteration $i = 2$

$$\begin{aligned} \mathbf{z} &= \mathbf{a}_2 - (\mathbf{a}_2^\top \mathbf{q}_1)\mathbf{q}_1 = \mathbf{a}_2 - R_{1,2}\mathbf{q}_1 = \mathbf{a}_2 - 4\mathbf{q}_1 = (1,1,1,1)^T \\ R_{2,2} &= \|\mathbf{z}\| = 2 \\ \mathbf{q}_2 &= \frac{\mathbf{z}}{R_{2,2}} = (1/2, 1/2, 1/2, 1/2)^T \end{aligned}$$

Iteration $i = 3$

$$\begin{aligned} \mathbf{z} &= \mathbf{a}_3 - (\mathbf{a}_3^\top \mathbf{q}_1)\mathbf{q}_1 - (\mathbf{a}_3^\top \mathbf{q}_2)\mathbf{q}_2 = \mathbf{a}_3 - R_{1,3}\mathbf{q}_1 - R_{2,3}\mathbf{q}_2 \\ &= \mathbf{a}_3 - 2\mathbf{q}_1 - 8\mathbf{q}_2 = (-2, -2, 2, 2)^T. \\ R_{3,3} &= \|\mathbf{z}\| = 4 \\ \mathbf{q}_3 &= \frac{\mathbf{z}}{R_{3,3}} = (-1/2, -1/2, 1/2, 1/2)^T \end{aligned}$$

Finally we get:

$$Q = \frac{1}{2}\begin{bmatrix} -1 & 1 & -1 \\ 1 & 1 & -1 \\ -1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, R = \begin{bmatrix} 2 & 4 & 2 \\ 0 & 2 & 8 \\ 0 & 0 & 4 \end{bmatrix}$$

Remark 6 (Full QR factorization)**.** *The QR factorization above is the "economic" QR where for a matrix $A \in \mathbb{R}^{m\times n}$ with $m > n$ we find $Q \in \mathbb{R}^{m\times n}$ and $R \in \mathbb{R}^{n\times n}$. Alternatively, we can also find an equivalent factorization such that $Q \in \mathbb{R}^{m\times m}$ and $R \in \mathbb{R}^{m\times n}$. Some packages compute it by default. To achieve this we can complement Q with the missing subspace of size $m - n$, such that Q would be orthogonal sized $m \times m$, and its columns would span $\mathbb{R}^m$, and placing zeros on the bottom $m - n$ rows of R. This is an equivalent factorization, which has mostly theoretical value and not practical.*

5.2 Least Squares solution using QR factorization

One interpretation of orthogonal matrices is that they are multi-dimensional rotation matrices, which do not change the lengths of vectors. This is easy to show: assume that $Q \in \mathbb{R}^{n\times n}$ is an orthogonal matrix ($Q^\top Q = I$), and that $\mathbf{v} \in \mathbb{R}^n$ is a vector, then multiplying Q with $\mathbf{v}$ preserves the ℓ_2 norm of $\mathbf{v}$:

$$\|Q\mathbf{v}\|_2^2 = \mathbf{v}^\top Q^\top Q\mathbf{v} = \mathbf{v}^\top \mathbf{v} = \|\mathbf{v}\|_2^2. \tag{27}$$

This is an important property that we're about to exploit.

Back to our least squares problem

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|_2^2.$$

This time we will see another approach for solving the problem, which is also more numerically stable. We will later see, that the condition number of $A^\top A$ is the square of the condition number of A. Therefore, solving the normal equations may be ill-conditioned. Suppose that $A \in \mathbb{R}^{m\times n}$ with $m \geq n$, and suppose that we have an economic QR factorization $A = QR$, where $Q \in \mathbb{R}^{m\times n}$ and $R \in \mathbb{R}^{n\times n}$. Note that

$$A^\top A = (QR)^\top(QR) = R^\top Q^\top QR = R^\top R.$$

We know that the solution is:

$$\hat{\mathbf{x}} = (A^\top A)^{-1}A^\top\mathbf{b} = (R^\top R)^{-1}R^\top Q^\top\mathbf{b} = R^{-1}(R^\top)^{-1}R^\top Q^\top\mathbf{b} = R^{-1}Q^\top\mathbf{b},$$

where here we used the fact that R is square and invertible (full rank).

Example 17 (Least squares via QR)**.** *Suppose that we need to minimize* $\|A\mathbf{x} - \mathbf{b}\|_2$ *with*

$$A = \begin{bmatrix} -1 & -1 & 1 \\ 1 & 3 & 3 \\ -1 & -1 & 5 \\ 1 & 3 & 7 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} -1 \\ 23 \\ 15 \\ 39 \end{bmatrix}.$$

We will now solve the minimization by two equivalent approaches:

1. *Using the normal equations, we will form* $A^\top A$*, and* $A^\top \mathbf{b}$*, and solve* $A^\top A\mathbf{x} = A^\top \mathbf{b}$*:*

$$A^\top A = \begin{bmatrix} 4 & 8 & 4 \\ 8 & 20 & 24 \\ 4 & 24 & 84 \end{bmatrix}, \quad A^\top \mathbf{b} = \begin{bmatrix} 48 \\ 172 \\ 416 \end{bmatrix}, \quad \hat{\mathbf{x}} = (A^\top A)^{-1} A^\top \mathbf{b} = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}.$$

The solution of the normal equation is done using LU *or Cholesky factorizations.*

2. *Alternatively, we can use the QR factorization (found in the previous example):*

$$Q = \frac{1}{2}\begin{bmatrix} -1 & 1 & -1 \\ 1 & 1 & -1 \\ -1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, R = \begin{bmatrix} 2 & 4 & 2 \\ 0 & 2 & 8 \\ 0 & 0 & 4 \end{bmatrix}.$$

Now we solve $R\mathbf{x} = Q^\top \mathbf{b}$*:*

$$Q^\top \mathbf{b} = \begin{bmatrix} 24 \\ 38 \\ 16 \end{bmatrix}, \quad \mathbf{x} = R^{-1} Q^\top \mathbf{b} = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}.$$

5.3 Spectral Decomposition - reminder and some facts

Theorem 12 (Spectral Decomposition). *Let $A \in \mathbb{C}^{n\times n}$, and $A\mathbf{v}_i = \lambda_i \mathbf{v}_i$ for $i = 1, ..., n$. If the vectors $\{\mathbf{v}_i\}$ are linearly independent then A is diagonalizable such that*

$$\begin{aligned} A &= V\Lambda V^{-1} \\ \Lambda &= diag(\lambda_1, ..., \lambda_n) \\ V &= [\mathbf{v}_1|\mathbf{v}_2|...|\mathbf{v}_n]. \end{aligned}$$

This is just a writing in matrix form. Note that $A\mathbf{v}_i = \lambda_i \mathbf{v}_i$ for $i = 1, ..., n$ can be written as $AV = V\Lambda$ is matrix form.

Theorem 13 (Normal and Hermitian Spectral Decomposition). *Let $A \in \mathbb{C}^{n\times n}$ be normal (or Hermitian) then there exist a unitary matrix $U \in \mathbb{C}^{n\times n}$ such that*

$$\begin{aligned} U^*U &= I \\ A &= U\Lambda U^* \\ \Lambda &= diag(\lambda_1, ..., \lambda_n) \\ U &= [\mathbf{u}_1|\mathbf{u}_2|...|\mathbf{u}_n], \end{aligned}$$

where $\mathbf{u}_i \in \mathbb{C}^n$ are the eigenvectors of A, and λ_i are the eigenvalues of A. If A is normal then $\lambda_i \in \mathbb{C}$, and if Hermitian then $\lambda_i \in \mathbb{R}$ (real numbers). ***If A is real and symmetric, all the matrices and eigenvalues above are real-valued.***

```
%% MATLAB code for demonstraing eigenvalues and eigenvectors:
>> A = randn(3)
[V,D] = eig(A)
norm(A - V*D*inv(V),'fro')

B = A'*A % B is now a symmetric matrix
[U,D] = eig(B)
norm(B - U*D*U','fro')
%% Output:
A =
    0.3129   -0.1649    1.1093
   -0.8649    0.6277   -0.8637
   -0.0301    1.0933    0.0774
V =
   0.8215 + 0.0000i   0.3669 + 0.3391i   0.3669 - 0.3391i
   0.2570 + 0.0000i  -0.6522 + 0.0000i  -0.6522 + 0.0000i
  -0.5090 + 0.0000i  -0.2962 + 0.4871i  -0.2962 - 0.4871i
D =
  -0.4261 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.7220 + 1.0946i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.7220 - 1.0946i
% This means: A = V*D*inv(V)
ans =
   2.3265e-15
---------------------------------------------------------------
B =
    0.8468   -0.6273    1.0917
   -0.6273    1.6164   -0.6404
    1.0917   -0.6404    1.9824
U =
    0.8717    0.1041    0.4788
    0.1724    0.8496   -0.4985
   -0.4587    0.5171    0.7227
D =
    0.1483         0         0
         0    1.1498         0
         0         0    3.1475
% This means: B = U*D*U'
ans =
   3.5804e-15
```

5.4 SVD - Singular Value Decomposition

In this section we will deal with the singular value decomposition, which is one of the more important and useful matrix factorizations in linear algebra.

Theorem 14. *Let $A \in \mathbb{R}^{m\times n}$, then there exist orthogonal matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$, such that $V^*V = I_n$, $U^*U = I_m$ and*

$$A = U\Sigma V^*, \quad \Sigma = diag(\sigma_1, \sigma_2, ..., \sigma_p) \in \mathbb{R}^{m\times n} \quad p = \min(m, n),$$

where $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq ... \geq \sigma_p \geq 0$. This is called the full SVD factorization of A.

You may see the proof in Section 5.4.1. Another way to look at SVD, is that the singular triplets satisfy:

$$A\mathbf{v}_i = \sigma_i \mathbf{u}_i, \quad \text{for} \quad i = 1, ..., \min(m, n),$$

which is somewhat similar to the definition of eigenvalue and eigenvector.

5.4.1 Proof for the SVD existence Theorem

Theorem 15. *Let $A \in \mathbb{R}^{m\times n}$, then there exist orthogonal matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$, such that $V^TV = I_n$, $U^TU = I_m$ and*

$$A = U\Sigma V^\top, \quad \Sigma = diag(\sigma_1, \sigma_2, ..., \sigma_p) \in \mathbb{R}^{m\times n} \quad p = \min(m, n),$$

where $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq ... \geq \sigma_p \geq 0$. This is called the full SVD factorization of A.

Proof. Let $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^m$ be two unit norm vectors, such that $A\mathbf{x} = \sigma\mathbf{y}$ with $\sigma = \|A\|_2$. Let $V_1 \in \mathbb{R}^{n\times(n-1)}$ and $U_1 \in \mathbb{R}^{m\times(m-1)}$ be any orthogonal matrices that complement $\mathbf{x}$ and $\mathbf{y}$ to a full space, such that $V = [\mathbf{x}|V_1] \in \mathbb{R}^{n\times n}$ and $U = [\mathbf{y}|U_1] \in \mathbb{R}^{m\times m}$ are also orthogonal matrices. The matrix U^TAV has the following structure:

$$U^\top AV = U^\top[\sigma\mathbf{y}|AV_1] = \begin{bmatrix} \sigma & \mathbf{w}^\top \\ 0 & B \end{bmatrix} = A_1,$$

where $\mathbf{w} \in \mathbb{R}^{n-1}$ is an unknown vector. We will now show that $\mathbf{w} = 0$, and this will allow to complete the proof by induction argument. We have that

$$\left\| A_1 \begin{bmatrix} \sigma \\ \mathbf{w} \end{bmatrix} \right\|_2^2 \geq (\sigma^2 + \mathbf{w}^\top \mathbf{w})^2,$$

so $\|A_1\|_2^2 \geq \sigma^2 + \mathbf{w}^\top \mathbf{w}$. But $\|A_1\|_2 = \|A\|_2 = \sigma$, and so we must have that $\mathbf{w} = 0$. It is also easy to show that $\sigma \geq \|B\|_2$, otherwise this would contradict that $\|A_1\|_2 = \sigma$.

The rest of the proof can be completed by induction, assuming that

$$U^\top A V = \begin{bmatrix} \Sigma & 0 \\ 0 & B \end{bmatrix}, \tag{28}$$

for $\Sigma_k = \text{diag}(\sigma_1, \sigma_2, ..., \sigma_k) \in \mathbb{R}^{k \times k}$, and $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq ... \geq \sigma_k \geq 0$, are the k largest singular values and $\sigma_k \geq \|B\|_2$. Applying the steps above to the matrix $B \in \mathbb{R}^{(m-k)\times(n-k)}$, we can find orthogonal matrices $U_2 \in \mathbb{R}^{(m-k)\times(m-k)}$ and $V_2 \in \mathbb{R}^{(n-k)\times(n-k)}$ such that

$$U_2^\top B V_2 = \begin{bmatrix} \sigma_{k+1} & 0 \\ 0 & C \end{bmatrix},$$

for some matrix C and $\sigma_{k+1} = \|B\|_2$. Now U and V in Eq. (28) can be multiplied by the two orthogonal matrices blockdiag$(I_{k\times k}, U_2)$, and blockdiag$(I_{k\times k}, V_2)$ respectively, resulting in new orthogonal matrices U, V that lead to $U^\top A V = \begin{bmatrix} \Sigma_{k+1} & 0 \\ 0 & C \end{bmatrix}$ □

Remark 7 (Economic (practical) SVD). *Similarly to the "economic" and full versions of QR, the SVD factorization also has "economic" and full factorizations, where the economic factorization is more useful in practice, and the full one is mostly used in theoretical results. For start, note that in the full SVD factorization, the matrix Σ may actually be rectangular with blocks of zeros. For example, if $m > n$ then the bottom $m - n$ rows in Σ are zeros. Given a matrix $A \in \mathbb{R}^{m\times n}$, let $p = min(m, n)$. In the economic SVD we find orthogonal matrices $U \in \mathbb{R}^{m\times p}$, and $V \in \mathbb{R}^{p\times n}$, and a diagonal matrix $\Sigma \in \mathbb{R}^{p\times p} = diag(\sigma_1, \sigma_2, ..., \sigma_p)$. This is the economic SVD decomposition, and in this case Σ is square and diagonal.*

Remark 8 (Complex SVD). *In the case of $A \in \mathbb{C}^{m\times n}$, there is a complex SVD decomposition, where the matrices U and V are unitary (complex and orthogonal), but Σ remains real with a positive diagonal.*

Remark 9 (Computing SVD). *Computing the SVD decomposition is considered to be a rather expensive and complicated task and we will not address it in this course. Generally, there are different algorithms for computing the SVD, and in all of them, maintaining the accuracy of the computation is necessary. In general, it is not numerically "safe" to compute $A^\top A$, since we square the condition number. In essence, the SVD of a full matrix A starts with a Bi-orthogonalization process (See Appendix). Then, $A^\top A$ is essentially transformed to be tri-diagonal and we can compute its determinant efficiently to find its eigenvalues.*

5.4.2 The relation between SVD and the Spectral Decomposition

As we saw earlier, if a matrix A is symmetric, then it has the spectral decomposition

$$A = U\Lambda U^*, \tag{29}$$

where $\Lambda = \text{diag}(\lambda_1, ..., \lambda_n)$ is the diagonal matrix of eigenvalues, and $U = [\mathbf{u}_1|\mathbf{u}_2|...|\mathbf{u}_n]$ is the matrix of orthogonal eigenvectors $\mathbf{u}_i \in \mathbb{C}^n$ $(U^*U = I)$. We have noted that if A is symmetric (Hermitian), then λ_i are real, and if A is also semi-positive definite then λ_i are non-negative.

We will now see that the singular values are the square root of the eigenvalues of the matrix A^*A. Using the SVD of A we have

$$\begin{aligned} A^*A &= (U\Sigma V^*)^* U\Sigma V^* && (30)\\ &= V\Sigma^* U^* U\Sigma V^* \quad (\text{where } U^*U = I) && (31)\\ &= V\Sigma^*\Sigma V^* \quad (\text{where } \Sigma^*\Sigma = \Sigma^2 \text{ since it's a diagonal matrix}) && (32)\\ &= V\Sigma^2 V^* && (33)\end{aligned}$$

From here we can see that similarly to the spectral decomposition in (29), V is the eigenvector matrix of A^*A, and the diagonal Σ^2 corresponds to the eigenvalues of A^*A. We can see that the diagonal entries of Σ^2 are non-negative, which agrees with A^*A being symmetric semi positive definite by definition. Also, σ_i are the square root of the eigenvalues of A^*A. In principle, the SVD can be computed by finding the eigenvalues and eigenvectors of A^*A, however, the process is not stable and expensive, and in practice the SVD is computed using more sophisticated approaches.

5.5 Least squares via SVD

Suppose again that we are solving our least squares problem ($A \in \mathbb{R}^{m\times n}, m \geq n$)

$$\hat{\mathbf{x}} = \underset{\mathbf{x}\in\mathbb{R}^n}{\arg\min} \|A\mathbf{x} - \mathbf{b}\|_2^2.$$

and this time assume that we have the (economic) SVD decomposition of A: $A = U\Sigma V^\top$ ($U \in \mathbb{R}^{m\times n}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal matrices). Similarly to the QR case,

$$\begin{aligned} A^\top A &= (U\Sigma V^\top)^\top (U\Sigma V^\top) = V\Sigma^\top U^\top U\Sigma V^\top = V\Sigma^\top \Sigma V^\top. \\ A^\top \mathbf{b} &= (U\Sigma V^\top)^\top \mathbf{b} = V\Sigma^\top U^\top \mathbf{b}. \end{aligned}$$

We get an equation:

$$V\Sigma^\top \Sigma V^\top \hat{\mathbf{x}} = V\Sigma^\top U^\top \mathbf{b}.$$

Since V and Σ are square and invertible, we can "divide from the left"

$$\Sigma V^\top \hat{\mathbf{x}} = U^\top \mathbf{b},$$

and applying a change of variables $\hat{\mathbf{y}} = V^\top \hat{\mathbf{x}}$ we get a diagonal system

$$\Sigma \hat{\mathbf{y}} = U^\top \mathbf{b}.$$

After solving this equation, our solution is $\hat{\mathbf{x}} = (V^\top)^{-1}\hat{\mathbf{y}} = V\hat{\mathbf{y}}$.

Example 18 (Least squares via the SVD)**.** *Suppose that we need to minimize* $\|A\mathbf{x} - \mathbf{b}\|_2$ *with*

$$A = \begin{bmatrix} -1 & -1 & 1 \\ 1 & 3 & 3 \\ -1 & -1 & 5 \\ 1 & 3 & 7 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} -1 \\ 23 \\ 15 \\ 39 \end{bmatrix}.$$

We will now solve this problem using a third approach (in addition to the two in the previous example), via the economic SVD decomposition which is the default in Julia's `svd()`*.*

```
A = [-1.0 -1.0 1.0 ; 1.0 3.0 3.0 ; -1 -1 5.0 ; 1 3 7];
b = [ -1.0 ; 23 ; 15 ; 39];
(U,S,V) = svd(A);
U
4x3 Array{Float64,2}:
 0.0574049  -0.410658   0.760306
 0.402203    0.509243   0.573502
 0.450295   -0.732826  -0.100999
 0.795093    0.187076  -0.287803
S # Julia returns only a vector of singular values, not a diagonal matrix
3-element Array{Float64,1}:
 9.61534
 3.91982
 0.424511
V
3x3 Array{Float64,2}:
 0.0717183   0.469358  -0.88009
 0.320757    0.824639   0.465924
 0.944442   -0.31571   -0.0914083
Utb = U'*b;
Utb
3-element Array{Float64,1}:
 46.9563
  8.42681
 -0.309061
y = Utb./S; # here we invert Sigma
y
3-element Array{Float64,1}:
  4.88348
  2.14979
 -0.728041
x = V*y;
x
3-element Array{Float64,1}:
 2.0
 3.0
 4.0
```

5.6 Minimum norm solution to rank-deficient LS and the pseudo inverse

So far we've looked at a full-rank version of the least-squares problem, where the matrix $A \in \mathbb{R}^{m\times n}$, with $m \geq n$, is full rank (rank n), which also means that $A^\top A$ is full rank and invertible. This means that there is a unique solution to the normal equation. Also, the matrix R in the QR factorization of A is invertible.

Now we consider the LS problem

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|_2^2.$$

where $\text{rank}(A) < n$, and $A^\top A$ is singular, and the normal equation

$$A^\top A\mathbf{x} = A^\top \mathbf{b},$$

has infinitely many solutions. One way to see that is to place the SVD decomposition $A = U\Sigma V^\top$ ($U \in \mathbb{R}^{m\times n}$, $\Sigma \in \mathbb{R}^{n\times n}$,$V \in \mathbb{R}^{n\times n}$) in the normal equation (similarly as before):

$$V\Sigma^2 V^\top \mathbf{x} = V\Sigma U^\top \mathbf{b}.$$

After multiplying by $V^\top$ from the left, this equation can also be written as:

$$\Sigma^2\mathbf{y} = \Sigma\hat{\mathbf{b}},$$

for $\hat{\mathbf{b}} = U^\top\mathbf{b}$, and $\mathbf{y} = V^\top\mathbf{x}$. If there is a zero diagonal entry in Σ, there will also be a zero in the right-hand-side, indicating that this equation has infinitely many solutions.

Out of all these solutions, in many cases we wish to find the one with minimum ℓ_2 norm. Since Σ is diagonal, $y_i = \frac{\hat{b}_i}{\sigma_i}$ for the entries where $\sigma_i \neq 0$ and y_i can get any value in the entries where $\sigma_i = 0$. Obviously, the solution $\mathbf{y}$ with the lowest ℓ_2 norm is the one with zeros where $\sigma_i = 0$, and so will be the solution $\mathbf{x}$ since $\|\mathbf{x}\|_2 = \|V\mathbf{y}\|_2 = \|\mathbf{y}\|_2$.

To sum up, if A is full rank, then the solution to the least squares problem is

$$\hat{\mathbf{x}} = (A^\top A)^{-1}A^\top\mathbf{b} = V(\Sigma)^{-1}U^\top = A^\dagger\mathbf{b}.$$

The matrix $A^\dagger = (A^\top A)^{-1}A^\top = V(\Sigma)^{-1}U^\top$ is also called the Moore–Penrose pseudoinverse of A. In the case where A is not full rank, we will define $A^\dagger$ to be $V(\Sigma)^\dagger U^\top$ where

$$(\Sigma^\dagger)_{ii} = \begin{cases} (\Sigma_{ii})^{-1} & \Sigma_{ii} \neq 0 \\ 0 & \Sigma_{ii} = 0 \end{cases}$$

Remark 10. *We can get the minimum norm LS solution using the pseudo inverse by taking a different approach. Consider the regularized problem* $\hat{x} = \arg\min \|A\mathbf{x} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{x}\|_2^2$ *for* $\lambda > 0$. *In this problem, the normal equation is always solvable, and because we penalize the norm of the solution, we will always get a minimum norm solution to the LS problem, if the matrix A is rank deficient. The pseudo inverse will be achieved for* $\lambda \longrightarrow 0$.

5.7 Best low-rank approximation of matrices

Another important role of the SVD factorization is answering the following question: Given a matrix $A \in \mathbb{R}^{m\times n}$, what is the matrix $X \in \mathbb{R}^{m\times n}$ of rank $p < \min(n, m)$ that best approximates A in Frobenius norm?

For starts, $X \in \mathbb{R}^{m\times n}$ will be a low-rank matrix, and we will be able to represent it as a sum of p outer products. To understand this, consider the following example, where the following outer product leads to a rank-one matrix:

$$\mathbf{w}\mathbf{q}^\top = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} [2 \quad 3 \quad 4] = \begin{bmatrix} 2 & 3 & 4 \\ 2 & 3 & 4 \\ 2 & 3 & 4 \end{bmatrix}.$$

Generally, if we have the sets $\{\mathbf{v}_i\}_{i=1}^p \in \mathbb{R}^n$, and $\{\mathbf{u}_i\}_{i=1}^p \in \mathbb{R}^m$, and $p \leq \min(m, n)$, then the sum

$$\sum_{i=1}^{p} \mathbf{u}_i \mathbf{v}_i^\top \in \mathbb{R}^{m\times n}$$

is a matrix of rank p at most. So, our approximation X will have to be defined as a sum of outer products. It happens that such a sum is exactly what we get if we look at the SVD, because by the definition of matrix multiplication (we assume here that A is full rank):

$$A = U\Sigma V^\top = \sum_{i=1}^{\min(m,n)} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top,$$

so if Σ will have only p non-zero values, this sum will be of p outer-products.

Next, we will recall that just like for the ℓ_2 norm, we have that for every matrix X and orthogonal matrix V

$$\|VX\|_F^2 = \text{trace}(X^\top V^\top V X) = \text{trace}(X^\top X) = \|X\|_F^2,$$

and this will hold for every matrix V satisfying $V^\top V = I$.

Back to our problem of approximating A. Let's that $A = U\Sigma_A V^\top$ is the full SVD decomposition where U and V are square. Then, for example, $\|UX\|_F = \|U^T X\|_F = \|X\|_F$. Following this:

$$\|A - X\|_F = \|U\Sigma_A V^\top - X\|_F = \|\Sigma_A - U^\top X V\|_F.$$

So, it is clear that $U^\top X V$ should be diagonal with entries as close as possible to Σ_A. This will be achieved is we set $X = U\Sigma_X V^\top$, and then we will get:

$$\|A - X\|_F = \|\Sigma_A - \Sigma_X\|_F.$$

Both Σ_A, Σ_X are diagonal. Which p diagonal entries will we choose to be non-zeros to best approximate A? The largest ones of course!

Example 19 (Low rank approximations via the SVD)**.** *First, we will generate a random matrix A (that fits our purposes) and form its SVD decomposition.*

```
julia>  B = rand(5,3)*diagm([1;0.1;0.001])*rand(3,3)
5x3 Array{Float64,2}:
 0.610097   0.371672   0.437624
 0.270363   0.163615   0.192502
 0.710042   0.425165   0.500611
 0.472682   0.31145    0.366526
 0.0786736  0.0559478  0.0653355

julia> (U,s,V) = svd(B);

julia> U
5x3 Array{Float64,2}:
 -0.559522    0.135717  -0.10364
 -0.247125    0.106349   0.28395
 -0.645929    0.451951   0.0593005
 -0.450122   -0.825446  -0.269576
 -0.0776604  -0.290945   0.912383

julia> s
3-element Array{Float64,1}:
 1.4973
 0.0281634
 0.000312524

julia> V
3x3 Array{Float64,2}:
 -0.725098   0.68863    0.00468583
 -0.445839  -0.474613   0.758927
 -0.524844  -0.548207  -0.651159

julia> vecnorm(B -  U*diagm(s)*V') # vecnorm is one way to get the Frobenius norm in Julia.
9.7223783957695e-16
```

Now we take a few largest elements of Σ and form a low-rank approximation of A by multiplying it with U and V. As we take more singular values, the approximation improves (in Frobenius norm).

```
# recall B:
julia>  B
5x3 Array{Float64,2}:
 0.610097   0.371672   0.437624
 0.270363   0.163615   0.192502
 0.710042   0.425165   0.500611
 0.472682   0.31145    0.366526
 0.0786736  0.0559478  0.0653355

# here we take only the largest element of Sigma (form a rank-1 approximation).
julia> s1 = copy(s); s1[2:end] = 0.0;

julia> U*diagm(s1)*V'
5x3 Array{Float64,2}:
 0.607465   0.37351    0.439698
 0.2683     0.164969   0.194202
 0.701276   0.431192   0.507601
 0.488691   0.30048    0.353727
 0.0843149  0.0518424  0.0610292

# here we take the two largest elements of Sigma (form a rank-2 approximation).

julia> s2 = copy(s); s2[end] = 0.0;

julia> U*diagm(s2)*V'
5x3 Array{Float64,2}:
 0.610097   0.371696   0.437603
 0.270363   0.163547   0.19256
 0.710041   0.42515    0.500623
 0.472683   0.311514   0.366471
 0.0786723  0.0557314  0.0655212

# here we compare the norms of the two approximations:
# the rank-one approximation
julia> vecnorm(B -  U*diagm(s1)*V')/vecnorm(A)
0.0188073

# the rank-two approximation
julia> vecnorm(B -  U*diagm(s2)*V')/vecnorm(A)
0.0002086888134648313
```

Example 20 (Low rank approximations of images)**.** *The following matlab code demonstrates low-rank approximation of images*

```
%%% A MATLAB code for demontrating low rank approximation of images
Im = imread('cameraman.tif');
%Im = imread('corn.tif',3);
figure(); imshow(Im); pause;
Im = double(Im)/255;
[U,S,V] = svd(Im,'econ');
pmax = min(size(Im));
figure()
S_I = diag(S);
plot(S_I,'LineWidth',2); pause;
S_X = zeros(pmax,1);
for k = [1,5,10,15,20,30,40,50,60,70,80,90,100]
    S_X(1:k) = S_I(1:k);
    X = U*diag(S_X)*V';
    imshow(X); title([num2str(k),'-rank approximation']); pause(2);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% END OF MATLAB CODE %%%%%%%%%%%%%%%%%%%%

### A Julia code for demontrating low rank approximation of images
using FileIO
using Colors
using PyPlot;
Im = load("cameraman.png")
#Im = load("Corn.png")
Im = Gray.(Im);
Im = convert(Array{Float64},Im)
figure(); imshow(Im,cmap="gray"); title("Original"); pause(5.0);
(U,S,V) = svd(Im);
pmax = minimum(size(Im));
S_X = zeros(pmax);
figure()
for k = [1,5,10,15,20,30,40,50,60,70,80,90,100]
    S_X[1:k] = S[1:k];
    X = U*diagm(S_X)*V';
    imshow(X,cmap="gray");
    title(string(k,"-rank approximation"));
    pause(2);
end
figure(); plot(S); title("Singular Values");
```

You may see another example on face recognition using PCA in:
`https://www.youtube.com/watch?v=_lY74pXWlS8`.