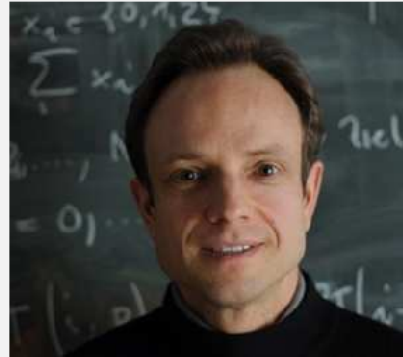


Course Video
Section 3.4.2: SVD in EIGEN

Prof. R. Hiptmair, SAM, ETH Zurich

Date: September 29, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 3.4.1, Section 1.2.1

Duration: 5 minutes

III. Direct Methods for Linear Least Squares Problems

3.4. Singular Value Decomposition (SVD)

3.4.2. SVD in Eigen

Theorem 3.4.1.1. Singular value decomposition $\rightarrow$,

For any $\mathbf{A} \in \mathbb{K}^{m,n}$ there are unitary/orthogonal matrices $\mathbf{U} \in \mathbb{K}^{m,m}$, $\mathbf{V} \in \mathbb{K}^{n,n}$ and a (generalized) diagonal (*) matrix $\mathbf{\Sigma} = \text{diag}(\sigma_1, \dots, \sigma_p) \in \mathbb{R}^{m,n}$, $p := \min\{m, n\}$, $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_p \geq 0$ such that

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H.$$

C++-code 3.4.2.1: Computing SVDs in EIGEN

```

2 #include <Eigen/SVD>
3
4 // Computation of (full) SVD  $A = U\mathbf{\Sigma}V^H \rightarrow$  Thm. 3.4.1.1
5 // SVD factors are returned as dense matrices in natural order
6 std::tuple<MatrixXd, MatrixXd, MatrixXd> svd_full(const MatrixXd& A) {
7     Eigen::JacobiSVD<MatrixXd> svd(A, Eigen::ComputeFullU | Eigen::ComputeFullV);
8     MatrixXd U = svd.matrixU(); // get unitary (square) matrix  $U$ 
9     MatrixXd V = svd.matrixV(); // get unitary (square) matrix  $V$ 
10    VectorXd sv = svd.singularValues(); // get singular values as vector
11    MatrixXd Sigma = MatrixXd::Zero(A.rows(), A.cols());
12    const unsigned p = sv.size(); // no. of singular values
13    Sigma.block(0,0,p,p) = sv.asDiagonal(); // set diagonal block of  $\Sigma$ 
14    return std::tuple<MatrixXd, MatrixXd, MatrixXd>(U, Sigma, V);
15 }
16
17 // Computation of economical (thin) SVD  $A = U\mathbf{\Sigma}V^H$ , see (3.4.1.5)
18 // SVD factors are returned as dense matrices in natural order
19 std::tuple<MatrixXd, MatrixXd, MatrixXd> svd_econ(const MatrixXd& A) {
20     Eigen::JacobiSVD<MatrixXd> svd(A, Eigen::ComputeThinU | Eigen::ComputeThinV);
21     MatrixXd U = svd.matrixU(); // get matrix  $U$  with orthonormal columns
22     MatrixXd V = svd.matrixV(); // get matrix  $V$  with orthonormal columns
23     VectorXd sv = svd.singularValues(); // get singular values as vector
24     MatrixXd Sigma = sv.asDiagonal(); // build diagonal matrix  $\Sigma$ 
25     return std::tuple<MatrixXd, MatrixXd, MatrixXd>(U, Sigma, V);
26 }

```

sorted in descending order

Cost (eco-SVD of $A \in \mathbb{K}^{m,n}$) = $O(\min\{m,n\}^2 \cdot \max\{m,n\})$
for $m, n \rightarrow \infty$

$m = n$: cost(SVD) = $O(n^3)$

Ex. 3.4.2.3 : SVD-based computation of rank

Lemma 3.4.1.13. SVD and rank of a matrix $\rightarrow$

If, for some $1 \leq r \leq p := \min\{m, n\}$, the singular values of $A \in \mathbb{K}^{m,n}$ satisfy $\sigma_1 \geq \dots \geq \sigma_r > \sigma_{r+1} = \dots = \sigma_p = 0$, then

- $\text{rank}(A) = r$ (no. of non-zero singular values),
- $\mathcal{N}(A) = \text{Span}\{(V)_{:,r+1}, \dots, (V)_{:,n}\}$,
- $\mathcal{R}(A) = \text{Span}\{(U)_{:,1}, \dots, (U)_{:,r}\}$.

! Sing. vals. σ_i are affected by roundoff

$\Rightarrow$ no testing $\sigma_i == 0.0$

Instead: test for relative smallness

C++-code 3.4.2.5: Computing rank of a matrix through SVD

```

2 // Computation of the numerical rank of a non-zero matrix by means of
3 // singular value decomposition, cf. (3.4.2.4).
4 MatrixXd::Index rank_by_svd(const MatrixXd &A, double tol = EPS) {
5     if (A.norm() == 0) return MatrixXd::Index(0);
6     Eigen::JacobiSVD<MatrixXd> svd(A);
7     const VectorXd sv = svd.singularValues(); // Get sorted singular values as
8     // vector
9     MatrixXd::Index n = sv.size();
10    MatrixXd::Index r = 0;
11    // Test relative size of singular values
12    while ((r < n) && (sv(r) >= sv(0)*tol)) r++;
13    return r;

```

"numerical rank"

largest sing. val.

C++-code 3.4.2.6: Using rank () in EIGEN

```

2 // Computation of the numerical rank of a matrix by means of SVD
3 MatrixXd::Index rank_eigen(const MatrixXd &A, double tol = EPS) {
4     return A.jacobiSvd().setThreshold(tol).rank();
5 }

```

Ex 3.4.2.7 : Computation of $\mathcal{N}(A)/\mathcal{R}(A)$
 $\hookrightarrow$ provide ON basis

Lemma 3.4.1.13. SVD and rank of a matrix $\rightarrow$

If, for some $1 \leq r \leq p := \min\{m, n\}$, the singular values of $A \in \mathbb{K}^{m,n}$ satisfy $\sigma_1 \geq \dots \geq \sigma_r > \sigma_{r+1} = \dots = \sigma_p = 0$, then

- $\text{rank}(A) = r$ (no. of non-zero singular values),
 - $\mathcal{N}(A) = \text{Span}\{(\mathbf{V})_{:,r+1}, \dots, (\mathbf{V})_{:,n}\}$
 - $\mathcal{R}(A) = \text{Span}\{(\mathbf{U})_{:,1}, \dots, (\mathbf{U})_{:,r}\}$
- ONBs

use numerical rank

C++-code 3.4.2.8: ONB of $\mathcal{N}(A)$ through SVD

```

2 // Computation of an ONB of the kernel of a matrix
3 MatrixXd nullspace(const MatrixXd &A, double tol = EPS) {
4     using index_t = MatrixXd::Index;
5     Eigen::JacobiSVD<MatrixXd> svd(A, Eigen::ComputeFullV);
6     index_t r = svd.setThreshold(tol).rank();
7     // Rightmost columns of V provide ONB of N(A)
8     MatrixXd Z = svd.matrixV().rightCols(A.cols() - r);
9     return Z;
10 }

```

$= n$

C++-code 3.4.2.9: ONB of $\mathcal{R}(A)$ through SVD

```

2 // Computation of an ONB of the image space of a matrix
3 MatrixXd rangespace(const MatrixXd &A, double tol = EPS) {
4     using index_t = MatrixXd::Index;
5     Eigen::JacobiSVD<MatrixXd> svd(A, Eigen::ComputeFullV);
6     index_t r = svd.setThreshold(tol).rank();
7     // r left columns of U provide ONB of R(A)
8     return svd.matrixU().leftCols(r);
9 }

```



Correct option ?

④

Review questions :

Examine the review questions 3.4.2.10

5

7