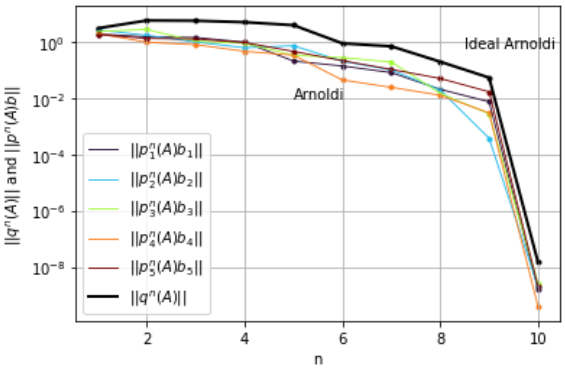
Arnoldi iteration and Chebyshev polynomials

Kateryna Ufymtseva

Bachelor Thesis
Spring Term 2021



LUND UNIVERSITY

Faculty of Science
Centre for Mathematical Sciences
Numerical Analysis

Abstract

In this thesis, we examine the Arnoldi iteration - an iterative algorithm used for finding a Hessenberg form of a matrix as well as approximating its eigenvalues by forming an orthonormal basis of a Krylov subspace. We explore the mechanism behind the work of the algorithm and how the values it finds approximate the eigenvalues.
In the process of answering these questions we consider the concepts of the ideal Arnoldi approximation problem, ideal Arnoldi polynomial and Chebyshev polynomial of a matrix, and how they are related to the original Arnoldi approximation problem, which is solved by the iteration. We also look into the pseudospectra of a matrix, its connection to eigenvalue estimates, also known as Ritz values, and Chebyshev polynomials.

Acknowledgements

I want to extend gratitude to my supervisor Claus Führer, who helped me to write this thesis even in the summer and amongst the challenges of the pandemic, and also to my family, who supported me in this work.

Contents

1 Background

Solving systems of linear equations and finding eigenvalues of a matrix are two very important problems in linear algebra. There are methods to solve them analytically, but they generally cannot be used to compute solutions of large problems numerically. That is why a number of methods were developed to solve these problems using different approaches.

When solving the problems numerically, one uses algorithms employing matrix factorizations, such as LU and QR factorizations and a Hessenberg form of a matrix. They are used to represent a matrix as a product of two or more matrices with special structure: lower- and upper-triangular matrices in LU factorization and orthogonal and upper-triangular matrices in QR decomposition, as an example. There exist a group of known and studied methods that solve a problem exactly, such as, for example, Gram-Schmidt algorithm for solving linear systems of equations and Gaussian elimination. However, they have the drawback of being expensive to compute.

Another group of algorithms that are called iterative methods go around this issue. Unlike direct methods, that can take $O(n^3)$ to complete and have to be finished to achieve results, iterative methods provide useful results even after a small (less than the dimension of a problem) number of iterations. Algorithms of this type are often used to approximate solutions to large problems. One of their common uses are problems concerning sparse matrices - matrices that have very few non-zero elements (roughly equal to the number of rows or columns). These types of problems often arise in engineering when solving partial differential equations, in combinatorics and network theory.

The Arnoldi iteration is an iterative method used for approximating eigenvalues that was first introduced in 1950 in the paper "The principle of minimized iterations in the solution of the matrix eigenvalue problem" by W. E. Arnoldi [1]. The paper describes a generalization of a Lanczos' algorithm - an algorithm for diagonalizing symmetric matrices - to be applied to all matrices.
After its first formulation, the algorithm was studied and there was established a link between the way it operates and approximation theory of polynomials. The mechanism of the iteration is now thought about as projections onto Krylov subspaces, while the process by which it locates eigenvalues is not yet completely clear.

This thesis is concerned with explaining which eigenvalues are approximated and why, as well as exploring the connection between the algorithm, Chebyshev polynomials, and pseudospectra.

2 Algorithm overview

Notation:
We consider real $m \times m$ matrices A here, the norm $\|.\|$ is a vector two norm or a matrix two norm.

Definition 1 An *upper Hessenberg matrix* is a matrix whose elements $a_{i,j} = 0,\ \forall i, j$ with $i > j + 1$. It is a matrix of the form:

$$H = \begin{bmatrix} h_{11} & h_{12} & \dots & h_{1m} \\ h_{21} & h_{22} & & \\ & \ddots & \ddots & \vdots \\ & & & h_{m-1,m} \\ & & h_{m,m-1} & h_{mm} \end{bmatrix}$$

The matrix H is called the Hessenberg form of A, if $A = QHQ^*$ or $AQ = QH$, where Q is an orthogonal matrix [2].

Definition 2 *Krylov subspaces* are a nested set of spaces K_i defined as follows: for an $n \times n$ matrix A and a vector b of length n,

$$K_n = [b, Ab, A^2b, A^3b, ..., A^{n-1}b]. \tag{1}$$

The Arnoldi iteration is an algorithm, which aims to recursively construct an orthonormal basis for the Krylov subspaces K_i.
At every step, the eigenvalues of a section of the Hessenberg matrix H approximate some of the eigenvalues of the original matrix A. These approximations are called Arnoldi estimates or Ritz values. At each step n there are exactly n of them.

The iteration is derived as follows [2]:
Let m, $n < m$ be positive integers, A a real $m \times m$ matrix, $\|.\|$ a two-norm, b an arbitrary m-vector. Let Q_n be the $m \times n$ matrix consisting of the first n columns of Q:

$$Q_n = \left[\begin{array}{c|c|c|c|c} & & & & \\ q_1 & q_2 & q_3 & \dots & q_n \\ & & & & \end{array}\right]$$

and H_n be the $(n + 1) \times n$ upper-left section of H:

$$H_n = \begin{bmatrix} h_{11} & & \dots & h_{1n} \\ h_{21} & h_{22} & & \\ & \ddots & \ddots & \vdots \\ & & h_{n,n-1} & h_{nn} \\ & & & h_{n+1,n} \end{bmatrix}$$

Then $AQ_n = Q_{n+1}H_n$. Writing this equality in more detail we get:

$$\begin{bmatrix} & & \\ & A & \\ & & \end{bmatrix} \left[\begin{array}{c|c|c} & & \\ q_1 & \dots & q_n \\ & & \end{array}\right] = \left[\begin{array}{c|c|c} & & \\ q_1 & \dots & q_{n+1} \\ & & \end{array}\right] \begin{bmatrix} h_{11} & \dots & h_{1n} \\ h_{21} & & \\ & \ddots & \vdots \\ & & h_{n+1,n} \end{bmatrix}$$

And performing the matrix multiplications explicitly we get:

$$\left[\begin{array}{c|c|c} & & \\ Aq_1 & \dots & Aq_n \\ & & \end{array}\right] = \left[\begin{array}{c|c|c|c} & & & \\ h_{11}q_1 + h_{21}q_2 & h_{12}q_1 + h_{22}q_2 + h_{32}q_3 & \dots & h_{1n}q_1 + ... + h_{n+1,n}q_{n+1} \\ & & & \end{array}\right]$$

Writing down the nth column of this equation separately gives the following expression:

$$Aq_n = h_{1n}q_1 + ... + h_{nn}q_n + h_{n+1,n}q_{n+1} \tag{2}$$

It is a recurrence definition for q_{n+1} involving the previous vectors q_i for $i \leq n$, which means that they can be computed consecutively, given the first vector q_1. The algorithm implements this recurrence relation with the initial vector chosen arbitrarily, with the only condition that it is of length 1.

The following is a pseudocode that describes the algorithm [3, p. 3]:

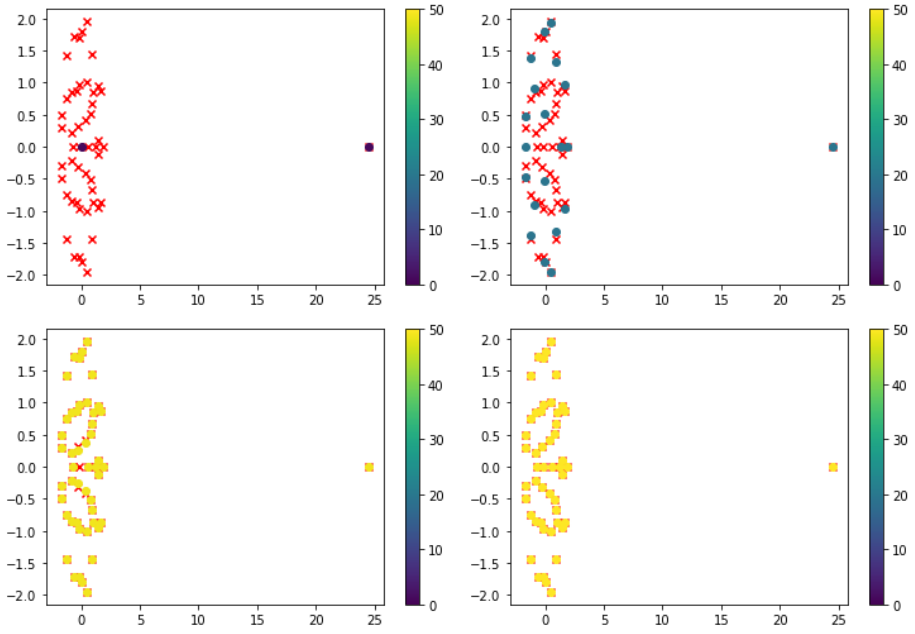
Algorithm 1: Arnoldi iteration

```
b = arbitrary;
q_1 = b/||b||;
for n = 1, 2, ... do
    v = A q_n;
    for j = 1 to n do
        h_jn = q_j^* v;
        v = v − h_jn q_j;
    end
    h_{n+1,n} = ||v||;
    q_{n+1} = v/h_{n+1,n};
end
```

The eigenvalues of H_n, derived via an additional algorithm, are the Arnoldi estimates of the matrix A.
The algorithm was implemented in Python and is displayed in the appendix.

Figure 1 displays the outcome of the Arnoldi iteration performed for a random matrix 50×50 on steps 2, 20, 49 and 50. Red crosses represent true eigenvalues of the matrix, the dots are Arnoldi estimates for a given step.
In the last step the iteration locates the eigenvalues exactly. In practice, the algorithm is stopped before the last iteration, because the dimensions of the matrix are large and the goal is to obtain estimated values of the eigenvalues.
We can observe that the algorithm locates the eigenvalues on the edges first, and

Figure 1: The graph of eigenvalues (red crosses) and eigenvalue estimates of a matrix at an iteration (dots, with the colour indicating the number of the iteration)



gradually "moves" closer to the center of the cluster.

2.1 Arnoldi iteration and minimal polynomial

It is known that matrix iterative methods are closely related to approximation theory [4]. In the case of the Arnoldi iteration, it is polynomial approximation. In the following two sections we will discuss the theory behind the Arnoldi algorithm and state the Arnoldi(/Lanczos) approximation problem that is solved by it [2].

2.2 Projections onto successive Krylov subspaces

According to Definition 2, Krylov subspaces are a nested set of spaces

$$K_n = [b, Ab, A^2b, A^3b, ..., A^{n-1}b],$$

for an $n \times n$ matrix A and a vector b of length n.

By definition, the columns of the matrix Q, q_i, form an orthonormal basis for Krylov subspaces generated by A and q_1.

$$K_n = [q_1, Aq_1, A^2q_1, ..., A^{n-1}q_1] = [q_1, q_2, ..., q_{n-1}] \tag{3}$$

The matrix $\tilde{H}_n = Q_n^T A Q_n$ is the Hessenberg matrix H_n with the last row removed. It represents the orthogonal projection of A onto K_n in the orthogonal basis $q_1, ..., q_n$.

Definition 3 We define the *Krylov matrix* K_n as a matrix of columns $A^i b$:

$$K_n = \left[\begin{array}{c|c|c|c} & & & \\ b & Ab & ... & A^{n-1}b \\ & & & \end{array}\right]$$

Theorem 4 [2, p. 255]The matrices Q_n generated by the Arnoldi iteration are reduced QR factors of the Krylov matrix:

$$K_n = Q_n R_n \tag{4}$$

The Hessenberg matrices H_n are the corresponding projections

$$\tilde{H_n} = Q_n^* A Q_n, \tag{5}$$

and the successive iterates are related by the formula

$$AQ_n = Q_{n+1} H_n, \tag{6}$$

2.3 Minimal polynomial

Consider a vector $x \in K_n$. It can be written as

$$x = c_0 b + c_1 Ab + c_2 A^2 b + ... + c_{n-1} A^{n-1} b \tag{7}$$

for some $c_i \in \mathbb{C}$.
Let

$$q(z) = c_0 + c_1 z + c_2 z^2 + ... + c_{n-1} z^{n-1}, \tag{8}$$

then $x = q(A)b$ and can be analysed in terms of matrix polynomials.
Monic polynomials are polynomials with the leading coefficient one. Define $P^n = \{$monic polynomials of degree n$\}$, then

Arnoldi Approximation Problem:
Find the minimal polynomial $p_b^n \in P^n$ such that $\|p_b^n(A)b\|_2 =$ minimum (9)

Theorem 5 [2, p. 259] As long as the Arnoldi iteration does not break down (i.e. K_n is of full rank n), the Arnoldi approximation problem has a unique solution p_b^n, namely the characteristic polynomial of H_n.

3 Chebyshev polynomials

Chebyshev polynomials were first described and studied by the Russian mathematician Pafnuty Chebyshev, who worked, among other things, on number theory, integral theory and problems in mechanics [5].
They are used in many areas of applied mathematics, such as interpolation problems, approximating functions with a polynomial and trigonometry.
We will describe the original Chebyshev polynomials defined on a closed interval. There were also developed generalisations of Chebyshev polynomials to compact subsets of $\mathbb{C}$ and finitely many smooth regions, which we will not go into [6][p. 193].

3.1 Chebyshev polynomials on a closed interval

Definition 6 *Chebyshev polynomials* $T_n(x)$ are defined as

$$T_n(x) = \cos(n \cdot \arccos(x)),\ n \in \mathbb{N}_0. \tag{10}$$

$T_n(x)$ are polynomials of degree n that can also be defined by a recurrent relation:

Definition 7 *Chebyshev polynomials* $T_n(x)$ are defined by

$$T_0(x) = 1,\ T_1(x) = x \text{ and } T_{n+1}(x) = 2x \cdot T_n(x) - T_{n-1}(x) \tag{11}$$

Since the functions are a cosine function performed on some argument, their attained values lie in $[-1, 1]$. This also makes it easier to determine their zeros and extreme values.

The zeros of T_n are called *Chebyshev points* x_k^* and are described by:

$$x_k^* = \cos\left(\frac{2k+1}{2n} \cdot \pi\right) \ \in [-1, 1],\ 0 \le k \le n-1. \tag{12}$$

The extreme values $|T_n(\overline{x_k})| = 1$ are attained at *Chebyshev abscissae*:

$$\overline{x_k} = \cos\left(\frac{k}{n}\pi\right) \ \in [-1, 1],\ 0 \le k \le n. \tag{13}$$

Chebyshev polynomials have a minimal property that is formulated in the following theorem.

Theorem 8 (Minimal property of Chebyshev polynomials)
1. Let $\mathcal{P}$ be a set of polynomials of degree n that have a leading coefficient $a_n \neq 0$, then

$$\exists \xi \in [-1, 1] :\ |p(\xi)| \ge \frac{|a_n|}{2^{n-1}}.$$

2. Let $\mathcal{P}_n^*$ be a set of polynomials of degree n with leading coefficient $a_n = 1$, and $\|p\|_\infty = \max_{\xi \in [-1,1]} |p(\xi)|$. Then

$$\|2^{-(n-1)} T_n\|_\infty \le \min_{w \in \mathcal{P}_n^*} \|w\|_\infty.$$

By the recurrent definition, the leading coefficient of $T_n(x)$ is 2^{n-1}, therefore monic Chebyshev polynomials are defined as

$$\tilde{T}_n(x) = \frac{1}{2^{n-1}} T_n(x) = \frac{1}{2^{n-1}} \cos(n \cdot \arccos(x)) \quad \text{, for } n \in \mathbb{N}_0.$$

3.2 Chebyshev polynomials of matrices

The properties of Chebyshev polynomials on compact sets are well-known, however there hasn't been a lot of theoretical work in the field of the Chebyshev polynomials of matrices.
In a 2010 work [7], V. Faber, Tichy and Liesen explored general properties of Chebyshev polynomials of matrices in analogy to the ones on compact sets, as well as considered special cases of matrices. Among the properties common or related to Chebyshev polynomials of compact sets, are behaviour under shifts or scalings, and even/odd properties.
In the next section, we consider Chebyshev polynomials of matrices and their connection to the Arnoldi iteration.

4 Ideal Arnoldi approximation problem and Chebyshev polynomials of matrices

4.1 Ideal Arnoldi approximation problem

The Arnoldi iteration depends on the starting vector b. However, it can be removed from the discussion and the following problem can be posed:

$$\text{Find } q^n \in P^n \text{ such that } \|q^n(A)\| = \text{ minimum.} \tag{14}$$

It is called the **ideal Arnoldi approximation problem**.

The outcome of the Arnoldi iteration - the matrix H_n and, hence, the quality of approximation - depends on the starting vector b, as well as the matrix A. However, the choice of b is arbitrary, and although its special qualities can be important in some cases, they do not usually impact the algorithm very much. In general, properties of the matrix A are more significant to the problem.

By passing from the initial approximation problem to the ideal one, we can remove the effects of the starting vector and focus on the matrix essence of the problem.

There is an inequality relation between the polynomial solutions to the original and ideal Arnoldi approximation problems:

Theorem 9 [4] The original and ideal Arnoldi polynomials are related by

$$\frac{\|p_b^n(A)b\|}{\|b\|} \leq \|q^n(A)\| \leq \|p_b^n(A)\|. \tag{15}$$

The existence and uniqueness of the solution of (9) and (14) is stated in the following theorem:

Theorem 10 (2 in [4]) The optimal polynomials p_b^n and q^n exist. Provided that the minima in (9) and (14) are nonzero, they are unique.

(16)

Proof
We can regard the original Arnoldi problem of minimizing $\|p_b^n(A)b\|$ as a different problem:
Find $v^* \in span\{b, Ab, ..., A^{n-1}b\}$ such that

$$\|v^* - A^n b\| \leq \|v - A^n b\| \ \ \forall v \in span\{b, Ab, ..., A^{n-1}b\},$$

which is a problem of finding the best approximation of $A^n b$ in the space spanned by $b, Ab, ..., A^{n-1}b$.
Similarly, the ideal Arnoldi problem of minimizing $\|q^n(A)\|$ can be thought of as finding $v^* \in span\{I, A, ..., A^{n-1}\}$ such that

$$\|v^* - A^n\| \leq \|v - A^n\| \ \ \forall v \in span\{I, A, ..., A^{n-1}\},$$

We will consider both of these problems in terms of vectors: find the closest to y point v in vector space V.

1. Existence
Existence of p_b^n and q^n stems from compactness of the spaces spanned by $b, Ab, ..., A^{n-1}b$ and $I, A, ..., A^{n-1}$.

2. Uniqueness
The proof of uniqueness can be divided in two parts:
a) is the closest point $v \in V$ to y unique?
b) does it have a unique representation in terms of the vectors?

We will consider the latter question first.
b) For the ideal Arnoldi problem the uniqueness of the representation of v is equivalent to the vectors $I, A, ..., A^{n-1}$ are linearly independent.
We will do a proof by contradiction. Assume that the vectors are linearly dependent, then there exists a linear combination of them that is equal to zero with a non-zero coefficient by $A^{n-1} \Rightarrow A^{n-1} \in span\{I.A, ..., A^{n-2}\} \Leftrightarrow A^n \in span\{A, A^2, ..., A^{n-1}\}$ and, therefore, $\|q^n(A)\| = 0$, which is a contradiction.

The reasoning is analogous for the original Arnoldi problem.
We will give the outline for the proof of part a.
a) For the original problem the proof follows from strict convexity of the vector two-norm.

For the ideal problem, the proof is by contradiction, assuming that there exist two distinct solutions $q_1(z)$ and $q_2(z)$. Then we form $q(z) = \frac{1}{2}\big(q_1(z) + q_2(z)\big)$ and consider the right singular values of A. It is then proven that $(q_2 - q_1)(A)w_j = 0$ for a number of singular values w_j, $1 \leq j \leq J$. Then we consider the convex linear combination $q_\epsilon = (1 - \epsilon)q(z) + \epsilon\delta q(z)$, where $\delta q(z)$ is a monic polynomial derived from $(q_2 - q_1)(z)$. It is then shown that q_1 and q_2 are not minimal, which leads to a contradiction.
■

It was stated in [4] that the core of the process by which Arnoldi iteration locates eigenvalues is the solution to the ideal problem, not the original one. The outcome of the algorithm hinges on the implicit hope that the solution to (9) is a good approximation of the solution to (14).

The solution to the ideal approximation problem, q^n, might be called a Chebyshev polynomial of the matrix A of degree n, in an analogy to a Chebyshev polynomial of a subset of the complex plane, which is a monic polynomial that minimizes the sup-norm on that set. This analogy becomes an identity in the special case when A is normal [4].

4.2 Special case of normal matrices

When A is normal, it is diagonalizable $A = U\Lambda U^T$ and a polynomial of A multiplied by the initial vector can be written as:

$$p(A)b = Up(\Lambda)U^T b = \sum_{j=1}^{m} w_j p(\lambda_j) u_j, \tag{17}$$

where $w_j = u_j^T b$, $\{\lambda_j\}$ are eigenvalues and $\{u_j\}$ are corresponding orthonormal eigenvectors of A.
Therefore, the norm of this expression is given by

$$\|p(A)b\| = \left((p(A)b)(p(A)b)^T\right)^{\frac{1}{2}} = \left(\sum_{j=1}^{m}\sum_{i=1}^{m} w_j w_i p(\lambda_j)p(\lambda_i)\cdot u_j u_i^T\right)^{\frac{1}{2}} = \left(\sum_{j=1}^{m} w_j^2 p(\lambda_j)^2\right)^{\frac{1}{2}} \tag{18}$$

since $\{u_j\}$ are orthonormal. Then the original Arnoldi problem (9) is equivalent to a weighted least squares approximation problem in the complex plane.
For the ideal Arnoldi approximation problem, the identity

$$\|p(A)\| = \sup_{\lambda\in\Lambda(A)} |p(\lambda)| \tag{19}$$

shows that it is equivalent to the Chebyshev approximation problem in the complex plane: find a polynomial that has a minimum supremum norm on $\Lambda(A)$.
In particular, the ideal Arnoldi polynomial q^n is the same as the Chebyshev polynomial for $\Lambda(A)$.

4.3 Computations

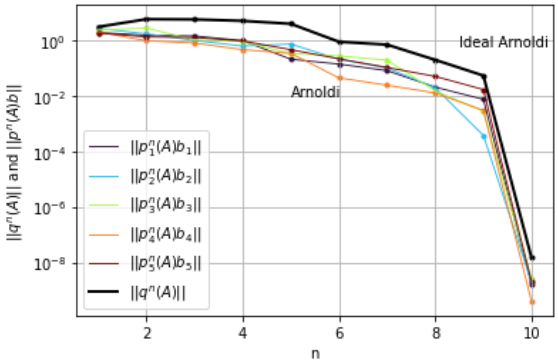
In this section we approximate the ideal Arnoldi polynomial q^n and plot it with true Arnoldi polynomials for various initial vectors.
For normal matrices, the ideal Arnoldi polynomial is simply the Chebyshev polynomial of the set $\Lambda(A)$, as explained in the previous section, and can be computed by known algorithms. For the general case, there also exist an algorithm involving semidefinite programming described in [9] by Trefethen and Toh. However, here, we used a simpler and more computationally expensive procedure discussed in the earlier paper [4].

It is based on Theorem 9 in the case of $\|b\| = 1$. The polynomial q^n can be computed with an optimization code to find b with $\|b\| = 1$ such that $\|p_b^n(A)b\|$ is maximum on a fixed step n. From Theorem 9 we have

$$\|p_b^n(A)b\| \le \|q^n(A)\| \le \|p_b^n(A)\| \tag{20}$$

$\forall b$ s.t. $\|b\| = 1$. If such a b can be found that $\|p_b^n(A)b\| = \|p_b^n(A)\|$, then $p_b^n = q^n$. It is not known whether such b always exist, but it is assumed that it does in [4].
We used an optimization code *minimize* from *scipy.optimize* in Python. It implements a Nelder-Mead method of optimization, which was chosen because it requires only function evaluations to run. The arguments of the method are the starting

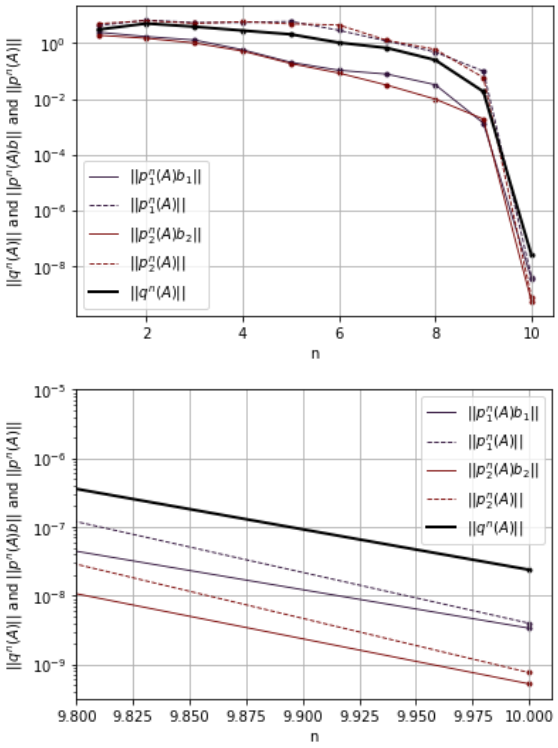
Figure 2: The graph of the norms of ideal and true Arnoldi polynomials for a random 10×10 matrix A. Dots represent values at each step.



vector b and the function that has the vector b as the argument. The function performs the Arnoldi iteration until the step number n and returns the value of the norm $\|p_b^n(A)b\|$. The optimization code finds a b that maximizes this norm.

The black line in the graph represents the norm of the ideal Arnoldi polynomial $\|q^n(A)\|$ and the colored thin lines represent the vector norms $\|p_b^n(A)b\|$ of original Arnoldi polynomials for different starting vectors b at all steps 1-10.
We can see that the norm of the ideal polynomial is larger than the norms of original polynomials.

Figure 3: The graph of the norms of ideal and original Arnoldi polynomials for a random 10×10 matrix A, with dots representing values at each step.



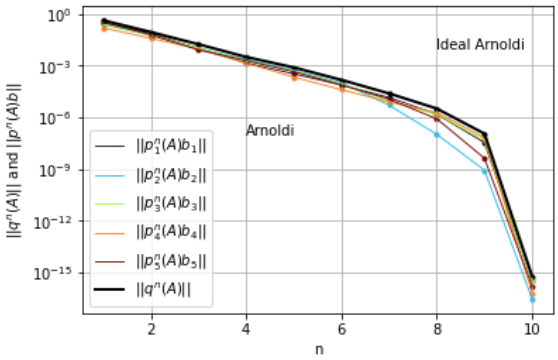
The second graph with dashed lines, shows the norm $\|p_b^n(A)\|$ in addition to

$\|p_b^n(A)b\|$ and $\|q^n(A)\|$. The first graph shoes the norms of polynomials on steps 1-10, and the second graph provides a closer look on the values obtained on step 10. According to the inequality in Theorem 9, the norms $\|p_b^n(A)\|$ are larger or equal to the norm of the ideal polynomial. This inequality is also illustrated by the graph, barring the last step m.
In fact, according to the theory, all the norms have the same value in the last step. Both the true and ideal Arnoldi polynomials are equal to the characteristic polynomial of A on the final step of the algorithm (since H is unitarily similar to A by definition and, thus, has the same characteristic polynomial), and their norms are equal to zero.
We argue that in our implementation the norms are not identical on the last step due to computational errors, as the differences are of order of magnitude 10^{-10} or less.
The fact that the norms of each individual Arnoldi polynomial multiplied by the initial vector is closest to the norm of the same polynomial not multiplied by the vector in our graph, is most likely also due to the implementation.

The program was also run for matrices with special structure. Arnoldi iteration is widely used for sparse matrices, thus we used special sparse matrices in our examples.

1. Diagonal

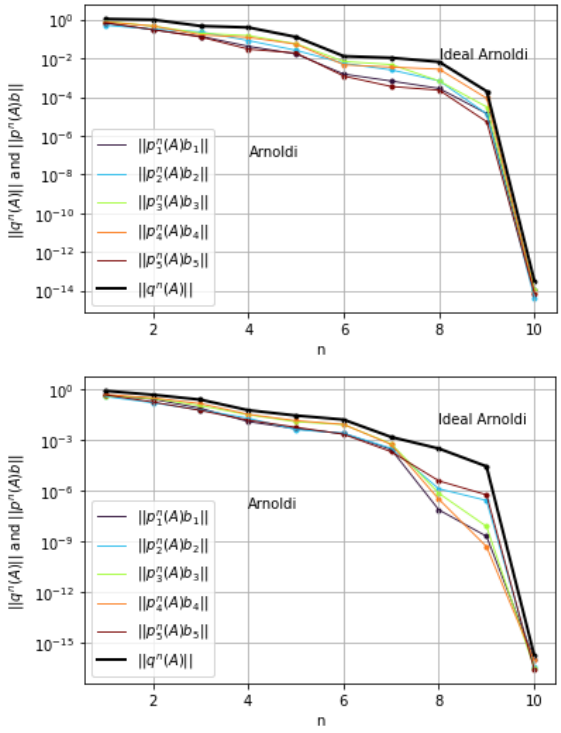
Figure 4: The graph of the norms of ideal and true Arnoldi polynomials for a diagonal 10×10 matrix A



2. Bidiagonal

In both cases, no special properties with regards to Arnoldi polynomials or ideal Arnoldi polynomials were found. The only notable difference from a general case is the heightened accuracy - at the last step the norms are of order of magnitude around 10^{-15}, not 10^{-8}.

Figure 5: Graphs of the norms of ideal and true Arnoldi polynomials for a bidiagonal 10×10 matrix A



5 Arnoldi iteration and pseudospectra

Definition 11 of *eigenvalues and eigenvectors*
Let $A \in \mathbb{R}^{m \times m}$ be a square matrix. A nonzero vector $x \in \mathbb{C}^m$ is an eigenvector of A, and $\lambda \in \mathbb{C}$ is its corresponding eigenvalue, if and only if

$$Ax = \lambda x \tag{21}$$

The set of all eigenvalues of A is called the spectrum of A and is denoted by $\Lambda(A) \subset \mathbb{C}$. Eigenvalues and eigenvectors provide valuable information about matrices and the way they operate, as well as reduce problems to a collection of scalar ones. Examples of the use of eigenvalues are problems of resonance and stability. [2]

Pseudospectra of a matrix, like spectra of a matrix, is a property that describes the way a matrix "behaves". Eigenvalues are often useful to determine what a matrix "does", for example in modelling physical events, however, in some cases, particularly when the matrix in question is far from normal - that is, does not have a full set of orthogonal eigenvectors, - its eigenvalues cannot accurately describe the matrix. [2, p. 258] In cases like these, another, more complex, concept is used - pseudospectra.

There are several equivalent definition of pseudospectra. [10]

Definition 12 Let $A \in \mathbb{R}^{m\times m}$ and $\epsilon > 0$ be arbitrary. The ϵ-pseudospectrum $\sigma_\epsilon(A)$ of A is the set of $z \in \mathbb{C}$ such that

$$\|(z - A)^{-1}\| > \epsilon^{-1}. \tag{22}$$

Using a convention that $\|(z-A)^{-1}\| = \infty$ for $z \in \sigma(A)$, where $\sigma(A)$ is the spectrum (the set of eigenvalues).

Definition 13 $\sigma_\epsilon(A)$ is the set of $z \in \mathbb{C}$ such that

$$z \in \sigma(A + E) \tag{23}$$

for some $E \in \mathbb{C}^{m\times m}$ with $\|E\| < \epsilon$.

Definition 2 defines ϵ-pseudospectra as a set of points that are eigenvalues of a perturbed matrix $A + E$ with $\|E\| < \epsilon$.

Definition 14 $\sigma_\epsilon(A)$ is the set of $z \in \mathbb{C}$ such that

$$\|(z - A)v\| < \epsilon \tag{24}$$

for some $v \in \mathbb{C}^m$ with $\|v\| = 1$.

For an arbitrary $A \in \mathbb{R}^{m\times m}$ and $\epsilon > 0$, $\sigma_\epsilon(A)$ is nonempty, open, and bounded, with at most m connected components, each containing one or more eigenvalues of A.

It was shown numerically that the ideal Arnoldi polynomial approximates pseudospectra for matrices that are far from normal.[9]
The way the polynomials approximate pseudospectra is through their lemniscates, regions defined as $\mathcal{L}_n(A) = \{z \in \mathbb{C} : |q^n(z)| \leq \|q^n(A)\|\}$.
Since $\|q^n\|_{\Lambda(A)} \leq \|q^n(A)\|$, the lemniscates contain the spectrum $\Lambda(A) \subset \mathcal{L}_n(A)$.

In their 1998 study "Chebyshev polynomials of a matrix" [9] Kim-Chuan Toh and Lloyd Trefethen found a link between pseudospectra and ideal Arnoldi lemniscates. They conducted a number of numerical experiments that revealed a connection between the pseudospectra of a certain, arbitrarily chosen, ϵ and lemniscate regions of the Chebyshev polynomial of the same matrix. Matrices considered included a diagonal, a bidiagonal, a Grcar, an ellipse, a Beam-Warming matrices and others.
The nature of the approximation of pseudospectra by Chebyshev lemniscates is not outlined in that paper, but the results open an area for further research on the topic.

6 Conclusions

In this thesis we looked into the Arnoldi iteration, an iterative algorithm that provides useful approximations of eigenvalues of a matrix. We relayed its implementation as well as the way it works and delivers the results on a theoretical level. We also looked into questions of how the iteration produces the values and why they are used as approximations of the eigenvalues.

Expanding on the latter point, we explored the ideal Arnoldi approximation problem and the way it relates to the original Arnoldi approximation problem. It also provides insight into the way the original algorithm approximates properties of the matrix.

As an illustration to an inequality involving the norms of the Arnoldi polynomial and the Chebyshev polynomial we created a program that plots the norms. We plotted the norms for several types of matrices.

We also explored the link between pseudospectra of a matrix and lemniscates of the ideal Arnoldi polynomial.

Some further work can be done into the question of how and why Chebyshev polynomial approximates pseudospectra, and how it reflects other properties of a matrix. Another area of potential study can be directed towards studying the Chebyshev polynomials of a matrix themselves. Some of their properties were discussed and proven in [7], but there is room for further work.

7 The Code

The Arnoldi iteration

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.animation import FuncAnimation

N = 20; #big number
m = 20; #the size of matrix
b = np.random.rand(m);  #arbitrary

A = np.random.rand(m, m);  #matrix A is arbitrary

np.reshape(A, (m , m));

q = b / np.linalg.norm(b);
Q = np.array([q]);
H = np.zeros((N + 1, N + 1));
np.reshape(H, (N + 1, N + 1));

eigenvaluesH = []; #array of arrays with Ritz values

for n in range(N):
    v = np.dot(A, q);  #use a blackbox procedure
    for j in range(n + 1):
        qj = np.transpose(Q[j, :]);
        h = np.dot(qj, v);
        v = v - h * qj;
        H[j][n] = h;
    h = np.linalg.norm(v);
    H[n + 1][n] = h;
    q = v / h;
    Q = np.concatenate((Q, [q]), axis = 0);

    ev, eigenvectorsH = np.linalg.eig(H[0:(n + 1), 0:(n + 1)]);
    eigenvaluesH.append(ev);

Q = np.transpose(Q)


eigenvaluesA, eigenvectorsA = np.linalg.eig(A)
```

The Arnoldi iteration function

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.animation import FuncAnimation


def arnoldi(A, N, b = np.zeros(2)):
    (m, m) = np.shape(A);
    if b.all() == 0:
        b = np.random.rand(m);

    q = b / np.linalg.norm(b);
    Q = np.array([q]);
    H = np.zeros((N + 1, N + 1));
    np.reshape(H, (N + 1, N + 1));

    eigenvaluesH = [];

    for n in range(N):
        v = np.dot(A, q);  #use a blackbox procedure
        for j in range(n + 1):
            qj = Q[j, :].T;
            h = np.dot(qj, v);
            v = v - h * qj;
            H[j][n] = h;
        h = np.linalg.norm(v);
        H[n + 1][n] = h;
        q = v / h;
        Q = np.concatenate((Q, [q]), axis = 0);

        ev, eigenvectorsH = np.linalg.eig(H[0:(n + 1), 0:(n + 1)]);
        eigenvaluesH.append(ev);

    return eigenvaluesH;
```

The code for plotting the norms of ideal Arnoldi polynomials $\|q^n(A)\|$

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
from arnoldi_function import *
from scipy.optimize import minimize


def pFunc(p, X):
    ans = np.zeros((N, N))
    #print('x', x)
    Xi = np.eye(N)
    for i in range(np.size(p)):
        #print('p[', i, ']: ', p[i])
        #print('X^', i, ': ')
        #print(Xi)
        ans += p[i] * Xi
        Xi = np.dot(X, Xi)
    return ans


#comparison of the ideal Arnoldi polynomial and the real Arnoldi
    polynomials with different b
N = 10; #big number
m = 10; #the size of matrix
A = np.random.rand(m, m);  #matrix A is arbitrary


l = 2 #number of true arnoldi polynomials
color = plt.get_cmap('turbo', l) #'plasma'

Y = np.zeros((l, N))
Y1 = np.zeros((l, N))

for k in range(l):

    b = np.random.rand(m);  #arbitrary

    np.reshape(A, (m , m));

    q = b / np.linalg.norm(b);

    eigenvaluesH = arnoldi(A, N, q)

    X = np.array(range(1, N + 1))

    for i in range(N):
        p = np.poly(eigenvaluesH[i])
        p = p[::-1]
        Y[k, i] = np.linalg.norm(np.dot(pFunc(p, A), q), ord = 2)
        Y1[k, i] = np.linalg.norm(pFunc(p, A), ord = 2)


    plt.plot(X, Y[k], color = color(k), linewidth = 0.9, label = r'
    $||p^n_{}(A)b_{}||$'.format(k + 1, k + 1))
    plt.plot(X, Y1[k], color = color(k), linewidth = 0.9, linestyle
    = '--', label = r'$||p^n_{}(A)||$'.format(k + 1))

```

```
55
56 #compute and plot the ideal arnoldi polynomial
57
58 YY = np.zeros(N)
59
60 for n in range(1, N + 1):
61     pp = np.zeros(n)
62
63     def bFunc(b):
64         global n
65         global pp
66         b = b / np.linalg.norm(b, ord = 2)
67         eigenvaluesH = arnoldi(A, n, b)
68         p = np.poly(eigenvaluesH[n - 1])
69         p = p[::-1]
70         pp = p
71         norm = np.linalg.norm(np.dot(pFunc(p, A), b), ord = 2)
72         return -norm
73
74
75     b0 = b / np.linalg.norm(b, ord = 2)
76
77     res = minimize(bFunc, b0, method='nelder-mead',
78                    options={'xatol': 1e-4, 'disp': True}) #1e-8
79
80     print("res.fun = ", res.fun)
81     print('pp = ', pp)
82     print('matrix norm: ', np.linalg.norm(pFunc(pp, A), ord = 2))
83
84     YY[n - 1] = -res.fun
85
86
87
88 plt.plot(X, YY, color = 'black', linewidth = 2.0, label = r'$||q^n(
    A)||$')
89
90 plt.grid(True)
91 plt.yscale('log')
92 #plt.ylim(10**(-10), 10**(-4))
93 #plt.xlim(9.6, 10.1)
94 plt.xlabel('n')
95 plt.ylabel(r'$||q^n(A)||$ and $||p^n(A)b||$ and $||p^n(A)||$')
96 #plt.text(7, 10**0, r'Ideal Arnoldi')
97 #plt.text(5, 10**(-3), r'Arnoldi')
98 plt.legend()
99 plt.show()
100
101 #----------------------closer look at the last step
    ----------------------------------------------------
102
103 plt.grid(True)
104 plt.yscale('log')
105
106 for k in range(1):
107     plt.plot(X[8:10], Y[k, 8:10], color = color(k), linewidth =
    0.9, label = r'$||p^n_{}(A)b_{}||$'.format(k + 1, k + 1))
108     plt.plot(X[8:10], Y1[k, 8:10], color = color(k), linewidth =
    0.9, linestyle = '--', label = r'$||p^n_{}(A)||$'.format(k + 1))
```

```
#plt.plot(X, Y, color = color(l), linewidth = 0.9)
#plt.plot(X, Y1, color = color(l), linewidth = 0.9, linestyle =
    '--')
plt.plot(X[8:10], YY[8:10], color = 'black', linewidth = 2.0, label
     = r'$||q^n(A)||$')

plt.xlabel('n')
plt.ylabel(r'$||q^n(A)||$ and $||p^n(A)b||$ and $||p^n(A)||$')
plt.ylim(10**(-9.5), 10**(-5))
plt.xlim(9.8, 10.01)
plt.legend()
plt.show()
```

The code for creating the animation

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.animation as animation

from arnoldi import eigenvaluesH, eigenvaluesA, N;

evs = eigenvaluesH;
m = np.size(evs);

#initialize the writer
Writer = animation.writers['ffmpeg']
writer = Writer(fps = 5, metadata = dict(artist = 'Me'),
                bitrate = 1800)

#create a figure
fig = plt.figure(figsize=(10,6))
plt.xlim(np.min(evs[m - 1].real) - 1,
         np.max(evs[m - 1].real) + 1)
plt.ylim(np.min(evs[m - 1].imag) - 1,
         np.max(evs[m - 1].imag) + 1)
plt.xlabel('Realz', fontsize = 20)
plt.ylabel('Imagz', fontsize = 20)
plt.title('Arnoldi eigenvalues', fontsize = 20)


colors = plt.get_cmap('viridis', m);
norm = plt.Normalize(0, N)
sm = plt.cm.ScalarMappable(cmap = "viridis", norm = norm)
sm.set_array([])

f = 0

#animation function
def animate(i):
    global f
    data = evs[i]; #select data range
    if i > 0:
      p = sns.scatterplot(x = evs[i - 1].real,
                y = evs[i - 1].imag, data = evs[i - 1],
                color = (1, 1, 1))
    p = sns.scatterplot(x = eigenvaluesA.real,
                y = eigenvaluesA.imag, data = eigenvaluesA,
                color = 'red', marker = 'x')
    p = sns.scatterplot(x = data.real,
                y = data.imag, data = data,
                color = colors(i))
    p.tick_params(labelsize=17)
    if i == 0 and f == 0:
        f = 1
        p.figure.colorbar(sm)


#start the animation
ani = matplotlib.animation.FuncAnimation(fig, animate,
```

```
                                    frames = N, repeat = True)

#save the animation
ani.save('ArnoldiEvs.mp4', writer = writer)
```

References

[1] W. E. Arnoldi, *The principle of minimized iterations in the solution of the matrix eigenvalue problem*, Quarterly of Applied Mathematics, volume 9, pages 17–29, 1951

[2] L. N. Trefeten, David Bau, *Numerical Linear Algebra*, 1997

[3] K.-C. Toh and L. N. Trefethen, *Calculation of pseudospectra by the Arnoldi iteration*, SIAM J. Sci. Comput. 17 (1996), 1-15

[4] A. Greenbaum and L. N. Trefethen, *GMRES/CR and Arnoldi/Lanczos as matrix approximation problems*, SIAM J. Sci. Comput. 15 (1994), 359-368

[5] *Pafnuty Lvovich Chebyshev*, sketch by A. M. Lyapunov (in Russian), http://escriptorium.univer.kharkov.ua/bitstream/1237075002/5967/2/Tom_4_17_Lyapunov.pdf (Online; accessed 13-August-2021)

[6] P. Deuflhard and A. Hohmann, *Numerical Analysis in Modern Scientific Computing*

[7] V. Faber, J. Liesen, and P. Tichy, *On Chebyshev polynomials of matrices*, SIAM J. Matrix Anal. Appl., Vol. 31, No. 4, pp. 2205–2221, 2010

[8] K.-C. Toh, *Matrix Approximation Problems and Nonsymmetric Iterative Methods*, Ph.D. thesis, Cornell University, Ithaca, NY, 1996.

[9] K.-C. Toh and L. N. Trefethen, *The Chebushev polynomials of a matrix*, SIAM J. Matrix Anal. Appl. 20 1998, 400-419

[10] L. N. Trefethen and M. Embree, *Spectra and pseudospectra of matrices, The behaviour of nonnormal matrices and operators*, Princeton University Press, Princeton and Oxford, 2005