Expanding the solutions of implicit sets of ordinary differential equations in power series

A. C. Norman

*IBM T. J. Watson Research Center, Yorktown Heights, New York 10598, USA**

Given a general analytic set of implicit ordinary differential equations the method described here will find an algorithm for expanding solutions to the equations in power series. The method is entirely mechanical, and does not have to rely on its user for prompting about the order, degree or form of the equations. In simple cases, such as sets of equations that can trivially be reduced to explicit form, the algorithm generated will be that of the Taylor Series method of Barton, Willers and Zahar (1972a, b) and so it can generally be expected to form the basis for effective and efficient numerical procedures.
(Received July 1974)

1. Introduction

The Taylor Series method for solving ordinary differential equations (Barton, Willers and Zahar, 1972a; Norman, 1973) has proved to be reliable and efficient (Barton *et al.*, 1972b). It has also been found to be a convenient starting point for a number of advanced numerical procedures (Norman, 1972; Willers, 1974; Norman, 1974). As described in the above references it is only applicable to explicit sets of equations—this limitation being intrinsic to the algorithms described. Van de Riet (1968) has shown how certain low order implicit equations can be treated by a Taylor Series approach, but demonstrates that his programs fail if used blindly on general implicit sets of equations. The algorithms described here redevelop this previous work from the standpoint of regular perturbation theory, and in so doing show how general implicit equations must be handled.

2. Preliminaries

The context in which the present algorithms are expected to be used is similar to that present in the system TAYLOR (Norman, 1973). A set of differential equations is read in by a special program which processes them and writes out special purpose code that can later be run to solve the equations numerically. Given any particular set of equations the first thing such a processor must do is to decide what the order of the system is and which names represent parameters and which represent dependent variables. Neither of these problems is as easy as might at first be imagined. The most obvious way of computing the order of a differential system is to identify the highest derivative of each variable mentioned, and to say that $d^r y/dx^r$ contributes $(r-1)$ to the total degree. This approach is good enough for explicit sets of equations provided that only derivatives on the left hand sides of equations are examined. For implicit equations it is possible to have things like:

$$\begin{aligned} y' - y &= 0 \\ z' + y''' &= 0 \end{aligned} \qquad (1)$$

which really have order two (only two initial conditions are needed, one for y and one for z) despite the appearance of a third derivative of y. Example (1) can be explained away by claiming that it should have been solved as two separate sub-problems, the first for y, the second (depending on the solution to the first) for z. This does not, however, sidestep the problem. If f and g are any non-degenerate functions of their arguments, the equations

$$\begin{aligned} f(y, z) &= 0 \\ g(y, z, y', z') &= 0 \end{aligned} \qquad (2)$$

may reasonably be considered. Here again the equations have fewer degrees of freedom than would be suggested by a count of derivatives. Since f and g are arbitrary, and there is some sort of symmetry between y and z in (2), it will normally not be feasible or reasonable to solve $f(y, z) = 0$ to express y as a function of z or vice versa. A general solution to this problem of finding the order of a differential system is contained in the algorithms presented later in this paper.

Identifying dependent variables and parameters is also not simple. If the equations

$$\begin{aligned} x^2 + y^2 &= r^2 \\ (x-1)^2 - y^2 &= s^2 \end{aligned} \qquad (3)$$

appear in the company of many further equations involving x, y, r and s, it may well be that s was intended to be an auxiliary function of x and y defined by

$$s = \mathrm{sqrt}((x-1)^2 - y^2)$$

while r was supposed to be a constant defining a relationship between x and y. An equally valid interpretation of (3) would make s a constant and r an auxiliary variable, and the two equations making up (3) are sufficiently alike that they can not be used as a guide to which interpretation was wanted. It would appear that this problem can not be overcome by reference just to the equations—the specification of initial conditions and parameter values provided with the problem must be used.

3. Perturbation theory

The expansions that will be generated for solutions to ODE's are going to be treated as perturbation expansions about the initial conditions, expressed in powers of the independent variable. A key result from perturbation theory that makes the present treatment practical is that (after a few initial difficulties) all the coefficients in the expansions can be found by solving *linear* equations. This result is true even though the original problem may have been highly non-linear. The result is easy to substantiate, but it should be stressed that the following demonstration is not the mathematical expression of a computationally feasible scheme. The computational method proposed is derived and justified in later sections.

Suppose a set of equations is given, and that the dependent variables have been identified and the order of the system has been computed. By the introduction of new variables and similar formal manipulation the equations can be reduced to the standard form

$$\mathbf{f}(t, \mathbf{y}, \mathbf{y}') = 0 \qquad (4)$$

where t is the independent variable, $\mathbf{y}$ a vector of dependent variables and $\mathbf{y}'$ represents $d\mathbf{y}/dt$. The order of the differential system will be the length of the vector $\mathbf{y}$. The algorithms for reducing general sets of equations to this form and the various short cuts that will have to be taken in a practical system are

*Present address: University of Cambridge Computer Laboratory, Corn Exchange St., Cambridge.

described later. Now the solution $\mathbf{y}$ of equation (4) is expressed as a power series:

$$\mathbf{y} = \sum_{i=0}^{\infty} \mathbf{y}_i t^i$$

where the coefficients $\mathbf{y}_i$ are the reduced derivatives of $\mathbf{y}$ evaluated at $t = 0$. $\mathbf{y}_0$ is just the vector of initial conditions that equation (4) needs, and $\mathbf{y}_1$ can be found by solving the equation

$$\mathbf{f}(0, \mathbf{y}_0, \mathbf{y}_1) = 0$$

This equation will normally be non-linear and so solving it may well be difficult: the present analysis provides no assistance at this stage in the processing. Further $\mathbf{y}_i$'s are found by differentiating equation (4) with respect to t, and then setting t to zero: thus to obtain $\mathbf{y}_r$, (4) is differentiated $r - 1$ times to give

$$f_{r-1,0,0} + \ldots + f_{0,0,1}\, d^r\mathbf{y}/dt^r = 0 \qquad (5)$$

where $f_{i,j,k}$ means

$$\frac{d^{i+j+k}\,\mathbf{f}(t, \mathbf{y}, \mathbf{p})}{dt^i\, d\mathbf{y}^j\, d\mathbf{p}^k}$$

with $\mathbf{p}$ standing for $\mathbf{y}'$. The terms represented by the ellipsis in (5) are all fixed functions of the initial conditions $\mathbf{y}_0$ and $\mathbf{y}_1$ and the derivatives of $\mathbf{y}$ with order less than r. Setting $t = 0$ in equation (5) and substituting for the derivatives of $\mathbf{y}$ in terms of the known reduced derivatives $\mathbf{y}_i$ leads to a linear equation that can be solved for $\mathbf{y}_r$. The fact that $\mathbf{f}$ is a vector valued function and that its derivatives are multi-dimensional objects does not disturb the formal calculation being performed. It should be noted that for all $r > 1$, $\mathbf{y}_r$ satisfies an equation of the form

$$f_{0,0,1}\mathbf{y}_r = Y_r \qquad (6)$$

where Y_r is an expression known in terms of the $\mathbf{y}_s$, $s < r$, and the matrix on the left hand side always precisely $f_{0,0,1}$. If this matrix is non-singular it is now clearly possible to compute as many of the coefficients $\mathbf{y}_r$ as are needed. If this matrix is singular, the expansion process will break down in the attempt to calculate $\mathbf{y}_2$, never at some later stage.

4. The practical method

The object of this section is to present efficient ways of computing the quantities Y_r that appear on the right hand side of equation (6). The term 'efficient' is taken to mean that the method must not involve the calculation of the partial derivatives that Y_r was originally defined in terms of, either numerically or by algebraic differentiation. First observe that if $\mathbf{f}$ is represented by a power series with coefficients $\mathbf{f}_i$ and the power series coefficients $\mathbf{y}_i$ are known for $i \leqslant r$ it is easy to compute the $\mathbf{f}_i$ for $i < r$. The algorithms involved have been discussed by Gibbons (1960) and by Barton, Willers and Zahar (1971), particularly with respect to the treatment of elementary functions. The general idea involves mapping simple combinations of terms into the corresponding combinations of their power series coefficients. Thus

$$a = b + c \longrightarrow a_r = b_r + c_r$$

$$a = b * c \longrightarrow a_r = \sum_{i=0}^{r} b_i c_{r-1}$$

are the transformations applicable to sums and products. Rules of the same general style (but marginally more complex) cope with quotients, exponentials, logarithms, sines, square roots and all the other simple functions that may appear in a set of equations.

The problem of interest is deciding what to do when $\mathbf{y}_r$ is not yet known, but is to be determined. Suppose that $\mathbf{f}_{r-1}$ is calculated in terms of a symbolic value of $\mathbf{y}_r$. The analysis of the previous section shows that the result will be, as in equation (5), linear in the unknown $\mathbf{y}_r$. The quantity Y_r is now obtained by setting $\mathbf{y}_r = 0$ in this linear expression. It is now clear that to obtain Y_r it was not necessary to perform any symbolic calculation: all that was needed was to go through the motions of calculating $\mathbf{f}_{r-1}$ with $\mathbf{y}_r$ temporarily taken to have the value zero. This, of course, corresponds to a rather simple piece of programming, and many of the program generation algorithms and techniques used in the previous (explicit) Taylor Series systems can be directly applied. To find the true value for $\mathbf{y}_r$ it is now necessary to solve equation (6). Because of the scaling of the reduced derivatives used in the power series expansions, $\mathbf{y}_r$ is related to the corresponding derivative of $\mathbf{y}$ by a factor of $r!$ while the quantity just calculated for Y_r (coming from the $(r-1)$st derivative of f) is out by a factor of $(r-1)!$. Thus the equation that is actually solved is

$$r f_{0,0,1}\, \mathbf{y}_r = \mathbf{f}_r^* \qquad (7)$$

where the f_r^* represents the negative of the quantity that the above paragraph showed how to compute.

When the true value of $\mathbf{y}_r$ has been found the working leading to $\mathbf{f}_{r-1}$ can be updated to correct for the original setting of $\mathbf{y}_r$ to zero.

The matrix $f_{0,0,1}$ can be computed as a side effect of solving the original non-linear equation

$$\mathbf{f}(0, \mathbf{y}_0, \mathbf{y}_1) = 0$$

since $f_{0,0,1}$ is needed anyway if this equation is to be solved by an iteration (such as Newton's method) that uses derivatives.

5. Reduction to standard form

It is now reasonable to address the problems of determining the order of differential systems and reducing them to the standard form used in the previous section. These algorithms will first be described in versions that will indeed produce the desired effects, but which will often result in unnecessary expansion in the size of the problem being considered. Particular aspects of the algorithms that result in this undesired expansion will be noted, and further sections will show how the difficulties can be avoided. The condition needed for a set of equations to have a power series expansion derivable by the methods used here is that $f_{0,0,1}$, as defined in previous sections, should be non-singular. The major preprocessing that has to be done involves converting sets of equations for which $f_{0,0,1}$ is necessarily singular, such as equations (2) above, into equivalent sets that do not have this problem.

To do this a *structure matrix* A is defined for the given set of equations. If the equation $f_i = 0$ does not use the value of dependent variable y_j explicitly the entry $a_{i,j}$ of A will be zero. If f_i mentions y_j but no derivatives of it then $a_{i,j}$ will be one. When f_i mentions the r'th derivative of y_j the value of $a_{i,j}$ will be $r - 1$. The first thing to do, having set up this array, is to look just at its pattern of zeros/non-zeros. Any entry $a_{i,j}$ that is zero in A must correspond to a zero entry in the matrix $f_{0,0,1}$, and so by looking at A it may be possible to show that $f_{0,0,1}$ will be singular. In this case A is called *structurally singular*. It is, of course, possible for the numeric values that will appear in $f_{0,0,1}$ to be such that the problem is singular even though A did not indicate that there would be trouble. If A here is structurally singular, the given problem was ill posed. The sort of sets of equations that will give trouble at this stage will be partly overdefined and partly underdefined, like the following example which is supposed to define three dependent variables x, y and z:

$$x' = 0$$

$$x' = 1$$

$$x' + y' + z' = 0$$

The structure matrix for these equations is

$$\begin{matrix} 2 & 0 & 0 \\ 2 & 0 & 0 \\ 2 & 2 & 2 \end{matrix}$$

which will be singular whatever values get associated with the entries labelled '2'. The algorithm that determines whether A is structurally singular in this way will also find a row permutation of A that will arrange that none of its diagonal elements are zero. Indeed, the algorithm (Hall, 1956) works precisely by systematically searching for such a permutation. If one can not be found, A was structurally singular. An important practical point is that if A is large but sparse (as it often will be) and the matrix is stored in a way that takes advantage of its sparseness to conserve store, the run time of the algorithm is dependent on the number of non-zero entries in A, rather than on the size of the full matrix.

The next observation to make is that if an r^{th} derivative of some variable y_i occurs anywhere in the equations, it will be necessary to solve for that derivative rather than for any lower one of y_i. A new matrix B is therefore defined: $b_{i,j}$ is equal to $a_{i,j}$ if it is as large as any other $a_{k,j}$, otherwise it is zero. If B is structurally singular it will not be possible to solve the given equations for the highest derivatives mentioned. For instance for equations (2) the A and B matrices are

$$A: \begin{matrix} 1 & 1 \\ 2 & 2 \end{matrix} \qquad B: \begin{matrix} 0 & 0 \\ 2 & 2 \end{matrix}$$

and although A is well behaved, B is singular. This situation can be recovered from by differentiating the first of the two equations in (2) with respect to t. If this is done symbolically a new pair of equations are produced:

$$f^{(1)}(t, y, z, y', z') = 0 \qquad (8)$$
$$g(t, y, z, y', z') = 0$$

and the A and B matrices for the new problem both have all entries non-zero, and so are not (structurally) singular. Differentiating the equation in (2) will, of course, have changed the degree of the differential system to be solved: to retain the original solution it is necessary to add the consistency condition

$$f(t_0, y_0, z_0) = 0$$

to the list of initial conditions associated with (2). The new problem that has been derived clearly has the same solution as the one that was originally posed. It is now possible to compute the degree of the system in the normal way, counting explicit derivatives of all the variables. The equations now do need two initial conditions, and so taking into account the degree of freedom removed by the consistency condition, the user will have to provide a single initial value. It should of course be noticed that the consistency condition is not (necessarily) linear and so it will have to be counted in with the general non-linear equation solving activity that occurs in setting up the first terms of the required expansion.

It is now possible to state that the condition on a set of equations that ensures that the order computed by counting highest derivatives of each variable is correct is just that the structure matrix B should not be structurally singular. Furthermore, given any set of equations where the structure matrix A is non-singular, it is possible to differentiate a subset of the equations (adjoining consistency conditions to preserve the original solution) to make the B matrix non-singular. The next section is devoted to the algorithms involved in identifying this subset. The existence of these algorithms serves as a proof of the above assertion.

6. Preprocessing algorithms

To convert a set of equations to standard form it is necessary to form the associated A structure matrix, and see if it is singular. As mentioned above the process of checking for singularity provides a row permutation of this matrix that moves zero elements off the diagonal. From now on it will be convenient to work with this permuted matrix, assuming that A is non-singular. Differentiating a single equation of the set making up the problem alters the matrix A by adding one to

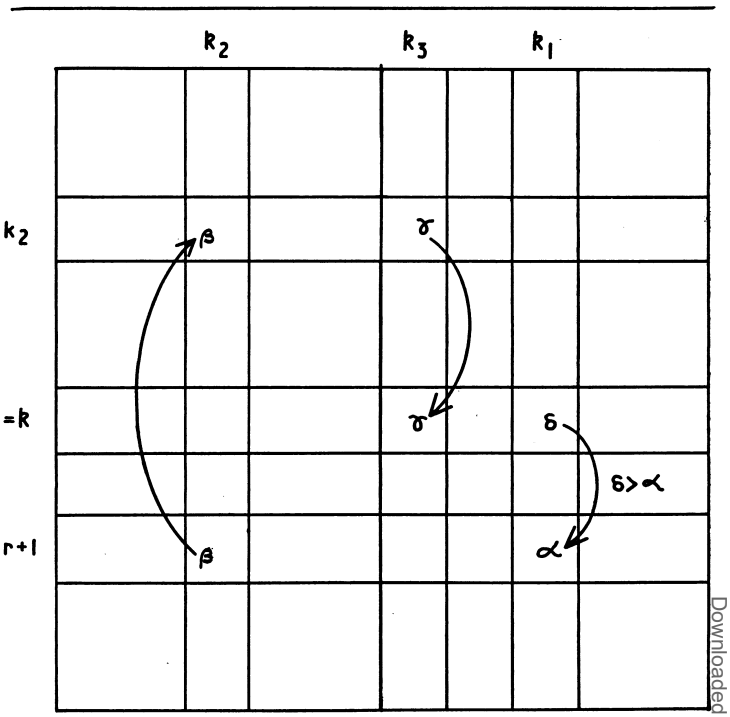

Fig. 1 Permutation of the rows of A to make $a_{r+1,\,r+1}$ maximal in its column

each non-zero element in the row corresponding to the differentiated quantity. This will alter the dominance relationship between elements in the same column, and so is reflected in a change in B. A combination of such incrementing of rows and row and column permutations will make it possible to remove zeros from the diagonal of B, and so make B non-singular.

Let A and B be $n \times n$ matrices, and suppose that the first r ($r < n$) diagonal elements of B are non-zero. An algorithm to make B non-singular will be based on a step that makes element $r + 1$ of the diagonal of B non-zero. This step has six stages:

1. If $b_{r+1,r+1}$ is already non-zero, do nothing.
2. If not, differentiate equation $r + 1$ until $a_{r+1,r+1}$ is maximal in its column
3. If the first r elements of the diagonal of the resulting B are still non-zero there is nothing more to do.
4. Otherwise, for each $k \leqslant r$ with $b_{k,k}$ zero, differentiate equation k until $b_{k,k}$ becomes non-zero. Continue with this step until the first r diagonal elements of B are again non-zero. If equation $r + 1$ had to be differentiated m times to make $b_{r+1,r+1}$ non-zero this process must finish when at most $m \times r$ differentiations have been performed. This worst case will correspond to having to differentiate each of the first r equations m times—a process that clearly restores the non-zero diagonal elements $b_{1,1}$ to $b_{r,r}$. Normally of course the amount of work to be done will be much less than this.
5. If $b_{r+1,r+1}$ is still non-zero, the step is complete.
6. If not, find the $a_{k,r+1}$ that is greatest in its column. The previous steps have made sure that $k \leqslant r$. Equation k must have been differentiated (or else this algorithm would have terminated at step 5) and there will be a chain of applications of step 4 that identified this differentiation as necessary. Let the sequence of equations differentiated in this chain be $(k_1, k_2, \ldots, k_m)$ where of course $k_1 = r + 1$ and $k_m = k$. Now since no equation was ever differentiated more times than was necessary to make the diagonal element of A corresponding to it as large as any other element in the same column of A, $a_{p,q} = a_{q,q}$ where p and q are any pair of consecutive k_i's. It is now evident that applying the cyclic

permutation that maps k_1 onto k_2, k_2 onto k_3 and so up to k_m onto k_1 to the rows of A leaves the first r diagonal elements of A unchanged. It does, however, move a column-maximal element into the position $a_{r+1,r+1}$ and so completes the extension of B's diagonal. **Fig. 1** illustrates this process when $m = 3$.

The full algorithm for making B non-singular now amounts to applying the above steps repeatedly to the set of equations, each application increasing the number of non-zero elements on the diagonal of B. The fact that this process always terminates guarantees that *any* set of equations corresponding to a non-singular A can be converted into a set for which B is non-singular.

As stated, the above algorithm will often differentiate sets of equations that could have been left unchanged. Although this will not alter the formal process of solving the differential system it will adversely affect efficiency. This difficulty can be overcome by adding a further step:

7. If in steps 2 and 4 *all* of the first $r + 1$ equations were differentiated at least m times ($m > 0$), integrate them all m times and remove the corresponding consistency conditions that were introduced when they were differentiated.

In many cases steps 1 to 7 will now just permute the matrix A, whereas without step 7 they would have done some differentiation. A practical system will not, of course, have to carry out any symbolic integration in step 7. When performing the above algorithm it will not actually do any differentiation, it will just record, for each equation, the number of differentiations that will have to be performed. The integration in step 7 corresponds just to decrementing this number by one. In later sections it will be seen that even the symbolic differentiation indicated here can often be avoided.

It has now been shown that any set of equations can be converted into a set for which the B structure matrix is non-singular. Using this conversion process it will now be demonstrated that any set of equations can be converted into a set in the standard form (4). What has to be shown is that the equations can be reduced to a set where all variables have no derivative higher than the first mentioned, and that non-singular B implies non-singular derivative $f_{0,0,1}$. The first transformation that must be applied gets rid of quantities defined by non-differential equations. Suppose a variable z appears in the set of equations but z' is not mentioned at all. Choose any equation mentioning z and replace it by its derivative (which will be an equation relating z and z' to the other variables). Remember to add a consistency condition corresponding to this differentiation. Now all dependent variables in the given equations can be considered as appearing with at least a first derivative. Apply the algorithm described above to make the matrix B associated with the equations non-singular. Now, for each second or higher derivative mentioned in the equations, auxiliary variables are introduced and new equations are added. If the quantity z''' appears somewhere in the equations it gets replaced by z_2' and the equations $z_2 - z_1' = 0$ and $z_1 - z' = 0$ are added. This does not alter the solution to the problem, and arranges things so that no derivatives higher than the first ever appear. It can be seen that the B matrix for the extended set of equations is non-singular, and has elements that are either two or zero. To be more specific, a single entry in the original A matrix corresponding to z''' would have been the number 4. In the expanded set of equations using z_1 and z_2 this entry has been replaced by a submatrix of the form:

	z	z_1	z_2
$z_1 = z'$	2	1	0
$z_2 = z_1'$	0	2	1
(z_2')	?	?	2

It is now possible to consider the complete vector **f** of equations and see what happens if it is differentiated partially with respect to the derivatives **y'** it is defined in terms of. The derivative will be a matrix $f_{0,0,1}$ and things have now been arranged so that the structure (in terms of zero/non-zero elements) is now given by B. Thus since B is not structurally singular $f_{0,0,1}$ is not. There is still, of course, the possibility that the particular numerical values that will appear in $f_{0,0,1}$ for some particular sets of initial conditions will make it numerically singular. That sort of difficulty will generally correspond to the equations not having a solution that can adequately be represented by a power series, and is beyond the scope of the present work.

7. Problem factoring

The next problem to be considered is the solution of the equations represented by (6). For small sets of equations the order of the matrix involved will not be excessive, and a direct approach will be adequate. For even moderate sets of equations, however, there will often be substantial gains to be achieved by treating the equations specially. Since the structure of $f_{0,0,1}$ is known, some of the sparse matrix methods of Gustavson, Liniger and Willoughby (1970) can be used. A number of particularly important sub-cases are likely to arise. First consider a set of equations that was originally given in explicit form and could have been solved by the previous Taylor Series systems. The fact that the equations could have been written explicitly will mean that there will be a permutation of rows and columns that will put B directly into triangular form. Solving equation (6) will then just amount to a process of back substitution. If B is diagonal the variables can be solved for in any order. In other cases the permutation that has to be applied to B to make it triangular defines an order in which the equations can be used. The algorithm that tries to permute rows and columns to make B triangular can be arranged to find a block decomposition of B when it cannot be made strictly triangular (Tarjan, 1972). Like other algorithms intended for use with sparse matrices, this one takes an amount of time dependent on the number of non-zero elements of B, not on the total size of the matrix, and so it can reasonably be used even for large problems. From this block decomposition it will be possible to extract the information necessary to see that equations (1) should have been solved as two problems (as originally suggested in the first section). The block decomposition algorithm should be applied to the matrix A after it has been rearranged to have a non-zero diagonal but before any of the equations are differentiated in the process of making B non-singular. Blocks that show up at this stage reveal gross dependencies between the variables, and show which sets of variables and equations can be factored out as sub-problems.

8. Avoiding differentiation

The process of reducing a general set of equations to standard form, as described above, is full of references to symbolically differentiating the equations. This is reasonable for a description of the processes involved, but needs to be avoided as much as possible in a practical scheme. The techniques for avoiding this differentiation, and some of the other manipulation of the equations that has been indicated, is discussed here. Avoiding having to differentiate is rather simple because of the power series environment in which this work is proceeding. To calculate the rth term in the power series expansion of the derivative of a function f it is sufficient to compute $(r + 1)$ times the $(r + 1)$st element in the expansion of f. All the differentiation mentioned so far can therefore be done purely formally, leaving the code generation parts of the power series manipulators to set up multipliers and index offsets as appropriate. Some sort of symbolic differentiation is still going to be necessary when generating the consistency conditions, since they must be expressed in closed form in terms of the low order

power series coefficients of the variables involved. The next thing to do is to avoid, where possible, the introduction of auxiliary variables that are just there to reduce the given equations to first order. Mixed order equations are almost as easy to solve as first order ones, the difference being that equation (7) has to be recast as

$$f_{0,0,1}R\mathbf{y}_r = \mathbf{f}_r^* \qquad (9)$$

where the matrix R is diagonal and has elements depending on the order of each variable y_j.

9. An example

Working through the algorithms presented here by hand on a small example will clearly show the complications that can arise in the method. What should be noticed is that all this complication is in setting up the recurrence relations between successive terms in the solution, and that this is work done only once. The application of the recurrence relationships, which is done many times, is rather simple. It should also be borne in mind that the initial manipulation of the problem, while so messy that it would make the present method unreliable and tedious to use by hand, is all mechanisable, and that the programmed version of it will not take an inordinate time to generate the required recurrences. The example chosen here is of a circular pendulum. The classical way of solving this problem would be to express the location of the bob in terms of an angle theta, as:

$$x = r\sin(\text{theta})$$
$$y = -r\cos(\text{theta})$$

and then express the forces involved in terms of theta to obtain an equation for theta:

$$\text{theta}'' = -(g/r)\sin(\text{theta})$$

This is reasonable for the very particular problem in hand, but is not easily extended to cope with non-circular pendulums. Instead, then, start from the equation relating x and y that states that the pendulum bob lies on a circle. This is

$$u = x^2 + y^2 - r^2 = 0 .$$

Add to this the equation of conservation of energy, which is

$$v = ((1/2)((x')^2 + (y')^2) + gy)' = 0 .$$

When setting up the structure matrix A the function v has to be counted as depending on x'' and y'', since if it was simplified both these terms would appear. The matrices A and B are thus initially:

$$A: \begin{matrix}1 & 1\\ 3 & 3\end{matrix} \qquad B: \begin{matrix}0 & 0\\ 3 & 3\end{matrix}$$

B is structurally singular, and so the equations have to be adjusted. In this case it is easy to see that the equation $u = 0$ has to be differentiated twice. This will introduce the two consistency conditions

$$x_0^2 + y_0^2 = r^2$$

and

$$2(x_0x_1 + y_0y_1) = 0 .$$

The equations are now fourth order, and there are two consistency conditions and so the user will have to provide two constraints as initial conditions. The first set of equations that have to be solved in developing the power series expansions are now $(u'')_0 = 0$ and $v_0 = 0$. There is no *a priori* reason for expecting these equations to be linear, although for this particular problem they are. It is necessary to write down these equations literally, and differentiate them to obtain the matrix $f_{0,0,1}$ that will be used through the rest of the calculation. Thus:

$$(u'')_0 = 2(2x_0x_2 + x_1^2 + 2y_0y_2 + y_1^2) = 0$$
$$v_0 = 2x_1x_2 + 2y_1y_2 + gy_1 = 0$$

and so

$$f_{0,0,1} = \begin{matrix}2x_0 & 2y_0\\ x_1 & y_1\end{matrix}$$

the factor of 2 having been taken out as representing the quantity R in equation (9). When the above equations have been solved for x_2 and y_2 it will be possible to set up the general recurrence for the rest of the coefficients x_i and y_i. For any $k > 0$ the quantities $(u'')_k$ and v_k that have to be calculated are given by

$$(u'')_k = (k+2)(k+1)u_{k+2} = \sum_{i=0}^{r+2} x_ix_{k-i+2} + y_iy_{k-i+2}$$
$$v_k = (k+1)gy_{k+1} + ((k+1)/2)\sum_{i=1}^{k+2} i(k-i+3)(x_ix_{k-i+3} + y_iy_{k-i+3}) .$$

Although these two expressions are complex enough to discourage hand calculation, they correspond to rather short and straightforward programs. Using these pieces of program, the algorithm for calculating the expansions of x and y (given the first few coefficients x_0, x_1, x_2, y_0, y_1 and y_2) is:

for $k = 1, 2, \ldots$
  $x_{k+2} := 0$; $y_{k+2} := 0$
  calculate (u'') and v_k
  solve the 2×2 linear system of equations
    $Fa = b$
  where F is the matrix $f_{0,0,1}$ discovered above,
  and b is the pair of values $(u'')_k$ and v_k.
  x_{k+2} and y_{k+2} are $-1/((k+1)(k+2))$ times the solution vector a that was found.

This program will generate as high an order expansion as is wanted. Transcribing the program above into FORTRAN is an error-prone process because indices of the coefficient vectors x and y will have to be offset by one to account for the fact that FORTRAN arrays start with element one, not element zero. When the complete program is generated mechanically allowance for such offsets is of course trivial.

This particular formulation of the circular pendulum equations of motion illustrates a pitfall in the indiscriminate use of implicit sets of equations. The equations of motion as set up using conservation of energy have a singular solution $x' = y' = 0$ (for arbitrary x and y), which does not correspond to a real motion of the pendulum. This difficulty results from $f_{0,0,1}$ being singular when the bob is at rest, and so it is not possible to expand a solution to the equations about this state. The problem is in the equations being considered, and not in the method used to solve them, and so no amount of preprocessing or special treatment can get round the singularity. Apart from $x' = y' = 0$ the linear equations that have to be solved in this problem are all well behaved; even the places $x = 0$ and $y = 0$ where it would have been difficult to solve for x in terms of y or vice versa are handled easily.

10. Further applications

So far the algorithms described here have been considered purely as ways of getting numerical values for power series coefficients of the solutions of well behaved ordinary differential equations. They actually have a rather wider class of applications. The first of these to consider is in the field of symbolic algebra. Many problems in this domain amount to finding an expansion of a function in terms of some small parameter. When the required expansion is a power series the present techniques can be used to produce an algorithm for developing the coefficients. Multiplication and division by the small parameter take the place of integration and differentiation in shifting power series orders. In terms of currently popular algebraic methods for producing such expansions, the new method is a variation on the method of repeated approximation (Barton and Fitch, 1972), where the user is relieved of the problems of selecting and coding an iteration, and the algebra system does not have to re-calculate any known values. As a

simple demonstration of the method consider the Kepler equation

$$E = u + e \sin(E) \qquad (10)$$

where it is necessary to expand E in terms of the small quantity e. The classical repeated approximation method notes the zeroth order solution $E = u$ and substitutes it into the right hand side to obtain a better solution $E = u + e \sin(u)$. Higher order approximations to the solution are obtained by repeatedly substituting into the right hand side of (10) and expanding $\sin(E)$ in powers of e. The new method observes that the given equation is explicit, and so expresses the power series expansion of $\sin(E)$ in terms of that of E. This is done by introducing new variables v and w with $v = \sin(E)$ and $w = \cos(E)$. Following the way previous Taylor Series schemes have coped with sines and cosines, it is possible to write

$$v' = wE'$$
$$w' = -vE'$$

where $'$ represents formal differentiation of the power series with respect to e. The required solution for E can now be found by using these values v and w in equation (10) and expressing the algebraic operations involved in terms of power series. It is easy to obtain the relationships

$$E_r = u_r + v_{r-1}$$
$$v_r = (1/r) \sum_{i=0}^{r-1} w_i (r - i) E_{r-i}$$
$$w_r = (-1/r) \sum_{i=0}^{r-1} v_i (r - i) E_{r-i}$$

These recurrences have to be used with initial conditions corresponding to the zeroth order solution to the equations, i.e.

$$E_0 = u_0$$
$$v_0 = \sin(E_0)$$
$$w_0 = \cos(E_0) \ .$$

Using these recurrences involves a rather more complex analysis of the problem that was needed for the straightforward use of repeated approximation. The algorithm generated will, however, be substantially more efficient, and since it is now possible to have a program generate the recurrence relationships automatically, their complexity does not matter. It should be noticed that the new scheme will cope naturally with the forcing term u in (10) being a function of e: it uses the coefficients u_i of the expansion of u in powers of e anyway.

The next class of problems that the new method can be extended to cover includes certain ordinary differential equations with removable singularities. A simple instance of the sort of equation involved is given by Bessel's equation of order zero, expanded about the origin:

$$f = xy'' + y' + xy = 0 \ .$$

At $x = 0$ (x being the independent variable) this equation does permit a solution that behaves like a polynomial. When calculating the order of terms that appear in this equation it is necessary to count multiplications by x as integrations and divisions by x as differentiations. It can then be seen that the first two terms in f have the same order. Thus y_r will be found by solving the equation $f_{r-1} = 0$: away from the origin where multiplication by x has no special significance y_r will be found in the normal way by solving $f_{r-2} = 0$. As for the previous sorts of problems considered, the low order terms in the solution being generated need special attention. For this equation it is easy to see that y_0 is arbitrary while y_1 must be zero. Bessel's equation is linear and so expanding its solution in power series is particularly simple. The same ideas can be extended to cope with some non-linear systems. Here automating the generation of expansion algorithms is a good deal harder. The new problems that have to be overcome involve dependent variables which have several of their early power series coefficients identically zero; multiplication or division by one of these alters the order of power series expansions, but the amount by which it does so can not be determined until the exact number of early zeros is known. The present suggestion for coping with this difficulty is to work purely algebraically with the equation, substituting low order power series expansions, and picking out coefficients to solve for as they appear. Eventually a non-zero coefficient will be found in each series, and the normal analysis can proceed. It is still the case that once the initial difficulties have been overcome only linear equations will have to be solved.

11. Conclusions

Algorithms developed here form the basis for a package that solves implicit sets of ordinary differential equations by an explicit method, closely akin to the normal Taylor Series one. The algorithms are arranged so that many important special case optimisation possibilities can be detected, and the treatment of these optimisation transformations accounts for most of the complexity of the method. Extensions to the method make it applicable to certain singular sets of equations, where the singularity is at a known place. All the detailed analysis of a given set of equations can be done once and for all, and the numeric process of solving the equations will reduce to using recurrence relationships found. If a set of equations is to be solved over a long interval, or will be used many times with different values for some parameter in it, the algorithms here will form the basis of an efficient numerical method, and the cost of problem analysis, program generation and compilation will be offset by the efficiency of the generated program.

References

BARTON, D., and FITCH, J. P. (1972). Applications of algebraic manipulative programs in physics, *Reports on progress in physics*, Vol. 35, No. 3, pp. 235-314.

BARTON, D., WILLERS, I. M., and ZAHAR, R. V. M. (1972a). The automatic solution of systems of ordinary differential equations by the method of Taylor series, *The best computer papers of* 1971, Edited Petrocelli, pp. 147-163, Auerbach Publishers Inc.

BARTON, D., WILLERS, I. M., and ZAHAR, R. V. M. (1972b). Taylor series methods for ordinary differential equations—an evaluation, *Mathematical Software*, Edited J. Rice, pp. 369-390, Academic Press, NY.

GIBBONS, A. (1960). A program for the automatic integration of differential equations using the method of Taylor series, *The Computer Journal*, Vol. 3, pp. 108-111.

GUSTAVSON, F, G., LINIGER, W., and WILLOUGHBY, R. (1970). Symbolic generation of an optimal Crout algorithm for sparse systems of linear equations, *JACM*, Vol. 17, pp. 87-109.

HALL, M. (1956). An algorithm for distinct representatives, *The American Mathematical Monthly*, Vol. 63, pp. 716-717.

NORMAN, A. C. (1972). A system for the solution of initial and two-point boundary value problems, *Proc. ACM72*, pp. 826-834.

NORMAN, A. C. (1973). *TAYLOR users manual*, University Computer Laboratory, Corn Exchange Street, Cambridge CB2 3QG, England.

NORMAN, A. C. (1974). The solution of ordinary differential equations involving step discontinuities, submitted to *ACM Transactions on Mathematical Software*.

VAN DE RIET, R. P. (1968). Formula manipulation in Algol 60, *MC tracts* 17 *and* 18, Mathematisch Centrum, Amsterdam.

TARJAN, R. (1972). Depth-first search and linear graph algorithms, *SIAM J. Computing*, Vol. 1, No. 2, pp. 146-160.

WILLERS, I. M. (1974). A new integration algorithm for ordinary differential equations based on continued fraction approximants, *CACM*, Vol. 17, No. 9, pp. 504-508.