On Symmetric Powers of Differential Operators

Manuel Bronstein Thom Mulders
Institute for Scientific Computation
ETH Zentrum IFW
CH-8092 Zürich
{bronstein,mulders}@inf.ethz.ch
http://www.inf.ethz.ch/personal/{bronstein,mulders}/

Jacques-Arthur Weil
Département de Mathématiques
Faculté des Sciences, 123 Av. Albert Thomas
F-87060 Limoges
weil@unilim.fr
http://Medicis.Polytechnique.fr/gage/weil.html

Abstract

We present alternative algorithms for computing symmetric powers of linear ordinary differential operators. Our algorithms are applicable to operators with coefficients in arbitrary integral domains and become faster than the traditional methods for symmetric powers of sufficiently large order, or over sufficiently complicated coefficient domains. The basic ideas are also applicable to other computations involving cyclic vector techniques, such as exterior powers of differential or difference operators.

Introduction

Let R be an integral domain of characteristic 0, D be a derivation on R, K be the quotient field of R and $R[\partial; D]$ be the corresponding ring of linear differential operators with coefficients in R. Let $L = \sum_{i=0}^{n} a_i \partial^i \in R[\partial; D]$ with $n > 0$ and $a_n \neq 0$, $Y_0, \ldots, Y_{n-1}$ be indeterminates, and consider the extension of D to $K[Y_0, \ldots, Y_{n-1}]$ given by $DY_i = Y_{i+1}$ for $0 \le i < n-1$ and $DY_{n-1} = -a_n^{-1} \sum_{i=0}^{n-1} a_i Y_i$. For any integer $m > 0$, the m^{th} *symmetric power of* L, denoted $L^{\circledS m}$, is defined to be a nonzero element of $R[\partial; D]$ of minimal degree such that $L^{\circledS m}(Y_0^m) = 0$. Let $H_m \subset K[Y_0, \ldots, Y_{n-1}]$ be the set of homogeneous polynomials of degree m. Since H_m is a vector space of dimension $N = \binom{n+m-1}{n-1}$ over K and is closed under D, it follows that the elements $Y_0^m, D(Y_0^m), \ldots, D^N(Y_0^m)$ of H_m are linearly dependent over K, hence that $L^{\circledS m}$ always exists and has degree at most N. Symmetric powers have many applications within algorithms for solving linear differential equations:

- the factorization patterns of $L^{\circledS m}$ determine the structure of the Galois groups of second and third order equations, and provide necessary and sufficient conditions for the existence of Liouvillian solutions [13];
- the radical solutions of $L^{\circledS m}$ provide explicitely the Liouvillian solutions of second and third order equations [14];
- the rational solutions of $L^{\circledS m}$ provide explicitely the Liouvillian solutions of completely reducible second order equations [16], and are needed as a first step in computing the Liouvillian solutions of higher-order equations [15];

For computing invariants of operators, the algorithms presented in this paper provide a rational alternative to the local method of [18].

The basic algorithm [11, 13] for computing $L^{\circledS m}$ is to look for a linear dependence over K between $Y_0^m, D(Y_0^m), \ldots, D^i(Y_0^m)$ for increasing i's until $i = N$, the coefficients of the first dependence found being the coefficients of $L^{\circledS m}$. Expressing each $D^j(Y_0^m)$ in the monomial basis of H_m converts the problem of looking for a linear dependence over K to finding the kernel of an $N \times (i+1)$ matrix with coefficients in K. This kernel computation becomes however rapidly expensive as n and m increase, so we present in this paper algorithms for computing $L^{\circledS m}$ that either avoid this linear algebra step (when L has order 2), or reduce its cost through preprocessing of the corresponding matrix (when L has arbitrary order). In addition, our algorithms can be carried out in a fraction–free fashion over an arbitrary integral domain rather than over its quotient field, avoiding gcd computations and allowing the use of efficient fraction–free linear algebra algorithms.

Notation: for any $x \in R$ and integer $m \ge 0$, $x^{\underline{m}}$ is the descending factorial of x of degree m, *i.e.* $x^{\underline{m}} = \prod_{i=0}^{m-1}(x-i)$. Note that $m^{\underline{m}} = m^{\underline{m-1}} = m!$ and $m^{\underline{m+1}} = 0$. For any rational number q, $\lfloor q \rfloor$ is the largest integer m such that $m \le q$.

1 An iteration for second-order operators

In the case where $\deg(L) = 2$, it is known that $L^{\circledS m}$ has degree $m+1$. Furthermore, in the appropriate basis of H_m the matrix of $Y_0^m, D(Y_0^m), \ldots, D^m(Y_0^m)$ is upper triangular, so the kernel computation can be avoided altogether, resulting in the following iteration for computing $L^{\circledS m}$.

Theorem 1 *Let $L = \partial^2 + a\partial + b \in R[\partial; D]$, $m > 0$ be an integer and consider the sequence given by $L_0 = 1$, $L_1 = \partial$ and*

$$L_{i+1} = (\partial + ia)L_i + i(m-(i-1))bL_{i-1}$$

for $0 < i \le m$. Then, $L_{m+1} = L^{\circledS m}$.

Proof. Note that $\deg(L_i) = i$ for $0 \leq i \leq m+1$ and that each L_i is monic. Let Y_0, Y_1 be indeterminates and consider the extension of D to $K[Y_0, Y_1]$ given by $DY_0 = Y_1$ and $DY_1 = -aY_1 - bY_0$. We show by induction on i that

$$L_i(Y_0^m) = m^{\underline{i}} Y_0^{m-i} Y_1^i \tag{1}$$

for $0 \leq i \leq m+1$. We have $L_0(Y_0^m) = Y_0^m$ and $L_1(Y_0^m) = \partial(Y_0^m) = mY_0^{m-1}Y_1$, so let $1 \leq j \leq m$ and suppose that (1) holds for $0 \leq i \leq j$. Then,

$$\begin{aligned}
L_{j+1}(Y_0^m) &= (\partial + ja)L_j(Y_0^m) + j(m-(j-1))bL_{j-1}(Y_0^m) \\
&= m^{\underline{j}}\partial(Y_0^{m-j}Y_1^j) + jam^{\underline{j}}Y_0^{m-j}Y_1^j \\
&\quad + j(m-(j-1))bm^{\underline{j-1}}Y_0^{m-(j-1)}Y_1^{j-1} \\
&= m^{\underline{j}}(m-j)Y_0^{m-(j+1)}Y_1^{j+1} \\
&\quad + jm^{\underline{j}}Y_0^{m-j}Y_1^{j-1}(-aY_1 - bY_0) \\
&\quad + jam^{\underline{j}}Y_0^{m-j}Y_1^j + jbm^{\underline{j}}Y_0^{m-(j-1)}Y_1^{j-1} \\
&= m^{\underline{j+1}}Y_0^{m-(j+1)}Y_1^{j+1}
\end{aligned}$$

which implies that (1) holds for $j+1$. Therefore,

$$L_{m+1}(Y_0^m) = m^{\underline{m+1}}Y_0^{-1}Y_1^m = 0$$

which implies that $L_{m+1} = L^{\circledS m}$ since it has degree $m+1$. □

When $L = p\partial^2 + a\partial + b$ and p is not a unit in R, we want to avoid going to the quotient field in order to avoid computing gcd's. While we can replace L_i by $p^{i-1}L_i$ in the iteration of Theorem 1 and obtain a sequence of operators in $R[\partial; D]$, those operators may have increasing contents, in a similar fashion to what happens in polynomial remainder sequences. Large parts of those contents can however be predicted, yielding a fraction–free iteration over R, which we present in the rest of this section.

Lemma 1 *If R is a unique factorization domain then*

$$r \mid q \mid \gcd(p, Dp) \implies qr \mid pDr$$

for any $p, q, r \in R$.

Proof. We can assume that $p \neq 0$, the result being trivial otherwise. Suppose first that r is irreducible. If $r \mid Dr$, then $qr \mid pDr$ since $q \mid p$, so suppose that $r \nmid Dr$ and write $p = r^n h$ where $n > 0$ and $r \nmid h$. Then, $Dp = nr^{n-1}hDr + r^nDh$, so $r \nmid hDr$ implies that $r^n \nmid Dp$, hence that $r^n \nmid q$ since $q \mid Dp$. Therefore, $r \mid (p/q)$, which implies that $qr \mid p$, hence that $qr \mid pDr$.
Let now r_1, r_2 be such that $r_i \mid q$ and $qr_i \mid pDr_i$ for $i = 1, 2$. Then,

$$\frac{p}{q}\frac{D(r_1r_2)}{r_1r_2} = \frac{p}{q}\left(\frac{Dr_1}{r_1} + \frac{Dr_2}{r_2}\right) = \frac{pDr_1}{qr_1} + \frac{pDr_2}{qr_2} \in R$$

so $qr_1r_2 \mid pD(r_1r_2)$ and the lemma follows. □

Theorem 2 *Suppose that R is a gcd domain. Let $L = p\partial^2 + a\partial + b \in R[\partial; D]$ where $p \neq 0$, $p_e = \gcd(p, Dp, a - Dp)$, $p_o = \gcd(p_e, bp/p_e)$, $m > 1$ be an integer and consider the sequence given by $L_0 = 1$, $L_1 = \partial$, $L_2 = p\partial^2 + a\partial + mb$,*

$$L_3 = \left(\frac{p}{p_e}\partial + 2\frac{a - Dp}{p_e} + \frac{Dp}{p_e}\right)L_2 + 2(m-1)b\frac{p}{p_e}L_1\,, \tag{2}$$

$$\begin{aligned}
L_{2i} &= \left(\frac{p}{p_o}\partial + (2i-1)\frac{a-Dp}{p_o} + \frac{Dp}{p_o} + (i-1)\frac{pDp_e}{p_op_e}\right. \\
&\quad \left. + (i-2)\frac{pDp_o}{p_o^2}\right)L_{2i-1} \\
&\quad + (2i-1)(m-(2i-2))\frac{bp}{p_op_e}L_{2i-2}
\end{aligned} \tag{3}$$

and

$$\begin{aligned}
L_{2i+1} &= \left(\frac{p}{p_e}\partial + 2i\frac{a-Dp}{p_e} + \frac{Dp}{p_e} + (i-1)\frac{pDp_e}{p_e^2}\right. \\
&\quad \left. + (i-1)\frac{pDp_o}{p_ep_o}\right)L_{2i} \\
&\quad + 2i(m-(2i-1))\frac{bp}{p_op_e}L_{2i-1}
\end{aligned} \tag{4}$$

for $2 \leq i \leq \lfloor (m+1)/2 \rfloor$. Then, $L_{m+1} = L^{\circledS m}$.

Proof. Define $Q_0 = L_0 = 1$, $Q_1 = L_1 = \partial$, and

$$Q_j = \frac{p_e^{\lfloor (j-1)/2 \rfloor}p_o^{\lfloor j/2 \rfloor - 1}}{p^{j-1}}L_j \tag{5}$$

for $2 \leq j \leq m+1$. We first prove that

$$Q_{j+1} = \left(\partial + j\frac{a}{p}\right)Q_j + j(m-(j-1))\frac{b}{p}Q_{j-1} \tag{6}$$

for $0 < j \leq m$. We have

$$Q_2 = \frac{1}{p}L_2 = \partial^2 + \frac{a}{p}\partial + m\frac{b}{p} = \left(\partial + \frac{a}{p}\right)Q_1 + m\frac{b}{p}Q_0$$

so (6) holds for $j = 1$. Using (2) we get

$$\begin{aligned}
Q_3 &= \frac{p_e}{p^2}L_3 = \left(\frac{1}{p}\partial + 2\frac{a-Dp}{p^2} + \frac{Dp}{p^2}\right)L_2 + 2(m-1)\frac{b}{p}L_1 \\
&= \frac{1}{p}\partial(pQ_2) + \left(2\frac{a-Dp}{p} + \frac{Dp}{p}\right)Q_2 + 2(m-1)\frac{b}{p}Q_1 \\
&= \left(\partial + 2\frac{a}{p}\right)Q_2 + 2(m-1)\frac{b}{p}Q_1
\end{aligned}$$

so (6) holds for $j = 2$. Let $3 \leq j \leq m$. If j is odd, then $j+1 = 2i$ for some $i \geq 2$, so using (3) and (5) we get

$$\begin{aligned}
Q_{j+1} &= \frac{p_e^{i-1}p_o^{i-1}}{p^j}L_{j+1} = \frac{p_e^{i-1}p_o^{i-1}}{p^j}L_{2i} = \frac{p_e^{i-1}p_o^{i-1}}{p^j}\Bigg\{ \\
&\quad \left(\frac{p}{p_o}\partial + j\frac{a-Dp}{p_o} + \frac{Dp}{p_o} + (i-1)\frac{pDp_e}{p_op_e}\right. \\
&\quad \left. + (i-2)\frac{pDp_o}{p_o^2}\right)\frac{p^{j-1}}{p_e^{i-1}p_o^{i-2}}Q_j \\
&\quad + j(m-(j-1))\frac{bp}{p_op_e}\frac{p^{j-2}}{p_e^{i-2}p_o^{i-2}}Q_{j-1}\Bigg\} \\
&= \frac{p_e^{i-1}p_o^{i-2}}{p^{j-1}}\partial\left(\frac{p^{j-1}}{p_e^{i-1}p_o^{i-2}}Q_j\right) + \left(j\frac{a-Dp}{p} + \frac{Dp}{p}\right. \\
&\quad \left. + (i-1)\frac{Dp_e}{p_e} + (i-2)\frac{Dp_o}{p_o}\right)Q_j + j(m-(j-1))\frac{b}{p}Q_{j-1} \\
&= \left(\partial + j\frac{a}{p}\right)Q_j + j(m-(j-1))\frac{b}{p}Q_{j-1}
\end{aligned}$$

so (6) holds for j odd. A similar calculation using (4) and (5) shows that (6) holds also for j even.
Since $Q_0 = 1$, $Q_1 = \partial$ and (6) holds for $0 < j \le m$, Theorem 1 applied to K implies that $Q_{m+1} = (L/p)^{\circledS m}$ hence that $L_{m+1} = L^{\circledS m}$. $\square$

Note that since $p_o \mid p_e \mid \gcd(p, Dp)$, Lemma 1 implies that if R is a unique factorization domain, then the coefficients of the iteration of Theorem 2 are in R, *i.e.* that all the quotients are exact.

Furthermore, if follows from (6) that each Q_j has leading coefficient 1, so (5) implies that the leading coefficient of $L^{\circledS m}$ is

$$c_m = \frac{p^m}{p_e^{\lfloor m/2 \rfloor} p_o^{\lfloor (m+1)/2 \rfloor - 1}} \,.$$

If R is a unique factorization domain, then $c_m \in R$ and every irreducible factor of c_m must divide p, which implies in particular that if R is of the form $k[x]$ with $Dx = 1$, then all the singularities of $L^{\circledS m}$ are singularities of L. This last point follows also from the iteration of Theorem 1.

2 A general fraction–free method

Let $L = \sum_{i=0}^n a_i \partial^i \in R[\partial; D]$ with $n > 0$ and $a_n \neq 0$, $Y_0, \dots, Y_{n-1}$ be indeterminates and consider the derivation $\Delta = a_n D$ on $K[Y_0, \dots, Y_{n-1}]$. Then, $\Delta Y_i = a_n Y_{i+1}$ for $0 \le i < n-1$ and $\Delta Y_{n-1} = -\sum_{i=0}^{n-1} a_i Y_i$, which implies that $R[Y_0, \dots, Y_{n-1}]$ is closed under Δ.

Lemma 2 *Let $m > 0$ be an integer,*

$$w_0 = Y_0^m \quad \text{and} \quad w_{i+1} = \Delta w_i - i D(a_n) w_i \quad \text{for } i \ge 0 \,. \tag{7}$$

If $\sum_{i=0}^M c_i w_i = 0$ for some integer $M \ge 0$ and $c_0, \dots, c_M \in R$, then

$$\left(\sum_{i=0}^M c_i a_n^i \partial^i \right) (Y_0^m) = 0 \,.$$

Proof. We first prove by induction on i that

$$w_i = a_n^i D^i \left(Y_0^m\right)$$

for $i \ge 0$. We have $w_0 = Y_0^m = a_n^0 D^0 \left(Y_0^m\right)$, so suppose that $w_i = a_n^i D^i \left(Y_0^m\right)$ for some $i \ge 0$. Then,

$$\begin{aligned} w_{i+1} &= \Delta w_i - i D(a_n) w_i \\ &= a_n D\left(a_n^i D^i \left(Y_0^m\right)\right) - i D(a_n) a_n^i D^i \left(Y_0^m\right) \\ &= a_n^{i+1} D^{i+1} \left(Y_0^m\right) + i a_n^i D(a_n) D^i \left(Y_0^m\right) \\ &\quad - i D(a_n) a_n^i D^i \left(Y_0^m\right) = a_n^{i+1} D^{i+1} \left(Y_0^m\right) \,. \end{aligned}$$

It follows that

$$\left(\sum_{i=0}^M c_i a_n^i \partial^i \right) (Y_0^m) = \sum_{i=0}^M c_i a_n^i D^i \left(Y_0^m\right) = \sum_{i=0}^M c_i w_i = 0 \,.$$

$\square$

It follows that $L^{\circledS m}$ can be obtained from the first linear dependence found over R between $w_0, w_1, \dots, w_i$ for increasing i until $i = N$, which is equivalent to finding the kernel of an $N \times (i+1)$ matrix with coefficients in R. However, as in the case of second-order operators, the iteration (7) produces increasing contents in the rows of the resulting matrix.

Lemma 3 *Let $(w_i)_{i \ge 0}$ be given by (7) and $Y \in K[Y_0, \dots, Y_{n-1}]$ be a primitive monomial of total degree m. Then for any $i \ge 0$, the coefficient of Y in w_i is divisible by $a_n^{P(Y,i)}$ where*

$$P(Y, i) = \begin{cases} F(Y) & \text{if } i < F(Y); \\ i & \text{if } F(Y) \le i \le F(Y) + G(Y); \\ F(Y) + G(Y) & \text{if } i > F(Y) + G(Y), \end{cases}$$

$$F(Y) = \sum_{k=1}^{n-1} k \deg_{Y_k}(Y)$$

and

$$G(Y) = n - 1 - \max\{k \mid \deg_{Y_k}(Y) > 0\} \,.$$

Proof. By induction on i. We have $w_0 = Y_0^m$ and $F(Y_0^m) = 0$, so the lemma holds for $i = 0$. Suppose now that $i \ge 0$ and that the coefficient of Y in w_i is divisible by $a_n^{P(Y,i)}$ for any primitive monomial in $Y \in H_m$. Since $w_i \in H_m$, we can write $w_i = \sum_Y b_Y Y$, where $b_Y \in R$ and the sum is taken over all primitive monomials of total degree m. We have

$$\begin{aligned} w_{i+1} &= \Delta w_i - i D(a_n) w_i \\ &= \sum_Y \left(a_n D(b_Y) Y - i D(a_n) b_Y Y + a_n b_Y D(Y)\right) . \end{aligned}$$

It suffices to show that for all primitive monomials Y and $\tilde{Y}$ of total degree m, $a_n^{P(\tilde{Y}, i+1)}$ divides the coefficient of $\tilde{Y}$ in $a_n D(b_Y) Y - i D(a_n) b_Y Y + a_n b_Y D(Y)$. Writing $b_Y = b a_n^{P(Y,i)}$ we get

$$\begin{aligned} a_n D(b_Y) - i D(a_n) b_Y &= \\ D(b) a_n^{P(Y,i)+1} &+ (P(Y,i) - i) b a_n^{P(Y,i)} D(a_n) \,. \end{aligned} \tag{8}$$

When $P(Y,i) \neq i$ we have $P(Y, i+1) = P(Y, i)$. From this and (8) we see that $a_n^{P(Y,i+1)}$ divides $a_n D(b_Y) - i D(a_n) b_Y$.

For a primitive monomial $Y \in H_m$ and $0 \le j, k < n$ such that $\deg_{Y_j}(Y) > 0$, we define $Y^{[j,k]}$ to be the primitive monomial $Y Y_k / Y_j$. Whenever $0 \le j < n-1$ and $\deg_{Y_j}(Y) > 0$, we have

$$F(Y^{[j,j+1]}) = F(Y) + 1 \tag{9}$$

and when $0 \le j < n$ and $\deg_{Y_{n-1}}(Y) > 0$, then

$$F(Y^{[n-1,j]}) = F(Y) - (n-1) + j. \tag{10}$$

Writing $Y = \prod_{j=0}^{n-1} Y_j^{e_j}$, we have

$$a_n b_Y D(Y) = a_n b_Y \sum_{j=0}^{n-2} e_j Y^{[j,j+1]} - e_{n-1} b_Y \sum_{j=0}^{n-1} a_j Y^{[n-1,j]}.$$

For $\tilde{Y} = Y^{[j,j+1]}$ we have $G(\tilde{Y}) \le G(Y)$. From this and (9) it follows easily that $P(\tilde{Y}, i+1) \le P(Y, i) + 1$ and so $a_n^{P(\tilde{Y}, i+1)}$ divides $a_n b_Y$.

For $\tilde{Y} = Y^{[n-1,j]}$ we have $G(\tilde{Y}) \le n - 1 - j$. From this and (10) it follows that $P(\tilde{Y}, i+1) \le F(\tilde{Y}) + G(\tilde{Y}) \le F(Y) \le P(Y, i)$, so $a_n^{P(\tilde{Y}, i+1)}$ divides b_Y. $\square$

From the previous lemma we see that $a_n^{F(Y)}$ divides the content of the row corresponding to Y. In fact various entries of the matrix are divisible by even higher powers of a_n.

The following matrix shows the powers of a_n in the general case of the 2^{nd} symmetric power of an order 3 operator. The numbers on the left are the corresponding $F(Y)$. ∞ means that the corresponding entry is 0.

$$
\begin{array}{c}
0 \\ 1 \\ 2 \\ 2 \\ 3 \\ 4
\end{array}
\quad
\begin{pmatrix}
0 & \infty & \infty & 2 & 2 & 2 & 2 \\
\infty & 1 & \infty & 2 & 2 & 2 & 2 \\
\infty & \infty & 2 & 2 & 2 & 2 & 2 \\
\infty & \infty & 2 & \infty & 3 & 3 & 3 \\
\infty & \infty & \infty & 3 & 3 & 3 & 3 \\
\infty & \infty & \infty & \infty & 4 & 4 & 4
\end{pmatrix}
$$

Depending on the coefficients of the operator it can happen that even higher powers of a_n than predicted by Lemma 3, divide some entry of the matrix as the following example shows.

Example 1 *For the 2^{nd} symmetrix power of*

$$(x^2+5)\partial^3 + (5x^2-2x+25)\partial^2 + (x^4-1)\partial + x^4 - x^3 - x - 1$$

we get the following matrix of powers (of x^2+5) dividing the corresponding entries.

$$
\begin{pmatrix}
0 & \infty & \infty & 2 & 3 & 3 & 3 \\
\infty & 1 & \infty & 2 & 4 & 3 & 3 \\
\infty & \infty & 2 & 2 & 3 & 4 & 4 \\
\infty & \infty & 2 & \infty & 3 & 4 & 4 \\
\infty & \infty & \infty & 3 & 3 & 4 & 4 \\
\infty & \infty & \infty & \infty & 4 & 4 & 4
\end{pmatrix}
$$

In this section, we have used the operator $\Delta = a_n D$ to get a matrix with all entries in R. In fact it might be the case that (depending on the coefficients of the operator) it is sufficient to use an operator $\Delta = cD$ with c dividing a_n, in order to avoid fractions. For example, for third–order operators with $a_3 = p^3$ it suffices to use $c = p^2$. However, notice that in this case $R[Y_0, \dots, Y_{n-1}]$ is not closed under Δ. As will be seen in the next section, the extra contents brought in by using $a_n D$ instead of cD can be removed through an adequate preprocessing of the matrix.

In order to avoid fractions one might also use the original operator D and multiply a column by a_n only when fractions do appear.

3 Preprocessing the matrix

When applying the fraction–free method described in the previous section, we have to compute a linear dependence between the w_j. For this we can use any method from linear algebra and in particular fraction–free Gaussian elimination ([5] §9.3) or modular methods [8] are suitable.

Choosing some order of the monomials of total degree m in $Y_0, \dots, Y_{n-1}$ we can write the coefficients of w_j in a matrix $A = (a_{ij})$ (*i.e.* a_{ij} is the coefficient of the ith monomial in w_j). Computing a linear dependence between the w_j and between the columns of A are now equivalent problems.

As shown in the previous section, the entries in A can be divisible by high powers of a_n. In this section we will show how we can preprocess the matrix A in order to make the computation of a linear dependence between its columns more efficient.

It is clear that the kernel of a matrix does not change by multiplying the rows of that matrix by scalars. By multiplying the columns of a matrix by scalars the kernel does change, but there is an easy relationship between the kernels. Our preprocessing strategy consists of multiplying the rows and columns of the matrix A by powers of a_n in order to decrease the powers of a_n dividing the entries of the matrix. Let $B = (b_{ij})$ be a matrix, having entries in $\mathbb{N} \cup \{\infty\}$, such that $b_{ij} = \infty$ when $a_{ij} = 0$ and $a_n^{b_{ij}}$ divides a_{ij} when $a_{ij} \neq 0$. For this we can use for example the powers predicted by Lemma 3 or, when R is a unique factorization domain, we can use the true exponents of a_n in the entries of A. Multiplication of the ith row (resp. jth column) of A by a_n^k can now be translated to adding k to each entry of the ith row (resp. jth column) of B. We can state the preprocessing problem now as follows:

> Let B be an $n \times m$ matrix with coefficients in $\mathbb{N} \cup \{\infty\}$. Find $r_i \in \mathbb{Z}$ $(1 \leq i \leq n)$ and $c_j \in \mathbb{Z}$ $(1 \leq j \leq m)$ such that the following holds:
>
> 1. $b_{ij} + r_i + c_j \geq 0$ for all i, j.
> 2. $\sum (b_{ij} + r_i + c_j)$ should be minimal, where the sum is taken over all i, j such that $b_{ij} \neq \infty$ (i.e. the remaining powers of a_n should be minimal).

When we omit the condition that the variables r_i and c_j should have integer values, this is an example of a linear programming problem. For details on linear programming we refer to [9]. It is clear that this problem has a solution.

We can write the constraints as $Cx \leq b$, where C is a matrix whose columns are labeled by r_i and c_j and whose rows are labeled by $b_{ij} \neq \infty$, and b is a vector whose entries are labeled by (and are equal to) $b_{ij} \neq \infty$. Then all entries in the b_{ij}'s row of C are 0, except for the entries in columns r_i and c_j, which are equal to -1. We see that C is the negative of the incidence matrix of the bipartite graph G, whose vertices are labeled by r_i and c_j and which has an edge from r_i to c_j when $b_{ij} \neq \infty$.

An important class of matrices in linear programming are the totally unimodular matrices. One of the characterizations of these matrices is that all minors equal $-1, 0$ or 1. It is well known that the incidence matrix of a bipartite graph is totally unimodular, so C is totally unimodular. From this it follows that a solution to our problem has integer components.

In our problem the variables r_i and c_j are free variables (*i.e.* their value may be positive or negative) and the constraints are inequalities. If we want to use the standard simplex method to solve this problem we first have to transform it to a problem having only non-negative variables (i.e. their values are non-negative), which doubles the number of variables. If N is the number of constraints in the original problem then solving the new problem using the simplex method would need pivoting in an $(N+1) \times (N + 2(n+m) + 1)$ matrix.

The dual problem of the original problem only has N non-negative variables and $n + m$ equalities. So solving the problem via the dual problem would need pivoting in an $(n+m+1) \times (N+1+n+m)$ matrix (we not only have to solve the dual problem but also the dual of the dual which equals the original problem).

Since $N = O(nm)$, the solution via the dual problem seems to be the most efficient one.

Note that the dual problem is essentially the Hitchcock problem (or transportation problem) for which efficient algorithms do exist ([6]).

As before, we can see that the matrix corresponding to the dual problem is totally unimodular. Next we will show that during the process of pivoting the matrix will stay totally unimodular.

Lemma 4 *Let $S = (s_{ij})$ be a totally unimodular $n \times m$ matrix, $1 \leq i_0 \leq n$, $1 \leq j_0 \leq m$ such that $s_{i_0 j_0} \neq 0$. Let $T = (t_{ij})$ be the matrix we get by pivoting matrix S using $s_{i_0 j_0}$ as pivot (i.e. clearing the ${j_0}^{\text{th}}$ column of S). Then T is totally unimodular.*

Proof. Since $s_{i_0 j_0} = \pm 1$ and by permuting the rows and columns of S we see that it suffices to prove the lemma for $i_0 = j_0 = 1$ and $s_{11} = 1$. Notice that

$$t_{ij} = \begin{cases} s_{ij} & \text{if } i = 1 \\ s_{11}s_{ij} - s_{i1}s_{1j} & \text{if } i > 1 \end{cases}$$

When $1 \leq i_1 < \cdots < i_k \leq n$ and $1 \leq j_1 < \cdots < j_k \leq m$, we denote by $([i_1, \ldots, i_k], [j_1, \ldots, j_k])_U$ the submatrix of matrix U consisting of rows $i_1, \cdots, i_k$ and columns $j_1, \ldots, j_k$.

Since $t_{i1} = 0$ for $i > 1$, we have when $i_1 > 1$

$$\det\left(([i_1, \ldots, i_k], [1, j_2, \ldots, j_k])_T\right) = 0\,.$$

When $i_1 = 1$ then

$$\det\left(([i_1, \ldots, i_k], [j_1, \ldots, j_k])_T\right) = \det\left(([i_1, \ldots, i_k], [j_1, \ldots, j_k])_S\right)$$

since $([i_1, \ldots, i_k], [j_1, \ldots, j_k])_T$ is obtained from $([i_1, \ldots, i_k], [j_1, \ldots, j_k])_S$ by adding or subtracting the first row from some other rows.

When $i_1 > 1$ and $j_1 > 1$ then Sylvester's identity ([5], §9.3) shows that

$$\begin{aligned} &\det\left(([i_1, \ldots, i_k], [j_1, \ldots, j_k])_T\right) \\ &\quad = s_{11}^{k-1} \det\left(([1, i_1, \ldots, i_k], [1, j_1, \ldots, j_k])_S\right) \\ &\quad = \det\left(([1, i_1, \ldots, i_k], [1, j_1, \ldots, j_k])_S\right)\,. \end{aligned}$$

□

This lemma shows that during the process of pivoting the matrix is always totally unimodular, which implies that the pivot (which is a 1-minor) always equals ± 1. This means that only integers appear during the pivoting and this fact can be used to produce a faster implementation of the algorithm.

Example 2 *The solution to the linear program corresponding to the matrix B of Example 1 is*

$$(r_1, \ldots, r_6, c_1, \ldots, c_7) = (-3, -3, -4, -4, -4, -4, 3, 2, 2, 2, 1, 0, 0)\,.$$

Adding $r_i + c_j$ to b_{ij} we get:

$$\begin{pmatrix} 0 & \infty & \infty & 1 & 1 & 0 & 0 \\ \infty & 0 & \infty & 1 & 2 & 0 & 0 \\ \infty & \infty & 0 & 0 & 0 & 0 & 0 \\ \infty & \infty & 0 & \infty & 0 & 0 & 0 \\ \infty & \infty & \infty & 1 & 0 & 0 & 0 \\ \infty & \infty & \infty & \infty & 1 & 0 & 0 \end{pmatrix}$$

The sum of all entries in B is 78, while the sum of all entries in the optimized matrix is only 7.

If one has some kind of factorization of a_n one can also use this optimization technique to all factors seperately. So depending on the algorithms available for R (factorization, gcd computations or none) one can use a complete factorization, balanced factorization or no factorization of a_n. Using this preprocessing at each factor compensates for using $a_n D$ as derivation (instead of cD for some factor c of a_n) when generating the matrix.

4 Implementation issues and timings

We have implemented the above algorithms in Σ^{IT} 0.1.8b [2] and in MAPLE V.3 [4]. In order to benefit fully from the advantages of the fraction–free methods, the following points must be considered:

- for second order operators, all the exact quotients appearing in Theorem 2 should be precomputed before starting the iteration;
- because of the high powers (Lemma 3) of the leading coefficient a_n of L appearing in the iteration (7), the elements of R should be manipulated in the form $a_n^d b$ where $d \in \mathbb{N}$ and $b \in R$, so that the powers a_n^d are not expanded;
- since the fraction–free algorithms work over arbitrary integral domains, when $R = k[x]$ and k is the quotient field of an integral domain S, then $cL \in S[x][\partial; D]$ for some nonzero $c \in S$, so we can take advantage of the faster arithmetic in $S[x]$ and compute $(cL)^{\circledS m}$ rather than $L^{\circledS m}$. An important case is $R = \mathbb{Q}[x]$, since multiplication in $\mathbb{Z}[x]$ has better asymptotic complexity than in $\mathbb{Q}[x]$[1].
- when R is a univariate polynomial ring (with arbitrary derivation), then the preprocessing of the previous section should be done for each balanced factor of a_n.
- if a ring-homomorphism (or even $\mathbb{Z}$-module homomorphism) $\sigma : R \to \mathbb{F}$ is available for a finite field $\mathbb{F}$, then the linear independence over $\mathbb{F}$ of $w_0^\sigma, w_1^\sigma, \ldots, w_i^\sigma$ can be checked while the w_i's are being computed. Let d be the smallest integer such that $w_0^\sigma, \ldots, w_d^\sigma$ are linearly dependent over $\mathbb{F}$. Then, $\deg(L^{\circledS m}) \geq d$ and for suitable R, $\mathbb{F}$ and σ, that degree is d with high probability [10]. We can then apply the preprocessing of Section 3 to the matrix A whose columns consist of $w_0, \ldots, w_d$ and continue the iteration (7) beyond w_d only if $\ker A = 0$. When $R = \mathbb{Z}[x]$, we can also attempt to directly lift the linear dependence [8]. Incidentally, this heuristic either proves that $\deg(L^{\circledS m}) = \binom{n+m-1}{n-1}$ or indicates that it is stricly smaller with high probability, an important information for the algorithm of [15].

We present in the rest of this section empirical timings of the various methods presented in this paper. In addition, since our MAPLE implementation of the fractional kernel method takes advantage of the many cancellations appearing during the linear algebra, we also give the timings obtained with version "Nov.21 1996" of the `diffop` package of [17]. All the timings in this section were obtained on a single

[1] The Karatsuba multiplication algorithm is not effective in $\mathbb{Q}[x]$ since multiplication and addition have the same complexity in $\mathbb{Q}$.

CPU DEC Alpha 500/333 with 256Mb main memory running Digital Unix V4.0a. All timings are given in seconds of CPU, garbage collection included, and a $*$ indicates that the computation was aborted after one hour of CPU. In order to be able to verify the results, we use examples with known Galois groups and/or invariants from the literature.

4.1 Timings for second-order operators

For rational coefficients, we use the following operators from [16]:

$$D_2 = D^2 - \frac{2}{2x-1}D + \frac{3(2x-1)^2(x^4-2x^3+x+1)}{16x^2(x-1)^2(x^2-x-1)^2},$$

$$D_3 = D^2 - \frac{2}{2x-1}D + \frac{(27x^4-54x^3+5x^2+22x+27)(2x-1)^2}{144x^2(x-1)^2(x^2-x-1)^2},$$

$$A_4 = D^2 + \frac{3}{16x^2} + \frac{2}{9(x-1)^2} - \frac{3}{16x(x-1)}$$

and for an example with algebraic numbers in its coefficients, the following non-completely reducible operator from [1]:

$$\begin{aligned} L_{\sqrt{-222}} \;=\; & \Big(x^{12} + 2x^{10} + \frac{151}{3}x^8 + \frac{296}{3}x^6 + \frac{5920}{9}x^4 \\ & + \frac{10952}{9}x^2 + \frac{5476}{9}\Big) D^2 - \frac{104}{25}x^{10} \\ & - \Big(\frac{274}{25} - \frac{22}{15}\sqrt{-222}\Big)x^8 - \Big(\frac{7754}{75} - \frac{68}{15}\sqrt{-222}\Big)x^6 \\ & - \Big(\frac{11248}{75} - \frac{194}{15}\sqrt{-222}\Big)x^4 \\ & - \Big(\frac{29452}{75} - \frac{296}{5}\sqrt{-222}\Big)x^2 + \frac{10952}{5} + \frac{148}{3}\sqrt{-222}. \end{aligned}$$

Table 1 contains the times needed for computing symmetric powers by the various methods of this paper: Iteration $\mathbb{Z}[x]$ is the fraction–free iteration of Theorem 2, Iteration $\mathbb{Q}(x)$ is the fractional iteration of Theorem 1, Kernel $\mathbb{Z}[x]$ is the fraction–free kernel method of Section 2, and Kernel $\mathbb{Q}(x)$ is the classical kernel method[2]. We see from Table 1 that the iteration methods of Section 1 are faster than the kernel methods, and that the fraction–free iteration is faster than the fractional one. The fraction–free kernel method of Section 2 is not always faster than the fractional kernel method, which is to be expected since it reduces the cost of the linear algebra, but the matrices generated in the case of second order operators are triangular, so the linear algebra cost is negligible compared to the simplex preprocessing and overhead. As will be seen in the next section, the fraction–free kernel method becomes better when the linear algebra cost increases, *i.e.* for operators of sufficiently high order, or with sufficiently complicated coefficients. We remark also that the fraction–free iteration together with a fraction–free rational kernel finder makes the algorithm of [16] quite efficient: Darboux curves for D_2, D_3 and A_4 were computed in Σ^{IT} in 1.2, 1.4 and 1.1 seconds respectively, without a priori knowledge of their degrees, while the MAPLE differential equation solver was only able to solve D_3.

Table 2 contains the times needed for computing symmetric powers of $L_{\sqrt{-222}}$ by the fraction–free (ffree) and fractional (frac) iterations of Section 1. It shows as expected that fraction–free methods become more advantageous when the cost of computing in the quotient field of R increases. The difference between the two methods is more drastic in Σ^{IT}, which uses the classical Euclidean algorithm for computing greatest common divisors in $\mathbb{Q}(\sqrt{-222})[x]$, while MAPLE uses better algorithms, making the cost of normalizing quotients less important.

[2] $\mathbb{Q}(x)$ is represented as the quotient field of $\mathbb{Z}[x]$, which is a more efficient type than the quotient field of $\mathbb{Q}[x]$.

	Σ^{IT}		MAPLE		
	ffree	frac	ffree	frac	`diffop`
$L_{\sqrt{-222}}^{\circledS 2}$	1.3	0.5	0.3	0.3	1.7
$L_{\sqrt{-222}}^{\circledS 3}$	2.2	71.4	0.6	0.6	4.9
$L_{\sqrt{-222}}^{\circledS 4}$	3.8	286.3	0.9	1.3	13.2
$L_{\sqrt{-222}}^{\circledS 5}$	5.9	2252	1.6	2.5	34.5
$L_{\sqrt{-222}}^{\circledS 6}$	8.7	$*$	2.1	4.4	69.2
$L_{\sqrt{-222}}^{\circledS 12}$	60	$*$	10	41.5	$*$

Table 2: second order operators over $\mathbb{Q}(\sqrt{-222})(x)$

4.2 Timings for higher-order operators

For rational coefficients, we use the following operators from [18]:

$$F_{36} = D^3 + \frac{5(9x^2+14x+9)}{48x^2(x+1)^2}D - \frac{5(81x^3+185x^2+229x+81)}{432x^3(x+1)^3},$$

$$\begin{aligned} G_{168} \;=\; & D^3 + \frac{7x-4}{x(x-1)}D^2 \\ & + \frac{2592x^2-2963x+560}{252x^2(x-1)^2}D + \frac{57024x-40805}{24696x^2(x-1)^2}, \end{aligned}$$

$$\begin{aligned} PSL_3 \;=\; & xD(xD-\frac{1}{2})(xD-\frac{1}{4})(xD+\frac{1}{4})(xD-\frac{1}{8}) \\ & (xD-\frac{5}{8})(xD+\frac{1}{8})(xD+\frac{5}{8}) \\ & -x(xD+\frac{1}{3})(xD-\frac{1}{3}), \end{aligned}$$

as well as the following operator with larger coefficients from [15]:

$$\begin{aligned} \tilde{L} \;=\; & D^3 + \frac{128\,x^5 - 98\,x^4 + 108\,x^3 + 378\,x^2 - 486\,x + 162}{x(x-1)p(x)}D^2 \\ & + \frac{a(x)}{36\,x^2(x-1)^2p(x)}D - \frac{b(x)}{72\,x(x-1)^2p(x)}, \end{aligned}$$

where

$$p(x) = 72\,x^6 - 216\,x^5 + 280\,x^4 - 126\,x^3 + 27\,x^2 + 81\,x - 54,$$

$$\begin{aligned} a(x) \;=\; & 2304\,x^8 - 13464\,x^7 + 24872\,x^6 - 24840\,x^5 + 2106\,x^4 \\ & + 29565\,x^3 - 26514\,x^2 + 9477\,x - 1458, \end{aligned}$$

and

$$\begin{aligned} b(x) \;=\; & 4608\,x^6 - 28872\,x^5 + 67240\,x^4 - 115560\,x^3 \\ & + 131166\,x^2 - 78165\,x + 27135. \end{aligned}$$

	Σ^{IT}				MAPLE				
	Iteration		Kernel		Iteration		Kernel		`diffop`
	$\mathbb{Z}[x]$	$\mathbb{Q}(x)$	$\mathbb{Z}[x]$	$\mathbb{Q}(x)$	$\mathbb{Z}[x]$	$\mathbb{Q}(x)$	$\mathbb{Z}[x]$	$\mathbb{Q}(x)$	$\mathbb{Q}(x)$
${D_2}^{\circledS 4}$	0.04	0.2	0.3	0.3	0.08	0.2	0.4	0.3	1.2
${D_2}^{\circledS 12}$	0.8	3.3	6.2	4.9	0.7	1.9	5.8	3.2	37.4
${D_2}^{\circledS 20}$	3.5	15.5	42.1	28	3.0	8.1	28.4	14.7	398.8
${D_3}^{\circledS 4}$	0.06	0.2	0.3	0.3	0.1	0.1	0.4	0.2	1.3
${D_3}^{\circledS 12}$	0.8	3.4	7.1	6.4	0.7	2.1	6.1	3.6	38.1
${D_3}^{\circledS 20}$	3.7	16.2	50	31.7	3.2	9.1	30.3	16.6	490
${A_4}^{\circledS 6}$	0.02	0.1	0.1	0.3	0.07	0.07	0.4	0.2	0.8
${A_4}^{\circledS 12}$	0.1	0.7	0.4	2.1	0.2	0.4	1.6	0.8	4.9
${A_4}^{\circledS 20}$	0.3	2.9	3	5.8	0.5	1.4	6.5	2.7	24.4

Table 1: second order operators over $\mathbb{Q}(x)$

Finally an example with algebraic numbers in its (small) coefficients:

$$L_{\sqrt{2}} = (x^2+1)D^3 - \sqrt{2}D + x\,.$$

Table 3 contains the times needed for computing symmetric powers by the fraction–free ($\mathbb{Z}[x]$) and fractional ($\mathbb{Q}(x)$) kernel methods, while Table 4 contains the times needed for computing symmetric powers of $L_{\sqrt{2}}$ by the fraction–free (ffree) and fractional (frac) kernel methods.

	Σ^{IT}		MAPLE		
	$\mathbb{Z}[x]$	$\mathbb{Q}(x)$	$\mathbb{Z}[x]$	$\mathbb{Q}(x)$	`diffop`
${F_{36}}^{\circledS 5}$	8.3	10.3	10.9	3.9	61.6
${F_{36}}^{\circledS 6}$	31.4	25.7	29.4	14.3	340.7
${G_{168}}^{\circledS 5}$	15.4	26.3	14.0	8.3	60.8
${G_{168}}^{\circledS 6}$	105	79.1	35.6	21.0	500.7
$\tilde{L}^{\circledS 3}$	5.7	12	6.7	6.0	33.6
$\tilde{L}^{\circledS 4}$	61.2	118.2	197.8	144.6	1201.2
$\tilde{L}^{\circledS 5}$	895.6	1384	$*$	2635.3	$*$
${PSL_3}^{\circledS 2}$	163.1	$*$	218.5	197.9	200.4

Table 3: higher order operators over $\mathbb{Q}(x)$

	Σ^{IT}		MAPLE		
	ffree	frac	ffree	frac	`diffop`
${L_{\sqrt{2}}}^{\circledS 2}$	0.3	1.1	1.4	1.4	5.1
${L_{\sqrt{2}}}^{\circledS 3}$	33.7	$*$	9.6	76.5	81.4
${L_{\sqrt{2}}}^{\circledS 4}$	$*$	$*$	60.2	$*$	$*$

Table 4: higher order operators over $\mathbb{Q}(\sqrt{2})(x)$

The Σ^{IT} columns of Table 3 illustrate the cutoff when the fraction–free method starts becoming better than the fractional one in a system with canonical expanded forms for polynomials and fractions such as Σ^{IT} or `axiom` [7]: ${F_{36}}^{\circledS m}$ and ${G_{168}}^{\circledS m}$ have orders smaller than the generic $(m+1)(m+2)/2$ and their coefficients are rather small, so the fractional method remains faster. On the other hand, $\tilde{L}$ and its powers have quite larger coefficients, and the fraction–free method is then faster, as well as on the higher-order example PSL_3. This cutoff is further away on MAPLE because of two characteristics of the arithmetic in MAPLE:

- the preprocessing step of Section 3 consists of a simplex on a matrix of machine-integers. Machine integers are however not accessible in the MAPLE user language, causing the preprocessing step to take a more significant portion of the computing time than in compiled languages, thereby pushing the cutoff further away;
- the `normal` function in MAPLE keeps the numerators and denominators of elements of $\mathbb{Q}(x)$ factorized, and this allows an appropriately coded Gaussian elimination to take advantage of the many cancellations that happens on the matrices that arise in the symmetric powers computations, thereby giving an extra advantage to the fractional method; At the same time, the intermediate results tend to become expanded during the fraction-free elimination, giving it an additional disadvantage;

Both of the above points disappear when the constant field is a proper extension of $\mathbb{Q}$, as illustrated by Table 4, where the fraction–free method is significantly faster than the fractional one on all systems.

Conclusions

For second order operators, the fraction–free iteration method of Theorem 2 is the most efficient way of computing symmetric powers, and it turns the algorithm of [16] into a very efficient second order linear ordinary equation solver. For higher-order operators, the fraction–free kernel method of Section 2 is more efficient than the fractional method whenever the order of the operator or its coefficients are sufficiently large, or whenever the underlying constant field is not $\mathbb{Q}$, in which case it seems to be the only practical method. Its performance on third-order operators with smaller coefficients in $\mathbb{Q}(x)$ seems to indicate that a polyalgorithm is not needed, and that the fraction–free method can be used in general.

In addition, the techniques of applying a fraction–free iteration of a multiple of D (Lemma 2) followed by the preprocessing of Section 3 are applicable to other problems that involve computing a cyclic vector of a matrix constructed from an operator, such as the computation of exterior powers of differential or difference operators [3], or the factorization of completely reducible differential operators [12], although that last problem can also be solved without computing a cyclic vector.

Acknowledgements

We are thankful to Marco Petkovšek, Loic Pottier and Emo Welzl for their help and advice on the linear programming aspects of the preprocessing step.

References

[1] Bronstein, M. On radical solutions of linear ordinary differential equations. In *Proceedings of the Nijmegen Cathode Workshop* (1995), T. Levelt, Ed., pp. 23–24.

[2] Bronstein, M. $\Sigma^{\rm IT}$– a strongly-typed embeddable computer algebra library. In *Proceedings of DISCO'96* (1996), LNCS 1128, Springer, pp. 22–33.

[3] Bronstein, M., and Petkovšek, M. An introduction to pseudo–linear algebra. *Theoretical Computer Science 157* (1996), 3–33.

[4] Char, B., Geddes, K., Gonnet, G., Benton, L., Monagan, M., and Watt, S. *Maple V Library Reference Manual.* Springer, New York, 1991.

[5] Geddes, K., Czapor, S., and Labahn, G. *Algorithms for Computer Algebra.* Kluwer Academic Publishers, Boston, 1992.

[6] Hénon, M. A mechanical model for the hitchcock problem. *Comptes Rendus de l'Académie des Sciences, Paris, Série I, Mathématiques 321* (1995), 741–745.

[7] Jenks, R., and Sutor, R. *Axiom – The Scientific Computation System.* Springer, New York, 1992.

[8] McClellan, M. The exact solution of systems of linear equations with polynomial coefficients. *Journal of the ACM 20* (1973), 563–588.

[9] Schrijver, A. *Theory of Linear and Integer Programming.* Wiley, 1986.

[10] Schwartz, J. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM 27* (1980), 701–717.

[11] Singer, M. Liouvillian solutions of $n^{\rm th}$ order homogeneous linear differential equations. *American Journal of Mathematics 103* (1980), 661–682.

[12] Singer, M. Testing reducibility of linear differential operators: a group theoretic perspective. *Applicable Algebra in Engineering, Communication and Computing 7* (1996), 77–104.

[13] Singer, M., and Ulmer, F. Galois groups for second and third order linear differential equations. *Journal of Symbolic Computation 16* (1993), 1–36.

[14] Singer, M., and Ulmer, F. Liouvillian and algebraic solutions of second and third order linear differential equations. *Journal of Symbolic Computation 16* (1993), 37–73.

[15] Singer, M., and Ulmer, F. Linear differential equations and products of linear forms. In *Proceedings of MEGA'96* (to appear).

[16] Ulmer, F., and Weil, J.-A. Note on kovacic's algorithm. *Journal of Symbolic Computation 22* (1996), 179–200.

[17] van Hoeij, M. *Factorization of Linear Differential Operators.* Computer science dissertation, Nijmegen, 1996. The package `diffop` is available from `http://www-math.sci.kun.nl/math/compalg/diffop/`.

[18] van Hoeij, M., and Weil, J.-A. An algorithm for computing invariants of differential galois groups. In *Proceedings of MEGA'96* (to appear).