universität ulm Universität Ulm

Timo Tonn
Tobias Born
Summer term 2023
Points: 40
Submit until: 05.05.2023

Exercise Sheet 2 to NMDS

1. [**Givens rotations**, 3+3+3+3+5+3 points]

At first, you will get to know a second tool to zero entries of matrices, namely the Givens rotations.

Theorem 1. *Givens rotations $G(i,k,\theta) \in \mathbb{R}^{n\times n}$ are orthogonal matrices that differ from unity matrices only in four entries:*

$$G(i,i) = \cos(\theta), \quad G(k,k) = \cos(\theta), \quad G(i,k) = \sin(\theta), \quad G(k,i) = -\sin(\theta)$$

For $x = \big(x_1, \ldots, x_n\big)^T \in \mathbb{R}^n$, one sets

$$\cos(\theta) = x_i/\sqrt{x_i^2 + x_k^2} \qquad \text{and} \qquad \sin(\theta) = -x_k/\sqrt{x_i^2 + x_k^2}.$$

Then the k-th entry of $y = G(i,k,\theta)^T x$ is zero.

a) In the setting of theorem 1 what are the entries of y?

b) If one wants to zero an entry of a matrix, one can multiply the matrix by the left with a suitable Givens rotation $G(i,k,\theta)^T$. What does the rotation look like?

c) Let $A \in \mathbb{R}^{m\times n}$ be a matrix and $\tilde{A} = G(i,k,\theta)^T A$ the result of the multiplication addressed in task b). How do A and $\tilde{A}$ differ? How can one use this to compute $\tilde{A}$ efficiently? How many arithmetic operations are needed to compute $\tilde{A}$ efficiently?

d) Is it possible to use Givens rotations in a way such that a desired entry of a matrix is zeroed but two columns instead of rows are changed? If it is, how?

e) Write two MATLAB programs:

- *[cos,sin] = determine_givens(x,i,k)* which calculates the values *cos* and *sin* that build up the Givens rotation corresponding to the vector x
- *[B] = apply_givens(A,i,k,cos,sin,dir)* which multiplies a Givens rotation, specified by i, k, *cos* and *sin*, to a matrix A (*dir* specifies if from the left ('l') or right ('r')). The multiplication is supposed to be realized efficiently.

f) How could one make the computation even more efficient in case of a matrix with band structure? You do not have to implement this.

2. [**Golub-Kahan SVD Step**, 10+2+3 points]

On the previous exercise sheet we dealt with bidiagonalizing an arbitrary real matrix. Now, you will learn (a step of) how to compute the SVD of a bidiagonal matrix. Therefore recall the QR-algorithm for the eigenvalue decomposition (Algorithm 1). We will only consider the case of symmetric matrices whereby all eigenvalues are real. In case of convergence, A_i will converge towards a diagonal matrix and the orthogonal matrices Q_i will give rise to the matrix of eigenvectors.

Algorithm 1 QR-algorithm for eigenvalue decomposition of $A \in \mathbb{R}^{n\times n}$

Input: $A \in \mathbb{R}^{n\times n}$
$A_1 = A$
for $i = 1, 2, \ldots$ **do**
  determine shift μ_i
  QR-decomposition $A_i - \mu_i I = Q_i R_i$
  $A_{i+1} = R_i Q_i + \mu_i I$
end for

The shifts $\mu_i I$ do not change the result of the diagonalization, since

$$A_{i+1} = R_i Q_i + \mu_i I = Q_i^T (A_i - \mu_i I) Q_i + \mu_i I = Q_i^T A_i Q_i - \mu_i Q_i^T Q_i + \mu_i I = Q_i^T A_i Q_i \ . \quad (1)$$

But they speed up the convergence because the algorithm converges with the ratio between the smallest and the biggest eigenvalue (in absolute values) and one chooses μ_i such that it approximates the smallest eigenvalue. At the end, one gets analogously to (1) that

$$A_{i+1} = Q_i^T A_i Q_i = \ldots = Q_i^T Q_{i-1}^T \cdots Q_1^T A Q_1 \cdots Q_{i-1} Q_i =: Q^T A Q$$

and consequently the eigenvalue decomposition, since A_{i+1} converges towards a diagonal matrix and Q is orthogonal.

Now, what do we get out of this? Let B be a bidiagonal matrix and $T = B^T B$. It is clear that T is symmetric and tridiagonal and that the eigenvalues of T are the squares of the singular values of B. Since T is tridiagonal, it would be possible to compute its QR-decomposition in a cheap way. Unfortunately, it is not wise to explicitly calculate $B^T B$ for reasons of condition.

So one wants to compute one iteration of Algorithm 1 on T in an implicit way, only working on the bidiagonal matrix B. This can be done using Algorithm 3.3.2 (Golub-Kahan SVD Step) in the lecture notes. It is not obvious that it equals one iteration of Algorithm 1, but the *implicit Q-theorem* states this equality up to signs (Theorem 3.3.5 in the lecture notes; for better understanding: a Hessenberg matrix is an upper triangular matrix whose lower bidiagonal is also occupied, so a tridiagonal matrix is a special case of a Hessenberg matrix).

The *implicit Q-theorem* requires that the bidiagonal matrix is irreducible all the time, what means that there is no zero on the diagonal or bidiagonal. We assume that this is given on this sheet and will have a closer look on this on the next exercise sheet.

The Golub-Kahan SVD step on a bidiagonal square matrix $B \in \mathbb{R}^{n\times n}$ with $T = B^T B$ consists of the following steps:

- Determine the shift parameter μ as the eigenvalue of $T(n-1:n, n-1:n)$ that is closest to $T(n,n)$.
- Form the Givens rotation which would zero the entry $(2,1)$ of $T - \mu I$ (we call it G_1). Multiply it on B from the right. You will now see, that one more entry of B is occupied, namely the entry $(2,1)$.
- This extra entry can be chased downwards the bidiagonal by alternate Givens rotations from the left (U_i^T) and the right (V_i).

Then finally, one gets

$$\overline{B} = (U_1 \cdots U_{n-1})^T B (G_1 V_2 \cdots V_{n-1}) =: U^T B V$$

and $\overline{B}^T \overline{B} = V^T B^T B V$ is equal to one step of Algorithm 1 up to signs.
The whole procedure is explained in more detail in the lecture notes on pages 27-28 (list with three steps). The algorithm can be found on page 29 (Algorithm 3.3.2 Golub-Kahan SVD Step).

a) Implement the Golub-Kahan SVD Step in a method *[B] = golub_kahan_svd_step(B)*. Assume that the bidiagonal matrix is square. You can use your programs for Givens rotations from exercise 1 and you only have to compute $\overline{B}$, not U and V. Some hints:

- To calculate eigenvalues, you can use the command *eig(·)*.
- To get a random bidiagonal matrix, use the program *create_bidiagonalmatrix.m*.
- With the command *spy(abs(B) > 0.00000001)* you can track if the chasing downwards works.

b) Use the program *test_golub_kahan_svd_step.m* to test your implementation. The expression *err = norm( );* has to be completed by you.

c) Run the program *exercise_2c.m* and compare the error and condition values. Why does this motivate to work on B instead of B^TB?

Remark 1. *You can store the bidiagonal matrix in matrix format. To be efficient, one normally would store the diagonal and the bidiagonal in two vectors.*
To finally not only get the singular values but also the singular vectors, one has to accumulate all orthogonal matrices applied to the decomposed matrix. These are the Householder reflections from the bidiagonalization and the Givens rotations from the decomposition of the bidiagonal matrix. We will omit this for simplicity. We will put the whole SVD algorithm together on the next exercise sheet. Unfortunately, we cannot do this yet due to the assumption on the irreducibility of the bidiagonal matrix. So take this as a cliffhanger and be excited for the next sheet!

3. [**Rank-k-approximation $\mathbf{A_k}$**, 5 points]

Last but not least, we take a look at the rank-k-matrix

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T = U_k \Sigma_k V_k^T = \begin{bmatrix} u_1, \dots, u_k \end{bmatrix} \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{pmatrix} \begin{bmatrix} v_1, \dots, v_k \end{bmatrix}^T \ .$$

We already worked with this matrix on the previous sheet and called it *best approximation* there. Now we know what is meant with this, namely

$$\begin{aligned} \|A - A_k\|_F &= \inf_{\operatorname{rank}(B) \le k} \|A - B\|_F \qquad \text{and} \\ \|A - A_k\|_2 &= \inf_{\operatorname{rank}(B) \le k} \|A - B\|_2 \ . \end{aligned}$$

Prove the following lemma which is lemma 3.2.7 in the lecture notes.

Lemma 1. *Let $A = U\Sigma V^T$ be the SVD of a matrix $A \in \mathbb{R}^{m \times n}$, $A_k = U_k \Sigma_k V_k^T$ and define the subspace $\mathcal{U}_k = span(u_1, \dots, u_k) \subset \mathbb{R}^m$. Then the columns $\begin{bmatrix} A_k \end{bmatrix}_j$, $j = 1, \dots, n$ of A_k are the orthogonal projections of the columns a_j of the matrix $A = \begin{pmatrix} a_1, \dots, a_n \end{pmatrix}$ onto the subspace $\mathcal{U}_k$, i.e.*

$$\begin{bmatrix} A_k \end{bmatrix}_j = P_{\mathcal{U}_k} a_j, \quad \textit{for} \quad j = 1, \dots, n \ .$$