ELSEVIER Linear Algebra and its Applications 314 (2000) 75–89
LINEAR ALGEBRA AND ITS APPLICATIONS
www.elsevier.com/locate/laa

Construction of the Jordan decomposition by means of Newton's method

Dieter Schmidt

Fachbereich 6, Mathematik und Informatik, Universität GH Essen, 45117 Essen, Germany

Received 10 August 1998; accepted 13 March 2000

Submitted by H. Schneider

Abstract

Newton's method is applied to construct the semi-simple part of the Jordan decomposition of an algebraic element in an arbitrary algebra and to derive an efficient algorithm for its computation. Applications on the matrix case and on differential operators are discussed.

AMS classification: 15A21; 11Cxx; 34A20; 65Fxx

Keywords: Polynomial algorithms; Normal forms of operators; Meromorphic differential operators

1. Introduction

Let $\mathbb{K}$ be an arbitrary field and $\mathscr{A}$ an algebra over $\mathbb{K}$ with unit 1. Subject of our interest is the following well-known result on the Jordan decomposition, e.g., see [1,5,13].

Theorem 1.1. *If $\mathbb{K}$ is a perfect field, then, for each algebraic $A \in \mathscr{A}$, there exist unique $S, N \in \mathbb{K}[A]$ such that $A = S + N$, S is semi-simple and N is nilpotent.*

We recall that a field $\mathbb{K}$ is *perfect* if and only if $\operatorname{char} \mathbb{K} =: k > 0$ implies that the function $\mathbb{K} \ni z \to z^k \in \mathbb{K}$ is surjective. In particular, by definition, every field $\mathbb{K}$

E-mail address: dieter.schmidt@uni-essen.de (D. Schmidt).

0024-3795/00/$ - see front matter
PII: S 0 0 2 4 - 3 7 9 5 (0 0) 0 0 1 1 1 - 7

with char $\mathbb{K} = 0$ and thus every subfield of $\mathbb{C}$ is perfect. Obviously, every algebraically closed field is perfect.

An element $A \in \mathscr{A}$ is called *algebraic* if there exists a polynomial $p \in \mathbb{K}[x]\backslash\{0\}$ such that $p(A) = 0$; specifically, $A \in \mathscr{A}$ is called *semi-simple* if there exists a square-free $p \in \mathbb{K}[x]\backslash\{0\}$ such that $p(A) = 0$; finally, $A \in \mathscr{A}$ is called *nilpotent* if there exists an integer $\ell \in \mathbb{N}^*$ such that $A^\ell = 0$. A polynomial $p \in \mathbb{K}[x]$ is called *square-free* if $\gcd\{p, p'\} = 1$.

We remark that in the case when $\mathbb{K}$ is algebraically closed and $\mathscr{A}$ is a subalgebra of the matrix-algebra $\mathbb{K}^{n\times n}$, then S is semi-simple if and only if S is diagonalizable.

Since Theorem 1.1 has various applications, it is important to have efficient algorithms for the computation of S in terms of A. The proof of the existence of S as presented in the book of Hoffman and Kunze [5] is constructive and yields direct methods for computations. An algorithm, which is essentially based on these ideas, is given by Levelt [10]. Unfortunately, it is—as the author remarks—rather slow. The algorithm of Bourgoyne and Cushman [2] is faster, because higher derivatives are used. An analysis of the proof of Hoffman and Kunze shows that it is related to Newton's method, but their 'Ansatz' for the solution prevents to obtain a convergence rate as known from Newton's algorithm. The same holds also for [2,10].

The main goal of this article is to show that Newton's method can be applied directly in the present situation and that the corresponding algorithm has a well-known good convergence rate, namely, quadratic convergence. The results regarding this matter are to be found in Proposition 2.1, yielding a new and short proof of the existence of the Jordan decomposition and moreover a simple algorithm to construct it. Techniques which use Newton's method in similar situations are common, e.g., see [3,11]. A further important goal of the article is to make the obtained Newton-type algorithm efficient for calculations on a computer. This is achieved by reducing the degree of the polynomials occurring in the iteration process in an optimal manner. The result is our Theorem 2.2. Applications of this theorem on the matrix case and on singular differential operators are treated in a separate section. Examples prove the efficiency of our method.

The uniqueness statement of Theorem 1.1 is valid even in the case of an arbitrary field $\mathbb{K}$ and follows rather directly, e.g., [5.13]. In the following, only the questions of existence and construction will be treated. For that we consider an arbitrary $p \in \mathbb{K}[x]\backslash\{0\}$ with $p(A) = 0$. Following [10], we pose herewith:

Problem 1.2. Find $q, s \in \mathbb{K}[x]$ such that q square-free, $q^\ell = 0 \bmod p$ for a sufficiently large $\ell \in \mathbb{N}^*$, $x = s \bmod q$ and $q(s) = 0 \bmod p$.

If such polynomials q, s are found, we can put $S := s(A)$, $N := A - S$ and then have $q(S) = 0$, $N^\ell = 0$, which means S semi-simple and N nilpotent. This shows, in particular, that the general problem considered in Theorem 1.1 can be treated completely on the level of polynomials. (For further details see [1,5,10,13].)

On the other hand, Problem 1.2 can be interpreted in the frame of the quotient-algebra $\mathbb{K}[x]/p\,\mathbb{K}[x]$, where it just means to find the Jordan decomposition for the special element $x + p\,\mathbb{K}[x]$. Every s according to Problem 1.2 yields $s + p\,\mathbb{K}[x]$ as the semi-simple part of the Jordan decomposition of $x + p\,\mathbb{K}[x]$. By the uniqueness of the Jordan decomposition follows then, that there exists a unique $s_p \in \mathbb{K}[x]$ with $\mathrm{degree}(s_p) < \mathrm{degree}(p)$ such that $s_p + p\,\mathbb{K}[x] = s + p\,\mathbb{K}[x]$. s_p is the remainder of s divided by p.

Problem 1.2 can be treated in two steps: first find for a given p an appropriate q and afterwards determine s in dependence of this q.

For discussing the first step, we consider an arbitrary non-constant $p \in \mathbb{K}[x]$. We can assume that p is monic, which means that the leading coefficient is 1. By the unique factorisation theorem

$$p = \prod_{q\in\sigma} q^{\mu(q)}, \tag{1.1}$$

where σ is a finite subset of the monic prime polynomials and $\mu : \sigma \to \mathbb{N}^*$. Defining

$$\hat{p} := \prod_{q\in\sigma} q, \qquad \ell_p := \max_{q\in\sigma}\ \mu(q), \tag{1.2}$$

we obviously have $\hat{p}^{\ell_p} = 0 \bmod p$. $\hat{p}$ will be square-free if and only if every $q \in \sigma$ is square-free. However, a prime polynomial q is square-free if and only if $q' \neq 0$, which in particular holds true when $\mathbb{K}$ is a perfect field. In the case of $\mathrm{char}\,\mathbb{K} = 0$ this follows immediately. In this case we also have the formula

$$\hat{p} = p/\gcd\{p, p'\}. \tag{1.3}$$

The second step in solving Problem 1.2 is the central subject of Section 2. In [10] this step is settled with the following:

Lemma 1.3. *Let $q \in \mathbb{K}[x]$ be square-free and $\ell \in \mathbb{N}^*$. Then there exists $s \in \mathbb{K}[x]$ such that $x = s \bmod q$ and $q(s) = 0 \bmod q^{\ell}$.*

Our investigations using Newton's method also yield a new proof of this lemma.

2. Newton's algorithm for computing the Jordan splitting

Let $\mathbb{K}$ be an arbitrary field. We consider in the following a fixed non-constant monic $p \in \mathbb{K}[x]$ and assume that $\hat{p}$ is square-free. Then $\hat{p}^{\ell_p} = 0 \bmod p$.

According to Section 1, we are interested in solving the equation

$$\hat{p}(u) = 0 \bmod p \quad \text{for} \quad u \in M := x + \hat{p}\,\mathbb{K}[x]. \tag{2.1}$$

Every solution s of (2.1) yields s_p as the remainder of s divided by p.

Since $\gcd\{\hat{p}, \hat{p}'\} = 1$, there exist unique $a, b \in \mathbb{K}[x]$ with $\mathrm{degree}(b) < \mathrm{degree}(\hat{p})$ such that

$$1 = a\hat{p} + b\hat{p}'. \tag{2.2}$$

Herewith we introduce the operator

$$\Phi : \mathbb{K}[x] \ni u \to u - b(u)\,\hat{p}(u) \in \mathbb{K}[x]. \tag{2.3}$$

Since we have $b(u)\,\hat{p}'(u) = 1 \bmod \hat{p}(u)$ by (2.2), this is the correct operator to formulate Newton's algorithm for Eq. (2.1). We discuss the mapping properties of Φ.

For arbitrary $u, h \in \mathbb{K}[x]$ there obviously exists a unique polynomial $\tilde{p}[u, h]$ such that

$$\hat{p}(u + h) = \hat{p}(u) + h\,\hat{p}'(u) + h^2\,\tilde{p}[u, h]. \tag{2.4}$$

Inserting here $h = -b(u)\,\hat{p}(u)$ and then using (2.2) yields

$$\hat{p}(\Phi(u)) = \hat{p}(u)^2\,\Psi(u) \tag{2.5}$$

with

$$\Psi(u) := a(u) + b(u)^2\,\tilde{p}[u, -b(u)\,\hat{p}(u)] \in \mathbb{K}[x].$$

Applying (2.4) further with $u = x$ and $h \in \hat{p}\,\mathbb{K}[x]$ yields

$$\hat{p}(u) \in \hat{p}\,\mathbb{K}[x] \quad (u \in M). \tag{2.6}$$

With the definition of Φ then follows at once

$$\Phi(M) \subset M. \tag{2.7}$$

We now consider an arbitrary $u \in M$. Then by (2.7)

$$u_n := \Phi^n(u) \in M \quad (n \in \mathbb{N}), \tag{2.8}$$

and further by (2.6) and (2.5)

$$\hat{p}(u_n) = 0 \bmod \hat{p}^{2^n} \quad (n \in \mathbb{N}). \tag{2.9}$$

Here choosing $n = k_p$, where $k_p \in \mathbb{N}$ is uniquely determined by

$$2^{k_p - 1} < \ell_p \leqslant 2^{k_p}, \tag{2.10}$$

yields:

Proposition 2.1. *$\Phi^{k_p}(u)$ is a solution of* (2.1) *for every $u \in M$, in particular, for $u = x$.*

Thus, we have found solutions of our problem by Newton's algorithm, which moreover has the expected convergence rate. Unfortunately, the degrees of the polynomials u_n in (2.8) will in general become extremly large when n increases, which brings about that the computation of $\Phi(u_n)$ will take a very long time. This makes the above procedure unpracticable. In the following, we describe a modification of the procedure where the degree of the polynomials within the iteration process is limited by the degree of p.

We define, for $n \in \mathbb{N}$,

$$\varepsilon_n := \gcd\{p, \hat{p}^{2^n}\}, \qquad M_n := \{u \in M \mid \hat{p}(u) = 0 \bmod \varepsilon_n\}. \tag{2.11}$$

One verifies easily

$$\varepsilon_0 = \hat{p}, \qquad \varepsilon_{n+1} = \gcd\{p, {\varepsilon_n}^2\} \quad (n \in \mathbb{N}), \qquad \varepsilon_{k_p} = p, \tag{2.12}$$

in particular $\varepsilon_{n+1} = 0 \bmod \varepsilon_n$. This implies with (2.6) and (2.5) that

$$M_0 = M, \qquad M_{n+1} \subset M_n, \qquad \Phi(M_n) \subset M_{n+1} \quad (n \in \mathbb{N}), \tag{2.13}$$

and moreover for the solution manifold of (2.1)

$$M_{k_p} = \{u \in M \mid \hat{p}(u) = 0 \bmod p\}. \tag{2.14}$$

We define further for $n \in \mathbb{N}$ and $u \in M_n$ by $\pi_n(u) \in \mathbb{K}[x]$ the remainder of u divided by ε_n, which is uniquely determined through the relation $u = t\varepsilon_n + \pi_n(u)$ where $t \in \mathbb{K}[x]$ and $\text{degree}(\pi_n(u)) < \text{degree}(\varepsilon_n)$. Using (2.4) once more yields

$$\hat{p}(\pi_n(u)) = \hat{p}(u) - t\varepsilon_n \hat{p}'(u) + t^2\varepsilon_n^2 \tilde{p}[u, -t\varepsilon_n] = 0 \bmod \varepsilon_n.$$

Therefore, for all $n \in \mathbb{N}$,

$$\begin{aligned}&\pi_n(u) \in M_n,\\ &\quad \text{degree}(\pi_n(u)) < \text{degree}(\varepsilon_n) \leqslant \text{degree}(p) \quad (u \in M_n).\end{aligned} \tag{2.15}$$

Now considering for an arbitrary $u \in M$ the recursively defined sequence

$$\begin{aligned}&u_0 := \Phi_0(u) := \pi_0(u),\\ &u_{n+1} := \Phi_{n+1}(u) := (\pi_{n+1} \circ \Phi)(u_n) \quad (n \in \mathbb{N}),\end{aligned} \tag{2.16}$$

we have

$$u_n \in M_n, \quad \text{degree}(u_n) < \text{degree}(p) \qquad (n \in \mathbb{N}). \tag{2.17}$$

This yields by (2.14) with k_p according to (2.10):

Theorem 2.2. $\Phi_{k_p}(u) = s_p$ *for every* $u \in M$, *in particular, for* $u = x$.

The construction of the u_n in (2.16) suggests that the following representation is valid:

Remark 2.3. Let $\text{degree}(\hat{p}) > 1$. Then for $n \in \mathbb{N}$

$$u_n := \Phi_n(x) = x - \sum_{j=0}^{n-1} \alpha_j \varepsilon_j,$$

where the α_j can be computed recursively by

$$\alpha_n = \text{remainder of } b(u_n)\, \hat{p}(u_n)/\varepsilon_n \text{ divided by } \varepsilon_{n+1}/\varepsilon_n \quad (n \in \mathbb{N}).$$

In particular, $\text{degree}(\alpha_n) < \text{degree}(\varepsilon_{n+1}/\varepsilon_n) < \text{degree}(p)$ and thus $\alpha_n = 0$ $n \geqslant k_p$.

The proof follows by induction. For $n = 0$ the statement $\Phi_0(x) = \pi_0(x) = x$ follows directly by means of $\varepsilon_0 = \hat{p}$ with the assumption degree$(\hat{p}) > 1$. Now let $n \in \mathbb{N}$ and assume that the representation to be valid for $u_n \in M_n$. With $\hat{p}(u_n) = r_n\varepsilon_n$, $r_n \in \mathbb{K}[x]$ follows $\Phi(u_n) = u_n - b(u_n)r_n\varepsilon_n$. The unique representation $b(u_n)r_n = t_n(\varepsilon_{n+1}/\varepsilon_n) + \alpha_n$ with $t_n, \alpha_n \in \mathbb{K}[x]$, degree$(\alpha_n) <$ degree$(\varepsilon_{n+1}/\varepsilon_n)$ yields then

$$\Phi(u_n) = \left(x - \sum_{j=0}^{n} \alpha_j \varepsilon_j\right) - t_n\varepsilon_{n+1}.$$

Since the degree of the term in brackets is less than the degree of ε_{n+1}, we finally obtain that $u_{n+1} = (\pi_{n+1} \circ \Phi)(u_n)$ is equal to this term.

The special situation in Lemma 1.3 is settled with the following:

Remark 2.4. Let $q \in \mathbb{K}[x]$ be square-free and $\ell \in \mathbb{N}^*$. For $p := q^\ell$ we obviously have $\hat{p} = q$ and $\ell_p = \ell$. Thus by Theorem 2.2, $\Phi_k(x) = s_p$ with $k \in \mathbb{N}$ determined by $2^{k-1} < \ell \leqslant 2^k$. Moreover, if $\mathbb{N}^* \ni n < \ell$, then s_{q^n} is the remainder of s_p divided by q^n.

The following remark clarifies, in particular, the connection with the corresponding results in [2,5,10].

Remark 2.5. Let $\tau(n) \in \mathbb{N}$ with $\tau(0) = 1$ and $\tau(n) < \tau(n+1) \leqslant 2\tau(n)$ $(n \in \mathbb{N})$. Consider $\varepsilon_n := \gcd\{p, \hat{p}^{\tau(n)}\}$. Obviously, $\varepsilon_0 = \hat{p}$, $\varepsilon_{n+1} = 0 \bmod \varepsilon_n$ and $\varepsilon_k = p$ for $k \in \mathbb{N}^*$ with $\tau(k-1) < \ell_p \leqslant \tau(k)$. Then, in analogy to Theorem 2.2, $\Phi_k(u) = s_p$ holds for every $u \in M$. Moreover, the corresponding representation in Remark 2.3 is valid.

Our choice $\tau(n) = 2^n$ is optimal since in this case the best possible convergence rate is realized. The choice $\tau(n) = n + 1$ yields the weakest convergence rate. In this case, one needs $\ell_p - 1$ steps of iteration to obtain s_p. The series representation in Remark 2.3 becomes then

$$s_p = x - \sum_{j=0}^{\ell_p - 2} \tilde{\alpha}_j \hat{p}^{j+1}$$

with an analogous formula for the $\tilde{\alpha_j}$. This is exactly the ‘Ansatz’ for the solution considered in [2,5,10].

We close this section with an example of a polynomial p, where s_p can be determined explicitly. It is a straightforward generalization of an example in [2]. In this special case, the convergence rate is even better than in Theorem 2.2 and Remark 2.3 if $m > 2$.

Example 2.6. Let char $\mathbb{K} = k > 0$, $m = k^\nu$, $\nu \in \mathbb{N}^*$ and $\ell \in \mathbb{N}$. We consider $q := x^m + x + 1$ and $p := q^\ell$. Then $\hat{p} = q$, $\ell_p = \ell$ and

$$s_p = x - \sum_{j=0}^{n-1} (-1)^j q^{m^j}$$

with $n \in \mathbb{N}$ determined by $m^{n-1} < \ell \leqslant m^n$.

The proof follows by induction considering the Newton iterations and using the formula

$$\begin{aligned} q(u+h) &= (u+h)^m + u + h + 1 = u^m + h^m + u + h + 1 \\ &= q(u) + h + h^m, \end{aligned}$$

which is valid when $m = k^\nu$.

3. Applications and programs

First we give a program in MAPLE for the calculation of s_p in the case $\mathbb{K} = \mathbb{Q}$ using the algorithm in Theorem 2.2. The program begins with the calculation of $\hat{p}$ as in (1.3) and determines after that $a,\ b$ as in (2.2). Then, starting with

$$\varepsilon_0 = \hat{p}, \qquad u_0 = \mathrm{rem}(x, \varepsilon_0),$$

the quantities

$$\begin{aligned} \varepsilon_{n+1} &= \gcd\{p, \varepsilon_n^2\}, \\ \tilde{u}_n &= \mathrm{rem}(b(u_n)\,\hat{p}(u_n), \varepsilon_{n+1}), \\ u_{n+1} &= u_n - \tilde{u}_n, \\ \alpha_n &= \mathrm{quo}(\tilde{u}_n, \varepsilon_n) \end{aligned}$$

are recursively calculated for $n \in \mathbb{N}$ until $\varepsilon_n = p$. The final value of u_n yields s_p. When the degree of p is large, the evaluation of $\tilde{u}_n = \mathrm{rem}(b(u_n)\,\hat{p}(u_n), \varepsilon_{n+1})$ takes, in general, a long time. Thus, it is recommendable to evaluate these quantities by using the Horner form and taking the remainder by ε_{n+1} in each step. The call `s_poly(p,x)` returns $s_p = s_p(x)$.

```
> s_poly:=proc(p,x)
> local d,p1,r,q,q1,a,b,v,s,ss,e,ee,n,l,j;
> d:=degree(p,x); p1:=diff(p,x); r:=gcd(p,p1); q:=quo(p,r,x);
> q1:=diff(q,x); gcdex(q,q1,x,'a','b');
> v:=sort(expand(q*b),x); n:=degree(v,x);
> e:=q; l:=degree(e,x); s:=rem(x,e,x);
> while l < d do ee:=gcd(p,e*e); ss:=coeff(v,x,n);
> for j from 1 to n do ss:=coeff(v,x,n-j)+rem(ss*s,ee,x); od;
> s:=s-ss; e:=ee; l:=degree(e,x) od;
```

Table 1

(k, ℓ, m)	(3, 4, 2)	(5, 2, 4)	(6, 8, 4)	(7, 5, 11)	(9, 11, 9)	(16, 13, 9)	(15, 15, 15)
degree$(p)/k_p$	20/2	20/3	40/3	40/4	60/4	80/4	90/4
Time (s)	0.260	0.982	5.167	9.544	23.504	35.090	42.982

```
> s:=sort(expand(s));
> s
> end:
```

As an example, we consider the polynomials

$$p = u^k v^\ell w^m \quad \text{with} \quad u = x^2 - 2,\ v = x^3 - 3,\ w = x - 7 \tag{3.1}$$

for some special values of k, ℓ, m. Table 1 shows the computing time for the calculation of s_p by using the procedure `s_poly`.

When using (an improved version of) the original Levelt algorithm the computing time for the first two cases was 98 and 280 s, respectively.

All calculations were done by using MAPLE V.5 under Windows NT on a PC with Pentium 300MMX processor and 64 MB RAM.

In the following, we discuss how the Jordan decomposition of matrices can be calculated by using the previous results. As already mentioned in Section 1, the semi-simple part $S =: S_A$ of the Jordan decomposition of a matrix A can be obtained by the following scheme:

$$A \to p := \text{characteristic polynomial of } A \to s := s_p \to S := s(A). \tag{3.2}$$

In particular, for $A \in \mathbb{Q}^{n\times n}$, all of these three steps can principally be settled by using MAPLE procedures: the first by `p:=linalg[charpoly](A,x)`, the second by the procedure `s:=s_poly(p,x)` and the last step by `S:=evalm(subs(x=A,s))`. Unfortunately, MAPLE's builtin procedure `p:=linalg[charpoly](A,x)` is extremly slow. Furthermore, in particular, in the case of a big (say size 20×20) and dense matrix A, the computation of $S := s(A)$ by using `evalm(subs(x=A,s))` takes a large amount of time. Thus, though the routine `s_poly` is very fast, the calculation of S by this way remains unsatisfactory, in particular, as other strong algorithms are available. A good reference in this connection is the 'normform' package of Mulders and Levelt in the the MAPLE V share library. This package contains a very fast routine to calculate the rational Jordan normal form J of a matrix A and the corresponding transformation matrices C, C^{-1} such that $C^{-1} \circ A \circ C = J$. Since the semi-simple part S_J of J can be obtained directly from J by taking the diagonal blocks, one gets then immediately $S_A = C \circ S_J \circ C^{-1}$.

The algorithms of the normform package are described in [12]. From this it follows that the computation of the rational Jordan normal form J is carried out in three steps. First (based on the idea of cyclic vectors) a special 'cyclic form' $F = F_A$ and the corresponding transformation matrices W, W^{-1} such that $W^{-1} \circ A \circ W = F$ are computed. Then (by using the usual routine for computing the Smith normal form) the Frobenius normal form and after that finally the rational Jordan normal form (each with corresponding transformation matrices) are determined. Since the computation of the Smith normal form takes a large amount of time, it will be of advantage in any case if this step can be avoided (see [12, p. 95]).

The calculation of the 'cyclic form' F plus transformation matrices W, W^{-1} is carried out in the MAPLE procedure 'normform/cyclic_vectors'. F has the block structure $F = (F_{ij})$, where $F_{ij} = 0$ for $i > j$, the diagonal blocks F_{jj} are companion matrices of monic polynomials p_j and F_{ij} for $i < j$ have zero columns except (at most) for the last column. From this follows immediately that the product of the p_j yields the characteristic polynomial of F, which is also the characteristic polynomial of A. It is thus reasonable to modify the 'normform/cyclic_vectors' program of Levelt and Mulders slightly in order to obtain a fast routine for the calculation of the characteristic polynomial. The following MAPLE procedure `cyclic_form` realizes this idea. The call `cyclic_form(A,x,'U','V','W','F','f','ind')` with $A \in \mathbb{Q}^{n\times n}$ computes and returns the characteristic polynomial $p = p(x)$ of A. Besides that it computes the 'cyclic normal form' F, the transformation matrix W, an LU-type decomposition, $W = U \circ V$ and the integer-valued vectors f, ind which are required for later calculations. These quantities are assigned to the variables U, V, W, F, f, ind in the argument of `cyclic_form`.

```
> cyclic_form:=proc(A,x,U,V,W,F,f,ind)
> local n,i,j,k,r,r1,r2,u,v,w,m,a,temp,p,q,p0;
> n:=linalg[rowdim](A);
> U:=array(1..n,1..n); V:=array(1..n,1..n);
> W:=array(1..n,1..n); F:=array(1..n,1..n);
> u:=array(1..n); v:=array(1..n); w:=array(1..n); q:=array(1..n);
> ind:=array(sparse,1..n); f:=array(sparse,1..n);
> r:=0; p:=1;
> while r < n do r1:=r;
>  for i to n while ind[i] <> 0 do  od;
>  for j to n do w[j]:=0 od; w[i]:=1;
>  do u:=copy(w);
>   for i to n do v[i]:=0 od;
>   for i to n do k:=ind[i];
>    if k <> 0 and u[i] <> 0 then
>    a:=u[i]/U[i,k]; u[i]:=0;
>     for j from i+1 to n do u[j]:=u[j]-a*U[j,k] od;
>    v[k]:=a fi;
>   od;
>   i:=1; while i<=n and u[i]=0 do i:=i+1 od;
>   if i <= n then r:=r+1; ind[i]:=r;
>    for j to n do W[j,r]:=w[j] od;
```

```
>     for j from i to n do U[j,r]:=u[j] od;
>     for j to r-1 do V[j,r]:=v[j] od;
>     for i to n do temp:=0;
>      for j to n do temp:=temp+A[i,j]*w[j] od;
>     u[i] := temp od; w:=copy(u)
>    else break fi
>   od;
>   r2:=r-r1; f[r]:=1;
>   for j to r do temp:=v[r+1-j];
>    for m from r+2-j to r do temp:=temp-V[r+1-j,m]*q[m] od;
>   q[r+1-j]:=temp; F[r+1-j,r]:=temp od;
>   p0:=sort(x^r2-sum('q[r+1-j]*x^(r2-j)',('j')=1..r2));
>   p:=sort(expand(p*p0))
> od;
> p
> end:
```

The following MAPLE procedure `char_poly` is just a simplified call to `cyclic_form`. The call `char_poly(A,x)` with $A \in \mathbb{Q}^{n\times n}$ returns the characteristic polynomial $p = p(x)$ of A.

```
> char_poly:=proc(A,x)
> cyclic_form(A,'x','U','V','W','F','f','ind');
> end:
```

With reference to [8], one of the referees pointed out that the idea of computing the characteristic polynomial by using the cyclical form is classical.

The calculation of S_A according to (3.2) by using `char_poly` for the first, `s_poly` for the second and the Horner form for the third step yields quite satisfactory results in the case when A is not too large (say size $<$10) or in the case when A has a special structure (say lower or upper block triangular form and/or a band structure). In the case when A is a large and dense matrix, it is reasonable to make use of the additional results furnished by the procedure `cyclic_form`. The concept is then as follows: calculate first $p, F, W, \ldots$ with `cyclic_form`. Then calculate $s = s_p$ with `s_poly` as before. Since F has a simple structure, the calculation of $S_F = s(F)$ can be organized in a very efficient way. Herewith calculate finally $S_A = W \circ S_F \circ W^{-1}$ by using the decomposition $W = U \circ V$. This concept is realized in the following MAPLE procedure SND. The call SND(A) with $A \in \mathbb{Q}^{n\times n}$ returns the semi-simple part S_A of A.

```
> SND:=proc(A::matrix)
> local n,i,j,k,l,ind,U,V,W,F,f,S,S1,U1,W1,m,a,temp,p,s;
> n:=linalg[rowdim](A);
```

```
> U:=array(1..n,1..n); V:=array(1..n,1..n); W:=array(1..n,1..n);
> U1:=array(1..n,1..n); W1:=array(1..n,1..n); F:=array(1..n,1..n);
> p:=cyclic_form(A,'x','U','V','W','F','f','ind');
> s:=s_poly(p,x); l:=degree(s,x);
> S1:=array(1..n,1..n); S:=array(sparse,1..n,1..n);
> a:=coeff(s,x,l); for i to n do S[i,i]:=a od;
> for k from 1 to l do a:=coeff(s,x,l-k);
>  for i to n do for j to n do
>   if f[j]=1 then temp:=0;
>    for m to j do temp:=temp+S[i,m]*F[m,j] od;
>   S1[i,j]:=temp
>   else
>   S1[i,j]:=S[i,j+1] fi;
>   if i=j then S1[i,j]:=S1[i,j]+a fi;
>  od; od; S:=copy(S1);
> od;
> for i to n do for j to n do temp:=0;
>   for m to n do temp:=temp+W[i,m]*S[m,j] od;
> W1[i,j]:=temp od; od;
> for i to n do U1[i,1]:=W1[i,1] od;
> for j from 2 to n do for i to n do temp:=W1[i,j];
>  for k to j-1 do temp:=temp-U1[i,k]*V[k,j] od;
> U1[i,j]:=temp od; od;
> l:=ind[n]; for i to n do S[i,n]:=U1[i,l]/U[n,l] od;
> for j to n-1 do l:=ind[n-j]; for i to n do temp:=U1[i,l];
>  for k from n-j+1 to n do temp:=temp-S[i,k]*U[k,l] od;
> S[i,n-j]:=temp/U[n-j,l] od; od;
> S
> end:
```

Table 2 shows the computing time for the calculation of S_A for some examples $A \in \mathbb{Z}^{n\times n}$ by using the procedure SND. The second column ‘inner structure’ shows the structure of the Frobenius normal form of A, which has a strong influence on the computing time. Here $[p_1, \ldots, p_k]$ means that the Frobenius normal form of A is $\mathrm{diag}(\mathrm{comp}(p_1), \ldots, \mathrm{comp}(p_k))$, where $\mathrm{comp}(p_j)$ denotes the companion matrix of the monic polynomial p_j. The total computing time for SND is given in column t3. It includes the computing time for `char_poly` and `s_poly` which are shown separately in the columns t1 and t2, respectively. t4 gives the computing time for the rational Jordan normal form J of A plus corresponding transformation matrices $C,\ C^{-1}$ by using the normform package. Actually, the correct competition to t3 is not just t4, since the time for evaluating the product $S_A = C \circ S_J \circ C^{-1}$ must also be taken into account. When $C,\ C^{-1}$ are large and dense matrices, this enlarges t4 once more considerably.

Table 2

n	Inner structure	t1	t2	t3	t4
10	$[u^5]$	0.270	0.030	0.791	1.561
10	$[u^2, u^3]$	0.101	0.030	0.571	0.962
10	$[u, u^2, u^2]$	0.090	0.170	0.500	1.022
15	$[v^5]$	1.121	0.041	5.368	11.665
15	$[v^2, v^3]$	1.142	0.030	4.927	8.322
20	$[u^{10}]$	8.463	0.050	34.089	81.147
20	$[v^3, uv^3]$	5.057	0.511	21.161	37.242
20	$[u^2, u^2, u^2, u^2, u^2]$	2.093	0.050	9.434	18.315
25	$[u^5v^5]$	32.667	1.122	131.339	322.664
25	$[u^2v^2, u^3v^3]$	25.277	1.382	105.121	186.749
30	$[v^{10}]$	91.161	0.281	367.879	896.359
30	$[uv, u^2v^2, u^3v^3]$	65.495	1.612	342.172	470.426

The examples have been calculated with $u,\ v$ from (3.1). Similar results were obtained with other choices of u and v. In any of the treated cases, our 'direct' method is faster than the roundabout way via the computation of the rational Jordan normal form.

Finally, we show that how our method can be applied very successfully in the case of meromorphic differential operators. Originally, this was the actual starting point for our investigations. Fundamental papers in connection with the Jordan decomposition of differential operators are [4,9]. A short presentation of the theoretic background of the following investigations is to be found in [6,7].

We consider the (formal) differential operator

$$\mathscr{L}Y := -z\frac{\mathrm{d}}{\mathrm{d}z}Y + \omega(z) \circ Y, \quad \omega(z) = \sum_{j=0}^{\infty} z^j \omega_j,\ \omega_j \in \mathbb{K}^{m \times m} \tag{3.3}$$

with a singularity of the first kind at 0. For the moment let $\mathbb{K}$ be any perfect field. An important special case of (3.3) is

$$\mathscr{L}Y := -z\frac{\mathrm{d}}{\mathrm{d}z}Y + (\omega_0 + z\omega_1) \circ Y, \quad \omega_0, \omega_1 \in \mathbb{K}^{m \times m}. \tag{3.4}$$

The corresponding differential equation $\mathscr{L}Y = 0$ is known as Birkhoff's equation. Special cases hereof are the confluent hypergeometric differential equation and the Bessel equation.

The central problem in the study of (3.3) is to determine the structure of the fundamental solutions Y of the corresponding differential equation $\mathscr{L}Y = 0$. The really interesting case arises when ω_0 has eigenvalues which differ by integers $\neq 0$. It turns out that this problem can be solved by calculating the Jordan decomposition of a specific matrix.

In [6] is shown that the operator $\mathscr{L}$ admits a generalized Jordan decomposition and that the corresponding semi-simple part $\mathscr{S} = \mathscr{S}_{\mathscr{L}}$ has the same structure as $\mathscr{L}$, namely,

$$\mathscr{S}Y := -z\frac{\mathrm{d}}{\mathrm{d}z}Y + \sigma(z) \circ Y, \quad \sigma(z) = \sum_{j=0}^{\infty} z^j \sigma_j, \ \sigma_j \in \mathbb{K}^{m\times m}. \tag{3.5}$$

In order to obtain the first $k+1$ coefficients $\sigma_0, \ldots, \sigma_k$ of σ, one considers the lower triangular block-matrix

$$A := \begin{pmatrix} \omega_0 & 0 & 0 & \cdots & 0 \\ \omega_1 & \omega_0 - 1 & \ddots & \ddots & \vdots \\ \omega_2 & \omega_1 & \omega_0 - 2 & \ddots & \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \omega_{k-1} & \ddots & \ddots & \ddots & 0 \\ \omega_k & \omega_{k-1} & \cdots & \omega_1 & \omega_0 - k \end{pmatrix} \in \mathbb{K}^{n\times n}, \quad n = m(k+1). \tag{3.6}$$

A has a band structure. Then (3.5) yields that S_A has the same band structure with ω_j replaced by σ_j. Thus, $\sigma_0, \ldots, \sigma_k$ can be determined by calculating the first block column of S_A. Considering the case $\mathbb{K} = \mathbb{Q}$ this can principally be done by using the procedure SND. However, since the size $n = m(k+1)$ of A is in general very large, it is better to make here an individual approach.

The characteristic polynomial p of A in (3.6) is obviously

$$p(x) = \prod_{j=0}^{k} q(x+j), \tag{3.7}$$

where q is the characteristic polynomial of ω_0. The concept is then as follows: calculate the first q with `char_poly` and then p with (3.7). The calculation of $s = s_p$ is carried out with `s_poly` as before. By the band structure of A, the calculation of $S_A = s(A)$ requires only the calculation of the first block column. This concept is realized in the following MAPLE procedure ODE. The call ODE(L, k) with $L = [\omega_0, \ldots, \omega_r]$, $\omega_j \in \mathbb{Q}^{m\times m}$ $(j = 0, \ldots, r)$ and $r, k \in \mathbb{N}^*$ calculates the first $k+1$ coefficients $\sigma_0, \ldots, \sigma_k$ of σ in (3.5).

```
> ODE:=proc(L::list,k)
> local r,m,n,p0,p,s,t,l,i,j,a,d,temp,A,S,S1;
> r:=nops(L); m:=rowdim(L[1]); n:=m*(k+1);
> p0:=char_poly(L[1],x); p:=p0;
> for j to k do p0:=subs(x=x+1,p0); p:=p0*p; od;
> p:=sort(expand(p));
> s:=s_poly(p,x); d:=degree(s,x);
> A:=array(sparse,1..n,1..n);
> for t to r do for l to k+2-t do
```

Table 3

m	Structure of ω_0	k	t1	t2	t3
2	$[x^2]$	4	0.191	0.300	0.561
2	$[(x-2)(x+2)]$	4	0.230	0.321	0.931
2	$[x(x+3)]$	4	0.120	0.311	0.791
4	$[(x^2+2)(x^2-6x+11)]$	4	2.183	3.284	15.733
6	$[x^2, x^2(x-3)(x-4)]$	4	15.593	23.934	44.865
8	$[x(x-3), x^4(x-2)(x-3)]$	4	62.610	260.255	440.834

```
>   for i to m do for j to m do
>    if i=j and t=1 then temp:=l-1 else temp:=0 fi;
>    A[(t-1+l-1)*m+i,(l-1)*m+j]:=L[t][i,j]-temp
>   od; od;
> od; od;
> S1:=array(1..n,1..m); S:=array(sparse,1..n,1..m);
> for l from 0 to d do a:=coeff(s,x,d-l);
>  for i to n do for j to m do
>    if i=j then temp:=a else temp:=0 fi;
>    for t to n do temp:=temp+A[i,t]*S[t,j] od;
>  S1[i,j]:=temp od; od;
>  S:=copy(S1);
> od;
> S
> end:
```

Table 3 shows the computing time for calculating the first $k+1$ coefficients of (3.5) for some special cases of (3.4). The meaning of the columns is as follows: m and k are as before; t1 gives the total computing time by using the procedure ODE; t2 the total computing time by using SND for the corresponding matrix A in (3.6); t3 contains the computing time for the rational Jordan normal form J of the same A plus transformation matrices $C,\ C^{-1}$ by using the normform package.

The routine ODE is in any case much faster than the other routines.

A final remark concerning the MAPLE programs might be useful in order to avoid misunderstandings. The goal of this section was not to deliver a package of well-implemented routines. The programs are rather coarse and bare of sophisticated techniques. For example, no special techniques for handling fractions with large numbers have been used. A professional implementation in particular of the routines SND and ODE will furnish even much better results than those presented in the tables.

In the last section, we have restricted our considerations to the case $\mathbb{K}=\mathbb{Q}$. Analogous treatments in the cases when $\mathbb{K}$ is an algebraic extension of $\mathbb{Q}$ or a field of rational functions in one or more variables over $\mathbb{Q}$ or even in the non-zero characteristic case $\mathbb{K}=\mathbb{Z}/p\mathbb{Z}$, where p is a prime number, are possible.

Acknowledgement

I thank the referees for their valuable criticism concerning the discussion of the applications, in particular for pointing out the role and the technique of efficiently computing the characteristic polynomial.

References

[1] N. Bourbaki, Éléments de Mathématique; Algèbre, Hermann, Paris, 1958 (Chapter 7, Section 5; Chapter 8, Section 9).
[2] N. Burgoyne, R. Cushman, The decomposition of a linear mapping, Linear Algebra Appl. 8 (1974) 515–519.
[3] K.O. Geddes, S.R. Czapor, G. Labahn, Algorithms for Computer Algebra, Kluwer Academic Publishers, Dordrecht, 1992 (Chapter 4, Section 9).
[4] R. Gerard, A.H.M. Levelt, Sur les connexions a singularités régulières dans le cas de plusieurs variables, Funkcial. Ekvac. 19 (1973) 149–173.
[5] K. Hoffman, R. Kunze, Linear Algebra, second ed., Prentice-Hall, Englewood Cliffs, NJ, 1971 (Chapter 7, Section 7).
[6] P.F. Hsieh, M. Kohno, Y. Sibuya, Construction of a fundamental matrix solution at a singular point of the first kind by means of the SN decomposition of matrices, Linear Algebra Appl. 239 (1996) 29–76.
[7] P.F. Hsieh, Y. Sibuya, Basic Theory of Ordinary Differential Equations, Springer, New York, 1999 (Chapter 5, Sections 3 and 4).
[8] W. Keller-Gehring, Fast algorithms for the characteristical polynomial, Theoret. Comput. Sci. 36 (1985) 309–317.
[9] A.H.M. Levelt, A descent theorem for differential operators of the first kind, Proc. K. Ned. Akad. Wet., Amsterdam Ser. A 81 (2) (1978) 260–268.
[10] A.H.M. Levelt, The semi-simple part of a matrix, Algoritmen In De Algebra, A Seminar on Algebraic Algorithms, University of Nijmegen, 1993.
[11] J.D. Lipson, Newton's Method: A Great Algebraic Algorithm, in: R.D. Jenks (Ed.), Proceedings of the SYMSAC '76, ACM, New York, 1976, pp. 260–270.
[12] T. Mulders, Computation of normal forms for matrices, Algoritmen In De Algebra, A Seminar on Algebraic Algorithms, University of Nijmegen, 1993.
[13] T.A. Springer, Linear Algebraic Groups, Birkhäuser, Basel, 1981 (Chapter 2, Section 4).