NON-TRUNCATED POWER SERIES SOLUTION OF LINEAR ODE'S IN ALTRAN

K.O. Geddes

Department of Computer Science
University of Waterloo

RESEARCH REPORT CS-79-09

February 1979

This research was supported by the Natural Sciences and Engineering Research Council of Canada under Grant A8967.

ABSTRACT

An algorithm is presented for generating the power series solution of an arbitrary-order linear ordinary differential equation with polynomial coefficients, assuming initial-value conditions. The form of the power series solution is non-truncated in the sense that the algorithm computes a recurrence equation for generating successive power series coefficients, with the first few coefficients specified explicitly. The algorithm is implemented in ALTRAN and the results of applying it to some sample problems are presented.

TABLE OF CONTENTS

1. INTRODUCTION

It is a classical tool of applied mathematics to express the solution of a linear ordinary differential equation (ODE) in the form of a Taylor series expansion. The basic approach is to assume a solution in Taylor series form, formally substitute this form into the differential equation, and after some algebraic manipulations obtain equations to solve for the Taylor series coefficients. Obviously this process is well suited to automatic computation in a system for symbolic algebraic manipulation. In this paper we present algorithmic procedures for symbolically computing the Taylor series solution of an arbitrary-order linear ODE with polynomial coefficients and polynomial right hand side. The most interesting aspect of these procedures is that, for any ODE of this type expressed as an initial-value problem, the solution is obtained as an exact infinite series (i.e. not as a truncated power series).

The general Taylor series representation used in this paper takes the form of a linear list:

(1) $(a_0, a_1, \ldots, a_{k-1}, \mathit{recurrence})$

where a_i $(0 \le i \le k-1)$ are explicit Taylor series coefficients and *recurrence* is a recurrence relation for computing successive coefficients. More specifically, the last entry in the list (1) is an expression in the symbol k and in the symbols $a_{k-1}, \ldots, a_{k-n}$ (for some integer $n \ge 1$) such that

(2) $a_k = \mathit{recurrence}$

is a linear recurrence (difference) equation for computing successive co-efficients in the series

$$\sum_{k=0}^{\infty} a_k \, x^k$$

being represented. The number of coefficients which appear explicitly in the list (1) will always be at least n. For the class of linear ODE's with polynomial coefficients and polynomial right hand sides, the Taylor series solution (if it exists) can always be obtained in the form (1) by the procedures given in this paper.

2. OTHER SERIES REPRESENTATIONS

In the recent past several authors have discussed the problem of obtaining power series solutions of ODE's and it is appropriate to place the present paper in context with the previous papers. Fateman [2] discusses series expansions in general and series solutions of ODE's in particular. He illustrates how the current MACSYMA facilities can be used to generate a truncated power series (TPS) solution of a linear ODE. Lafferty [4] discusses the problem of obtaining closed-form power series solutions of linear ODE's. The approaches of Fateman and Lafferty can be contrasted with each other, and with the approach taken in this paper, by considering a simple example. The classical ODE initial-value problem

$$(3) \quad y' = y; \; y(0) = 1$$

has the known solution

$$(4) \quad y(x) = e^{x}.$$

The TPS approach (as discussed by Fateman) would yield the solution

$$(5) \quad 1 + x + \frac{1}{2} x^{2} + \frac{1}{6} x^{3} + \ldots$$

where the number of terms to be explicitly computed is under user control. The closed-form power series solution produced by Lafferty's program would be

$$(6) \quad \sum_{k=0}^{\infty} \frac{1}{k!} x^{k}.$$

The series solution produced by the algorithms in this paper is expressed in the form

$$(7) \quad (1, \frac{1}{k} a_{k-1})$$

which represents the series

$$\sum_{k=0}^{\infty} a_k x^k$$

where

$$a_0 = 1,$$

$$a_k = \frac{1}{k} a_{k-1}, \text{ for } k \geq 1.$$

Let us consider the various solutions expressed above with respect to (i) the desirability of the form of the solution and (ii) the generality of the method leading to the solution. Clearly the solution (4) would be the most desirable in most contexts and is indeed obtainable by MACSYMA's ODE solver. However the class of ODE's for which exact analytic solutions can be obtained is quite limited. TPS solutions of the form (5) are very general but lack preciseness (i.e. the solution explicitly obtained is only an "approximation"). Closed-form power series solutions of the form (6) are clearly the most desirable among series solutions but the method of solution will only succeed in very special cases. For linear ODE's with polynomial coefficients, series solutions of the form (7) combine the <u>generality</u> of the TPS approach with the <u>preciseness</u> of the closed-form power series approach. Clearly the TPS (5) of any desired order can be readily computed once the series is known in the form (7). Furthermore, Lafferty's method for computing the closed-form power series solution (6) is based on solving the recurrence equation (in this case, $a_k = \frac{1}{k} a_{k-1}$) appearing in (7). The limited success of the latter method is due to the fact that solving recurrence (difference) equations is a problem that ranks in difficulty along with the problem of solving ODE's exactly (cf. [1], [3]).

The approach taken in this paper is similar in spirit to the SCRATCHPAD system for power series manipulation developed by Norman [5]. However Norman's system does not recognize the simplicity of the recurrences required to specify the series solution for the class of ODE's considered here - namely, the class of arbitrary-order linear ODE's with polynomial coefficients and polynomial right hand sides. Specifically, the series representation (1) generated by the algorithms in this paper uses a <u>single</u> recurrence (along with some "initial coefficients") to specify the solution of a given ODE. In contrast, Norman's system typically introduces several "intermediate quantities" and thus represents the desired series solution by a (long) chain of recurrences. It should be pointed out, however, that Norman's system can handle <u>nonlinear</u> ODE's. The algorithms presented here offer an alternative approach to the handling of an important class of problems.

3. GENERATION OF THE RECURRENCE EQUATION

Consider the general order-ν linear ordinary differential equation (ODE) with polynomial coefficients:

$$(8) \quad p_\nu y^{(\nu)} + \ldots + p_1 y' + p_0 y = r$$

where p_i $(0 \le i \le \nu)$, $r \in D[x]$, where $D[x]$ denotes the set of polynomials in the indeterminate x over a coefficient domain D. (Typically, D is the field $\underline{Q}$ of rationals or a polynomial domain $\underline{Q}[\underline{\mu}]$ where $\underline{\mu}$ is a vector of indeterminates appearing in the problem.) Associated with the ODE (8) there will be ν conditions, which we assume to be of initial-value type:

$$(9) \quad y^{(i)}(0) = \alpha_i \qquad (0 \le i \le \nu-1)$$

where $y^{(0)}$ denotes y, $y^{(1)}$ denotes y', etc.

We seek a solution of (8) - (9) in the form of a Taylor series expansion about the initial point x = 0:

$$(10) \quad y = y(x) = \sum_{k=0}^{\infty} a_k x^k .$$

(The case where the initial conditons are specified at a point u ≠ 0, and hence the Taylor expansion is about the point u, can be handled by a change of variable and will not be pursued here.) Substituting (10) into (8) transforms the left hand side of (8) into the expression:

$$(11) \quad \sum_{k=0}^{\infty} a_k \{p_\nu (x^k)^{(\nu)} + \ldots + p_1 (x^k)' + p_0 (x^k)\}.$$

Noting that

$$(12)\quad (x^k)^{(i)} = k(k-1)\ \ldots\ (k-i+1)\ x^{k-i}$$

and considering the effect of multiplying (12) by a polynomial p_i, the expression (11) takes the general form

$$(13)\quad \sum_{k=0}^{\infty} a_k\ \{v_0\ x^{k-s} + v_1\ x^{k-s+1} + \ldots + v_n\ x^{k-s+n}\}$$

for some integer n, where the integer $s \leq \nu$ will be called the <u>shift</u>, and where the coefficients v_i $(0 \leq i \leq n)$ are polynomial expressions in the index k. Note that the apparent negative powers of x in expression (13) do not actually occur since the corresponding polynomial expression v_i will be zero (i.e. the coefficient of x^{k-i} in (12) is zero for all k in the range $0 \leq k < i$).

In order to equate coefficients on the left and right of (8), we choose to express the left-side expression (13) in the form

$$(14)\quad \sum_{k=s}^{\infty} \{u_0\ a_k + u_1\ a_{k-1} + \ldots + u_n\ a_{k-n}\}\ x^{k-s}$$

where, by convention, $a_i = 0$ for $i < 0$. This form is obtained by changing the index of summation in the expression (13), separately in each term of the expression, so as to obtain the coefficient of x^{k-s}. Specifically, the coefficients u_i in (14) are obtained from the coefficients v_i in (13) by performing the simple substitutions:

$$u_i = v_i\ (k \leftarrow k-i)$$

where the notation v_i $(k \leftarrow f(k))$ denotes, in an obvious way, an operation of substitution in the polynomial expression v_i. Note that the lower limit of the summation in (14) has been taken to be s, since the expression in braces $\{\cdot\}$ will be zero for all indices $k < s$ (corresponding to negative powers of x).

The ODE (8) can now be expressed in a form where the left side of the equation is expression (14). Equating coefficients on the left and right sides we see that, for k large enough, the Taylor series coefficients must satisfy the (n+1)-term recurrence equation

$$(15) \quad u_0\, a_k + u_1\, a_{k-1} + \dots + u_n\, a_{k-n} = 0,$$

where the recurrence coefficients u_i are polynomial expressions in k. In the following section we describe how the complete series solution is obtained, taking into account the initial conditions (9) and the polynomial r appearing on the right hand side of the ODE (8).

4. REPRESENTATION OF THE SERIES SOLUTION

The problem of determining the Taylor series coefficients for the solution of the initial-value problem (8) - (9) has been reduced to a problem of equating like terms in the equation

$$(16)\quad \sum_{k=s}^{\infty} \{u_0\, a_k + u_1\, a_{k-1} + \ldots + u_n\, a_{k-n}\}\, x^{k-s} = \sum_{k=s}^{d+s} r_{k-s}\, x^{k-s},$$

after noting that the initial conditions (9) explicitly specify the first ν Taylor coefficients. The left side of equation (16) is expression (14) and the right side comes from expressing the polynomial r appearing in the ODE (8) in the form

$$r = \sum_{i=0}^{d} r_i\, x^i,$$

where d denotes the degree of r. Specifically, the desired Taylor series coefficients can be obtained by solving, in the order specified, the following equations:

$$(17)\quad a_k = \alpha_k/k!, \qquad 0 \le k \le \nu-1$$

$$(18)\quad u_0\, a_k + u_1\, a_{k-1} + \ldots + u_n\, a_{k-n} = r_{k-s},\ \nu \le k \le d+s$$

$$(19)\quad u_0\, a_k + u_1\, a_{k-1} + \ldots + u_n\, a_{k-n} = 0,\ k > d+s$$

(recalling the convention that $a_i = 0$ for $i < 0$). Note that the range of the index k specified for (18) and (19) refers not only to the subscripts appearing explicitly in the above notation but also to the values to be assigned to k in the polynomial expressions u_i.

It was noted below expression (13) that the shift s satisfies the inequality $s \leq \nu$. If $s = \nu$ then equations (18) - (19) have been obtained by equating coefficients of x^i on the left and right of equation (16) for all powers $i \geq 0$. On the other hand, if $s < \nu$ then the coefficients of x^i in equation (16), for $0 \leq i < \nu-s$, have not been equated in forming (18) - (19). Therefore a consistency check must be made in the latter case to ensure that there exists a solution of the form (10) for the problem (8) - (9). This consistency check is made after evaluating (17) to obtain a_k $(0 \leq k \leq \nu-1)$ and before solving (18) for a_ν, and it involves checking the identities

$$(20) \quad u_0\, a_k + u_1\, a_{k-1} + \dots + u_n\, a_{k-n} = r_{k-s}, \quad s \leq k \leq \nu-1.$$

If this identity fails to hold for some value of k in the specified range then the initial-value problem (8) - (9) does not have a solution in the form of a Taylor series expansion about the origin.

The solution of equation (19), for k arbitrary, yields the k-th Taylor series coefficient as a linear recurrence involving the preceding n coefficients. Specifically, the general solution is

$$(21) \quad a_k = -\frac{u_1}{u_0}\, a_{k-1} - \dots - \frac{u_n}{u_0}\, a_{k-n}$$

where u_i $(0 \leq i \leq n)$ are polynomials in k. The complete set of Taylor series coefficients can therefore be represented in the form of the linear list (1) where *recurrence* is precisely the right hand side of equation (21).

The number of Taylor coefficients which must appear explicitly in the representation (1) is

$$\max\{n, d+s+1\}; \tag{22}$$

they are computed from equations (17) - (18) and, in case $n > d+s+1$, the first few cases of equation (19).

5. SPECIFICATION OF THE PROCEDURES

In this section a pseudo-Algol algorithmic notation is used to specify procedures for generating the Taylor series solution of a given linear ODE using the representation (1). In these procedures, the following five "system" functions for polynomial manipulation are assumed:

degree (p,x) - returns the degree of the polynomial p in the indeterminate x, with the convention that degree $(0,x) = -\infty$;

coefficient (p,x,n) - returns the coefficient in the polynomial p of the n-th power of the indeterminate x;

substitute (r, x_list, e_list) - returns the result of substituting into the rational expression r the i-th entry in e_list for every occurrence of the i-th entry in x_list, where i ranges from 1 to the length of x_list. [Note: x_list must be a list of indeterminates and e_list must be a list (of the same length) of expressions];

low_index (expr, x_array) - returns the index i such that x_array(i) appears explicitly in the expression expr while x_array(j) does not appear, for all $j < i$. [Note: x_array must be a one-dimensional array of indeterminates];

high_index (expr, x_array) - returns the index i such that x-array(i) appears explicitly in the expression expr while x_array(j) does not appear, for all $j > i$. [Note: x_array must be a one-dimensional array of indeterminates].

Procedure taylor is the high-level procedure which would be called by the user; it requires the three other procedures specified here: generate_recurrence, derivative_x**k_times_p, and solve_recurrence. The differential equation is passed to procedure taylor as a polynomial (ode) in the independent variable (x), the dependent variable (y), and the derivatives of the dependent variable (specified as dy(1), dy(2),... where dy is an array of indeterminates). The differential equation is then understood to be

ode = 0.

The initial conditions are specified by an array (initial) dimensioned from 0 to ν-1, where ν is the order of the differential equation, such that

$$y^{(i)}(0) = \text{initial}(i),\ 0 \le i \le \nu-1.$$

Note that an *error return* is possible from step 3 of procedure solve_recurrence indicating that the given initial-value problem has no Taylor series solution.

Throughout the three lower-level procedures the following names are used for indeterminates. The name k always represents an indeterminate; whenever a numerical value is to be assigned to k we assign the desired value to the variable k_value and then use an explicit substitution of k_value for k via the substitute function mentioned above. The name x_power_k stands for an array of indeterminates such that x_power_k(j) is used to represent the expression x^{k+j} as it appears in summation (13), where k is an indeterminate and j is an integer (positive, negative, or zero). The name ak stands for an array of indeterminates such that ak(j) is used to represent the expression a_{k+j} as it appears in summation (14) and in the resulting recurrence equation, where k is an indeterminate and j is a nonpositive integer.

```
procedure taylor (ode, x, y, dy, initial)
    [Input parameters:  ode, x, y, dy, initial as described above;
     Output:  The value returned is the Taylor series solution of the
              ode represented in the form (1)]

    [1.  Determine the order of the ode]
    ν ← high_index (ode, dy)

    [2.  Pick off the polynomial coefficients and right hand side]
    p_0 ← coefficient (ode, y, 1)
    for i = 1 step 1 until ν do
        p_i ← coefficient (ode, dy(i), 1)
    doend
    r ← (p_0*y + Σ_{i=1}^{ν} p_i*dy(i)) - ode

    [3.  Generate and solve the recurrence equation]
    generate_recurrence (ν, p, x, recurrence_equation, n, shift)
    series ← solve_recurrence (recurrence_equation, n, shift, r, x,
                                                          initial, ν)
    return (series)
end of procedure taylor
```

```
procedure generate_recurrence (ν, p, x, recurrence_equation, n, shift)
    [Input parameters:  ν, p, x;
     Output parameters:  recurrence_equation, n, shift]

    [1.  Compute factor, which is the expression in braces {  } in
         the summation (11)]
    jmax ← - ν
    factor ← 0
    for i = 0 step 1 until ν do
        jmax ← max (jmax, degree (p_i ,x) - i)
        factor ← factor + derivative_x**k_times_p(i, p_i, x)
    doend

    [2.  Compute n and shift, which are parameters associated with the
         recurrence equation as described below summation (13)]
    jmin ← low_index (factor, x_power_k) [Note: jmin ≥ - ν]
    n ← jmax - jmin
    shift ← - jmin

    [3.  Compute recurrence_equation, which is the expression in braces
         {  } in the summation (14)]
    recurrence_equation ← 0
    for j = jmin step 1 until jmax do
        coef ← coefficient (factor, x_power_k(j),1)
        coef ← substitute (coef, k, k + jmin - j)
        recurrence_equation ← recurrence_equation + coef * ak (jmin - j)
    doend
end of procedure generate_recurrence
```

```
procedure derivative_x**k_times_p (i, p, x)
    [Input parameters:  i, p, x;
     Output:  The value returned is the i-th derivative of x^k
              (where k is an indeterminate) multiplied by p which is
              a polynomial in the indeterminate x (recall the use of
              x_power_k as described above)]

    [1.  First form x^(k-i) * p]
    newp ← 0
    for   j = 0 step 1 until degree (p,x) do
        newp ← newp + coefficient (p, x, j) * x_power_k (-i+j)
    doend

    [2.  Now attach the factors k(k-1) ... (k-i+1) -- see equation [12]]
    for j = 0 step 1 until i-1 do
        newp ← (k-j) * newp
    doend

    return (newp)
end of procedure derivative_x**k_times_p
```

```
procedure solve_recurrence (recurrence_equation, n, shift, r, x,
                                                         initial, ν)

   [Input parameters: recurrence_equation, n, shift, r, x, initial, ν;
    Output:  The value returned is the series solution represented in
             the form (1)                                             ]

   [1.  The first ν coefficients are given by equation (17)]
   for k_value = 0 step 1 until ν-1 do
       series (k_value) ← initial (k_value)/(k_value)!
   doend

   [2.  Solve recurrence_equation = 0 (i.e. equation (15)) for
        recurrence, which is the right hand side of equation (21)]
   u0 ← coefficient (recurrence_equation, ak(0), 1)
   recurrence ← ak(0) - recurrence_equation / u0
   list_of_indeterminates ← (ak(-1), ak(-2),...,ak(-n))

   [3.  Make consistency check as specified by equation (20);
        error return indicates that no Taylor series solution exists]
   for k_value = shift step 1 until ν-1 do
        list_of_values ← (series(k_value - 1),
                           series(k_value - 2),...,series(k_value - n))
        temp ← substitute (recurrence, k, k_value)
        temp ← substitute (temp, list_of_indeterminates,
                                                  list_of_values)
        taylor_coefficient ← temp + coefficient (r, x, k_value - shift)/
                                          substitute (u0, k, k_value)
        if taylor_coefficient ≠ series(k_value) then error return
   doend     [Note:  series(i) = 0 if i < 0]
```

```
[4.  Solve equation (18), and perhaps the first few cases of
     equation (19), until the number of coefficients specified by
     (22) have been computed]
for k_value = ν step 1 until max (n-1, degree(r,x) + shift) do
     list_of_values ← (series(k_value - 1),
                        series(k_value - 2),...,series(k_value - n))
        [Note:  series(i) = 0 if i < 0]
     temp ← substitute (recurrence, k, k_value)
     temp ← substitute (temp, list_of_indeterminates,
                                                    list_of_values)
     series(k_value) ← temp + coefficient (r, x, k_value - shift)/
                                          substitute (u0, k, k_value)
doend

[5.  The last entry in the series representation (1) is recurrence]
series(max(n, degree(r,x) + shift + 1)) ← recurrence

return (series)
end of procedure solve_recurrence
```

6. SAMPLE PROBLEMS

In this section, the output from the ALTRAN program which implements the algorithm described in the preceding sections is presented for the following sample problems.

Problem 1: (First-order problem)

$(1+x^2)\, y' = 1$

$y(0) = 0$

Solution: $y(x) = \arctan(x)$

Problem 2: (Order 0 differential equation)

$(1+x^2)\, y = 1$

Solution: $y(x) = 1/(1+x^2)$

Problem 3: (Problem with polynomial solution)

$(x-x^2)\, y'' + (1/2-x)\, y' + 4y = 0$

$y(0) = 1;\ y'(0) = -8$

Solution: $y(x) = T_2(1-2x)$

$= 8x^2 - 8x + 1$

Remark: Special case of the hypergeometric equation.

Problem 4: (Fourth-order problem)

$y^{(4)} - y = 0$

$y(0) = 3/2;\ y'(0) = -1/2;\ y''(0) = -3/2;\ y'''(0) = 1/2.$

Solution: $y(x) = 3/2 \cos(x) - 1/2 \sin(x).$

Problem 5: (Indeterminate initial conditions)

$$(1+x^2)y'' - y' + xy = 2-x^2$$

$$y(0) = \mu_1;\ y'(0) = \mu_2$$

Solution: unknown

Remark: The Taylor series coefficients are bilinear polynomials in μ_1 and μ_2.

Problem 6: (Indeterminate in differential equation)

$$y' = \mu_1 y$$

$$y(0) = \mu_2$$

Solution: $\mu_2 e^{\mu_1 x}$

Problem 7: (Indeterminate in differential equation)

$$y'' + \mu_1 xy = 0$$

$$y(0) = 1;\ y'(0) = 1$$

Solution: unknown

Remark: Every third Taylor series coefficient is zero.

Problem 8: (Problem with several indeterminates)

$$(1 + \frac{1}{\mu_2^{\,2}}\ x^2)\ y' + \frac{1}{\mu_2}\ (\frac{2\mu_3}{\mu_2}\, x + \mu_4)\ y = 0$$

$$y(0) = \mu_1$$

Solution: unknown

Output for Problem 1

```
# DIFFEQ

    X**2*DY(1) + DY(1) - 1 [illegible]

# INIT(0)

    0

# THE INFINITE TAYLOR SERIES SOLUTION IS

# A

    {0,

    1,

    - AK(-2) * ( K - 2 ) / ( K ) }

# TIME (SEC.) TO COMPUTE TAYLOR SOLUTION WAS

# TNEW

    2.741172

# THE SERIES EXPANDED UP TO DEGREE 10 IS

# A

    { 0,

    1,

    0,

    - 1 / 3,

    0,

    1 / 5,

    0,

    - 1 / 7,

    0,

    1 / 9,

    0,

    133: - AK(-2) * ( K - 2 ) / ( K ) }

# TIME (SEC.) TO EXPAND THE SERIES WAS

# TNEW

    3.084531
```

Output for Problem 2

```
# DIFEQ

      X**2*Y + Y - 1

# THE INFINITE TAYLOR SERIES SOLUTION IS

# A

      ( 1 ,

      0 ,

      - AK(-2) )

# TIME (SEC.) TO COMPUTE TAYLOR SOLUTION WAS

# TNEW

      2.366422

# THE SERIES EXPANDED UP TO DEGREE 10 IS

# A

      ( 1 ,

      0 ,

      - 1 ,

      0 ,

      1 ,

      0 ,

      - 1 ,

      0 ,

      1 ,

      0 ,

      - 1 ,

      - AK(-2) )

# TIME (SEC.) TO EXPAND THE SERIES WAS

# TNEW

      1.721203
```

Output for Problem 3

```
# DIFFEQ

   [illegible]

# INIT(0)

     1

# INIT(1)

     - 8

# THE INFINITE TAYLOR SERIES SOLUTION IS

# A

     ( 1 ,

     - 8 ,

     2*AK(-1) * ( K**2 - 2*K - 3 ) / ( ( K ) * ( 2*K - 1 ) ) )

# TIME (SEC.) TO COMPUTE TAYLOR SOLUTION WAS

# TNEW

     5.113906

# THE SERIES EXPANDED UP TO DEGREE 10 IS

# A

     ( 1 ,

     - 8 ,

     8 ,

     0 ,

     0 ,

     0 ,

     0 ,

     0 ,

     0 ,

     0 ,

     0 ,

     129: 2*AK(-1) * ( K**2 - 2*K - 3 ) / ( ( K ) * ( 2*K - 1 ) ) )

# TIME (SEC.) TO EXPAND THE SERIES WAS

# TNEW

     3.750219
```

Output for Problem 4

```
# DIFFEQ
      - ( Y - DY(4) )
# INIT(0)
      3 / 2
# INIT(1)
      - 1 / 2
# INIT(2)
      - 3 / 2
# INIT(3)
      1 / 2
# THE INFINITE TAYLOR SERIES SOLUTION IS
# A
      ( 3 / 2 ,
      - 1 / 2 ,
      - 3 / 4 ,
      1 / 12 ,
      AK(-4) / ( ( K ) * ( K**3 - 6*K**2 + 11*K - 6 ) ) )
# TIME (SEC.) TO COMPUTE TAYLOR SOLUTION WAS
# TNEW
      4.467812
# THE SERIES EXPANDED UP TO DEGREE 10 IS
# A
      ( 3 / 2 ,
      - 1 / 2 ,
      - 3 / 4
```

```
      1 / 12 ,
      1 / 16 ,
    - 1 / 240 ,
    - 1 / 480 ,
      1 / 10080 ,
      1 / 26880 ,
    - 1 / 725760 ,
    - 1 / 2419200 ,
      136: AK(-4) / ( ( K ) * ( K**3 - 6*K**2 + 11*K - 6 ) ) )
# TIME (SEC.) TO EXPAND THE SERIES WAS
# TNEW
      3.567516
```

Output for Problem 5

```
# DIFFEQ
      X**2*DY(2) + X**2 - X*Y - DY(1) + DY(2) - 2
# INIT(0)
      MU(1)
# INIT(1)
      MU(2)
# THE INFINITE TAYLOR SERIES SOLUTION IS
# A
      ( MU(1) ,
      MU(2) ,
      ( MU(2) + 2 ) / 2 ,
      - ( MU(1) - MU(2) - 2 ) / 6 ,
      - ( MU(1) + 3*MU(2) + 4 ) / 24 ,
      - ( K**2*AK(-2) - 5*K*AK(-2) - K*AK(-1) + AK(-3) + 6*AK(-2) + AK(-1) ) /
      ( ( K ) * ( K - 1 ) ) )
# TIME (SEC) TO COMPUTE TAYLOR SOLUTION WAS
# TNEW
      7.568156
# THE SERIES EXPANDED UP TO DEGREE 10 IS
# A
      ( MU(1) ,
      MU(2) ,
      ( MU(2) + 2 ) / 2 ,
      - ( MU(1) - MU(2) - 2 ) / 6 ,
```

5 (cont.)


```
- ( MU(1) + 3*MU(2) + 4 ) / 24 ,
( 5*MU(1) - 12*MU(2) - 22 ) / 120 ,
( 21*MU(1) + 20*MU(2) + 18 ) / 720 ,
- ( 74*MU(1) - 275*MU(2) - 478 ) / 5040 ,
- ( 73*MU(1) + 253*MU(2) - 70 ) / 40320 ,
( 2227*MU(1) - 11943*MU(2) - 20132 ) / 362880 ,
( 43923*MU(1) + 25*MU(2) - 27876 ) / 3628800 ,
142: - ( K**2*AK(-2) - 5*K*AK(-2) + K*AK(-1) + AK(-2) + 6*AK(-2) +
AK(-1) ) / ( ( K ) * ( K - 1 ) ) )
# TIME (SEC) TO EXPAND THE SERIES WAS
# TNEW
   5.418391
```

Output for Problem 6

```
# DIFFEQ
      - ( Y*MU(1) - DY(1) )
# INIT(0)
      MU(2)
# THE INFINITE TAYLOR SERIES SOLUTION IS
# A
      ( MU(2) ,
      AK(-1)*MU(1) / ( K ) )
# TIME (SEC) TO COMPUTE TAYLOR SOLUTION WAS
# TNEW
      1.967422
# THE SERIES EXPANDED UP TO DEGREE 10 IS
# A
      ( MU(2) ,
      MU(1)*MU(2) ,
      MU(1)**2*MU(2) / 2 ,
      MU(1)**3*MU(2) / 6 ,
      MU(1)**4*MU(2) / 24 ,
      MU(1)**5*MU(2) / 120 ,
      MU(1)**6*MU(2) / 720 ,
      MU(1)**7*MU(2) / 5040 ,
      MU(1)**8*MU(2) / 40320 ,
      MU(1)**9*MU(2) / 362880 ,
      MU(1)**10*MU(2) / 3628800 ,
      AK(-1)*MU(1) / ( K ) )
# TIME (SEC) TO EXPAND THE SERIES WAS

      2.575953
```

Output for Problem 7

```
# DIFFEQ
      X*Y*MU(1) + DY(2)
# INIT(0)
      1
# INIT(1)
      1
# THE INFINITE TAYLOR SERIES SOLUTION IS
# A
      ( 1 ,
      1 ,
      0 ,
      - AK(-3)*MU(1) / ( ( K ) * ( K - 1 ) ) )
# TIME (SEC.) TO COMPUTE TAYLOR SOLUTION WAS
# TNEW
      3.771688
# THE SERIES EXPANDED UP TO DEGREE 10 IS
# A
      ( 1 ,
      1 ,
      0 ,
      - MU(1) / 6 ,
      - MU(1) / 12 ,
      0 ,
      MU(1)**2 / 180 ,
      MU(1)**2 / 504 ,
      0 ,
```

```
    - MU(I)**3 / 12960 ,
    - MU(I)**3 / 45360 ,
    HE - AK(-3)*MU(I) / ( ( K ) * ( K - 1 ) ) )
# TIME (SEC.) TO EXPAND THE SERIES WAS
# TNEW
      3.450797
```

Output for Problem 8

```
# DIFFEQ
      ( X**2*DY(1) + 2*X*Y*MU(3) + Y*MU(2)*MU(4) + DY(1)*MU(2)**2 ) /
      ( MU(2)**2 )
# INIT(0)
      MU(1)
# THE INFINITE TAYLOR SERIES SOLUTION IS
# A
      ( MU(1) ,
      - MU(1)*MU(4) / ( MU(2) ) ,
      - ( K*AK(-2) + 2*AK(-2)*MU(3) - 2*AK(-3) + AK(-1)*MU(2)*MU(4) ) /
      ( K*MU(2)**2 ) )
# TIME (SEC) TO COMPUTE TAYLOR SOLUTION WAS
# TNEW
      4.684281
# THE SERIES EXPANDED UP TO DEGREE 10 IS
# A
      ( MU(1) ,
      - MU(1)*MU(4) / ( MU(2) ) ,
      - MU(1) * ( 2*MU(3) - MU(4)**2 ) / ( 2*MU(2)**2 ) ,
      MU(1)*MU(4) * ( 6*MU(3) - MU(4)**2 + 2 ) / ( 6*MU(2)**3 ) ,
      MU(1) * ( 12*MU(3)**2 - 12*MU(3)*MU(4)**2 + 12*MU(3) + MU(4)**4 -
      8*MU(4)**2 ) / ( 24*MU(2)**4 ) ,
      - MU(1)*MU(4) * ( 60*MU(3)**2 - 20*MU(3)*MU(4)**2 + 100*MU(3) +
      MU(4)**4 - 20*MU(4)**2 + 24 ) / ( 120*MU(2)**5 ) ,
      - MU(1) * ( 120*MU(3)**3 - 180*MU(3)**2*MU(4)**2 + 360*MU(3)**2 +
      30*MU(3)*MU(4)**4 - 420*MU(3)*MU(4)**2 + 240*MU(3) - MU(4)**6 +
```

```
40*MU(4)**4 - 154*MU(4)**2 ) / ( 720*MU(2)**6 ),
MU(1)*MU(4) * ( 840*MU(3)**3 - 420*MU(3)**2*MU(4)**2 + 3360*MU(3)**2 +
42*MU(3)*MU(4)**4 - 1260*MU(3)*MU(4)**2 + 3528*MU(3) - MU(4)**6 +
70*MU(4)**4 - 784*MU(4)**2 + 720 ) / ( 5040*MU(2)**7 ),
MU(1) * ( 1680*MU(3)**4 - 3360*MU(3)**3*MU(4)**2 + 10080*MU(3)**3 +
840*MU(3)**2*MU(4)**4 - 16800*MU(3)**2*MU(4)**2 + 18480*MU(3)**2 -
56*MU(3)*MU(4)**6 + 3080*MU(3)*MU(4)**4 - 23744*MU(3)*MU(4)**2 +
10080*MU(3) + MU(4)**8 - 112*MU(4)**6 + 2464*MU(4)**4 -
8448*MU(4)**2 ) / ( 40320*MU(2)**8 ),
- MU(1)*MU(4) * ( 15120*MU(3)**4 - 10080*MU(3)**3*MU(4)**2 +
110880*MU(3)**3 + 1512*MU(3)**2*MU(4)**4 - 60480*MU(3)**2*MU(4)**2 +
263088*MU(3)**2 - 72*MU(3)*MU(4)**6 + 6552*MU(3)*MU(4)**4 -
106848*MU(3)*MU(4)**2 + 219168*MU(3) + MU(4)**8 - 168*MU(4)**6 +
6384*MU(4)**4 - 52352*MU(4)**2 + 40320 ) / ( 362880*MU(2)**9 ),
- MU(1) * ( 30240*MU(3)**5 - 75600*MU(3)**4*MU(4)**2 + 302400*MU(3)**4 +
25200*MU(3)**3*MU(4)**4 - 655200*MU(3)**3*MU(4)**2 + 1058400*MU(3)**3 -
2520*MU(3)**2*MU(4)**6 + 176400*MU(3)**2*MU(4)**4 - 1900080*MU(3)**2*
MU(4)**2 + 1512000*MU(3)**2 + 90*MU(3)*MU(4)**8 - 12600*MU(3)*MU(4)**6 +
372960*MU(3)*MU(4)**4 - 2080800*MU(3)*MU(4)**2 + 725760*MU(3) -
MU(4)**10 + 240*MU(4)**8 - 14448*MU(4)**6 + 229760*MU(4)**4 -
648576*MU(4)**2 ) / ( 3628800*MU(2)**10 ).
167: - ( K*AK(-2) + 2*AK(-2)*MU(3) - 2*AK(-2) + AK(-1)*MU(2)*MU(4) ) /
( K*MU(2)**2 ) )
# TIME (SEC.) TO EXPAND THE SERIES WAS
# TNEW
    7.962641
```

7. SOURCE LISTING OF ALTRAN PROCEDURES

The ALTRAN implementation of the algorithm described in this report is given in this section. Procedure MAIN is a driver program for the method. Procedures TAYLOR, GENREC, DXKP, and SOLREC correspond respectively to the procedures specified in section 5 under the names taylor, generate_recurrence, derivative_x**k_times_p, and solve_recurrence. Finally, procedure SEREVL is used for "series evaluation" in the sense that the general series representation specified by (1):

$$(a_0, a_1, \ldots, a_{k-1}, \textit{recurrence})$$

is expanded out to a specified degree N, yielding the representation

$$(a_0, a_1, \ldots \ldots, a_N, \textit{recurrence}).$$

<u>Index of Procedures</u>:

```
PROCEDURE MAIN

   # MAIN PROCEDURE FOR COMPUTING THE TAYLOR SERIES SOLUTION
   #    OF A LINEAR ORDINARY DIFFERENTIAL EQUATION WITH INITIAL-
   #    VALUE CONDITIONS.
   #
   # THE INPUT IS, IN THE FOLLOWING SEQUENCE:
   #    ODEORD - THE ORDER OF THE DIFFERENTIAL EQUATION;
   #    DIFFEQ - THE DIFFERENTIAL EQUATION AS A MULTINOMIAL IN
   #       THE INDETERMINATES:  X, Y, DY(1), ... , DY(ODEORD) ,
   #       WHERE X IS THE INDEPENDENT VARIABLE, Y IS THE
   #       DEPENDENT VARIABLE, AND DY(I) REPRESENTS THE I-TH
   #       DERIVATIVE OF THE DEPENDENT VARIABLE Y ;
   #    INIT(0); ...; INIT(ODEORD-1) - THE INITIAL CONDITIONS
   #       FOR THE DIFFERENTIAL EQUATION -- I.E. THE VALUES AT
   #       X = 0  OF Y AND ITS FIRST (ODEORD-1) DERIVATIVES.
   #
   # THE OUTPUT IS AN ECHO OF THE INPUT FOLLOWED BY THE TAYLOR
   #    SERIES SOLUTION OF THE ODE.


   INTEGER  ODEORD = SIREAD()
   INTEGER  I
   REAL  TOLD, TNEW

   LONG ALGEBRAIC ( K:31, POWK(-5:10):1, AK(-15:0):1,
      X:31, Y:1, DY(1:IMAX(ODEORD,1)):1, MU(1:5):31 )  DIFFEQ

         # THE SUBSCRIPT RANGE USED BY POWK IS  LOWER : UPPER ,
         # WHERE
         #
         #    LOWER >= -ODEORD ,  AND UPPER <= MAX. DEG. IN X
         #                                     OF DIFFEQ .
         #
         # THE SUBSCRIPT RANGE USED BY AK IS THEN
         #              -(UPPER - LOWER) : 0 .


   LONG ALGEBRAIC ARRAY (0:IMAX(ODEORD-1,0))  INIT
   LONG ALGEBRAIC ARRAY  A

   EXTERNAL ALGEBRAIC  XK=K
   EXTERNAL ALGEBRAIC ARRAY  XPOWK=POWK, XAK=AK

   LONG ALGEBRAIC ARRAY ALTRAN  TAYLOR, SEREVL
```

```
# READ IN THE DIFFERENTIAL EQUATION AND INITIAL CONDITIONS.

READ  DIFFEQ
WRITE  DIFFEQ

INIT(0)  =  DIFFEQ  # JUST DEFINES THE LAYOUT FOR INIT.
DO I = 0, ODEORD-1
   READ  INIT(I)
   WRITE  INIT(I)
DOEND


# OBTAIN THE TAYLOR SERIES SOLUTION OF THE ODE.

A  =  TAYLOR(DIFFEQ, X, Y, DY, INIT)
WRITE  "THE INFINITE TAYLOR SERIES SOLUTION IS"
WRITE  A

TNEW  =  TIME(TOLD)
WRITE  "TIME (SEC.) TO COMPUTE TAYLOR SOLUTION WAS", TNEW


# EXPAND THE SERIES UP TO DEGREE 10.

A  =  SEREVL(A, 10)
WRITE  "THE SERIES EXPANDED UP TO DEGREE 10 IS"
WRITE  A

TNEW  =  TIME(TOLD)
WRITE  "TIME (SEC.) TO EXPAND THE SERIES WAS", TNEW

END  # END OF PROCEDURE MAIN.
```

```
PROCEDURE TAYLOR (ODE, INDEP, DEP, DERIV, INIT)
   LONG ALGEBRAIC VALUE  ODE
   ALGEBRAIC VALUE  INDEP, DEP
   ALGEBRAIC ARRAY VALUE  DERIV
   LONG ALGEBRAIC ARRAY VALUE  INIT

   # PROCEDURE TO SOLVE A LINEAR ODE (INITIAL-VALUE PROBLEM)
   #    WITH POLYNOMIAL COEFFICIENTS AS AN INFINITE TAYLOR
   #    SERIES.
   #
   # INPUT PARAMETERS:
   #    ODE - THE ORDINARY DIFFERENTIAL EQUATION EXPRESSED AS
   #       A POLYNOMIAL (THE DIFFERENTIAL EQUATION IS UNDER-
   #       STOOD TO BE  ODE = 0);
   #    INDEP - THE NAME OF THE INDEPENDENT VARIABLE IN ODE;
   #    DEP - THE NAME OF THE DEPENDENT VARIABLE IN ODE;
   #    DERIV - ARRAY DIMENSIONED FROM 1 TO V, WHERE V IS
   #       GREATER THAN OR EQUAL TO THE ORDER OF THE ODE,
   #       CONTAINING THE NAMES OF THE DERIVATIVES OF THE
   #       DEPENDENT VARIABLE AS THEY APPEAR IN ODE;
   #    INIT - ARRAY DIMENSIONED 0 TO V-1 (V >= ORDER OF ODE)
   #       CONTAINING THE INITIAL CONDITIONS FOR THE PROBLEM.
   #
   # OUTPUT:
   #    THE VALUE RETURNED IS A SERIES REPRESENTED AS A TPS
   # IN WHICH ALL BUT THE LAST ENTRY ARE ACTUAL COEFFICIENTS,
   # WHILE THE LAST ENTRY IS A RECURRENCE EQUATION FOR
   # GENERATING SUCCESSIVE COEFFICIENTS.
   #
   #    ERROR CONDITION:  IF THE VALUE RETURNED IS NULL THEN
   # EITHER THE ODE WAS FOUND TO BE NONLINEAR OR ELSE THE GIVEN
   # LINEAR INITIAL-VALUE PROBLEM DOES NOT HAVE A SOLUTION IN
   # THE FORM OF A TAYLOR SERIES EXPANSION ABOUT THE ORIGIN.
   #
   # PROCEDURES REQUIRED:
   #    GENREC; SOLREC .
```

```
[illegible]
[illegible]
LONG ALGEBRAIC ARRAY ([illegible]) POLY
LONG ALGEBRAIC ARRAY  SERIES
LONG ALGEBRAIC ARRAY ALTRAN  SOLREC


# DETERMINE THE ORDER OF THE ODE.

DO ODEORD = V, 1, -1
   IF ( DEG(ODE, DERIV(ODEORD)) <> 0 )  GO TO OUT
DOEND
ODEORD = 0
OUT:


# PICK OFF THE POLYNOMIAL COEFFICIENTS AND RIGHT HAND SIDE.

POLY(0) = GETBLK (ODE, DEP, 1)
LHS = POLY(0) * DEP
DO I = 1, ODEORD
   POLY(I)  =  GETBLK (ODE, DERIV(I), 1)
   LHS  =  LHS + POLY(I) * DERIV(I)
DOEND
RHS  =  LHS - ODE


# VERIFY THAT THE ODE IS LINEAR.

DO I = 0, ODEORD

   IF ( DEG(POLY(I), DEP) <> 0 )  RETURN

   DO J = 1, ODEORD
      IF ( DEG(POLY(I), DERIV(J)) <> 0 )  RETURN
   DOEND

DOEND

IF ( DEG(RHS, DEP) <> 0 )  RETURN

DO J = 1, ODEORD
   IF ( DEG(RHS, DERIV(J)) <> 0 )  RETURN
DOEND


# GENERATE AND SOLVE THE RECURRENCE EQUATION.

GENREC (ODEORD, POLY, INDEP, EQN, N, SHIFT)

SERIES  =  SOLREC (EQN, N, SHIFT, RHS, INDEP, INIT, ODEORD)


RETURN ( SERIES )

END  # END OF PROCEDURE TAYLOR.
```

```
PROCEDURE GENREC (ODEORD, POLY, X, EQN, N, SHIFT)
  INTEGER VALUE  ODEORD
  LONG ALGEBRAIC ARRAY VALUE  POLY
  ALGEBRAIC VALUE  X
  LONG ALGEBRAIC  EQN
  INTEGER  N, SHIFT

  EXTERNAL ALGEBRAIC  XK
  EXTERNAL ALGEBRAIC ARRAY  XAK, XPOWK

  # PROCEDURE TO GENERATE THE RECURRENCE EQUATION FOR THE
  #    TAYLOR SERIES SOLUTION OF A LINEAR ODE WITH POLYNOMIAL
  #    COEFFICIENTS.
  #
  # INPUT PARAMETERS:
  #    ODEORD - THE ORDER OF THE DIFFERENTIAL EQUATION;
  #    POLY - ARRAY OF LEFT HAND SIDE POLYNOMIALS SUCH THAT
  #       POLY(ODEORD) IS THE COEFFICIENT OF THE HIGHEST-ORDER
  #       DERIVATIVE, ETC.;
  #    X - THE NAME OF THE INDETERMINATE IN THE POLYNOMIALS.
  #
  # OUTPUT PARAMETERS:
  #    EQN - THE LEFT SIDE OF THE RECURRENCE EQUATION;
  #    N - THE LENGTH OF THE RECURRENCE EQUATION (I.E. EQN IS
  #       OF THE FORM
  #          U0*AK(0) + U1*AK(-1) + ... + UN*AK(-N) );
  #    SHIFT - INDICATES THAT THE L.H.S. OF THE ODE HAS BEEN
  #       EXPRESSED IN THE FORM
  #             SUM ( EQN * X**(K-SHIFT) )
  #       WHERE THE SUM IS OVER  K = SHIFT, SHIFT+1, ... .
  #
  # EXTERNAL VARIABLES:
  #    XK - THE NAME OF THE INDETERMINATE INDEX (K) WHICH WILL
  #       APPEAR IN THE COEFFICIENTS OF THE RECURRENCE EQUATION;
  #    XAK - THE ARRAY OF INDETERMINATES SUCH THAT XAK(J) WILL
  #       APPEAR IN THE RECURRENCE EQUATION REPRESENTING
  #       A(K+J) WHERE K IS AN INDETERMINATE;
  #    XPOWK - THE ARRAY OF INDETERMINATES SUCH THAT XPOWK(J)
  #       IS USED TO REPRESENT THE MONOMIAL X**(K+J) WHERE K
  #       IS AN INDETERMINATE.
  #
  # PROCEDURES REQUIRED:
  #    DXKP.


  INTEGER  DEGREE, I, J, JMAX, JMIN
  LONG ALGEBRAIC  FACTOR, U0, COEF
  LONG ALGEBRAIC ALTRAN  DXKP
```

```
# THE LEFT SIDE OF THE ODE IS
#       PM*(D**M)Y  +  . . .  +  P1*(D)Y  +  P0*Y ,
# WHERE  D = D/DX  AND  M = ODEORD .  THE SUBSTITUTION
#       Y  =  SUM ( A(K) * X**K )
# (WHERE THE SUM IS OVER  K = 0, 1, 2, ..) CONVERTS THE
# LEFT SIDE OF THE ODE INTO THE FORM
#       SUM ( A(K) * FACTOR ),  K = 0, 1, 2, ..
# WHERE
#       FACTOR  =  P0 * X**K  +  P1 * (D)X**K  +  . . .
#                                  +  PM * (D**M)X**K .
# THE FOLLOWING CODE COMPUTES THE EXPRESSION FACTOR.

JMAX  =  - ODEORD
FACTOR  =  0

DO I = 0, ODEORD
   IF ( POLY(I) <> 0 )  DO
      DEGREE  =  DEG (POLY(I), X)
      JMAX  =  IMAX (JMAX, DEGREE-I)
      FACTOR  =  FACTOR  +  DXKP (I, POLY(I), X)
   DOEND
DOEND


# THE EXPRESSION FACTOR (SEE COMMENT ABOVE) IS OF THE FORM
#       U0*XPOWK(JMIN)  +  U1*XPOWK(JMIN+1)  +  . . .
#                                  +  UN*XPOWK(JMAX)
# WHERE  N = JMAX-JMIN  AND WHERE  JMIN >= -ODEORD.  IN EACH
# TERM OF THIS EXPRESSION, A CHANGE OF INDEX IS PERFORMED
# TO OBTAIN THE COEFFICIENT OF  X**(K+JMIN).  THIS YIELDS
# THE RECURRENCE EQUATION IN THE FORM
#       U0*AK(0) + U1*AK(-1) + ... + UN*AK(-N) .


# COMPUTE N AND SHIFT, AFTER DETERMINING THE VALUE OF JMIN.

     JMIN  =  - ODEORD
LOOP: U0  =  GETBLK (FACTOR, XPOWK(JMIN), 1)
     IF ( U0 == 0 )  DO;  JMIN = JMIN+1;  GO TO LOOP;  DOEND

N  =  JMAX - JMIN
SHIFT  =  - JMIN


# FORM THE RECURRENCE EQUATION INTO VARIABLE EQN.

EQN  =  U0 * XAK(0)
DO J = JMIN+1, JMAX
   COEF  =  GETBLK (FACTOR, XPOWK(J), 1)
   COEF  =  COEF (XK = XK+JMIN-J)
   EQN  =  EQN  +  COEF * XAK(JMIN-J)
DOEND


END  # END OF PROCEDURE GENREC
```

```
PROCEDURE DXKP (ORDER, P, X)
   INTEGER VALUE  ORDER
   ALGEBRAIC VALUE  X
   LONG ALGEBRAIC VALUE  P

   EXTERNAL ALGEBRAIC  XK
   EXTERNAL ALGEBRAIC ARRAY  XPOWK

   # PROCEDURE TO COMPUTE THE POLYNOMIAL  (D**ORDER)X**K * P ,
   #    WHERE K IS AN INDETERMINATE, D STANDS FOR THE
   #    DIFFERENTIATION OPERATOR D/DX, AND P IS AN ORDINARY
   #    POLYNOMIAL IN THE INDETERMINATE X.  THE MONOMIAL
   #    X**(K+J) IS REPRESENTED BY THE INDETERMINATE XPOWK(J),
   #    WHERE XPOWK IS AN EXTERNAL ARRAY OF INDETERMINATES,
   #    AND THE EXTERNAL VARIABLE XK SPECIFIES THE NAME OF THE
   #    INDETERMINATE WHICH WILL REPRESENT K IN THE RESULT.
   #    THE FOLLOWING DECLARATION MUST APPEAR IN THE CALLING
   #    PROCEDURE:
   #          LONG ALGEBRAIC ALTRAN  DXKP .


   INTEGER  J
   LONG ALGEBRAIC  NEWP, COEF


   # USING D TO DENOTE THE DIFFERENTIATION OPERATOR D/DX,
   # (D**ORDER)X**K  =  K*(K-1)* ... *(K-ORDER+1) * X**(K-ORDER).

   # FIRST FORM  X**(K-ORDER) * P .

   NEWP  =  0

   DO J = 0, DEG(P,X)
      COEF  =  GETBLK (P, X, J)
      NEWP  =  NEWP  +  COEF * XPOWK(-ORDER+J)
   DOEND


   # NOW ATTACH THE FACTORS  K*(K-1)* ... *(K-ORDER+1) .

   DO J = 0, ORDER-1
      NEWP  =  (XK-J) * NEWP
   DOEND

   RETURN (NEWP)

END  # END OF PROCEDURE DXKP.
```

```
PROCEDURE SOLREC (EQN, N, SHIFT, RHS, X, INIT, ODEORD)
   INTEGER VALUE  N, SHIFT, ODEORD
   ALGEBRAIC VALUE  X
   LONG ALGEBRAIC VALUE  EQN, RHS
   LONG ALGEBRAIC ARRAY VALUE  INIT

   EXTERNAL ALGEBRAIC  XK
   EXTERNAL ALGEBRAIC ARRAY  XAK


   # PROCEDURE TO SOLVE THE RECURRENCE EQUATION FOR THE TAYLOR
   #    SERIES SOLUTION OF A LINEAR ODE WITH INITIAL-VALUE
   #    CONDITIONS.
   #
   # INPUT PARAMETERS:
   #    EQN - THE LEFT SIDE OF THE RECURRENCE EQUATION;
   #    N - THE LENGTH OF THE RECURRENCE EQUATION (I.E. EQN IS
   #       OF THE FORM
   #          U0*AK(0) + U1*AK(-1) + ... + UN*AK(-N)  );
   #    SHIFT - INDICATES THAT THE L.H.S. OF THE ODE HAS BEEN
   #       EXPRESSED IN THE FORM
   #             SUM ( EQN * X**(K-SHIFT) )
   #       WHERE THE SUM IS OVER K = SHIFT, SHIFT+1, ... ;
   #    RHS - THE RIGHT HAND SIDE OF THE ODE, AS AN ORDINARY
   #       POLYNOMIAL;
   #    X - THE VARIABLE APPEARING IN RHS;
   #    INIT - ARRAY DIMENSIONED FROM 0 TO ODEORD-1 CONTAINING
   #       THE INITIAL CONDITIONS FOR THE ODE -- IF INIT IS
   #       NULL THEN IF SHIFT <= 0 THERE IS ONLY ONE SOLUTION
   #       IN THE FORM OF A TAYLOR SERIES EXPANSION ABOUT THE
   #       ORIGIN AND THIS SOLUTION IS COMPUTED;
   #    ODEORD - THE ORDER OF THE ODE.
   #
   # OUTPUT:
   #    THE VALUE RETURNED IS A SERIES REPRESENTED AS A TPS IN
   # WHICH ALL BUT THE LAST ENTRY ARE ACTUAL TAYLOR SERIES
   # COEFFICIENTS FOR THE SOLUTION OF THE ODE, WHILE THE LAST
   # ENTRY IS A RECURRENCE EQUATION FOR GENERATING SUCCESSIVE
   # TAYLOR SERIES COEFFICIENTS.  THIS RECURRENCE EQUATION IS
   # IN A FORM SUCH THAT, UPON SUBSTITUTING FOR THE INDETER-
   # MINATE (SPECIFIED BY EXTERNAL VARIABLE XK) A VALUE K,
   # THE K-TH TAYLOR SERIES COEFFICIENT IS OBTAINED IN TERMS
   # OF THE PRECEDING N COEFFICIENTS.
   #    ERROR CONDITION:  IF THE VALUE RETURNED IS NULL THEN
   # ONE OF THE FOLLOWING TWO SITUATIONS HAS OCCURRED:
   #    (1) THERE WAS AN INCONSISTENCY IN THE EQUATIONS, SIG-
   #       NIFYING THAT THE GIVEN PROBLEM DOES NOT HAVE A
   #       SOLUTION IN THE FORM OF A TAYLOR SERIES EXPANSION
   #       ABOUT THE ORIGIN;  OR
   #    (2) IN CASE INIT IS NULL AND SHIFT > 0, A NULL VALUE
   #       IS RETURNED BECAUSE THERE IS NOT A UNIQUE TAYLOR
   #       SERIES SOLUTION FOR THE ODE WITHOUT SPECIFYING
   #       SOME INITIAL CONDITIONS.
   #
```

```
# EXTERNAL VARIABLES:
#    XK -- THE NAME OF THE INDETERMINATE APPEARING IN THE
#        COEFFICIENTS OF THE RECURRENCE EQUATION;
#    XAK -- THE ARRAY OF INDETERMINATES SUCH THAT XAK(J)
#        APPEARS IN THE RECURRENCE EQUATION REPRESENTING
#        A(K+J), WHERE K IS AN INDETERMINATE.
```

```
INTEGER  I, FACT, KVAL
LONG ALGEBRAIC  U0, AK0, AKVAL, COEF
LONG ALGEBRAIC ARRAY  SOLN, UNLIST, VALIST

# USE THE INITIAL CONDITIONS TO DEFINE THE FIRST ODEORD
#    COEFFICIENTS, UNLESS INIT IS NULL.

IF ( NOT NULL(INIT) )  DO
   FACT  =  1
   DO KVAL = 0, ODEORD-1
      SOLN  =  ( SOLN, INIT(KVAL) / FACT )
      FACT  =  FACT * (KVAL + 1)
   DOEND
 DOEND

ELSE  IF ( SHIFT > 0 )  RETURN   # NOT A UNIQUE SOLUTION.

# IN EQN, DETERMINE THE LEADING COEFFICIENT U0 AND SOLVE
#    THE EQUATION "EQN = 0" FOR AK(0).

U0  =  GETBLK (EQN, XAK(0), 1)

AK0  =  XAK(0)  -  EQN / U0

# SET UP THE LIST OF UNKNOWNS WHICH MAY APPEAR IN THE
#    EXPRESSION AK0

DO I = 1, N
  UNLIST  =  ( UNLIST, XAK(-I) )
DOEND
```

```
# THE NEXT BLOCK OF CODE IS EXECUTED ONLY IF  SHIFT < ODEORD.
# THERE ARE THREE POSSIBLE ACTIONS:
#    (1) IF SHIFT < 0 THEN IF THE APPROPRIATE R.H.S. COEF-
#        FICIENTS ARE NOT ZERO, THE COMPUTATION IS ABORTED.
#    (2) IF INIT IS NOT NULL THEN IF THE GIVEN INITIAL
#        CONDITIONS ARE NOT CONSISTENT WITH A TAYLOR SERIES
#        SOLUTION, THE COMPUTATION IS ABORTED.
#    (3) IF INIT IS NULL THEN THE FIRST ODEORD TAYLOR SERIES
#        COEFFICIENTS ARE COMPUTED HERE.

DO KVAL = SHIFT, ODEORD-1

   VALIST  =  0 $ VALIST

   DO I = 1, IMIN(KVAL,N)
      VALIST  =  ( VALIST, SOLN(KVAL-I+1) )
   DOEND

   IF (KVAL < 0)  VALIST  =  N $ 0
   ELSE  IF (N > KVAL)  VALIST  =  ( VALIST, (N-KVAL) $ 0 )


   COEF  =  GETBLK (RHS, X, KVAL-SHIFT)
   AKVAL  =  AK0 (XK = KVAL) (UNLIST = VALIST)
   AKVAL  =  AKVAL  +  COEF / U0 (XK = KVAL)


   IF ( KVAL < 0  .AND.  AKVAL <> 0 )  RETURN   # NO TAYLOR
                                                #  SOLUTION.

   IF ( NULL(INIT) )  SOLN  =  ( SOLN, AKVAL )
   ELSE  IF ( AKVAL <> SOLN(KVAL+1) )  RETURN   # NO TAYLOR
                                                #  SOLUTION.

DOEND


# THE FIRST N COEFFICIENTS MUST BE EXPLICITLY SPECIFIED FOR
#    THE SERIES.

DO KVAL = ODEORD, N-1

   VALIST  =  0 $ VALIST

   DO I = 1, KVAL
      VALIST  =  ( VALIST, SOLN(KVAL-I+1) )
   DOEND

   VALIST  =  ( VALIST, (N-KVAL) $ 0 )


   COEF  =  GETBLK (RHS, X, KVAL-SHIFT )
   AKVAL  =  AK0 (XK = KVAL) (UNLIST = VALIST)
   AKVAL  =  AKVAL  +  COEF / U0 (XK = KVAL)


DOEND
```

```
# ALL COEFFICIENTS WHICH DEPEND ON THE RHS POLYNOMIAL MUST
#    BE SPECIFIED.

IF ( RHS <> 0 ) _

   DO  KVAL  =  IMAX(N, ODEORD),  DEG(RHS,X) + SHIFT

      VALIST  =  0 $ VALIST

      DO I = 1, N
         VALIST  =  ( VALIST, SOLN(KVAL-I+1) )
      DOEND

      COEF  =  GETBLK (RHS, X, KVAL-SHIFT)
      AKVAL  =  AK0 (XK = KVAL) (UNLIST = VALIST)
      AKVAL  =  AKVAL  +  COEF / U0 (XK = KVAL)

      SOLN  =  ( SOLN, AKVAL )

   DOEND

# THE LAST ENTRY IN THE SOLUTION IS THE RECURRENCE EQUATION
#    FOR GENERATING SUCCESSIVE TAYLOR SERIES COEFFICIENTS.

SOLN  =  ( SOLN, AK0 )

TPSORD ( SOLN )  # CONVERTS TO A TPS -- I.E. INDEXED FROM 0.

RETURN ( SOLN )

END  # END OF PROCEDURE SOLREC.
```

```
PROCEDURE SEREVL (SERIES, APPDEG)
LONG ALGEBRAIC ARRAY VALUE  SERIES
INTEGER VALUE  APPDEG

EXTERNAL ALGEBRAIC  XK
EXTERNAL ALGEBRAIC ARRAY  XAK


# PROCEDURE TO EVALUATE A SERIES REPRESENTED AS A TPS IN
#    WHICH ALL BUT THE LAST ENTRY ARE ACTUAL COEFFICIENTS
#    WHILE THE LAST ENTRY IS A RECURRENCE FORMULA FOR
#    GENERATING SUCCESSIVE COEFFICIENTS.  IT IS ASSUMED
#    THAT THE NUMBER OF TERMS IN THE RECURRENCE IS LESS
#    THAN OR EQUAL TO THE NUMBER OF ACTUAL COEFFICIENTS IN
#    SERIES.  THE VALUE RETURNED IS A SERIES USING THE SAME
#    REPRESENTATION AS THE INPUT SERIES (I.E. THE LAST
#    ENTRY IS THE RECURRENCE FORMULA), BUT THE COEFFICIENTS
#    THROUGH DEGREE APPDEG (AT LEAST) APPEAR EXPLICITLY IN
#    THE RETURNED SERIES.  THE INPUT SERIES IS PRECISELY
#    THE RETURNED SERIES IN THE CASE WHERE THE FORMER
#    SERIES ALREADY EXPLICITLY CONTAINS THE COEFFICIENTS
#    THROUGH DEGREE APPDEG.  THE FOLLOWING DECLARATION MUST
#    APPEAR IN THE CALLING PROCEDURE:
#          LONG ALGEBRAIC ARRAY ALTRAN  SEREVL .


INTEGER  I, N, NLIM, LAST, KVAL
LONG ALGEBRAIC  RECURR
LONG ALGEBRAIC ARRAY  UNLIST, VALIST
LONG ALGEBRAIC ARRAY (0 : APPDEG+1)  NEWSER


# CHECK FOR IMMEDIATE RETURN.

LAST  =  DBINFO(SERIES)(1,1)

IF ( LAST >= APPDEG+1 )  RETURN ( SERIES )


# DETERMINE THE NUMBER OF TERMS IN THE RECURRENCE FORMULA.

RECURR  =  SERIES(LAST)

NLIM  =  - DBINFO(XAK)(1,0)

DO  N  =  (MIN(LAST, NLIM), 1, -1
  IF ( DEG ( RECURR, XAK(-N) ) <> 0 )  GO TO OUT
DOEND
  OUT:
```

```
# SET UP THE LIST OF UNKNOWNS WHICH MAY APPEAR IN THE
#    RECURRENCE FORMULA.

DO I = 1, N
   UNLIST  =  ( UNLIST, XAK(-I) )
DOEND


# THE FIRST "LAST" COEFFICIENTS ARE ALREADY AVAILABLE.

DO  KVAL  =  0,  LAST-1
   NEWSER(KVAL)  =  SERIES(KVAL)
DOEND


# THE REMAINING COEFFICIENTS MUST BE COMPUTED FROM THE
#    RECURRENCE FORMULA.

DO KVAL = LAST, APPDEG

   VALIST  =  0 $ VALIST

   DO I = 1, N
      VALIST  =  ( VALIST, NEWSER(KVAL-I) )
   DOEND

   NEWSER(KVAL)  =  RECURR (XK = KVAL) (UNLIST = VALIST)

DOEND


# ATTACH THE RECURRENCE FORMULA.

NEWSER(APPDEG+1)  =  RECURR


RETURN ( NEWSER )

END  # END OF PROCEDURE SEREVL .
```

REFERENCES

1. J. Cohen and J. Katcoff, Symbolic solution of finite-difference equations. ACM Trans. Math. Softw., 3(3), Sept. 1977, pp. 261-271.

2. Richard J. Fateman, Some comments on series solutions. Proc. of the 1977 MACSYMA Users' Conference, NASA: Washington, D.C., 1977, pp. 327-346.

3. John Ivie, Some MACSYMA programs for solving recurrence relations. ACM Trans. Math. Softw., 4(1), Mar. 1978, pp. 24-33.

4. Edward L. Lafferty, Power series solutions of ordinary differential equations in MACSYMA. Proc. of the 1977 MACSYMA Users' Conference, NASA: Washington, D.C., 1977, pp. 347-360.

5. A.C. Norman, Computing with formal power series. ACM Trans. Math. Softw., 1(4), Dec. 1975, pp. 346-356.