J. Symbolic Computation (1993) **16**, 423–443

Computing Closed Form Solutions of First Order ODEs Using the Prelle-Singer Procedure

YIU-KWONG MAN†

School of Mathematical Sciences, QMW, University of London, U.K.

(Received 26 April 1993)

The Prelle-Singer procedure is an important method for formal solution of first order ODEs. Two different REDUCE implementations (PSODE versions 1 & 2) of this procedure are presented in this paper. The aim is to investigate which implementation is more efficient in solving different types of ODEs (such as exact, linear, separable, linear in coefficients, homogeneous or Bernoulli equations). The test pool is based on Kamke's collection of first order and first degree ODEs. Experimental results, timings and comparison of efficiency and solvability with the present REDUCE differential equation solver (ODESOLVE) and a MACSYMA implementation (ODEFI) of the Prelle-Singer procedure are provided. Discussion of technical difficulties and some illustrative examples are also included.

1. Introduction

Prelle and Singer (1983) proved that if a system of differential equations has an elementary first integral (i.e. a first integral expressible in terms of exponentials, logarithms and algebraic functions) then it must be of a very special form. For example, if a two dimensional autonomous system

$$\dot{x} = P(x, y), \quad \dot{y} = Q(x, y) \tag{1.1}$$

where P and Q are polynomials with coefficients in the complex field $\mathbb{C}$, has an elementary first integral, it has one of the form

$$F(x, y) = v_0(x, y) + \sum_i c_i \log(v_i(x, y))$$

where the c_i are constants and the v_i are algebraic functions of x and y. The first order ordinary differential equation associated with (1.1) is

$$P(x, y)\frac{dy}{dx} = Q(x, y).$$

One can prove (see Prelle and Singer (1983)) that if an elementary first integral exists, then we can find an integrating factor R with $R^n \in \mathbb{C}(x, y)$ for some nonzero integer n, such that $\frac{\partial RP}{\partial x} + \frac{\partial RQ}{\partial y} = 0$ and hence we can solve the differential equation by quadrature.

† This research is supported by the ORS awards committee in U.K.

0747–7171/93/110423+21 $08.00/0

Let $R = \prod_i f_i^{n_i}$ where f_i are irreducible polynomials and n_i are nonzero integers. Since R is an integrating factor, we have $f_i | Df_i$ where D is the differential operator $P\frac{\partial}{\partial x} + Q\frac{\partial}{\partial y}$. Conversely, Darboux showed that if one could find all irreducible f such that $f|Df$, then one could decide if such an integrating factor R exists. It is known (see Jouanolou (1979) or Singer (1992)) that if f is irreducible and $f|Df$, then deg $f \leq N$ for some integer N. But how we can find such N effectively is still an unsolved problem and that is why the Prelle-Singer method is only a semi-decision procedure. However one can use the procedure outlined in Prelle and Singer (1983) by arbitrarily assigning a bound to the degree of the f's such that $f|Df$. The drawback is that the method sometimes may not find a first integral even if it exists. This approach has been implemented in MACSYMA (see Shtokhamer et al (1986)) with surprising success and the present implementation in REDUCE (see below) also uses such an approach. From now on, I will use the abbreviation ODEFI to refer to the MACSYMA program, SGC to refer to the experimental results mentioned in Shtokhamer et al (1986) and PSODE (versions 1 & 2) to refer to the REDUCE programs reported in this paper.

2. Prelle-Singer Procedure

In this section, a brief review of the Prelle-Singer Procedure is given. But the actual implementation in REDUCE is quite different from ODEFI and this is one of the reasons why there are discrepancies in degree bounds and number of solved examples between PSODE and SGC (see details in later sections). Below, D, P, Q, R and f_i have the same meanings as before.

Prelle-Singer Procedure.

1 Set $N = 1$.

2 Find all monic irreducible polynomials f_i such that deg $f_i \leq N$ and $f_i | Df_i$.

3 Let $Df_i = f_i g_i$.† Decide if there are constants n_i, not all zero, such that

$$\sum_{i=1}^{m} n_i g_i = 0.$$

If such n_i exist then $w = \prod_{i=1}^{m} f_i^{n_i}$ is a first integral and we return $w - c = 0$, where c is an arbitrary constant, as the general solution to the first order differential equation $P(x,y)\frac{dy}{dx} = Q(x,y)$. If no such n_i exist then go to the next step.

4 Decide if there are constants‡ n_i, such that

$$\sum_{i=1}^{m} n_i g_i = -\left(\frac{\partial P}{\partial x} + \frac{\partial Q}{\partial y}\right).$$

If such n_i exist, then $R = \prod_{i=1}^{m} f_i^{n_i}$ is an integrating factor for the given differential equation and we return the general solution $w - c = 0$, where c is an arbitrary constant

† Since this is analogous to an eigenvalue problem $Ax = \lambda x$ in linear algebra, we sometimes call f_i an eigenpolynomial.

‡ n_i can be all zeros here. In this case, it means the integrating factor is equal to unity, i.e. the equation was already exact. In Prelle and Singer (1983), it was mentioned that one needs to decide if there are rational n_i at this stage, but in practice we discover that it also works for non-rational n_i, so we say n_i are constants here.

and w is determined either by

$$\int RQ\,dx - \int \left(RP + \frac{d}{dy}\int RQ\,dx \right) dy$$

or

$$-\int RP\,dy + \int \left(RQ + \frac{d}{dx}\int RP\,dy \right) dx.^{\dagger}$$

If no such n_i exist, then go to the next step.

5 Increase the value of N by 1. If N is greater than the preset bound then return failure, otherwise repeat the whole procedure.

Note that this procedure as originally described in Prelle and Singer (1983) assumed P, Q are polynomials in x and y over $\mathbb{C}$. The extension of it to solve differential equations with transcendental terms or algebraic terms (heuristically) was described in Shtokhamer et al (1986). The main idea is to regard the different transcendental terms or algebraic terms that appear in P and Q as new variables which are then used to construct the polynomial(s) f_i (with undetermined coefficients) in addition to the original variables x and y (e.g. see Example 3.2). But since we are working on differential field extensions over $\mathbb{C}(x,y)$, we must guarantee that all the derivatives with respect to the derivations $\frac{\partial}{\partial x}$ and $\frac{\partial}{\partial y}$ lie in the same differential field extension. Therefore one needs to determine all the derivatives of the new variables. If the derivatives obtained are new transcendental terms or algebraic terms then consider them as new variables too and repeat the differentiations again until no more new variables can be derived. For instance, if we are given a differential equation $\frac{dy}{dx} + y\cos x = e^{-\sin x}$, then the transcendental terms are $\cos x$ and $e^{\sin x}$. So the new variables obtained will be $\{\cos x, \sin x, e^{\sin x}\}$.[‡] However, if we regard $t = \tan(\frac{x}{2})$ as a transcendental generator of $\cos x$ and $\sin x$, then we can also use $\{t, e^{\frac{2t}{1+t^2}}\}$ as our new variables (similarly if we express $\cos x$, $\sin x$ in terms of exponentials). In the latter case, we will have just 4 variables (including x and y) to work on and in general this will speed up the subsequent calculations. In addition, we need to modify P and Q in step 2 by $P \leftarrow P \times den\_lcm$, $Q \leftarrow Q \times den\_lcm$ where den_lcm means the lcm of all the denominators of the derivatives of the new variables and the right hand side of the equation in step 4 is replaced by $-(\frac{\partial P}{\partial x} + \frac{\partial Q}{\partial y}) \times den\_lcm$. In Shtokhamer (1988), the D operator is modified to $\sum_{i=1}^{n}(P\frac{\partial t_i}{\partial x} + Q\frac{\partial t_i}{\partial y}) \times den\_lcm \times \frac{\partial}{\partial t_i}$ where $t_1 = x, t_2 = y$ and the remaining t_i's ($i \geq 3$) are the new variables obtained, but one can see that this is equivalent to modifying P, Q as above. The advantage of our approach here is to keep the D operator in simple form. Now once all these small steps have been done, we can apply the Prelle-Singer procedure to solve differential equations with transcendental or algebraic terms. For more details, see section 3.2 where two examples are given.

† Mathematically, they are equal expressions, but this certainly does not mean that the actual computations of the integrals involved are equally easy.

‡ It is because $\frac{d}{dx}e^{\sin x} = e^{\sin x}\cos x$ and $\frac{d}{dx}\cos x = -\sin x$ and further differentiations give us no extra transcendental terms.

3. Implementations

3.1. PSODE (VERSION 1)

The implementation (PSODE version 1) of the Prelle-Singer Procedure (abbreviated as P_S from now on) in REDUCE has three main procedures

1 *excoef(poly,varlist)* — Given a multivariate polynomial, this procedure will return the extracted coefficients with respect to the variables specified in *varlist*.
2 *ps_1(P,Q,varlist,degreebound)* — Given P, Q in the differential equation $P(x,y)\frac{dy}{dx} = Q(x,y)$ and a preset degree bound, this procedure will return all the monic irreducible polynomials f_i and the associated polynomials g_i such that $Df_i = f_i g_i$.
3 *ps_2(P,Q,gflist,varlist)* — This procedure uses the g_i and f_i polynomials (called *gflist*) returned by *ps_1* to construct a first integral (if step 3 of P_S succeeds) or an integrating factor (if step 4 of P_S succeeds); otherwise it returns failure.

The whole idea behind the implementation is to treat step 2, step 3 and step 4 of P_S as problems of solving systems of algebraic equations — they come separately from the extraction of coefficients from $Df_i = f_i g_i$ (step 2), $\sum_{i=1}^{m} n_i g_i = 0$ (step 3) and $\sum_{i=1}^{m} n_i g_i = -(\frac{\partial P}{\partial x} + \frac{\partial Q}{\partial y})$ (step 4). Therefore, PSODE uses *solve* and *groebner* in REDUCE†. When an integrating factor R is found, then the evaluation of integrals is done by calling *int*‡ (top level function call for the integrator in REDUCE). In general, the systems of equations arising from step 2 are more complicated (they contain non-linear equations) than those arising from steps 3 or 4. In fact, the implementation of steps 3 and 4 is quite straightforward compared with step 2, and in most cases, the time complexity is less than that in step 2. Because of these reasons, more detailed descriptions of the procedure *ps_1* are provided below.

ps_1$(P, Q, varlist, degreebound)$.

$S_{fg} \leftarrow \emptyset$; $k \leftarrow 1$
while $k \leq degreebound$ do

1 construct all monic polynomials (with undetermined coefficients) f_i of degree $\leq k$ (see explanations below).
2 for each f_i do

- calculate $Df_i = P\frac{\partial f_i}{\partial x} + Q\frac{\partial f_i}{\partial y}$.
- $lt\_f_i \leftarrow$ leading term of f_i ; $lt\_Df_i \leftarrow$ leading term of Df_i
- if lt_f_i divides lt_Df_i then
  * $n \leftarrow \deg Df_i - \deg f_i$
  * if $n < 0$, then $n \leftarrow 0$.
  * $g_i \leftarrow$ construct a polynomial (with undetermined coefficients) of degree n
  * $eqns \leftarrow excoef(f_i g_i - Df_i, varlist)$
  * split *eqns* into two sets: *geqns* ← equations which come from multiplying g_i by lt_f_i.§ *feqns* ← rest of the equations.

† SGC used a self-written equation solver in addition to the standard solver in MACSYMA.

‡ PSODE uses ALGINT (a package in REDUCE which can handle integration of algebraic functions) rather than the standard integrator by default.

§ e.g. If $lt\_f_i = xy$ and $g_i = g_1 + g_2 x$, then geqns will consist of all extracted coefficients of the terms xy and $x^2 y$ in the product $f_i g_i - Df_i$.

* *geqns* ← solve *geqns* in terms of the coefficients of f_i;
  feqns ← *sub*(*geqns*, *feqns*)
* calculate *groebner*(*feqns*) with respect to the undetermined coefficients in f_i
* if the system is consistent, then all unknown coefficients in f_i and g_i can be determined; otherwise jump the next step and try the next f_i.
* if f_i is irreducible[†], then $S_{fg} \leftarrow (f, g) \cup S_{fg}$.

3 end_for_loop

$k \leftarrow k + 1$
end_while_loop
return S_{fg}

In Prelle and Singer (1983), a Darboux bound is mentioned, which is defined as $2 + m(m+1)/2$, where $m = \max(\deg P, \deg Q)$ and P, Q are pure polynomials in x and y. This is used to inform us that in case the number of elements in S_{fg} is greater than or equal to this bound, then we are guaranteed to have a rational first integral, which means the g_i's are linearly dependent (i.e. non-zero n_i's exist in step 3) and we can obtain a solution without the need to integrate at all. But since we allow P, Q to contain some transcendental terms or algebraic terms, we cannot use this bound directly and need to go through step 3 of the Prelle-Singer procedure to check for the dependencies of the g_i's.[‡] The step constructing f_i and g_i with undetermined coefficients is done like this: for instance, if $k = 1$, two polynomials f_i can be constructed, namely, $a_1 + x$ and $a_1 + a_2 x + y$, where a_1, $a_2 \in \mathbb{C}$ are undetermined coefficients. (Note: we must consider the *ordering* of x and y in the actual program). When we construct g_i, if the leading term of g_i is known, say, xy, then g_i is defined as $b_1 + b_2 x + b_3 y + b_4 x^2 + b_5 xy$; otherwise it is defined as $b_1 + b_2 x + b_3 y + b_4 x^2 + b_5 xy + b_6 y^2$ — a general bivariate polynomial of deg 2, where $b_i \in \mathbb{C}\ (1 \leq i \leq 6)$ are undetermined coefficients. The reason for splitting *eqns* into *geqns* and *feqns* is that we want to determine all the unknown coefficients in f_i first and then determine those in g_i. Normally, *feqns* is a non-triangular system of non-linear algebraic equations, while *geqns* is a triangular system of linear equations. More details can be found in the examples in section 3.

3.2. Examples

This section will give some examples to illustrate the procedures in finding a general solution of a first order ordinary differential equation by P_S. Some examples on how to run PSODE in REDUCE can be found in the appendix.

Example 3.1. *Suppose we want to solve a linear ordinary differential equation* $(x^2+1)\frac{dy}{dx} + xy = x(x^2+1)$. *First we define a differential operator* D *as* $(x^2+1)\frac{\partial}{\partial x} + (x^3 + x - xy)\frac{\partial}{\partial y}$. *For* $N = 1$, *there are 2 monic polynomial candidates, namely, (1)* $f = f_1 + x$ *and (2)* $f = f_1 + f_2 x + y$. *For (1),* $Df = x^2 + 1$. *Since the leading term of* f *divides that of* Df, *then we can define* $g = g_1 + g_2 x$. *Putting this into* $Df = fg$ *and equating coefficients, we get a consistent system of equations. In this case,* $geqns = \{g_2 - 1, g_1 + f_1 g_2\}$ *and* $feqns =$

[†] This can be checked by dividing f_i by each element in S_{fg}. If none of them divide f_i, then f_i is irreducible.

[‡] It may be one of the reasons why Shtokhamer et al (1986) did not mention this bound explicitly in the description of this procedure.

$\{f_1 g_1 - 1\}$. *Solving this system, we obtain* $f_1 = \pm i$, $g_1 = \mp i$ *and* $g_2 = 1$. *Therefore we get two sets of* f *and* g, *namely,* $\{f = x + i,\ g = x - i\}$ *and* $\{f = x - i,\ g = x + i\}$. *Next, we consider (2). In this case,* $Df = (x^2+1)f_2 + (x^3 + x - xy)$. *But since the leading term of* f *cannot divide* x^3 *(which is the leading term of* Df *in total degree ordering), there is no solution for this case. So, we have to consider*[§] $n_1(x-i) + n_2(x+i) = -(\frac{\partial P}{\partial x} + \frac{\partial Q}{\partial y})$, *where* $P = x^2 + 1$ *and* $Q = x^3 + x - xy$. *Solving, we get* $n_1 = n_2 = -\frac{1}{2}$ *and hence an integrating factor* $R = (x+i)^{-\frac{1}{2}}(x-i)^{-\frac{1}{2}} = (x^2+1)^{-\frac{1}{2}}$ *is obtained. Putting it into one of the formulas in step 4 of P_S, we get the general solution* $c = -y\sqrt{x^2+1} + \frac{1}{3}(x^2+1)^{\frac{3}{2}}$, *where* c *is an arbitrary constant.*

EXAMPLE 3.2. *Suppose we want to solve a separable equation* $\frac{dy}{dx} = \log x$. *Here we have* $P = 1$ *and* $Q = \log x$. *Since* $\log x$ *is a transcendental term and* $\frac{d}{dx}\log x = \frac{1}{x}$, *no extra transcendental term can be obtained by further differentiations and we have only three variables to work on, namely* $\{x, y, \log x\}$. *The differential operator* D *is defined as* $xP\frac{\partial}{\partial x} + xQ\frac{\partial}{\partial y} = x\frac{\partial}{\partial x} + x\log x\frac{\partial}{\partial y}$, *where the extra* x *comes from the denominator of* $\frac{d}{dx}\log x$. *For* $N = 1$, *there are 3 monic polynomial candidates, namely, (1)* $f = f_1 + x$, *(2)* $f = f_1 + f_2 x + y$ *and (3)* $f = f_1 + f_2 x + f_3 y + \log x$. *For (1),* $Df = x$, *so it is obvious that* $g = 1$ *and* $f = x$. *For (2),* $Df = f_2 x + x\log x$, *but the leading term of* f *cannot divide* $x\log x$, *so* f *cannot divide* Df. *For (3),* $Df = 1 + f_2 x + f_3 x\log x$. *Define* $g = g_1 + g_2 x$ *and solve for the unknown coefficients in* $Df = fg$. *But the system of equations obtained by equating coefficients is inconsistent. In this case,* $geqns = \{g_2 - f_3, g_1\}$ *and* $feqns = \{f_1 g_1 - 1, g_1 f_2 + f_1 g_2 - f_2, g_1 f_3, f_2 g_2, f_3 g_2\}$. *Therefore, there is only one set of* f *and* g *polynomials. Next, consider* $n_1 g = -(\frac{\partial P}{\partial x} + \frac{\partial Q}{\partial y})x$.[†] *Solving, we get* $n_1 = 0$ *and an integrating factor* $R = 1$ *(as expected). Putting it into one of the formulas in step 4 of P_S, we obtain the general solution* $c = x\log x - x - y$, *where* c *is an arbitrary constant.*

Compared with pattern matching techniques (i.e. identifying appropriate type and then applying standard techniques), P_S will certainly not be so efficient in some cases, but it provides a systematic way of finding an integrating factor (provided the solution is elementary) without a prior knowledge of what type the equation is. In fact, by doing experiments, we can see that it is an important and useful algorithmic tool in solving first order differential equations. Compared with ODESOLVE (the differential equation solver in REDUCE), PSODE can solve far more equations, based on the Kamke's (1959) collection of ODEs (see table 6 below).

3.3. PSODE (VERSION 2)

As mentioned before, PSODE uses the packages GROEBNER and ALGINT as support. Experimental results indicated that the computational effort in step 2 is usually much more than that in steps 3 & 4. We have seen in the above examples that not every case within each degree bound N can lead to solution(s), and quite often we have to 'pass' a lot of nasty inconsistent cases before coming to a solvable case for the polynomials f and g. PSODE (version 1) relies solely on GROEBNER to detect inconsistency as well as

§ We do not go through step 3 of P_S here because we can recognize that $x - i$ and $x + i$ are linearly independent.

† Again, the extra x comes from the denominator of $\frac{d}{dx}\log x$.

solving consistent systems of equations, which is sometimes computationally expensive. Can we have an alternative way of doing it ? The following description provides such an alternative. The notations used below have the same meaning as before and the new notations will be introduced in the context.

new_ps_1$(P, Q, varlist, degreebound)$.

$S_{fg} \leftarrow \emptyset$; $k \leftarrow 1$
while $k \leq degreebound$ do

1 construct all monic polynomials (with undetermined coefficients) f_i of degree $\leq k$.
2 for each f_i do

- calculate $Df_i = P\frac{\partial f_i}{\partial x} + Q\frac{\partial f_i}{\partial y}$.
- $g_i \leftarrow 0$
- *indivisible* $\leftarrow$ *false*
- $feqns \leftarrow \emptyset$
- while not *indivisible* and $Df_i \neq 0$ do
  - * if lt_f_i divides lt_Df_i then
    - · $g_i \leftarrow g_i + \frac{lt\_Df_i}{lt\_f_i}$
    - · $Df_i \leftarrow Df_i - f_i \times \frac{lt\_Df_i}{lt\_f_i}$
  - * else if lc_Df_i (lc_Df_i is a constant then *indivisible* $\leftarrow$ *true*
  - * else if lc_Df_i contains one f_i variable (say var) only then
    - · $s \leftarrow solve(lc\_Df_i, var)$
    - · $f_i \leftarrow sub(s, f_i)$
    - · $g_i \leftarrow sub(s, g_i)$
    - · $feqns \leftarrow sub(s, feqns)$
  - * else
    - · $feqns \leftarrow lc\_Df_i \cup feqns$
    - · $Df_i \leftarrow Df_i - lt\_Df_i$
- end_while_loop
- if not *indivisible* and $feqns \neq \emptyset$ then
  - * calculate $groebner(feqns)$ with respect to the undetermined coefficients in f_i
  - * if the system is consistent, then all unknown coefficients in f_i and g_i can be determined; otherwise jump the next step and try the next f_i.
- if not *indivisible* and f_i is irreducible, then $S_{fg} \leftarrow (f, g) \cup S_{fg}$

3 end_for_loop

$k \leftarrow k + 1$
end_while_loop
return S_{fg}

The main difference between the current approach and the previous approach is that there is an inner *while_loop* inside the procedure which is basically performing 'long divisions' (i.e. Df_i divided by f_i). A few remarks concerning such an approach are listed below:

1 There is no need to construct g with unknown coefficients. g is just the quotient of Df_i over f_i (in the divisible case).

2 There is no need to split the determining system of equations into 2 parts (i.e. *feqns* and *geqns*); only one system of *feqns* is necessary.
3 An inconsistent case can sometimes be detected without calling 'groebner', namely the case when lc_Df_i is a constant.
4 The substitution part can help to reduce the size of the system of *feqns* and may even determine f and g completely sometimes, namely in the case when $feqns = \emptyset$ after the inner *while_loop*.
5 It is better to represent the polynomials f_i and Df_i in distributed forms† in this approach since we need to use lt_f_i and lc_Df_i quite often.

PSODE (version 2) was implemented using such an approach and the internal representations of f_i and Df_i (in RLISP) are distributed polynomials. By performing computer experiments and taking timings, this approach proves useful and more efficient than the previous approach in most cases (see section 4). Another useful ansatz is that it is *not* really necessary to separate the steps 2–4 in the P_S procedure — we can check the dependence of the g_i polynomials or try to construct an integrating factor whenever a new pair of $\{f_i, g_i\}$ has been found. The reason is that we may probably skip a lot of inconsistent cases (some may be nasty systems of polynomial equations which GROEBNER will take a long time to reduce to $\{1\}$) within a given value of N. If we look back at the examples 1 & 2 above, we can see that this strategy can save us one step in the first example and two steps in the second example. For this reason, such a strategy has been incorporated into PSODE (version 2) as well.

3.4. A SIMPLE EXAMPLE

Suppose we want to solve the differential equation $(2x^2y - x)\frac{dy}{dx} = 2xy^2 + y$ by using the 'division' method. Firstly, we define a differential operator D as $(2x^2y - x)\frac{\partial}{\partial x} + (2xy^2 + y)\frac{\partial}{\partial y}$ and assume we are using total degree ordering on x and y. For $N = 1$, there are 2 monic polynomial candidates, namely (1) $f = f_1 + x$ and (2) $f = f_1 + f_2x + y$. For (1), $Df = 2x^2y - x$, $lt\_f = x$ and $lt\_Df = 2x^2y$, so

$$g \leftarrow 2xy \quad \text{and} \quad Df \leftarrow -2f_1xy - x.$$

Next, we have $lt\_Df = -2f_1xy$,

$$g \leftarrow 2xy - 2f_1y \quad \text{and} \quad Df \leftarrow 2f_1^2y - x.$$

Since $lt\_Df = 2f_1^2y$ and x divides lt_Df only if $f_1 = 0$,

$$g \leftarrow 2xy \quad \text{and} \quad Df \leftarrow -x.$$

Hence one more division leads to

$$g \leftarrow 2xy - 1 \quad \text{and} \quad Df \leftarrow 0.$$

Therefore we obtain $f = x$ and $g = 2xy - 1$ in this case. For (2), we have $Df = 2xy^2 + 2f_2x^2y + y - f_2x$, $lt\_f = y$ and $lt\_Df = 2xy^2$, so

$$g \leftarrow 2xy \quad \text{and} \quad Df \leftarrow -2f_1xy + y - f_2x.$$

† e.g. $2x^2 + x(y^2 + 1) + y$ is a recursive form (assuming a total degree ordering of x and y) while $xy^2 + 2x^2 + x + y$ is a distributed form

Next we have $lt\_Df = -2f_1xy$ and

$$g \leftarrow 2xy - 2f_1x \quad \text{and} \quad Df \leftarrow 2f_1f_2x^2 + y + (2f_1^2 - f_2)x.$$

Since $lt\_Df = 2f_1f_2x^2$ and y divides lt_Df only when $f_1f_2 = 0$,

$$feqns \leftarrow \{f_1f_2\} \quad \text{and} \quad Df \leftarrow y + (2f_1^2 - f_2)x.$$

Next we have $lt\_Df = y$, so

$$g \leftarrow 2xy - 2f_1x + 1 \quad \text{and} \quad Df \leftarrow (2f_1^2 - 2f_2)x - f_1.$$

Now it is easy to see that we need to assign

$$feqns \leftarrow \{f_1f_2\} \cup \{2f_1^2 - 2f_2, f_1\} \quad \text{and} \quad Df \leftarrow 0.$$

Solving the $feqns$ gives $f_1 = f_2 = 0$, so we obtain $f = y$ and $g = 2xy + 1$. Since $2xy - 1$ and $2xy+1$ are linearly independent, so consider $n_1(2xy-1)+n_2(2xy+1) = -(\frac{\partial P}{\partial x}+\frac{\partial Q}{\partial y})$, where $P = 2x^2y - x$ and $Q = 2xy^2 + y$. Solving we get $n_1 = n_2 = -2$ and hence an integrating factor $R = (xy)^{-2}$. Performing the integration steps, we can get the general solution $c = 2\log x - 1/xy - 2\log y$, where c is an arbitrary constant.

4. Comparison of Efficiencies

In this section, ten examples are chosen from Kamke (1959) and comparison of efficiencies are done for PSODE (version 1 & 2), ODEFI and ODESOLVE†. All testings are done interactively on a Sun Sparcstation II and the timings are in terms of milliseconds. Those examples which a particular program failed to solve or no answer was returned in more than 30 minutes will be marked with a **F** and an asterisk repectively. The REDUCE version is 3.4.1 and the MACSYMA version is 417.1. On can recognize that the second version of PSODE is faster than the first version in general. The next observation is that although ODESOLVE failed in most of these examples, we can see that it is significantly more efficient in those cases which it can solve. This suggests that if we want to build up a general, efficient and powerful first order ODE solver, then pattern-matching is always worthwhile to try first ! In table 1, PS(I) and PS(II) refer to the first and second version of PSODE respectively. For convenience, I have used *intfactor* as an abbreviation for 'integrating factor'.

5. Experimental Results

From this section onwards, the name PSODE will mean PSODE (version 2) unless stated otherwise. Table 2 is a summary of the experimental results obtained by running PSODE through 331 Kamke examples. Altogether there are 367 first order and first degree Kamke examples: 210 of them are polynomial type, 81 of them are transcendental type, 40 of them are algebraic type and 36 of them are non-explicit type (i.e. no explicit functions are given in the equations, only arbitrary functions $f(x)$ or $h(y)$ etc). Therefore the meaningful test set consists of 331 Kamke examples only. Whenever an answer obtained involves one or more unevaluated integrals, I will regard it as partially solved.

† Putting ODESOLVE here is just for reference purposes since it is based on pattern-matching rather than the P_S procedure.

Table 1. Comparison of Efficiencies

Example	Type	PS(I)	ODEFI	PS(II)	ODESOLVE
k41	Abel	4505	*	2227	F
k93	Linear	21641	15483	2278	323
k120	unknown	3213	10516	1513	F
k163	Riccati	1904	19583	1479	F
k182	Riccati	5032	36583	4964	F
k220	reducible to Linear	1717	18250	850	204
k249	reducible to Bernoulli	4998	*	6885	F
k275	special intfactor	629	5866	510	F
k291	reducible to Linear	1632	14083	1258	F
k341	Exact	7582	*	1139	340

In summary, with a degree bound $N \leq 4$,[‡] we discovered that 241 of them can be completely solved while 23 of them can be partially solved. Among these solvable cases, 161 of them are of polynomial type, 69 of them are of transcendental type and 34 of them are of algebraic type. More details about the partially solved examples can be found in table 6 and the timings for the examples solvable by PSODE are tabulated in table 3 (all timings are in terms of milliseconds and the garbage collection time is excluded).

6. Discrepancies between PSODE and SGC

In this section, the discrepancies in results obtained by PSODE and SGC[†] are discussed. They arise mainly from two causes: the first one is that the two implementations were done in different computer algebra systems — the equation solver and the standard integrator in the two systems are different; and the second one is that the actual approach is somewhat different — PSODE uses the Groebner Package while SGC did not, and SGC incorporates other methods for finding special integrating factors while PSODE did not. As a result, two kinds of discrepancies arise: degree bound set and missing examples. They are mentioned in the next two subsections.

6.1. Degree Bounds

Table 4 shows the differences in degree bound set in PSODE and SGC. Comparisons are done for the polynomial type of Kamke examples only since SGC did not mention the explicit degree bound set for transcendental or algebraic cases.

‡ Setting $N > 4$ will normally lead to a too large search space for the P_S procedure step 2, so the default maximal value of N in PSODE is 4, but the user can always alter it by specifying another value of N in the fourth argument of the input (see appendix).

† We are comparing the results obtained by PSODE with those results tabulated in Shtokhamer et al (1986) here, so the name SGC is used instead of the program's name ODEFI.

Table 2. Experimental Results

Type	N	Kamke examples solvable by PSODE	Total
Polynomial	1	12,17,19,23,26,29,31,39,94,96-98,101-103,130 135-138,148-150,153,155,156,158,160-162,165,167 171,174,175,177,178,180,183,188,204,207,210 213-216,218,221-229,231,232,236,238-247,249 252,254-256,258,260-264,270-273,275-277 279-282,284-287,290,291,293-303,306-310 313,315,317-319,321-325,327-330	128
	2	15,41,104,140,141,143,163,170,182,186,187,217 220,257,304,312	16
	3	42,106,151,172,181,248,274,289,316,320	10
	4	44,142,168,173,251,288,305	7
Transcendental	1	2-4,6-9,75-78,90-93,117-120,122-125,131,132 134,152,154,159,193,194,196-200,233,259,267 283,314,341,342,344-349,352-359,361-364	61
	2	32,81,108,109,195,208,278	7
	3	211	1
Algebraic	1	57-68,89,112-116,190-192,209,332-340,360	32
	2	38,52	2

6.2. Missing Examples

By "missing examples", we mean those Kamke examples which should be solvable by P_S method, but unfortunately do not appear in the solvable list of the test results. They are missed by PSODE due to technical difficulties and we will discuss the underlying problems in more detail in section 9. In table 5, N means the degree bound set reported in the successful program. Since there is no explicit degree bound N given in SGC for the example k350, we just put a '-' in such a case. Again, we can only give information for the polynomial or transcendental types due to the lack of reported results for the algebraic cases in SGC.

7. Applicability of PSODE

In this section, the applicability of PSODE in solving first order ODEs is investigated. For this purpose, the different types of equations are tabulated in tables 7-8 below. Those equations which cannot be solved by PSODE are entered in table 10. An analysis of the number of solvable examples by Kamke, PSODE and ODESOLVE can be found in table 6. Since there is no explicit result given by SGC for the algebraic type examples, so I will enter '-' for those non-comparable cases. The column for success rate (I) means the success rate if we consider all polynomial, transcendental and algebraic types together while the column for success rate (II) means the success rate if we consider polynomial and transcendental types only. In tables 7-8, there is a column called *adjusted success rate* — which means the success rate if we do not consider those Kamke examples which are known to be non-elementary or those having no explicit answer in Kamke (1959). The classification of types follows Kamke closely with only slight modifications. The

Table 3. Timings for solvable examples by PSODE

Time	$N = 1$	$N = 2$	$N = 3$	$N = 4$	Total
0-1000	2,12,17,19,26,29,57,91,94,96,101-103 118,119,130,131,135-138,148-150,153 155,156,158-161,165,171,174,175,177 193,194,207,210,218,221-229,232,236 238-247,252,254-256,258,260-263,270 271,273,275-277,280-282,284,293,295 297-303,309,313,317,318,321,323,324 328,344,353,354	170,217			103
1000-2000	3,4,6,8,9,23,31,58,89,97,112,117,120 123-125,132,134,162,167,196-198,204 209,214-216,264,267,272,279,283,286 287,291,294,306-308,314,315,319,330 334,337,341,345,349	15,108 109,140 143,163 220			57
2000-3000	75,76,98,113,114,122,180,183,213,259 310,322,327,342,346,362-364	141,257 304			21
3000-4000	7,61,90,92,93,115,152,154,192,199,200 332,347,355	104	248,320		17
4000-5000	233,285,339	41	289		5
5000-10000	59,60,62,63,190,191,249,333,335,336 338,348,352,356,357,358	182,187 312	316	251	21
10000-20000	65,178	208,278	274	168,288 305	8
20000-40000	64,67,77,231,359,361	32,186	172,181	173	11
> 40000	39,66,68,78,116,188,290,325,329,340 360	38,52 81,195	42,106 151,211	44,142	21

Table 4. Differences in Bounds

Example	PSODE	SGC	Example	PSODE	SGC
k98	N = 1	N = 2	k289	N = 3	N = 1
k180	N = 1	N = 2	k248	N = 3	N = 2
k204	N = 1	N = 2	k274	N = 3	N = 2
k264	N = 1	N = 2	k316	N = 3	N = 2
k323	N = 1	N > 3	k320	N = 3	N = 2
k325	N = 1	N > 3	k288	N = 4	N = 1
k181	N = 3	N > 3	k305	N = 4	N = 1
k42	N = 3	N = 1	k251	N = 4	N = 2

Table 5. Missing Examples

Name	N	Examples	Remarks
PSODE	1	311	polynomial type: 311, 326
	2	326	trancendental type: 350
	-	350	They are unsolved because of technical difficulties (see table 11).
SGC	1	39,188,231,291	polynomial type: all except 211
	2	41,163,182,220	trancendental type: 211 only
	3	211	39,188 and 211 are partially
	4	142,168	solved by PSODE

classification scheme is like this: (1) If there is an explicit type name mentioned in Kamke (1959), then it will be used, unless Kamke's classification was incorrect or it is more appropriate to classify the example concerned into a simpler type. (2) If Kamke did not give an explicit type name but mentioned a suitable transformation to reduce the example concerned to standard type, say *Linear*, then it will be tabulated as *reducible to Linear*. (3) If the above two rules cannot be applied, then I will classify the example according to what type of integrating factor (abbreviated as *intfactor* in the tables) it will require to convert it into an exact equation, e.g. integrating factor depending on y only or integrating factor depending on xy only, etc. If Kamke did not provide an explicit integrating factor (normally, an equivalent higher order equation was given), then such examples will not be classified (see table 8).

8. Comparison with ODESOLVE

In this section, we will compare the solvability by PSODE with ODESOLVE. The 331 tested examples were again used for testing ODESOLVE. We discovered that 137 of them can be completely solved by ODESOLVE while 36 of them can be partially solved. Among these solvable cases, 94 of them are of polynomial type, 52 of them are of transcendental type and 27 of them are of algebraic type. Some of the partially solved examples can be improved if the ALGINT package is used. For comparison purposes, the performance of ODESOLVE in solving different types of examples is tabulated in table 9. Since ODESOLVE uses a pattern-matching technique, several Kamke examples that are known to be non-elementary can be partially solved by it. These examples are k1,k5,k69-k71,k73,k129, k133 and k335-k338.

9. Technical Difficulties

From table 6, we can see that the success rate of PSODE in solving polynomial, transcendental and algebraic types of first order and first degree ODEs is 264/331, so there are 67 examples remaining unsolved. Of course, there is no reason why PSODE should be able to solve all of them – what we expect PSODE can solve are those examples which have elementary solutions, or occasionally some examples with non-elementary

Table 6. Analysis of solvable examples

Name	number of examples completely solved	number of examples partially solved	success rate (I)	success rate (II)	Remarks
Kamke	218	22	72.5 % (240/331)	73.2 % (213/291)	4 partially solved examples can be further simplified. 18 of them are non-elementary.
PSODE	241	23	79.8 % (264/331)	79.0 % (230/291)	6 partially solved examples are non-elementary, 9 of them have no answer in Kamke and 8 of them can be further simplified.
ODE-SOLVE	137	36	52.3 % (173/331)	50.2 % (146/291)	7 partially solved examples are non-elementary, 11 of them have no answer in Kamke and 18 of them can be further simplified.
SGC	-	-	-	76.0 % (221/291)	no information about the number of partially solved examples was mentioned in Shtokhamer et al (1986).

solutions which are returned from P_S step 3 with non-integers n_i (see section 2) or P_S step 4 with non-elementary integrals† contained in the answers. In order to clarify the situation, all these unsolved examples by PSODE are tabulated in table 10 and the explanations of why 3 examples known to have elementary solutions are missed can be found in table 11. The followings are further remarks on the technical difficulties encountered in step 2 of the P_S procedure:

1 In general, the system of *feqns* (see section 3.1 and 3.3) generated in step 2 can be huge and their inconsistency, or otherwise, may be hard to determine by using GROEBNER. Even when the system of equations is consistent, it may not be possible for the underlying computer algebra (CA) system to solve if the arithmetic needs to be done in algebraic extensions over the rationals. In fact, the latter is a common problem in all known CA systems.
2 The choice of the input variables in calling GROEBNER can seriously affect the efficiency of the subsequent calculations. For example, if we want to solve the example k305 and use the default ordering (lexicographical) in GROEBNER with the switch *groebopt*† on, it will still take much longer time than we just reverse

† It is because the integrator in REDUCE is based on the Risch-Norman semi-decision algorithm rather than the Risch decision algorithm.

† Supposed to be able to rearrange the input variables so that the subsequent calculations can be optimised.

Table 7. Solvability by PSODE

Type	Solved Kamke Examples	success rate	adjusted success rate	partially solved examples
Linear	2-4,6-8,90-94,130,134,148-150 153,154,161,174,175,192,193 196,198,200	26/28	26/26	
Homogeneous	112,113,117,123-125,136-138 167,204,223,232,239,246,262 271,272,276,281,284-286,290 295,297,306,308,310,315,325 337,349,363,364	35/35	35/35	325
Bernoulli	29,44,101,108,109,132,156,158 160,171,177,197,208,240,314	15/16	15/15	
Exact	245,248,251,263,270,273,274 288,289,299,305,309,322,330 336,341,347,348,352,353,355 356,361	23/24	23/24	
Separable	9,12,17,23,26,31,39,57,59-61 63-68,76,89,96,118,135,159 183,190,191,199,209,256,335 358,359	32/37	32/32	39,59 63 65-68
Abel	38,41,42,151,188	5/17	5/5	38,42 151,188
Riccati	19,32,98,102-104,106,140-143 155,162,163,165,068,170,172 173,178,180-182,186,187,194 195,207	28/50	28/28	106,173 178,181 186,195
Jacobi		0/1	0/0	
Linear in coefficients	213-216,221,222,224-229,231	13/13	13/13	

the order of the input variables or adopt a total degree ordering‡. Some examples like k44 and k173 cannot even be solved after spending several hours if we do not choose the total degree ordering. But certainly, we do not know yet (still an active research topic) which ordering will be optimal for all cases.

3 Multiparametric equations, such as k250 and k292† (with 8 and 6 parameters respectively) are also hard for GROEBNER to solve, even though the number of *feqns* generated in both cases is only 3. In Shtokhamer et al (1986), it was mentioned that ODEFI ran out of space for the equation k250 on a VAX/780 machine and could solve it only if 2 parameters are specified explicitly. That means such sets

‡ The current default ordering in PSODE version 2.

† Not expected to be solvable by PSODE (see table 10).

Table 8. Solvability by PSODE(cont'd)

Type	Solved Kamke Examples	success rate	adjusted success rate	partially solved examples
reducible to Linear	75,131,152,210,217,220,233 241,242,259,267,278,283,291 298,300,316,354	18/20	18/18	
reducible to Homogeneous	247,277,282,287	4/4	4/4	
reducible to Bernoulli	249,252,296,320,321,328,345 357	8/10	8/9	
reducible to Separable	15,52,77,97,115,116,119,211 218,254,258,279,280,294,303 307,329,338,360,362	20/20	20/20	52,116 211,360
reducible to Abel		0/2	0/0	
reducible to Riccati		0/3	0/0	
intfactor depends on y only	122,319,340,344	4/4	4/4	
intfactor depends on $x^2 + y^2$ only	275,339	2/2	2/2	
intfactor depends on xy only	243,255,260,304,318,324,333	7/7	7/7	
intfactor depends on xy^2 or x^2y only	301,302,317	3/3	3/3	
intfactor is a general function of x and y	244,293,327	3/4	3/4	
no explicit form given in Kamke	58,62,78,81,114,120,236,238 257,261,264,312,313,323,332 334,342,346	18/31	18/18	81

Table 9. Solvability by ODESOLVE

Type	Solved Kamke Examples	success rate	partially solved examples
Linear or reducible to Linear	2-8,75,90-94,130,131,133,134,148-150 153,154,161,174,175,192,193,196,198 200,210,220,233,241,242,259,267,298,300	39/48	5,8,89 133,148 174,192
Homogeneous or reducible to Homogeneous	112,113,117,123-125,136-138,167,204,223 232,239,246,262,271,272,276,281,284-287 290,295,297,306,308,310,315,325,337 349,363,364	36/39	112,113 325,337
Bernoulli or reducible to Bernoulli	29,44,101,108,109,129,132,156,158,160 171,177,197,208,240,314	16/26	129
Exact	245,248,251,263,270,273,274,288,289,299 305,309,311,322,330,336,341,347,348,352 353,355,356,361	24/24	336
Separable or reducible to Separable	1,9,12,17,23,26,31,39,57,59,60,61,63-71 73,76,89,96,118,135,159,183,190,191,199 209,211,256,258,280,307,335,338,358-360	43/57	1,39,59-61 67-71,73 89,190,191 211,280,335 338,360
Abel or reducible to Abel		0/19	
Riccati or reducible to Riccati	19,207	2/53	
Linear in coefficients	213-216,221,222,224-229,231	13/13	
special intfactor		0/20	
Jacobi		0/1	
no explicit form given in Kamke		0/31	

Table 10. Examples unsolved by PSODE

Type	known to be elementary but unsolved by PSODE	known to be non-elementary or no explicit answer in Kaike'c book.	claimed to be be solved by SGC
Linear or h reducible to Linear	none	5,133,435,343	none
Homogeneous or reducible to Homogeneous	none	none	none
Bernoulli or reducible to Bernoulli	350	129,351	350 only
Exact	311	none	311
Separable or reducible to Separable	none	1,69,70,71,73	none
Abel or reducible to Abel	none	36,37,40,43,45-48,111 145,147,169,185,237	none
Riccati or reducible to Riccati	none	13,14,18,20,21,22,24,25 27,28,30,88,95,99,105 107,121,139,144,157,164 166,176,179,184	none
Linear in coefficients	none	none	none
special intfactor	326	none	326
Jacobi	none	250	none
no explicit form given in Kamke	none	82,83,100,146,189,203 205,206,234,253,265 266,292	none

of equations pose common technical problems no matter whether the GROEBNER package‡ is being used or not. Other algorithms for solving parametric algebraic systems may be helpful in this respect (e.g. see Gao and Chou (1992)).

‡ ODEFI did not use GROEBNER basis approach.

Table 11. Technical Difficulties

Example	Difficulties	Remarks
k311	The eigenpolynomials can be solved, but the answers come from a quartic equation and so all the eigenpolynomials contain a lot of square root expressions. The main difficulty arises when we come to solve for a rational first integral because the arithmetic should be done in an algebraic extension of rationals and thus it is hard for REDUCE to solve the problem.	solved by SGC with reported N = 1
k326	Solving feqns and geqns will result in solving a quartic equation with 2 parameters. REDUCE cannot solve it after spending more than 8 hours.	solved by SGC with reported N = 2
k350	Solving for the irreducible eigenpolynomials in step one of P_S is hard - in a particular step, 307 *feqns* are generated and such systems of equations are hard to solve.	solved by SGC with 2 transcendentals and > 6 max order terms

10. Conclusion

A simple and efficient implementation of the Prelle-Singer procedure was developed to solve first order ODEs. By performing experiments on Kamke examples, we confirm that the Prelle-Singer procedure is an important method for formal solution of first order ordinary differential equations, even though it is still a semi-decision algorithm. The experimental results also indicated that by assigning a small degree bound ($N \leq 4$) in the program, we can still solve a large proportion of differential equations in Kamke (1959) and the solvable equations can cover most of the common types of differential equations (such as linear, homogeneous, Bernoulli, exact, separable, etc) and some special types of equations (such as Riccati and equations which require special integrating factors to solve). Besides, by comparing the results for the polynomial and transcendental types of equations, we discover that they are almost consistent with those reported in SGC, except for a few examples which have discrepancies in degree bounds and some missing due to technical difficulties. It was mentioned in MacCallum (1989) that the P_S procedure will be incorporated into the present differential equation solver in REDUCE and such a goal can be realized when the interfacing of PSODE with ODESOLVE is completed.

Acknowledgement

I would like to thank Prof. M.A.H. MacCallum for his useful help and advice in writing the program PSODE and this paper. Special thanks to Dr. F.J. Wright for his valuable suggestions to guide me how to translate the original version of PSODE from algebraic mode into symbolic mode in REDUCE.

Appendix

Suppose y is a dependent variable in a differential equation and x is the corresponding independent variable. The present input format of PSODE is as follows:
`psdesolve(ode,y,x [,deg_bound,convert_index])`.
The first four arguments should be self-explanatory. The fifth argument is an integer value— setting it to 1 will result in converting all the trigonometric terms in `ode` into tangents; while setting it to 2 will result in converting all of them into exponentials. These conversions are done before the actual P_S procedure is performed. That means if we have a trigonometric term like $\sin 2x$ in `ode`, then setting the `convert_index` to 1 will convert it into $\frac{2\tan x}{1+\tan^2 x}$, while setting the `convert_index` to 2 will convert it into $\frac{e^{2ix}-e^{-2ix}}{2i}$. In general, setting the `convert_index` to 1 will enable the subsequent calculations to run slightly faster than setting it into 2. If the user does not specify the fourth or the fifth arguments, then the default values in PSODE will be used, they are $N = 4$ and `convert_index` $= 1$ respectively. The following examples are chosen for demonstration purposes.

```
REDUCE 3.4.1, 15-Jul-92 ...

1: load psode;

2: on psdetimings; % a time switch in PSODE

3: psdesolve(x*df(y,x)-y*(x*log(x^2/y)+2),y,x);

Solved by Prelle-Singer Algorithm with N = 1

Time taken: 1938 ms

ARBCONSTANT=X + LOG(2*LOG(X) - LOG(Y))

4: psdesolve(df(y,x)*(2x^3*y^3-x)+2x^3*y^3-y,y,x);

Solved by Prelle-Singer Algorithm with N = 1

Time taken: 799 ms

                3  2      2  3
              4*X *Y  + 4*X *Y  + 1
ARBCONSTANT=-----------------------
                     2  2
                    X *Y

5: psdesolve(df(y,x)+2x*y-x*e^(-x^2),y,x);

Solved by Prelle-Singer Algorithm with N = 1

Time taken: 952 ms
```

```
                        2
                 2     X
ARBCONSTANT= - X  + 2*E  *Y

6: psdesolve(2y*df(y,x)-x*y^2=x^3,y,x);

Solved by Prelle-Singer Algorithm with N = 2

Time taken: 1037 ms

              2           2    2
ARBCONSTANT=X  - 2*LOG(X  + Y  + 2)
```

References

Gao, X.S., Chou, S.C. (1992). Solving Parametric Algebraic Systems. In *Proc. ISSAC 92 (ed. Paul S. Wang)* . California: ACM Press.

Jouanalou, J.P. (1979). Equations de Pfaff Algebriques. In *Lecture Notes in Mathematics* **708**, Springer Verlag.

Kamke, E. (1959). *Differentialgleichungen* . New York: Chelsea Publishing.

MacCallum, M.A.H. (1989). An ODE Solver for REDUCE. In *Lecture Notes in Comp. Sci.* **358**, 196–205, Springer Verlag.

Prelle, M.J., Singer, M.F. (1983). Elementary First Integrals of Differential Equations. *Trans. AMS* **279/1**, 215–229.

Shtokhamer, R., Glinos, N., Caviness, B.F. (1986). Computing Elementary First Integrals of Differential Equations. In *Computers and Mathematics Conference Manuscript.* Stanford.

Shtokhamer, R. (1988). Solving First Order Differential Equations using the Prelle–Singer Algorithm. *Technical Report* **88-09**, Centre for Mathematical Computation, University of Delaware.

Singer, M.F. (1990). Formal Solutions of Ordinary Differential Equations. *J. Symbolic Computation* **10/1**, 59–94.

Singer, M.F. (1992). Liouvillian First Integrals of Differential Equations. *Trans. AMS* **333/2**, 673–688.