ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Jordan Chevalley Decomposition

## Implementation and Numerical Analysis

Bachelor Thesis

Felix Sarnthein-Lotichius

Tuesday 7th April, 2020

Advisors: Prof. Dr. Markus Püschel and Chris Wendler

Department of Computer Science, ETH Zürich

**Abstract**

Introduction: Matrix decomposition is of great importance in the calculation of matrix functions. Simple solutions exist for matrices with an eigenvalue decomposition, but non-diagonalizable matrices pose a problem. In this case the function of a matrix is only defined with respect to its Jordan normal form. I here aim to provide a practical approach towards the Jordan-Chevalley decomposition.

Approach: First, I reformulated two basic algorithms and intuitive correctness proofs that show how the algorithms act on the input matrices. I observed that it might be possible to compute the Jordan-Chevalley decomposition exactly for integer matrices in polynomial-time. At the heart of this project, I programmed a Python extension. It provides sub-routines to compute the Jordan-Chevalley decomposition of integer matrices. I then evaluated my implementation on different classes of matrices and analyzed the numerical behavior of the decomposition.

Results: The *Chevalley iteration on matrices* does not require the computation of complicated intermediate results and is therefore significantly faster than the *Chevalley iteration on polynomials*. The asymptotic runtime complexity of both algorithms depends on the size of intermediate results and appears to be polynomial. In practical terms, we can compute the diagonalizable matrix $D$ more efficiently than the Chevalley polynomial.

Conclusions: I have shown both theoretically and empirically that it is possible to compute the Jordan-Chevalley decomposition of integer matrices without symbolic computation. One can simplify the Jordan structure of a matrix without knowing the structure itself.

Significance: This can be used to compute the eigenvalues and generalized eigenvectors of a defective integer matrix. As a possible application, this is useful to compute a subclass of diagonalizable filters in Graph Signal Processing.

# Contents

Chapter 1

# Introduction

The computation of matrix functions is a well-known mathematical problem with various applications in science and engineering. Specifically, matrix decomposition is of great importance in the calculation of matrix functions. Simple solutions exist for matrices with an eigenvalue decomposition, but non-diagonalizable matrices pose a problem. In this case the function of a matrix is only defined with respect to its Jordan normal form. It is well known that the problem of computing the Jordan structure of a matrix is ill-conditioned and thus not computable in practical applications [16] [1].

Graph signal processing (GSP) theory generalizes concepts of discrete signal processing like filters and Fourier transforms to signals on data indexed by graphs. Matrix polynomials play an important role in the GSP framework described in [25]. A filtered signal at a node $v$ is a weighted sum of the signals at its predecessors. The coefficient $h_i$ is applied to all signals at nodes that precede $v$ by $i$ edges. The matrix representation $H$ of a filter $h$ is therefore defined as the matrix polynomial $h(A)$, where $A$ is the adjacency matrix of the graph. The Fourier transform of $h$ is defined via the eigenvalue decomposition, respectively the Jordan normal form of $H$ [12].

In [20], it was recently proposed to approximate the adjacency matrix $A$ by a diagonalizable matrix $D$, obtained via the Jordan-Chevalley decomposition. Matrix functions of $D$ and $A$ are closely related, since $D$ has the same eigenvalues and (generalized) eigenvectors as $A$, but without the Jordan structure. An effective computation of the Jordan-Chevalley is thus desirable.

Is it possible to compute the Jordan-Chevalley decomposition without symbolic computation? This thesis discusses methods to compute the Jordan-Chevalley decomposition and some numerical properties of the diagonalizable part $D$. Our focus lies on directed graph adjacency matrices as a practical application. The existence of such a method would allow us to simplify the Jordan structure of a matrix without knowing the structure itself.

### Related work

In [20], the Jordan-Chevalley decomposition is calculated using symbolic computation. This works well but is not scalable to large matrices.

In the french literature the Jordan-Chevalley decomposition is widely known as "Dunford decomposition". Doctoral examinations seem to be particularly a motivating for algorithmic approaches towards the decomposition [11] [10]. A comprehensive introduction to the topic is provided by [24]. The notes by Burnol derive a plethora of algorithms [3] - [7]. In general, these are rather theoretic approaches and without practical considerations.

### Our contribution

We aim to provide a more practical approach towards the Jordan-Chevalley decomposition. First, we reformulated two basic algorithms and intuitive correctness proofs that show how the algorithms act on the input matrices. Then, we started an attempt to modify the algorithm from [20] to work without symbolic computation. This fails because the algorithm requires information about the Jordan structure. We observed that it might be possible to compute the Jordan-Chevalley decomposition exactly for integer matrices in polynomial-time. The implementation of the corresponding algorithm requires variable size numbers. The runtime is thus dependent on the size of intermediate results, which could grow exponentially. We evaluated our implementation on different classes of matrices and analyzed the numerical behavior of the decomposition. The results suggest that there might be ways to achieve a Jordan-Chevalley decomposition without exponential growth of intermediate results.

At the heart of this project I programmed a Python extension. It provides subroutines to compute the Jordan-Chevalley decomposition of integer matrices. The implementation consists of a C++ back-end and an front-end written in Cython. The back-end relies heavily on the linear algebra library Eigen [17] for C++. Eigen is a templated header-only library and most of its complexity is abstracted away at compile time. The Boost library is used for variable precision computations. The back-end provides subroutines for polynomial manipulations, as well as a selection of more advanced algorithms that allow computing the Jordan-Chevalley decomposition.

Chapter 2

# Background and Notation

The Jordan structure of a matrix reveals many properties of the underlying linear transformation. Since these concepts are used throughout the thesis, we will recall the most important definitions. We assume prior knowledge of basic linear algebra and use the following notation:

| Symbol | Meaning |
|---|---|
| $n$ | the size of a square matrix |
| $A$ | an $n \times n$ square matrix |
| $m$ | the number of distinct eigenvalues of a matrix $A$ |
| $\lambda_i$ | the $i^{th}$ distinct eigenvalue of a matrix $A$ |
| $ker(\lambda_i I - A)$ | the eigenspace $\lambda_i$ spanned by the eigenvectors $v_j$ |
| $\chi_A(x)$ | the characteristic polynomial of a matrix $A$ |
| $\alpha_i$, $\gamma_i$ | the algebraic and geometric multiplicity of $\lambda_i$ |
| $V\Lambda V^{-1}$ | the eigenvalue decomposition of a diagonalizable matrix $A$ |
| $ker(\lambda_i I - A)^k$ | the generalized eigenspace of $\lambda_i$ of rank $k$ |
| $\ell(\lambda_i, v_j)$ | the length of the Jordan chain of $\lambda_i$ starting at eigenvector $v_j$ |
| $\ell_i$ | the Jordan index of $\lambda_i$ := length of the longest Jordan chain of $\lambda_i$ |
| $\ell$ | the Jordan index of $A$ := length of the longest Jordan chain of $A$ |

We will now apply these concepts to a special example matrix, a Jordan block.

**Definition 2.1 (Jordan block)** *A Jordan block of size $\ell \geq 1$ and eigenvalue $\lambda \in \mathbb{C}$ is the square matrix*

$$J_{\lambda,\ell} = \begin{bmatrix} \lambda & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \lambda \end{bmatrix} \in \mathbb{C}^{\ell \times \ell}.$$

Note that the characteristic polynomial $\chi_{J_{\lambda,\ell}}(x)$ of a Jordan block is given by $(x-\lambda)^\ell$. There is the only one eigenvalue $\lambda$ with algebraic multiplicity $\alpha = \ell$. The eigenspace of $\lambda$ is given by $ker(\lambda I - J_{\lambda,\ell}) = \{e_0\}$ and therefore $\lambda$ has geometric multiplicity $\gamma = 1$. Since $\gamma < \alpha$, we know that $J_{\lambda,\ell}$ is not diagonalizable.

What about the generalized eigenvectors? By solving $(\lambda_i I - A)^k v = 0$ iteratively for $k = 1, \ldots$ we get a sequence of generalized eigenvectors $(e_0, e_1, \ldots)$. This is the Jordan chain of $\lambda$ starting at $e_0$, its length is $\ell(\lambda, e_0) = \ell$.

Using the generalized eigenvectors as a change of basis matrix allows us to generalize the concept of the eigenvalue decomposition.

**Theorem 2.2 (Jordan Decomposition)** *A matrix $A \in \mathbb{C}^{n\times n}$ can be expressed as*

$$A = VJV^{-1},$$

*where $V \in \mathbb{C}^{n\times n}$ is the Jordan basis of $A$ and $J \in \mathbb{C}^{n\times n}$ is its Jordan normal form. The decomposition is unique up to the ordering of the Jordan blocks.*

> *The **Jordan basis V** is a change of basis matrix. Its column vectors are the generalized eigenvectors, arranged according to the Jordan chains of A.*
>
> *The **Jordan matrix J** is a block diagonal matrix. Its blocks are Jordan blocks $J_{\lambda_i,\ell}$ each corresponding to a Jordan chain of A.*

Since a Jordan block is a banded matrix, we can also view the Jordan normal form as a banded matrix, i.e.

$$J = \begin{bmatrix} \ddots & \ddots & & & & & & \\ & \lambda_i & 1 & & & & & \\ & & \ddots & \ddots & & & & \\ & & & \ddots & 1 & & & \\ & & & & \lambda_i & 0 & & \\ & & & & & \ddots & \ddots & \\ & & & & & & \ddots & \end{bmatrix} \in \mathbb{C}^{n\times n}.$$

In particular, we observe that

1. the diagonal of $J$ contains the eigenvalues $\lambda_i$ of A, repeated according to their algebraic multiplicities $\alpha_i$,
2. the superdiagonal of $J$ consits of sequences of ones separated by zeros,
3. a sequence of length $\ell - 1$ corresponds to a Jordan chain of length $\ell$,
4. there are multiple sequences for every eigenvalue $\lambda_i$, the exact number is determined by its geometric multiplicity $\gamma_i$.

Now, lets examine the Jordan decomposition of to two special classes of matrices.

**Example 2.3** *Let $D$ be a diagonalizable matrix. The Jordan decomposition of $D$ is its eigenvalue decomposition, i.e.*

$$D = VJV^{-1} = V\Lambda V^{-1}$$

**Example 2.4** *Jordan normal form of a nilpotent matrix $N$ is superdiagonal since it has only one eigenvalue $\lambda = 0$. The nilpotency index of $N$ is given by its Jordan index, i.e.*

$$N^{\ell} = 0.$$

The next example shows that the Jordan decomposition is ill-conditioned and therefore only computable using symbolic computation.

**Example 2.5** *Consider the matrix*

$$A_{\varepsilon} = \begin{bmatrix} 1 & \varepsilon \\ 0 & 1 \end{bmatrix} \quad \Rightarrow \quad A_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad A_{\varepsilon>0} = V \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} V^{-1}.$$

*We observe that for a small change in the input matrix $A_{\varepsilon}$, the Jordan structure changes completely.*

### Matrix polynomials

The Jordan decomposition allows us to uncover how a polynomial acts on a square matrix. We observe that there are polynomials with $p(A) = 0$.

**Lemma 2.6** *Let $p(x)$ be a polynomial and $A$ be an $n \times n$ matrix with Jordan decomposition $A = VJV^{-1}$. Then $p(A) = Vp(J)V^{-1}$, where*

$$p(J) = \begin{bmatrix} \ddots & \ddots & & & & & \\ & p(\lambda_i) & p'(\lambda_i) & \cdots & \frac{p^{(\ell_i-1)}(\lambda_i)}{(\ell_i-1)!} & & \\ & & \ddots & \ddots & \vdots & & \\ & & & \ddots & p'(\lambda_i) & & \\ & & & & p(\lambda_i) & 0 & \\ & & & & & \ddots & \ddots \\ & & & & & & \ddots \end{bmatrix} \in \mathbb{C}^{n \times n}.$$

**Theorem 2.7 (Cayley-Hamilton)** *Let $A$ be a square matrix with characteristic polynomial $\chi_A(x) = \Pi(x - \lambda_i)^{\alpha_i}$. Evaluating $\chi_A(x)$ on the matrix $A$ yields the zero matrix, i.e.*

$$\chi_A(A) = 0.$$

**Proof** The characteristic polynomial $\chi_A(x) = \Pi(x - \lambda_i)^{\alpha_i}$ has multiple roots of multiplicity $\alpha_i \geq \ell_i$ at every eigenvalue $\lambda_i$ and thus $p(J) = 0$. □

**Definition 2.8 (Annihilating polynomials)** *A polynomial $p(x)$ annihilates $A$ if $p(A) = 0$. In particular all polynomial multiples of $\chi_A(x)$ annihilate $A$.*

**Definition 2.9 (Minimal polynomial)** *The minimal polynomial $\mu_A(x)$ of a matrix $A$ is the non-trivial polynomial of lowest degree that annihilates $A$. It is given by*

$$\mu_A(x) = \Pi(x - \lambda_i)^{\ell_i},$$

*where $\ell_i$ is the Jordan index of $\lambda_i$.*

### Polynomial arithmetic

Recall that the polynomial division is defined as

$$p(x) = q(x) \cdot d(x) + r(x),$$

with dividend $p(x)$, quotient $q(x)$, divisor $d(x)$ and remainder $r(x)$. There is a nice connection between polynomial division and polynomial evaluation.

**Theorem 2.10 (Taylor for polynomials)** *Consider a polynomial $p(x)$ and its Taylor approximation $T_{\alpha_0} p(x; x_0)$ at node $x_0$ of order $\alpha_0$, i.e.*

$$T_{\alpha_0} p(x; x_0) = p(x_0) + p'(x_0)(x - x_0) + \cdots + p^{(\alpha_0)}(x_0) \frac{(x - x_0)^{\alpha_0}}{\alpha_0!}.$$

*$T_{\alpha_0} p(x; x_0)$ is given by $p(x) \bmod (x - x_0)^{\alpha_0 + 1}$.*

**Proof** In the appendix. □

**Theorem 2.11 (Polynomial remainder)** *The value of a polynomial $p(x)$ at $x_0$ is given by*

$$p(x_0) = p(x) \bmod (x - x_0).$$

**Proof** This is a direct consequence of the Taylor theorem for polynomials. Let $r(x) = p(x) \bmod (x - x_0)$, then $p(x_0) = q(x)(x - x_0) + r(x_0) = r(x_0)$. Since $(x - x_0)$ is of degree 1, $r(x)$ must be constant and equal to $p(x_0)$. □

**Lemma 2.12** *Let $p(x)$ and $q(x)$ be a polynomials such that $q(x)$ anihilates a matrix $A$ and let $r(x) = p(x) \bmod q(x)$, then we have*

$$p(A) = r(A).$$

**Proof** This is a direct consequence of the Taylor theorem for polynomials, $p(A) = q(A)d(A) + r(A) = r(A)$. □

Chapter 3

# The Problem

We consider an input matrix $A \in \mathbb{C}^{n \times n}$ with

1. $m$ eigenvalues $\lambda_0, \ldots, \lambda_{m-1}$,
2. characteristic polynomial $\chi_A(x) = \Pi_{i=0}^{m-1}(x - \lambda_i)^{\alpha_i}$,
3. Jordan decomposition $A = VJV^{-1}$.

The goal is to compute the Jordan-Chevalley decomposition of A.

**Theorem 3.1 (Jordan-Chevalley decomposition)** *A matrix $A \in \mathbb{C}^{n \times n}$ can be uniquely decomposed into the sum of a diagonalizable and a nilpotent matrix*

$$A = D + N.$$

*The matrices $D \in \mathbb{C}^{n \times n}$ and $N \in \mathbb{C}^{n \times n}$ satisfy the following properties:*

1. *$D$ is diagonalizable and $N$ nilpotent,*
2. *$D$ and $N$ commute, i.e. $DN = ND$,*
3. *$D$ and $N$ are polynomials in $A$, i.e. $\exists p(x)\ D = p(A)$ and $N = A - p(A)$.*

**Proof** The Jordan normal form of A can be decomposed into the sum of a diagonal and a nilpotent part

$$J = \underbrace{\begin{bmatrix} \ddots & & & & & \\ & \lambda_i & & & & \\ & & \ddots & & & \\ & & & \ddots & & \\ & & & & \lambda_i & \\ & & & & & \ddots \end{bmatrix}}_{\Lambda} + \underbrace{\begin{bmatrix} \ddots & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & 0 \\ & & & & & \ddots \end{bmatrix}}_{\tilde{N}}.$$

Now, we construct the diagonal matrix $D = V\Lambda V^{-1}$ and the nilpotent matrix $N = V\tilde{N}V^{-1}$, which yields

$$A = V(\Lambda + \tilde{N})V^{-1} = D + N.$$

Later, we will show that $D$ and $N$ are polynomials in A Commutivity follows from the commuting property of polynomials:

$$DN = p(A)q(A) = (p \cdot q)(A) = (q \cdot p)(A) = q(A)p(A) = ND. \qquad \square$$

It is not feasible to compute the Jordan-Chevalley decomposition via the Jordan decomposition since the problem of finding the latter is ill-conditioned (see example 2.5). However, it might still be possible via computing the Chevalley polynomial.

**Definition 3.2 (Chevalley polynomial)** *The Chevalley polynomial $chev(x)$ realizes the Jordan-Chevalley decomposition*

$$chev(A) = D.$$

Computing the Jordan-Chevalley decomposition reduces to an either implicit or explicit computation of the Chevalley polynomial. We characterize the Chevalley polynomial as follows [20].

**Lemma 3.3** *For every eigenvalue $\lambda_i$ of $A$, the Chevalley polynomial has a fixpoint at $\lambda_i$ and roots at its derivatives. The Chevalley polynomial is given by the solution of the Hermite interpolation problem: for all eigenvalues $\lambda_i$*

$$\begin{aligned} chev(\lambda_i) &= \lambda_i \\ chev'(\lambda_i) &= 0 \\ &\vdots \\ chev^{(\ell_i - 1)}(\lambda_i) &= 0. \end{aligned}$$

**Proof** Using lemma 2.6 and the diagonalizability criterion, we get

$$\mathrm{chev}(A) = V \underbrace{\begin{bmatrix} \ddots & \ddots & & & & & \\ & \mathrm{chev}(\lambda_i) & \mathrm{chev}'(\lambda_i) & \cdots & \frac{\mathrm{chev}^{(\ell_i-1)}(\lambda_i)}{(\ell_i-1)!} & & \\ & & \ddots & \ddots & \vdots & & \\ & & & \ddots & \mathrm{chev}'(\lambda_i) & & \\ & & & & \mathrm{chev}(\lambda_i) & 0 & \\ & & & & & \ddots & \ddots \\ & & & & & & \ddots \end{bmatrix}}_{\Lambda} V^{-1} = D$$

$\square$

Now, we reformulate the problem as a system of congruences of polynomials, using the polynomial remainder theorem.

**Lemma 3.4** *The Chevalley polynomial of a matrix $A$ is given by the unique solution of the Chinese remainder problem*

$$\begin{aligned} chev(x_0) &\equiv \lambda_0 \qquad (\text{mod } (x-\lambda_0)^{\ell_0}) \\ &\vdots \\ chev(x_{m-1}) &\equiv \lambda_{m-1} \quad (\text{mod } (x-\lambda_{m-1})^{\ell_{m-1}}), \end{aligned}$$

*where $\lambda_i$ are a constant polynomials and $(x-\lambda_i)^{\ell_i}$ are the factors of any annihilating polynomial of $A$.*

**Proof** Since we have $\frac{d^k}{dx^k}\lambda_i = 0$ for all $k > 0$, we can apply theorem 2.10 and observe that for every eigenvalue $\lambda_i$ of $A$, the Chevalley polynomial has a fixpoint at $\lambda_i$ and roots at its derivatives. Now, we apply the argument from the proof of lemma 3.3. □

In this thesis we derive several algorithms to compute the Jordan-Chevalley decomposition, review an implementation and examine its numerical behavior.

Chapter 4

# Algorithms

The original proof of theorem 3.1, provided by Claude Chevalley in [8], shows that a modified Newton iteration on polynomials converges to the Chevalley polynomial. We use such an iteration to find a Jordan-Chevalley decomposition and provide a proof according to the reasoning of lemma 3.3.

The Chinese remainder problem in 3.4 is used as an entry point to derive algorithms for the Chevalley polynomial in [10] [6] and [4]. Some of them are close or equal to the iteration described by Chevalley. In [4] [5] and [3] higher-order iterative methods are derived, ultimately leading to an explicit formula in [7].

Using the Hermite interpolation in lemma 3.3 directly yields an algorithm to calculating the Chevalley polynomial as is described in [20]. The fundamental difference between the Hermite interpolation algorithm and all other approaches is that a computation of the eigenvalues is not required. Instead, a set of distinct eigenvalues is implicitly represented by polynomials.

## 4.1 Hermite interpolation

**Algorithm 4.1:** Hermite interpolation

**input:** matrix $A$ of size $n \times n$

1 **for** $i \leftarrow 0$ **to** $m$ **do**
2 $\quad \lambda_i \leftarrow$ the $i^{th}$ distinct eigenvalue of $A$
3 $\quad \ell_i \leftarrow$ the Jordan index of eigenvalue $\lambda_i$
4 **end**
5 chev$(x) \leftarrow$ the solution to the interpolation problem in lemma 3.3
6 $D \leftarrow chev(A)$, the evaluation of $chev(x)$ on the matrix $A$
7 **return** $D$

**Proof** The correctness follows directly from lemma 3.3. □

## 4.2 Chevalley iteration

One way to turn the proof by Chevalley into an algorithm is by iteration on matrices. This is not very efficient but serves the purpose of explanation well.

**Algorithm 4.2:** Chevalley iteration

**input:** matrix $A$ of size $n \times n$

$\chi_A(x) \leftarrow$ the characteristic polynomial of $A$

$\mu_D(x) \leftarrow \frac{\chi_A(x)}{gcd(\chi_A(x), \chi'_A(x))}$, the minimal polynomial of $D$

$inv(x) \leftarrow$ the inverse of $\mu'_D(x)$ modulo $\mu_D(x)$

$S_0 \leftarrow A$

**while** $S_{k+1} \neq S_k$ **do**

  $S_{k+1} \leftarrow S_k - \mu_D(S_k) \cdot inv(S_k)$

**end**

$D \leftarrow S_\ell$, the converged matrix

**return** $D$

**Proof** Recall that the characteristic polynomial is $\chi_A(x) = \Pi(x - \lambda_i)^{\alpha_i}$ and by definition of the polynomial greatest common devisor, we have

$$\frac{\chi_A(x)}{gcd(\chi_A(x), \chi'_A(x))} = \Pi_{i=0}^{m-1}(x - \lambda_i). \tag{4.1}$$

This is exactly the minimal polynomial of $D$, since D has no Jordan chains. Now $\mu'_D(x)$ and $\mu(D))$ are relatively prime and thus $inv(x)$ is always defined.

We notice that all $S_k$ are implicitly polynomials in $A$ and define the sequence of polynomial $s_k(x)$ such that $S_k = s_k(A)$. The Jordan decomposition of $S_k$ is

$$S_k = s_k(A) = V \begin{bmatrix} \ddots & \ddots & & & & & \\ & s_k(\lambda_i) & s'_k(\lambda_i) & \cdots & \frac{s_k^{(\ell_i-1)}(\lambda_i)}{(\ell_i-1)!} & & \\ & & \ddots & \ddots & \vdots & & \\ & & & \ddots & s'_k(\lambda_i) & & \\ & & & & s_k(\lambda_i) & 0 & \\ & & & & & \ddots & \ddots \\ & & & & & & \ddots \end{bmatrix} V^{-1}.$$

In theorem 4.1 we will show that the $k^{th}$ step of the Chevalley iteration turns the $k^{th}$ superdiagonal into zeros and keeps the $\lambda_i$s on diagonal unchanged. As described in lemma 3.4, $S_k$ will converge to $D$ in $\ell$ steps. □

**Theorem 4.1** *Consider the polynomials $f(x) = \Pi(x - x_i)$, without multiple roots, $u(x)$, such that $f'(x)u(x) = 1 \pmod{f(x)}$, and the recursively defined polynomial $s_{k+1}(x) = s_k(x) - f\big(s_k(x)\big)u\big(s_k(x)\big)$ with $s_0(x) = x$. For every root $x_i$ and every iteration $k > 0$ we have*

$$s_k(x_i) = x_i \quad \textit{and} \tag{4.2}$$

$$s_k^{(j)}(x_i) = 0, \quad \forall j = 1, \dots k. \tag{4.3}$$

**Proof** Recall that $\forall x_i \; f(x_i) = 0 \Rightarrow f'(x_i) \neq 0$.

We first proove 4.2 by induction on $k$.

$\underline{k = 1}$, i.e. we show that $s_1(x_i) = x_i$.

Since $f(x_i) = 0$, we have $s_1(x_i) = x_i - f(x_i)u(x_i) = x_i$

$\underline{k + 1 \leftarrow k}$, i.e. we assume $s_k(x_i) = x_i$ and show $s_{k+1}(x_i) = x_i$.

We have $s_{k+1}(x_i) = s_k(x_i) - f\big(s_k(x_i)\big)u(s_k(x_i)) = s_1\big(s_k(x_i)\big) = x_i$.

Now, we proove 4.3 by induction on $k$.

$\underline{k = 1}$, i.e. we show that $s_1'(x_i) = 0$.

Since $f'(x_i)u(x_i) = 1$, we have $s_1'(x_i) = f(x_i)u'(x_i) = 0$

$\underline{k + 1 \leftarrow k}$, i.e. we assume $s_k^{(j)}(x_i) = 0 \; \forall j = 1, \dots, k$ and show $s_{k+1}^{(k+1)}(x_i) = 0$.

First, we spilt the $k + 1$ into $k$ and 1, this yields

$$\frac{d^{k+1}}{dx^{k+1}} \, s_{k+1}(x) = \frac{d^k}{dx^k}\left(\frac{d}{dx} \, s_1\big(s_k(x)\big)\right) \quad = \frac{d^k}{dx^k}\Big(s_1'\big(s_k(x)\big) \cdot s_k'(x)\Big).$$

Then, we use the general Leibnitz rule and get

$$\frac{d^{k+1}}{dx^{k+1}} \, s_{k+1}(x) = \sum_{j=0}^{k} \binom{k}{j} \left(\frac{d^{k-j}}{dx^{k-j}} \, s_1'\big(s_k(x)\big)\right) \cdot s_k^{(j)}(x).$$

After evaluating at $x_i$ and applying the induction hypothesis, only the first summands remains:

$$\frac{d^{k+1}}{dx^{k+1}} \, s_{k+1}(x) \, \Big|_{x=x_i} = \left(\frac{d^k}{dx^k} \, s_1'\big(s_k(x)\big)\right) \cdot s_k(x) \, \Big|_{x=x_i}$$

Now we use Faà di Bruno's formula, then equation 4.2 and finally apply the induction hypothesis again.

$$s_{k+1}^{(k+1)}(x_i) = \Bigg( \underbrace{\sum_{j=1}^{k} s_1^{(j+1)}\big( \underbrace{s_k(x_i)}_{= x_i \ (3.2)} \big)}_{= 0 \ \forall j>0 \ (I.H.)} \cdot \mathcal{B}_{k,j}\big(s_k'(x_i), \dots\big) \Bigg) \cdot s_k(x_i) = 0 \qquad \square$$

### Correspondance to newton iteration on matrices

Note that $inv(S_k)$ could be replaced by an inverse matrix, which put us into the setting of a Netwon iteration over matrices:

**Algorithm 4.3:** Chevalley iteration on matrices

**input:** matrix $A$ of size $n \times n$

$\chi_A(x) \leftarrow$ the characteristic polynomial of $A$

$\mu_D(x) \leftarrow \frac{\chi_A(x)}{gcd(\chi_A(x), \chi_A'(x))}$, the minimal polynomial of $D$

$S_0 \leftarrow A$

**while** $S_{k+1} \neq S_k$ **do**

  $S_{k+1} \leftarrow S_k - \mu_D(S_k) \cdot \mu_D'(S_k)^{-1}$

**end**

$D \leftarrow S_\ell$, the converged matrix

**return** $D$

### Chevalley iteration on polynomials

The asympotic complexity of algorithm 4.2 is dominated by the computation of matrix polynomials in the loop. This can be avoided by computing the polynomials $s_k(x)$ explicitly, using function composition. To keep the the degree of $s_k(x)$ under control, we reduce $s_k(x)$ modulo $\chi_A(x)$ in each iteration.

**Proof** Correctness follows directly from theorem 4.1 which states that $s_k(x)$ converges towards the Chevalley polynomial $chev(x)$. $\square$

**Algorithm 4.4:** Chevalley iteration on polynomials

**input:** matrix $A$ of size $n \times n$

$\chi_A(x) \leftarrow$ the characteristic polynomial of $A$

$\mu_D(x) \leftarrow \frac{\chi_A(x)}{gcd(\chi_A(x), \chi'_A(x))}$, the minimal polynomial of $D$

$inv(x) \leftarrow$ the inverse of $\mu'_D(x)$ modulo $\mu_D(x)$

$s_0(x) \leftarrow x$, the linear polynomial

**while** $s_{k+1}(x) \neq s_k(x)$ **do**

  $s_{k+1}(x) \leftarrow s_k(x) - \mu_D(s_k(x)) \cdot inv(s_k(x)) \quad (\ \mathrm{mod} \chi_A(x)\ )$

**end**

chev$(x) \leftarrow s_\ell(x)$, the converged polynomial

**return** $chev(A)$

Chapter 5

# Implementation

For integer matrices, the characteristic polynomial $\chi_A(x)$ has integer coefficients. This means that the eigenvalues $\lambda_i$ are algebraic integers, exactly represented by $\chi_A(x)$. Usually, eigenvalues are computed and stored as floating-point approximations on computers. In the special case of integer matrices, however, it is possible to store an $\chi_A(x)$ exact and thus also an exact representation of the eigenvalues.

The Hermite interpolation problem in lemma 3.3 is described by eigenvalues. The Chevalley iteration solves this problem by using polynomials as an exact representation of these eigenvalues. We observe that the coefficients of all polynomials 4.4 are rational numbers and can be represented exactly. This suggests that it is possible to compute the Jordan-Chevalley decomposition exactly for unweighted graph adjacency matrices.

The Hermite interpolation algorithm requires knowledge about the multiplicities of the eigenvalues and is thus only implementable with symbolic computation. We will therefore focus the Chevalley iteration algorithms.

## 5.1 Computing the characteristic polynomial

It is well known that the coefficients of the characteristic polynomial of an $n \times n$ matrix can get very large, the bitsize grows in $\mathcal{O}(n \log n)$. In [13], Dumas provides an upper bound for the bitsize of the characteristic polynomial. If all entries of the matrix are bounded in absolute value by $B$ and $c_i$ are the coefficients of its characteristic polynomial, then

$$\log_2(|c_i|) \leq \frac{n}{2}\big(\log_2(n) + \log_2(B^2) + 0.21163175\big) \quad \forall c_i. \tag{5.1}$$

Consider for example a binary matrix $A$ of size $n > 26$. A 64-bit integer might be to small to store some coefficients of $\chi_A(x)$. This makes it very clear that we need variable precision integers to store the characteristic polynomial.

Algorithms for the computation of potentially sparse integer matrices are described in [18] [14] [22]. We decided to use La Budde's method, a numerical algorithm described in [23], since it is easy to implement and doesn't impose a limit in our use case. We use high-precision floating-point arithmetic and round the coefficients of the characteristic polynomial to the nearest integer. The algorithm runs in $\mathcal{O}(n^3)$, excluding the coefficient grow of $\mathcal{O}(n \log n)$.

## 5.2 Computing the polynomial gcd

Since every constant polynomial is a unit of the ring of polynomials, the polynomial greatest common divisor is only defined up to a constant, i.e.

$$gcd(p(x), q(x)) = c \cdot gcd(p(x), q(x)). \tag{5.2}$$

Usually the resulting polynomial is required to be monic, this applies also to the Chevalley Iteration. Consequently the coefficients of the gcd are not necessarily rationals. We provide an implementation of the Euclidean algorithm to compute the gcd, as described for example by Knuth in [19]. The coefficients are stored exactly in a rational number datatype. It runs in $\mathcal{O}(n^2)$ for constant size coefficient.

From equation 5.2 follows that the gcd can also be defined for polynomials with integer coefficients. An integer polynomial is called primitive when its coefficients are relatively prime, i.e. its representation is minimal. It is easy to modify the Euclidean algorithm to compute the primitive gcd. This algorithm has the drawback that the bitsizes of the intermediate results grow exponentially large. The bitsizes of the results, however, remain surprisingly moderate. The subresultant algorithm by Braun, Traub and Collins [9] [2] computes an integer gcd with moderate intermediate bitsizes:

$$\log_2(|c_i|) \leq n(\log_2(n) + \log_2(B)) \quad \forall c_i. \tag{5.3}$$

We implemented the subresultant algorithm as described by Knuth [19].

## 5.3 Computing the modulo inverse

The extended Euclidean algorithm computes a Bézout identity

$$p(x)u(x) + q(x)v(x) = gcd(p(x), q(x)).$$

If $p(x)$ and $q(x)$ are relatively prime, then $u(x)$ is the inverse of $p(x)$ mod $q(x)$. We did not find any literature about the bitsizes of $u(x)$ or an extended version of the subresultant algorithm.

We implemented the Euclidean algorithm with a rational number datatype. The implementation of this datatype always chooses a minimal representation by reducing the fractions. So if bitsizes of the intermediate results

stay moderate, then this should reflect in the asymptotic behavior of our implementation.

## 5.4 How to evaluate polynomials

The Horner scheme states that a polynomial can be rewritten as

$$\begin{aligned} p(x) &= p_n x^n + p_{n-1}x^{n-1} + \cdots + p_1 x^1 + p_0 x^0 \\ &= \Big(\Big(\big(\mathbf{0} + p_n x^0\big)x + p_{n-1}x^0\Big) + \cdots + p_1 x^0\Big)x + p_0 x^0, \end{aligned}$$

where $\mathbf{0}$ is the additive identity and $x^0$ is the multiplicative identity. This allows us two evaluate a polynomial with $n$ additions and $n$ multiplications.

We implemented polynomial evaluation for matrices using the Horner scheme, which runs in $\mathcal{O}(n^4)$. Note that there are more efficient algorithms, for example the Paterson-Stockmeyer method [21].

We also used the Horner scheme to implement function composition of polynomials. In this case, $\mathbf{0}$ and $x^0$ are the constant polynomials 0 respectively 1. Since we only use the Chevalley polynomial to evaluate it on the input matrix $A$, we can reduce it modulo $\chi_A(x)$ after every polynomial multiplication. This yields an asymptotic complexity of $\mathcal{O}(n^3)$.

## 5.5 Computing the Chevalley iteration

As described in the previous chapter, there are two variations of the Chevalley iteration. We also provide an implementation for both algorithms. For the iteration on matrices, however, we use the inverse of the matrix and don't compute the inverse polynomial.

Combining the asymptotic complexities of the subroutines yields the asymptotic complexities of the two Chevalley iteration algorithms.

| **Algorithm** | **Runtime** | **Bitsize** |
|---|---|---|
| Computing the characteristic polynomial $\chi_A(x)$ | $\mathcal{O}(n^3)$ | $\mathcal{O}(n \log n)$ |
| Computing the minimal polynomial $\mu_D(x)$ | $\mathcal{O}(n^2)$ | $\mathcal{O}(n \log n)$ |
| Computing the modulo inverse $inv(x)$ | $\mathcal{O}(n^2)$ | unknown |
| Chevalley iteration on matrices $S_k$ | $\mathcal{O}(n^5)$ | unknown |
| Chevalley iteration on polynomials $s_k(x)$ | $\mathcal{O}(n^4)$ | unknown |

We observe that the iteration on polynomials is more efficient if we neglect the coefficient growth. But since we do not need to compute the $inv(x)$ for the iteration on matrices, it could be possible that it is more efficient overall.

The asymptotic complexity of two Chevalley iteration algorithms depends on the bitsize of the intermediate results. These bitsizes are currently unknown. We analyze the asymptotic growth of the bitsize in the next chapter.

Chapter 6

# Evaluation

In this chapter we evaluate our implementation empirically on Euler at ETH.

## 6.1 Implementation of the characteristic polynomial

Recall that we compute the characteristic polynomial numerically using high-precision floating-point arithmetic and then retrieve the correct coefficients by rounding to the next integer. This means that we have to commit to a fixed precision before we compute $\chi_A(x)$. But for a fixed precision there is an upper limit on the size $n$ for which we can compute $\chi_A(x)$ exactly. So up to which size does our implementation compute the characteristic polynomial correctly?

### Tightness of Dumas bound

Let us start by analyzing the tightness of the Dumas bound empirically. For a fixed size $n = 45$ we randomly generate $N = 1000$ Bernoulli matrices $A$ with probability $p = 0.5$. Then we compute $\chi_A(x)$ and determine the bitsize of its largest coefficient. In figure 6.1 we show a histogram of the largest coefficients of $\chi_A(x)$. The Dumas bound at $n = 45$ predicts that the largest coefficient is 128 bits wide. We observe that in practice, the bitsize does not exceed 60. Thus, the Dumas bound is not very tight at $n = 45$.

In figure 6.2, we generate Bernoulli matrices for increasing sizes $n$, compute $\chi_A(x)$ and compare the bitsize of its largest coefficient to the Dumas bound. We observe that the Dumas bound diverges from the measured bitsize with increasing size. The Dumas bound is generally not very tight which gives us more flexibility to compute $\chi_A(x)$ exactly. 128-bit quadruple floating-point has 113-bit precision. This means that the all integers up to $2^{133}$ are represented exactly. According to our measurements it should be possible to store the characteristic polynomial of $n \approx 75$ exactly.

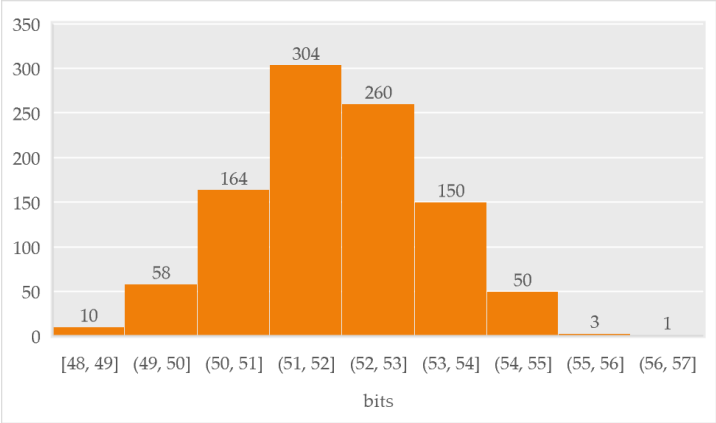



Figure 6.1: Histogram of the bitsizes of the largest coefficients for $\chi_{A_p}(x)$ with $n = 45$. The Dumas bound predicts $\log_2(|c_i|) < 128$. ($N = 1000$ samples)

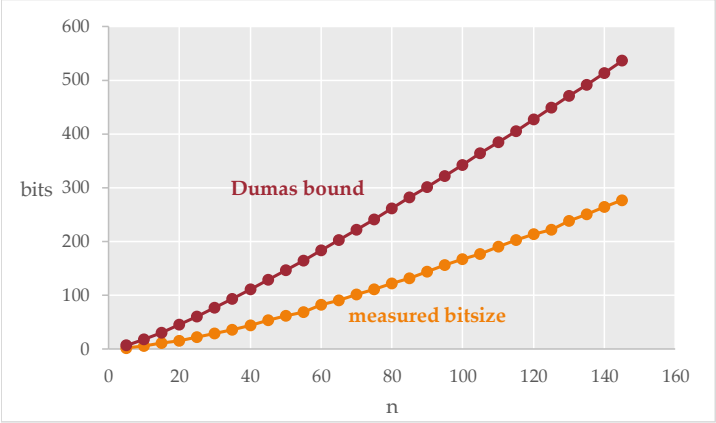



Figure 6.2: Comparison of the Dumas bound with measured bitsizes.

### Cayley-Hamilton criterion

In figure 6.3 we use 64-bit, 128-bit and 256-bit floating point precision to compute $\chi_A(x)$ on Bernoulli matrices. To access the correctness of our algorithm, we check if the Cayley-Hamilton theorem (thm. 2.7) holds. We evaluate $\chi_A(x)$ on $A$ and measure the squared Frobenius norm $\|\chi_A(A)\|_F^2$. We observe that there is no error up to a certain matrix size:

| | |
|---|---|
| 64-bit double: | exact up to size $n < 45$ |
| 128-bit quadruple: | exact up to size $n < 75$ |
| 256-bit octuple: | exact up to size $n < 130$ |

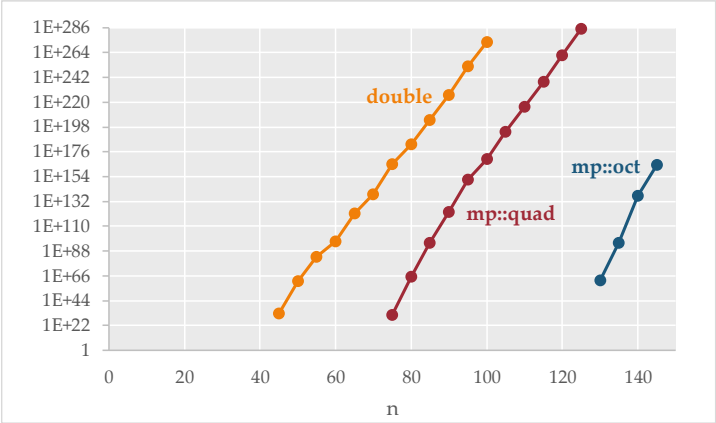



Figure 6.3: Measurements of the Cayley-Hamilton error $\|\chi_A(A)\|_F^2$ for 64-bit, 128-bit and 256-bit floating point precision.

## 6.2 Constructed Jordan matrices

On first thought, it might seem easy to construct a Jordan normal form to test the Chevalley iteration on matrices. However, it is rather difficult to construct a class of Jordan matrices that generate complicated intermediate polynomials. This is due to the fact that the Chevalley iteration acts directly on the Jordan normal form and makes use of its properties.

For example the Chevalley iteration of a Jordan block $J_{\lambda,\ell}$ will require a complex characteristic polynomial $\chi_A = (x-\lambda)^\ell$, but the minimal polynomial will be very simple $\mu_D(x) = x - \lambda$. So to enforce a complicated minimal polynomial, we need different eigenvalues. But choosing random eigenvalues seems to reduce comparability since the spectral radius and the condition number will vary a lot.

For better comparability, we construct the class of non-diagonalizable matrices as follows:

$$\begin{bmatrix} 0 & 1 & & & & & \\ & 0 & 0 & & & & \\ & & 1 & 1 & & & \\ & & & 1 & 0 & & \\ & & & & 2 & 1 & \\ & & & & & 2 & \ddots \\ & & & & & & \ddots \end{bmatrix}.$$

Note that all Jordan chains are of length 2 and have different eigenvalues. We will now examine the behavior of the Chevalley iteration on this class of

non-diagonalizable matrices.

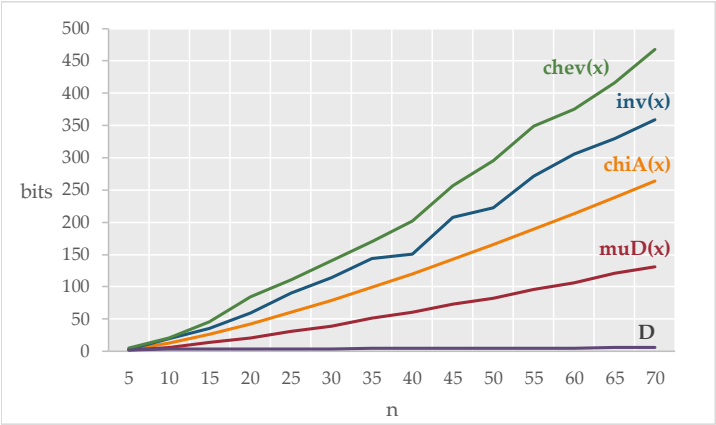



Figure 6.4: Growth of the bitsizes of the intermediate polynomials with constructed Jordan matrices as inputs.

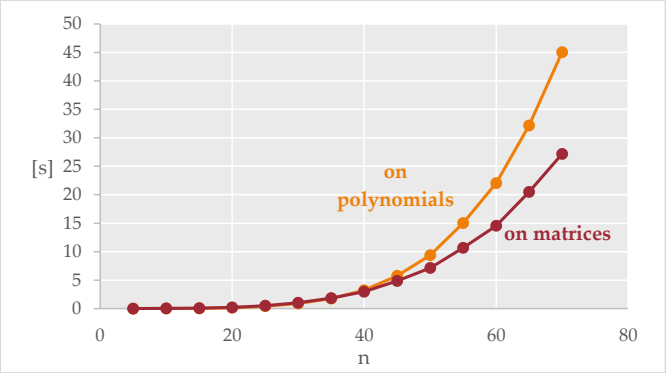



Figure 6.5: Runtime of the Chevalley iteration algorithms with constructed Jordan matrices as inputs.

In figure 6.4 we observe that the growth of the bitsizes of all intermediate polynomials is slightly superlinear, probably $\mathcal{O}(n \log n)$. This suggests that that both Chevalley iteration algorithms run in polynomial time on this special class of matrices.

Indeed, we observe in figure 6.5 that both algorithms compute the Jordan-Chevalley decomposition in polynomial time. The Chevalley iteration on matrices has a better asymptotic complexity since the intermediate polynomials with the largest bitsizes $chev(x)$ and $inv(x)$ are not required.

## 6.3 Bernoulli matrices

Now we want to tests our implementation on less synthetic matrices, for example random ones. The difficulty is that the majority of matrices is diagonalizable. Recall that it is costly to check whether a matrix is diagonalizable. It is thus unreasonable to generate random matrices and tests our algorithm only on the non-diagonalizable ones.

One way to generate random binary matrices is to fix a probability $p$ and then for each entry draw an independent sample from $\{0, 1\}$ where the probability of drawing 1 is $p$. We call such a matrix a Bernoulli matrix.

Randomly generated Bernoulli matrices are often diagonalizable. If we choose $p = 0.5$ and generate $N = 100$ matrices of size $n = 10$, then roughly 5% will be non-diagonalizable. The ratio decreases with larger sizes. In figure 6.6 we plot the ratio of non-diagonalizable matrices for increasing matrix size $n$ and different Bernoulli parameters $p$.

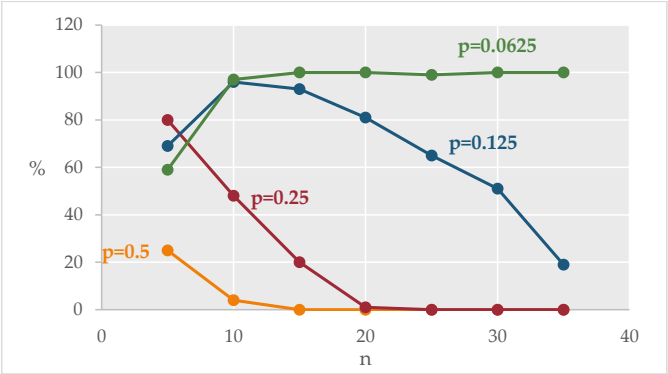



Figure 6.6: Ratio of non-diagonalizable Bernoulli matrices for increasing sizes $n$ and different parameters $p$ (with $N = 100$ samples).

We observe two effects. First, for increasing matrix sizes there are generally more diagonalizable matrices. Secondly, for decreasing $p$ and thus higher sparsity, there are more non-diagonalizable matrices. Note that very sparse matrices are usually nilpotent. This means that the Jordan-Chevalley decomposition is $D = 0$. Our tests for figure 6.6 show that with $p = 0.0625$, most of the matrices are are neither diagonalizable nor nilpotent.

We now examine the Chevalley iteration algorithms on two extremes of Bernoulli matrices. By choosing $p \in \{0.5, 0.0625\}$ we isolate two random classes of diagonalizable respectively non-diagonalizable matrices.

### Diagonalizable Bernoulli matrices

Let us first consider Bernoulli matrices with parameter $p = 0.5$. In figure 6.7 we observe that $\chi_A(x)$ and $\mu_D(x)$ require the same amount of bits and that the Chevalley polynomial requires 1 bit. These are all signs that the tested matrices were diagonalizable. Furthermore, we observe that steep curve of the bitsize of $inv(x)$ implies also a steep cure for the runtime of the iteration on polynomials, as seen in figure 6.8.

This means that the iteration on matrices is more efficient in determining that an input matrix is in fact diagonalizable.

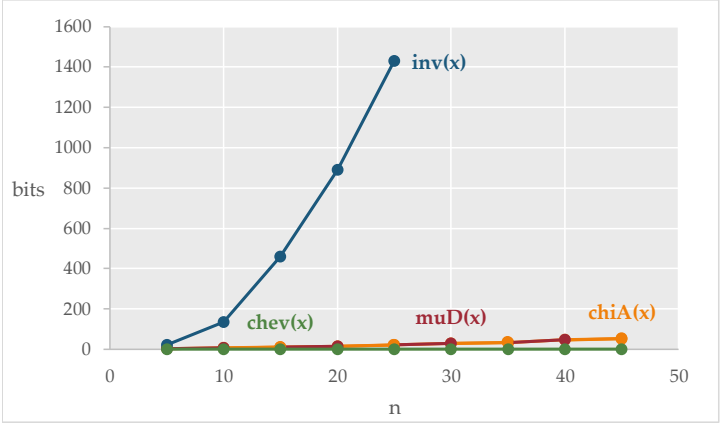



Figure 6.7: Growth of the bitsizes of the intermediate polynomials with Bernoulli matrices as inputs ($p = 0.5$).

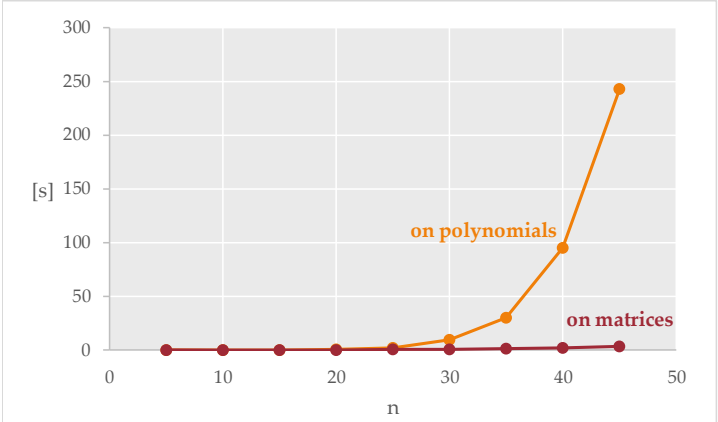



Figure 6.8: Runtime of the Chevalley iteration algorithms with Bernoulli matrices as inputs $p = 0.5$.

### Non-diagonalizable Bernoulli matrices

Now, we consider Bernoulli matrices with parameter $p = 0.0625$. In figure 6.9 we observe the bitsizes of the intermediate polynomials are more complicated, since the input matrices are not diagonalizable. The bitsize of $inv(x)$ still grows faster then the other polynomials and thus dominates the asymptotic complexity of the iteration on polynomials.

As expected, the iteration on matrices is slower for non-diagonalizable matrices than for diagonalizable ones.

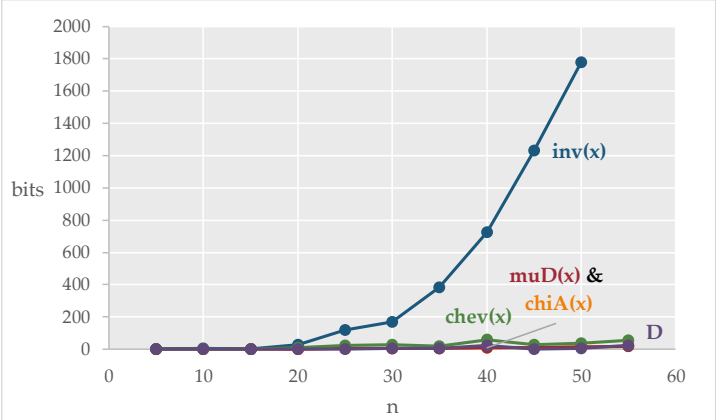



Figure 6.9: Growth of the bitsizes of the intermediate polynomials with Bernoulli matrices as inputs ($p = 0.0625$).

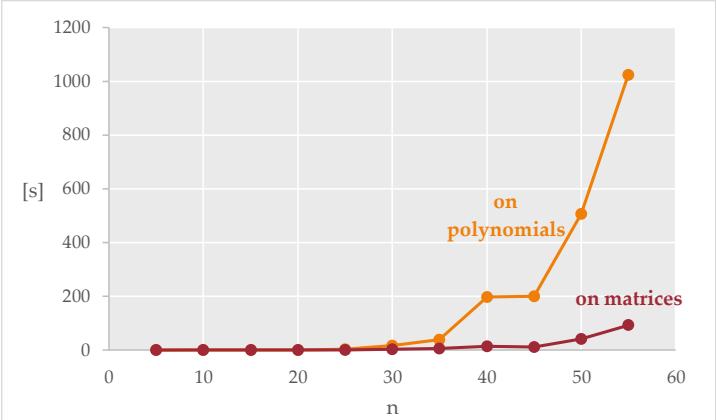



Figure 6.10: Runtime of the Chevalley iteration algorithms with Bernoulli matrices as inputs $p = 0.0625$.

## 6.4 Kronecker graphs

In the previous sections we have seen that our implementation of the Chevalley iteration on matrices is faster on all test matrices. Now we use this implementation to analyze the Jordan-Chevalley decomposition of statistical Kronecker graphs, as described [15].

To get a general idea of the Jordan structure of such adjacency matrices, we generate $N = 100$ matrices for the sizes $n = 4, 8, \ldots, 128$ starting with the initial probability configuration

$$\begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}.$$

In figure 6.11 we show the ratio of non-diagonalizable matrices for increasing sizes $n$. We observe that all matrices of size $n > 32$ are not diagonalizable.

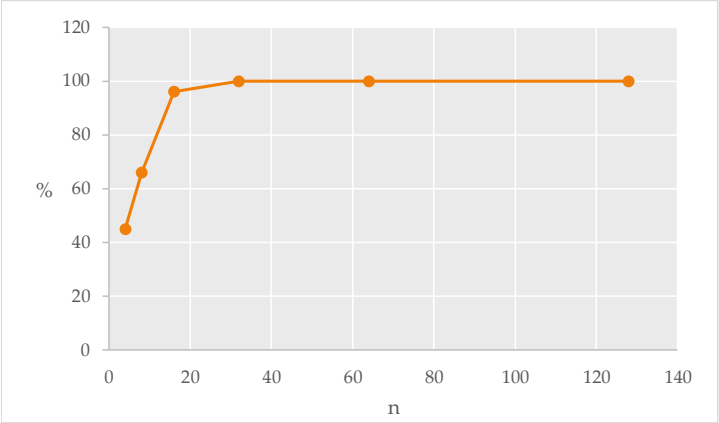



Figure 6.11: Ratio of non-diagonalizable Kronecker graph adjacency matrices for increasing sizes $n$ (with $N = 1000$ samples).

Now, we examine the behavior of the Chevalley iteration on Kronecker graph adjacency matrices of size $n = 2, 8, \ldots, 512$.

For the computation of the characteristic polynomial we used the software floating point type *mp::1000* provided by the Boost library. It has a mantissa of 3324 binary digits, which should allow us to compute matrices up to size $n = 1260$.

To be sure that the computed Jordan-Chevalley decomposition is correct, we successfully checked that $D$ and $N$ commute and that $N$ is nilpotent.

In figure 6.12 we observe that all bitsizes grow roughly linearly. Notice that the bitsizes of entries of the diagonalizable matrix $D$ are moderate even for large input matrices $A$.

Assuming, that the bitsizes of $muD(x)$ are in $\mathcal{O}(1)$ we derive that the Chevalley iteration on polynomials runs in $\mathcal{O}(n^5)$ on Kronecker Graph adjacency matrices. In figure 6.13 we computed the Jordan-Chevalley decomposition of a Kronecker graph with 1024 nodes in 19 hours and 20 minutes.

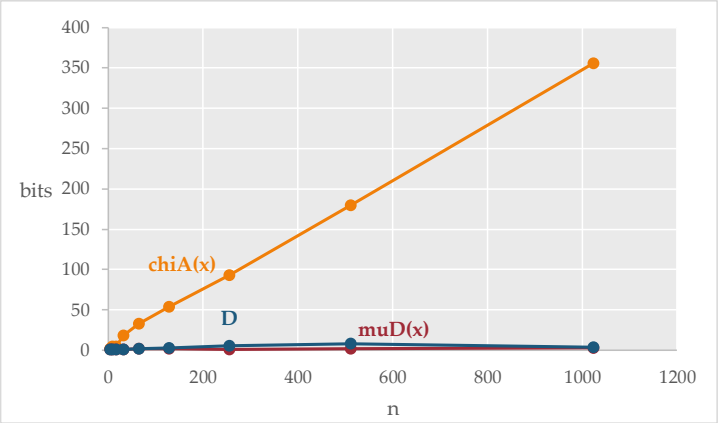



Figure 6.12: Growth of the bitsizes of the intermediate polynomials with Kronecker graph adjacency matrices as inputs.

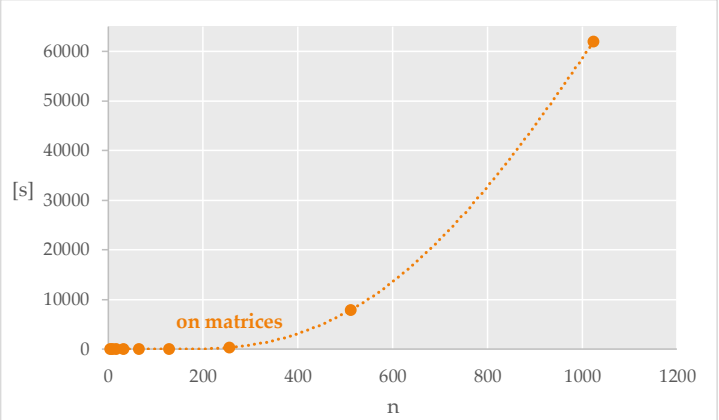



Figure 6.13: Runtime of the Chevalley iteration on matrices with Kronecker graph adjacency matrices as inputs.

Chapter 7

# Discussion

As a main result of the empirical evaluation, the polynomials $inv(x)$ and $chev(x)$ exhibited significant coefficient growth for all classes of test matrices. The *Chevalley iteration on matrices* does not require the computation of those polynomials and is therefore significantly faster than the *Chevalley iteration on polynomials*. In practical terms, we can compute the diagonalizable matrix $D$ more efficiently than the Chevalley polynomial $chev(x)$.

In the following we list an outlook on topics for further research:

**Asymptotic complexity** In the previous chapter we saw that the bit sizes of intermediate results did not grow exponentially. This suggests that both Chevalley iteration algorithms run in polynomial time. Certainly a mathematical proof of this evidence would be desirable. In particular, the Cramer rule might be helpful to prove asymptotic complexity for the *Chevalley iteration on matrices*.

**Floating-point implementation** The use of variable precision integers has the benefit that all computations are exact even for large coefficients. However, it introduces a significant computational overhead. A floating-point implementation would allow to speed up the computation dramatically while it is unclear whether it would suffer from numerical instabilities.

**Minimal polynomial and sparsity** In both Chevalley iteration algorithms, we used the characteristic polynomial as an entry point although they also work with the minimal polynomial. In [13], there is an upper bound for the coefficients of the minimal polynomial which depends on the spectral radius. The spectral radius is bound by the sparsity of the matrix, which is exactly the number of edges of the corresponding graph. Using the minimal polynomial therefore gives a more fine grained asymptotic time complexity.

Chapter 8

# Conclusions

We have shown both theoretically and empirically that it is possible to compute the Jordan-Chevalley decomposition of integer matrices without symbolic computation. The asymptotic runtime complexity depends on the size of intermediate results, which appears to be polynomial. In empirical evaluation, we can compute the diagonalizable matrix $D$ more efficiently than the Chevalley polynomial $chev(x)$.

Thereby one can simplify the Jordan structure of a matrix without knowing the structure itself. This can be used to compute the eigenvalues and generalized eigenvectors of a defective integer matrix. As a possible application, this is useful to compute a subclass of diagonalizable filters in Graph Signal Processing.

Chapter 9

# Acknowledgments

I thank Prof. Püschel and Chris Wendler for proposing the topic of the Bachelor thesis and supervising my work.

I thank Patrick Ziegler and Lukas Looser for intense discussions on several aspects of the topic.

I thank Camillo de Nardis for cooking for me during the final days of the work that took place in Corona quarantine.

I thank Tobias Looser, Laure Ciernik and Johannes Sarnthein for comments on earlier versions of the manuscript.

# Proof of Taylor theorem for polynomials

**Proof (Theorem 2.10)** Consider the polynomial division $p(x) = q(x)d(x) + r(x)$. We choose $d(x) = (x - x_0)^{\alpha_0+1}$ and show that $r(x)$ is the Taylor approximation $T_{\alpha_0}p(x; x_0)$.

The $k^{th}$ derivative of $p(x)$ is given by the general Leibnitz rule

$$p^{(k)}(x) = \sum_{j=0}^{k} \binom{k}{j} q^{(k-j)}(x) \cdot d^{(j)}(x) + r^{(k)}(x).$$

Since $d^{(j)}(x_0) = 0$ for all $j \leq \alpha_0$, the sum vanishes for $k \leq \alpha_i$ and we get

$$\begin{aligned} p(x_0) &= r(x_0) \\ &\vdots \\ p^{(\alpha_0)}(x_0) &= r^{(\alpha_0)}(x_0). \end{aligned}$$

Since these are exactly the properties of the Taylor approximation, we get $T_{\alpha_0}p(x; x_0) = r(x) := p(x) \bmod d(x)$. □

# Bibliography


[1] Theo Beelen and Paul Van Dooren. Computational aspects of the Jordan canonical form. Technical report.

[2] W S Brown and J F Traub. On Euclid's Algorithm and the Theory of Subresultants. Technical report, 1971.

[3] Jean-François Burnol. Décomposition de Jordan-Chevalley-Dunford effective en temps linéaire. Technical report, 2017.

[4] Jean-François Burnol. Décomposition de Jordan-Chevalley-Dunford et itérations de Newton-Halley-Householder. Technical report, 2017.

[5] Jean-François Burnol. Itération de Newton-Schröder-Householder et nouveau nouvel ( ?) algorithme pour la décomposition de Jordan-Chevalley-Dunford effective. Technical report, 2017.

[6] Jean-François Burnol. Un (nouvel ?) algorithme pour la décomposition de Dunford effective. Technical report, 2017.

[7] Jean-François Burnol. Une formule explicite réalisant la décomposition de Jordan-Chevalley-Dunford effective. Technical report, 2017.

[8] Claude Chevalley. *Théorie des groupes de Lie. Tome II, Tome II,*. Hermann, Paris, 1951.

[9] George E Collins. Subresultants and Reduced Polynomial Remainder Sequences. Technical report.

[10] Danielle Couty, Jean Esterle, and Rachid Zarouf. Décomposition effective de Jordan-Chevalley et ses retombées en enseignement. 3 2011.

[11] Daniel Ferrand. Une méthode effective pour la décomposition de Dunford. Technical report, 2003.

[12] Joya A. Deri and Jose M.F. Moura. Spectral Projector-Based Graph Fourier Transforms. *IEEE Journal on Selected Topics in Signal Processing*, 11(6):785–795, 9 2017.

[13] Jean-Guillaume Dumas and Jean Kuntzmann. Bounds on the Coefficients of the Characteristic and Minimal Polynomials. Technical report, 2007.

[14] Jean Guillaume Dumas, Clément Pernet, and Zhendong Wan. Efficient computation of the characteristic polynomial. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation, ISSAC*, volume 2005, pages 140–147, 2005.

[15] Jure@cs Stanford Edu, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin@eng Cam Ac Uk. Kronecker Graphs: An Approach to Modeling Networks Jure Leskovec Zoubin Ghahramani. Technical report, 2010.

[16] G H Golub and J H Wilkinson. Ill-conditioned Eigensystems and the Computation of the Jordan Canonical Form. Technical Report 4, 1976.

[17] Gaël Guennebaud, Benoît Jacob, and others. Eigen v3. http://eigen.tuxfamily.org, 2010.

[18] Walter Keller-Gehrig. Fast algorithms for the characteristics polynomial. *Theoretical Computer Science*, 36(C):309–317, 1985.

[19] Donald Ervin Knuth. Volume 2: Seminumerical Algorithms. In *The Art of Computer Programming*, page 216. 1998.

[20] Panagiotis Misiakos, Chris Wendler, and Markus Püschel. Diagonalizable Shift and Filters for Directed Graphs Based on the Jordan-Chevalley Decomposition. 2020.

[21] Michael S. Paterson and Larry J. Stockmeyer. On the Number of Nonscalar Multiplications Necessary to Evaluate Polynomials. *SIAM Journal on Computing*, 2(1):60–66, 3 1973.

[22] Clément Pernet and Arne Storjohann. Faster algorithms for the characteristic polynomial. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation, ISSAC*, pages 307–314, 2007.

[23] Rizwana Rehman and Ilse C F Ipsen. La Budde's Method for Computing Characteristic Polynomials. Technical report, 2011.

[24] Alaeddine Ben Rhouma. Autour de la déecomposition de Dunford réelle ou complexe. Théorie spectrale et méthodes effectives. 7 2013.

[25] Aliaksei Sandryhaila and José M.F. Moura. Discrete signal processing on graphs. *IEEE Transactions on Signal Processing*, 61(7):1644–1656, 4 2013.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

## Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

---

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

**Title of work** (in block letters):

Jordan Chevalley Decomposition
Implementation and Numerical Analysis

**Authored by** (in block letters):

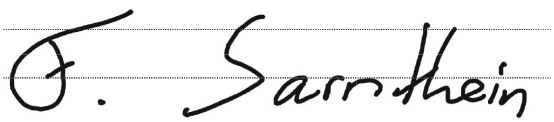
*For papers written by groups the names of all authors are required.*

| **Name(s):** | **First name(s):** |
|---|---|
| Sarnthein-Lotichius | Felix |
| | |
| | |
| | |
| | |

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

| **Place, date** | **Signature(s)** |
|---|---|
| Maloja, 07.04.2020 | |
| |  |
| | |

*For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.*