Univ. of Science and Technology) 19 p
HC A02/MF A01 CSCL 12A Unclas
G3/64 24631

SYMBOLIC SOLUTION OF LINEAR DIFFERENTIAL EQUATIONS

Robert B. Feinberg[(1)] Ronald G. Grooms[(2)]

ABSTRACT

This paper presents and justifies a new algorithm for solving linear constant-coefficient ordinary differential equations. It also discusses the computational complexity of the algorithm and describes its implementation in the FORMAC system. It concludes with a comparison between the algorithm and some classical algorithms for solving differential equations that have been previously implemented.

Key Words and Phrases: symbolic manipulation, linear ordinary differential equations, Euclidean algorithm.

C R Category: 5.7

(1) Mathematics Department, Iowa State University, Ames, Iowa 50011. This author's work is supported in part by the National Aeronautics and Space Administration under grant NSG-1538.
(2) Computation Center, Iowa State University, Ames, Iowa 50011.

INTRODUCTION

The class of problems we are solving are those of the type: find a particular solution y_p to the ordinary differential equation $L[D]y = f(x)$, where $L(t)$ is a real polynomial $a_n t^n + a_{n-1} t^{n-1} + \ldots + a_1 t + a_0$, D^r denotes r-fold differentiation and $f(x)$ is a linear combination of terms of the form $x^n \exp(\alpha+i\beta)x$, α and β real and n an integer. Note that we may restrict our attention to $f(x)$ which are monomials of the above form, by using linear superposition of solutions. Note also that we may obtain the solution of $L[D]y = x^n \exp \alpha x$ (cox βx or sin βx) by finding the real or imaginary part of the solution of $L[D]y = x^n \exp(\alpha+i\beta)x$.

The general solution of $L[D]y = f(x)$ may be obtained by adding a particular solution of this equation to the general solution of $L[D]y = 0$. The latter is determined by the roots of the polynomial $L(t)$, which in general cannot be found exactly. We shall not consider this problem further here.

ALGORITHM AND JUSTIFICATION

To present the algorithm for finding a particular solution of $L[D]y = f(x) = x^n \exp(\alpha+i\beta)x$, we consider separately the cases for $\beta = 0$ and $\beta \neq 0$.

Case 1 $\beta = 0$.

Step 1 Set $M(t) = (t-\alpha)^{n+1}$.

Step 2 Use the extended Euclidean algorithm to determine polynomials $A(t)$ and $B(t)$ such that $A(t)L(t)/(t-\alpha)^r + B(t)M(t) = I$, where $r \geq 0$ is the multiplicity of α as a root of $L(t)$.

Step 3 Set $y_p(x) = \exp \alpha x \, D^{-r} A[D+\alpha]x^n$, where $D^{-r} x^j = j! \, x^{j+r}/(j+r)!$.

Case 2 $\beta \neq 0$.

Step 1 Set $M(t) = (t^2-2\alpha t+\alpha^2+\beta^2)^{n+1}$.

Step 2 Determine $A(t)$, $B(t)$ such that $A(t)L(t)/(t^2-2\alpha t+\alpha^2+\alpha^2)^r + B(t)M(t) = I$, where r is the multiplicity of $\alpha \pm i\beta$ as a root of $L(t)$.

Step 3 Set $y_p = \exp \alpha x[D^2+\beta^2]^{-r} A[D+\alpha] \, x^n \exp(i\beta x)$, where

$[D^2+\alpha^2]^{-r} x^j \exp(i\beta x) = \exp(i\beta x) \, C_{r,j} \, U_{r,j}(x)$, with

$C_{r,j} = j!(-1)^r i^{r+j}/(r-1)!(2\beta)^{r+j}$ and

$$U_{r,j}(x) = \sum_{k=0}^{j} (r+j-k-1)!(-2i\beta)^k \, x^{r+k}/(j-k)!(r+k)! \quad .$$

We now establish the correctness of the algorithm in case 1.

Theorem The function $y_p(x)$ determined by the algorithm in case 1 is a particular solution to $L[D]y = f(x) = x^n \exp \alpha x$.

Proof It is readily verified that $M[D]f(x) = 0$, where $M(t)$ is as defined in step 1. Note also that $L(t)/(t-\alpha)^r$ and $M(t)$ are relatively prime, so that there are polynomials $A(t)$ and $B(t)$ as described in step 2. Note as well that $D^r \, D^{-r}x^j = x^j$, where D^{-r} is as described in step 3.

It then follows that $L[D]y_p = \{L[D]/[D-\alpha]^r\}\ [D-\alpha]^r \exp \alpha x\ D^{-r}\ A[D+\alpha]x^n =$

$\{L[D]/[D-\alpha]^r\} \exp \alpha x\ D^r\ D^{-r}\ A[D+\alpha]x^n$, using the exponential shift, $=$

$\{L[D]/[D-\alpha]^r\} \exp \alpha x\ A[D+\alpha]x^n = \{L[D]/[D-\alpha]^r\}\ A[D] \exp \alpha x\ x^n = \{I - B[D]M[D]\}f(x) = f(x)$,

since $M[D]f(x) = 0$. Q.E.D.

The following result is needed to prove the correctness of the algorithm in case 2.

Proposition 1 For all positive integers r, non-negative integers j, and real numbers β, $[D^2+\beta^2]^r \exp(i\beta x) = C_{r,j}U_{r,j}(x) = x^j \exp(i\beta x)$, $C_{r,j}$ and $U_{r,j}(x)$ being as defined in step 3.

Proof Set $F_{r,j}(x) = \exp(i\beta x)\ C_{r,j}\ U_{r,j}(x)$. The result is equivalent to $[D^2+\beta^2]^r\ F_{r,j}(x) = x^j \exp(i\beta x)$, for all positive integers r, non-negative integers j, and real numbers β. The latter result is established by induction on r.

For $r = 1$, j arbitrary,

$$[D^2+\beta^2]\ F_{1,j}(x) = \frac{-\exp(i\beta x)\ [D+2i\beta]D\ j!\ i^{j+1}}{(2\beta)^{j+1}} \sum_{k=0}^{j} \frac{(-2i\beta)^k\ x^{k+1}}{(k+1)!} =$$

$$-\frac{j!\ i^{j+1}\exp(i\beta x)}{(2\beta)^{j+1}} \left\{ \sum_{k=1}^{j} \left[\frac{(-2i\beta)^k\ x^{k-1}}{(k-1)!} - \frac{(-2i\beta)^{k+1}\ x^k}{k!} \right] - (-2i\beta) \right\} = x^j \exp(i\beta x)\ .$$

Hence the result is valid for $r = 1$, j arbitrary.

Suppose now that the result is valid for $r \le v$, j arbitrary. Then $C_{v+1,j} = (-i/2\beta v)\ C_{v,j}$ and it follows from the inductive hypothesis that $[D^2+\beta^2]^N\ F_{v,j}(x) = x^j \exp(i\beta x)$. Thus $[D+2i\beta]^v\ D^v\ C_{v,j}\ U_{v,j}(x) = x^j$. In addition, $[D^2+\beta^2]\ F_{v+1,j}(x) = (i/2\beta v)\ C_{v,j}\ \exp(i\beta x)[D+2i\beta]D\ U_{v+1,j}(x)$, and it follows from lengthy, but routine, computations that $[D+2i\beta]D\ U_{v+1,j}(x)$

$= \{(v+j)!x^{v-1}/j!(v-1)!\} + 2i\beta v\ U_{v,j}(x)$. Hence $[D^2+\beta^2]^{v+1}\ F_{v+1,j}(x)$

$$= (-i/2\beta v)\ C_{v,j}\ \exp(i\beta x)[D+2i\beta]^{v+1}\ D^{v+1}\ U_{v+1,j}(x)$$

$$= (-i/2\beta v)\ C_{v,j}\ \exp(i\beta x)[D+2i\beta]^{v}\ D^{v}\ \{(v+j)!x^{v-1}/j!(v-1)!\} + 2ivU_{v,j}(x)$$

$$= x^j\ \exp(i\beta x)$$

establishing the result for $r = v+1$, j arbitrary. Hence the result holds for all r and j. Q.E.D.

With this result, the proof of the correctness of the algorithm in case 2 is similar to the proof in case 1, hence we will omit it.

The next result is used to determine the polynomials $A(t)$ and $B(t)$ mentioned in step 2 of the algorithm (both cases). It is readily verified by induction. For completeness we state it for a Euclidean Domain, which is a generalization of the ring of polynomials over a field.

Proposition 2 Let $\bar{L}$ and N be relatively prime elements of a Euclidean Domain D, and A_0, $B_0 \in D$ be such that $A_0\bar{L} + B_0N = I$. For s a non-negative integer, define A_{s+1} and B_{s+1} recursively by $A_{s+1} = A_s[I+B_sN^{(2^s)}]$, $B_{s+1} = B_s^{\ 2}$. Then $A_s\bar{L} + B_sN^{(2^s)} = I$, all s.

IMPLEMENTATION

We have implemented the above algorithm as a program in the FORMAC73 system [2]. In the program's notation, the input is the polynomial L and the function F. The output is a particular solution YP to the equation L[D]Y = F(X), a FORMAC chain containing the individual terms of YP, and an integer giving the length of the chain. The function F is required to be of the form $X^N \exp \alpha X(\cos \beta X$ or $\sin \beta X)$ and X is used as the independent variable in L, F, and the solution. THe program includes a check on the correctness of the particular solution YP obtained, by direct substitution into the original differential equation.

Operations on coefficients are performed using rational mode arithmetic. Users who find large rational coefficients inconvenient to work with may wish to truncate them or convert them to floating point. Programs for doing this are given in [6].

The code for our program follows below.

```
DRIVER:  PROCEDURE OPTIONS(MAIN) REORDER;
/*                                                                          */
/* THIS TEST DRIVER IS GIVEN TO ILLUSTRATE THE METHOD                       */
/* OF CALLING LINODE.  FORMAL PARAMETERS ARE PASSED AS                      */
/* PL/I CHARACTER STRINGS OF LENGTH EIGHT.  FOR THIS EXAMPLE                */
/* L AND F ARE READ FROM INPUT FILE SYSIN.                                  */
/*                                                                          */
/* INPUT DATA FOLLOWS //GO.SYSIN DD * CARD.                                 */
/* NOTE:  IMBEDDED BLANKS ARE IGNORED; INPUT IS FREE FORM.                  */
/*        DATA FOR AS MANY ODES AS DESIRED MAY BE INCLUDED.                 */
/*                                                                          */
/* EXAMPLE:                                                                 */
/*  INPUT DATA FOR L(X)=5*X**2 + 3*X -1                                     */
/*            AND F(X)= X**2 * #E**(2*X) * SIN(3*X) IS OF THE FORM:         */
/* '5*X**2 + 3*X - 1'   'X**2 * #E**(2*X) * SIN(3*X)'                       */
/*                                                                          */
/* THE FOLLOWING DCL IS NECESSARY IN ANY DRIVER THAT USES                   */
/* LINODE AS AN EXTERNAL PROCEDURE.                                         */
/*                                                                          */
DCL LINODE ENTRY(CHAR(8),CHAR(8),CHAR(8),CHAR(8),CHAR(8));
DCL (LPIN,FIN) CHAR(72);
ON ENDFILE (SYSIN) GOTO DONE;
FORMAC_OPTIONS: OPTSET(EXPND);
OPTSET (LINELENGTH=72);
DO WHILE ('1'B);
  GET LIST (LPIN,FIN);
  PRINT_OUT (LX="LPIN"; FX="FIN");
  CALL LINODE ('LX', 'FX', 'SOLN', 'LIST', 'N');
  PRINT_OUT (R=SOLN; S=N; T=CHAIN(LIST));
END;
DONE:  PUT SKIP(2) EDIT ('END OF O.D.E. SOLVER')(A);
END DRIVER;
```

```
LINODE:  PROCEDURE (L,F,YP,TERMS,NTERMS) REORDER;
/*                                                                      */
/* THIS PROGRAM COMPUTES PARTICULAR SOLUTIONS OF N-TH ORDER             */
/* LINEAR ORDINARY DIFFERENTIAL EQUATIONS OF THE FORM L(D)Y=F(X)       */
/* WHERE L(X)=A(N)*X**N + A(N-1)*X**(N-1) + ... + A(1)*X + A(0)        */
/* WHERE THE A(I) ARE RATIONAL CONSTANTS AND F IS A (RATIONAL) LINEAR*/
/* COMBINATION OF TERMS OF THE FORM X**N * #E**(ALFA*X) * G(X)         */
/* WHERE G(X)= SIN(BETA*X) OR G(X)= COS(BETA*X).                        */
/* N IS A NON-NEGATIVE INTEGER.                                         */
/* ALFA AND BETA ARE RATIONAL CONSTANTS.                                */
/* (E.G.  F=X, F=COS(X), F=X + #E**(4*X)*SIN(4*X)).                    */
/*                                                                      */
/* FORMAL PARAMETERS ARE PASSED TO LINODE AS PL/I CHARACTER            */
/* STRINGS OF LENGTH EIGHT.                                             */
/*                                                                      */
/* DATA GOING IN:                                                       */
/*   FIRST ARGUMENT:  POLYNOMIAL L(X).                                  */
/*   SECOND ARGUMENT:  FORCING FUNCTION F(X).                           */
/* RESULTS COMING OUT:                                                  */
/*   THIRD ARGUMENT:  PARTICULAR SOLUTION TO L(D)Y=F(X).                */
/*   FOURTH ARGUMENT:  FORMAC CHAIN CONTAINING THE INDIV-               */
/*                     IDUAL TERMS OF THE PARTICULAR SOLUTION.          */
/*   FIFTH ARGUMENT:  INTEGER LENGTH OF THE ABOVE CHAIN.                */
/*                                                                      */
/* LINODE WILL EXPAND L.                                                */
/*                                                                      */
/* THE FOLLOWING DCL MUST BE IN ANY PROGRAM THAT USES LINODE           */
/* AS AN EXTERNAL PROCEDURE:                                            */
/*   DCL LINODE ENTRY(CHAR(8),CHAR(8),CHAR(8),CHAR(8),CHAR(8));         */
/*                                                                      */
/* LINODE CONTAINS THE FOLLOWING INTERNAL PROCEDURES:                   */
/*   POLYDIV:  WHICH DIVIDES A POLYNOMIAL BY ANOTHER POLY-              */
/*     NOMIAL AND RETURNS THE QUOTIENT AND REMAINDER.                   */
/*   RE:  WHICH COMPUTES THE REAL PART OF A COMPLEX NUMBER.             */
/*   IM:  WHICH COMPUTES THE IMAGINARY PART OF A COMPLEX                */
/*     NUMBER.                                                          */
/*   POLYDOP:  WHICH APPLIES A POLYNOMIAL ARGUMENT AS A                 */
/*     DIFFERENTIAL OPERATOR TO A SECOND ARGUMENT.                      */
/*   CU:  WHICH COMPUTES (COS(BETA*X)+#I*SIN(BETA*X))*C*U .             */
/*     NOTE:  FORMAC VARIABLES BETA & R USED IN CU ARE                  */
/*       GLOBAL TO LINODE.                                              */
/*   DIVMFAC:  WHICH DIVIDES A POLYNOMIAL BY A GIVEN FACTOR             */
/*     AND RETURNS A QUOTIENT AND MULTIPLICITY.                         */
/*                                                                      */
/* ADDITIONAL COMMENTS ON THE ABOVE MENTIONED INTERNAL                  */
/* PROCEDURES ARE GIVEN IN THE INDIVIDUAL PROCEDURES.                   */
/*                                                                      */
/* DEFINITION OF FORMAC VARIABLES USED IN LINODE:                       */
/*   ALFA:  ALPHA.                                                      */
/*   BETA:  BETA.                                                       */
/*   AP:  A(I).                                                         */
/*   BP:  B(I).                                                         */
/*   B2:  TEMPORARY VARIABLE USED IN COMPUTING A(0) & B(0).             */
/*   B3:  TEMPORARY VARIABLE USED IN COMPUTING A(0) & B(0)              */
/*        WHEN BETA IS PRESENT IN F.                                    */
/*   CP:  USED TO CHECK SOLUTION YP BY SUBSTITUTION.                    */
/*   F:  A TERM OF FT.                                                  */
/*   FT:  TOTAL FORCING FUNCTION AS GIVEN BY THE USER.                  */
/*   LP:  THE POLYNOMIAL L.                                             */
/*   LPR:  THE POLYNOMIAL LP/NP**R .                                    */
```

```
/*    N:  N.                                                           */
/*    NP:  X-ALPHA  OR  (X-ALPHA)**2 + BETA**2.                        */
/*    NTERMFT:  FORMAC EQUIVALENT OF DO  INDEX FROM 1 TO NARGSFT.      */
/*    NTERMS:  SAME AS NARGSFT.                                        */
/*    NULL:  USED TO INITIALIZE BETA FOR THE VACUOUS CASE.             */
/*    N1:  N+1.                                                        */
/*    Q1:  TEMPORARY VARIABLE USED IN COMPUTING B(0).                  */
/*    Q2:  TEMPORARY VARIABLE USED IN COMPUTING A(0) &                 */
/*         B(0) WHEN BETA IS PRESENT.                                  */
/*    R:  R.                                                           */
/*    TERMS:  CHAIN OF SOLUTIONS FOR TERMS OF FT.                      */
/*    X:  INDEPENDENT VARIABLE OF L AND F.  X IS USED AS               */
/*      THE INDEPENDENT VARIABLE FOR ALL PROCEDURES IN LINODE.         */
/*    YP:  SOLUTION FOR ONE TERM OF FT.                                */
/*    YT:  TEMPORARY VARIABLE USED TO COMPUTE AP(X+ALFA).              */
/*    YTOTAL: SUM OF PARTICULAR SOLUTIONS FOR ALL TERMS OF FT.         */
/*                                                                     */
/* FIXED BIN VARIABLES USED AS DO LOOP INDICES IN LINODE:              */
/*    IS:  USED IN COMPUTING AP AND BP.                                */
/*    NARGSFT:  NUMBER OF ARGUMENTS IN FT WHEN FT IS A SUM             */
/*      OF TERMS.                                                      */
/*    NTERMFT:  RANGES FROM 1 TO NUMBER OF TERMS IN FT.                */
/*                                                                     */
/* CHARACTER VARIABLES USED TO PASS FORMAL PARAMETERS TO               */
/*    AND FROM LINODE:  L, F, YP, TERMS, NTERMS.                       */

FORMAC_OPTIONS; OPTSET(EXPND);
F_FUNCTION (CU; RE; IM; POLYDOP);
DCL (L,F,YP,TERMS,NTERMS) CHAR(8);
DCL (NARGSFT, NTERMFT, IS) FIXED BIN;

ATOMIZE (TERMS);
LET( LP="L"; FT="F");
LET( YTOTAL=0); /* WILL CONTAIN SUM OF SOLUTIONS */
IF LOP(FT)=24 THEN NARGSFT=NARGS(FT); ELSE NARGSFT=1;
/* NARGSFT MAY BE 1 EVEN IF NARGS(FT)>1  */
DO NTERMFT=1 TO NARGSFT; /* LOOP FOR NR OF TERMS IN FT */
   LET( NTERMFT="NTERMFT");
   IF NARGSFT=1 THEN LET(F=FT); ELSE LET(F=ARG(NTERMFT,FT));
   LET( N1=HIGHPOW(F,X) + 1 );
   /* NOW EXAMINE F AND COMPUTE ALFA AND BETA */
   LET( ALFA=F/REPLACE(F, #E**($J*X), 1));
   IF IDENT(ALFA;1)
      THEN LET (ALFA=0);
      ELSE LET( ALFA=REPLACE(ALFA, #E**($J*X), $J));
   LET( BETA=F/REPLACE(F, COS($J*X), 1, SIN($J*X), 1));
   IF IDENT(BETA;1)
      THEN DO;
              LET( BETA=NULL);
              LET( NP=X-ALFA);
              /* NOW DIVIDE LP(X) BY POWERS OF NP; LPR IS QUOTIENT
                 AND R IS MULTIPLICITY OF NP IN LP */
              CALL DIVMFAC ('LP', 'NP', 'LPR', 'R');
              /* NOW COMPUTE A(0)=AP & B(0)=BP USING SINGLE
                 STEP EUCLIDEAN ALGORITHM */
              CALL POLYDIV ('LPR', 'NP', 'Q1', 'B2');
              LET( AP=1/B2; BP=-Q1/B2);
           END;
      ELSE DO;
              LET( BETA=REPLACE(BETA, COS($J*X), $J, SIN($J*X), $J));
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
            LET( NP=X**2 - 2*ALFA*X + ALFA**2 + BETA**2);
            /* NOW DIVIDE LP BY POWERS OF NP */
            CALL DIVMFAC ('LP', 'NP', 'LPR', 'R');
            /* NOW COMPUTE A(0) & B(0) */
            /* USING TWO STEP EUCLIDEAN ALGORITHM */
            CALL POLYDIV ('LPR', 'NP', 'Q1', 'B2');
            CALL POLYDIV ('NP', 'B2', 'Q2', 'B3');
            LET( AP=-Q2/B3; BP=(1+Q1*Q2)/B3);
         END;
   /*  COMPUTE A(S) */
   DO IS=0 BY 1 WHILE (2**IS < INTEGER(N1));
      LET( AP=AP * (1 + BP*NP**(2**"IS")));
      LET( BP=BP*BP);
   END;
   /* NOW FORM A(D+ALFA)(E**(-ALFA*X)*F(X)) */
   LET( YP=#E**(-ALFA*X) * F);
   LET( YT=FVAL(AP, X, X+ALFA));
   LET( YP=POLYDOP(YT,YP));
   /* NOW APPLY APPROPRIATE ANTI-DIFF OPERATOR */
   IF IDENT(BETA;NULL)
      THEN LET(YP=REPLACE(X*X*YP, X**$J,
                         FAC($J-2)*X**($J-2+R)/FAC($J-2+R)));
      ELSE IF ¬IDENT(R;0) THEN
           LET(YP=REPLACE(X*X*YP, X**$J*COS(BETA*X), RE(CU($J-2)),
                                  X**$J*SIN(BETA*X), IM(CU($J-2))));
   LET( YP=#E**(ALFA*X) * YP);
   /* NOW CHECK RESULT */
   LET( CP=POLYDOP(LP,YP));
   IF ¬IDENT(CP;F) THEN
      DO;
         PUT SKIP(2) EDIT
         ('L(D)YP ¬= TERM OF F, CP=L(D)YP PRINTED AS DEBUG AID')(A);
         PRINT_OUT(CP); RETURN;
      END;
   LET( QUEUE (TERMS)=YP);
   LET( YTOTAL=YTOTAL+YP);
END; /* OF DO NTERMFT=1 TO NARGSFT; */
LET ("YP"=YTOTAL; "TERMS"=CHAIN(TERMS));
LET (NTERMS="NARGSFT"; "NTERMS"=NTERMS);

POLYDIV: PROC(A,M,Q,REM) REORDER; /* REM=A (MOD M) */
/* THIS ROUTINE DIVIDES A POLYNOMIAL A BY A POLYNOMIAL M */
/* AND RETURNS THE QUOTIENT Q AND REMAINDER REM */
DCL (A,M,Q,REM) CHAR(8);
/* THE FOLLOWING ARE TEMPORARY VARIABLES USED IN POLYDIV: */
LOCALIZE (AT;CA;CM;NA;NM;Q;QT;S);
LET( QT=0; AT="A"; NA=HIGHPOW(AT,X); NM=HIGHPOW("M",X));
IF IDENT(NM;0) THEN DO; LET("Q"=AT/"M";"REM"=0); RETURN; END;
                ELSE LET(CM=COEFF("M",X**NM));
DO WHILE (INTEGER(NA) >= INTEGER(NM));
   LET( CA=COEFF(AT, X**NA));
   LET( Q=CA/CM * X**(NA-NM));
   LET( QT=QT+Q; S=Q*"M"; AT=AT-S);
   LET( NA=HIGHPOW(AT,X));
END;
LET( "Q"=QT; "REM"=AT);
END POLYDIV;

RE: F_PROC (Z) REORDER;
/* THIS ROUTINE COMPUTES THE REAL PART OF A COMPLEX NUMBER Z */
```

```
F_RETURN (EVAL(Z,#I,0));
F_END RE;

IM:  F_PROC (Z) REORDER;
/* THIS ROUTINE COMPUTES THE IMAGINARY PART OF A COMPLEX NUMBER Z */
F_RETURN (#I*(EVAL(Z,#I,0)-Z));
F_END IM;

POLYDOP: F_PROC (U,V) REORDER;
/* THIS ROUTINE RETURNS THE RESULT OF APPLYING THE POLYNOMIAL        */
/* FIRST ARGUMENT AS A DIFFERENTIAL OPERATOR TO THE SECOND           */
/* ARGUMENT.                                                         */
F_RETURN (REPLACE(X*X*U, X**$J, DERIV(V,X,$J-2)));
F_END POLYDOP;

CU: F_PROC (J) LOCAL (C;U) REORDER;
/* THIS ROUTINE COMPUTES (COS(BETA*X)+#I*SIN(BETA*X))*C*U */
/* BETA & R ARE GLOBAL VARIABLES , J IS A FORMAC INTEGER. */
DCL K FIXED BIN ;
LET( U=0);
DO K=0 TO INTEGER(J); LET( K="K");
   LET( U=U + FAC(R+J-K-1) / FAC(J-K) / FAC(R+K)
      * (-2*#I*BETA)**K * X**(R+K));
END;
LET( C=FAC(J) / FAC(R-1) * (-1)**R
   * #I**(R+J) / (2*BETA)**(R+J));
F_RETURN ((COS(BETA*X)+#I*SIN(BETA*X))*C*U);
F_END CU;

DIVMFAC: PROC(LP,NP,LPR,R) REORDER;
/* THIS ROUTINE DIVIDES A POLYNOMIAL LP REPEATEDLY BY A FACTOR */
/* NP AND RETURNS THE QUOTIENT POLY AS LPR AND THE MULTIPLICITY */
/* OF NP IN LP AS R */
DCL (LP,NP,LPR,R) CHAR(8);
/* DEFINE LOCAL TEMPORARY VARIABLES: */
DCL IR FIXED BIN;
LOCALIZE (DRX;LLP;RP;QP);
LET( RP=0; QP="LP");
DO IR=-1 BY 1 WHILE (IDENT(RP;0));
   LET( LLP=QP);
   CALL POLYDIV ('LLP', 'NP', 'QP', 'RP');
END;
LET( "LPR"=LLP; DRX="IR"; "R"=DRX);
END DIVMFAC;

END LINODE;
```

TEST PROBLEMS

The program was able to solve problems from a variety of sources. A sample of typical results follows below.

$$LX = X^{5} - 2 X^{4} + 18 X^{3} - 36 X^{2} + 81 X - 152$$

$$FX = X^{3} \#E^{2 X} + X^{2} \cos(3 X) \#E^{X} + \sin(3 X)$$

$$R = 1/676 X^{4} \#E^{2 X} - 8/2197 X^{3} \#E^{2 X} - 117/13690 X^{2} \sin(3 X) \#E^{X} + 1/468 X^{2} \sin(3 X) - 1/13690 X^{2} \cos(3 X) \#E^{X} + 1/312 X^{2} \cos(3 X) + 66/28561 X^{2} \#E^{2 X} + 42411/1266325 X \sin(3 X) \#E^{X} - 13152/1266325 X \cos(3 X) \#E^{X} + 336/371293 X \#E^{2 X} - 21294621/468540250 \sin(3 X) \#E^{X} + 9418897/468540250 \cos(3 X) \#E^{X}$$

$$S = 3$$

$$T = CHAIN(1/676 X^{4} \#E^{2 X} - 8/2197 X^{3} \#E^{2 X} + 66/28561 X^{2} \#E^{2 X} + 336/371293 X \#E^{2 X}, - 117/13690 X^{2} \sin(3 X) \#E^{X} - 1/13690 X^{2} \cos(3 X) \#E^{X} + 42411/1266325 X \sin(3 X) \#E^{X} - 13152/1266325 X \cos(3 X) \#E^{X} - 21294621/468540250 \sin(3 X) \#E^{X} + 9418897/468540250 \cos(3 X) \#E^{X}, 1/468 X^{2} \sin(3 X) + 1/312 X^{2} \cos(3 X))$$

$$LX = - 3/2 X^{3} + X^{2} - 1$$

$$FX = - 3/2 X + 2/5 \sin(4/7 X) \#E^{- 4/7 X}$$

$$R = 3/2 X - 73402/459281 \sin(4/7 X) \#E^{- 4/7 X} + 285376/2296405 \cos(4/7 X) \#E^{- 4/7 X}$$

$$S = 2$$

$$T = CHAIN(3/2 X, - 73402/459281 \sin(4/7 X) \#E^{- 4/7 X} + 285376/2296405 \cos(4/7 X) \#E^{- 4/7 X})$$

```
          2
LX = 5 X    + 3 X - 1
-------------------
     2                 2 X
FX = X    SIN ( 3 X ) #E
--------------------------
                2                 2 X                2                2 X
R =  - 20/5161 X    SIN ( 3 X ) #E     - 69/5161 X    COS ( 3 X ) #E
----------------------------------------------------------------------
                                   2 X
 + 366206/26635921 X SIN ( 3 X ) #E     + 134700/26635921 X COS ( 3 X )
----------------------------------------------------------------------
  2 X                                           2 X
#E     - 659109350/137467988281 SIN ( 3 X ) #E     + 441581922/137
----------------------------------------------------------------------
                         2 X
467988281 COS ( 3 X ) #E
---------------------------
S = 1
-----
                2                 2 X                2                2 X
T =  - 20/5161 X    SIN ( 3 X ) #E     - 69/5161 X    COS ( 3 X ) #E
----------------------------------------------------------------------
                                   2 X
 + 366206/26635921 X SIN ( 3 X ) #E     + 134700/26635921 X COS ( 3 X )
----------------------------------------------------------------------
  2 X                                           2 X
#E     - 659109350/137467988281 SIN ( 3 X ) #E     + 441581922/137
----------------------------------------------------------------------
                         2 X
467988281 COS ( 3 X ) #E
---------------------------

END OF O.D.E. SOLVER
```

ANALYSIS OF ALGORITHM

Following the discussion in [1], we assume that there are functions $\underline{M}(n)$ and $\underline{D}(n)$, where $\underline{M}(n)$ = time to multiply two nth degree polynomials and $\underline{D}(n)$ = time to divide a polynomial of degree at most 2n by a polynomial of degree n, time being measured in terms of the number of bit operations required on a computer. In addition, we assume that the running time of the program is essentially determined by the time to perform operations of the above types, so that we shall ignore the time-requirements of scalar multiplication and symbolic differentiation in our analysis. From this assumption it follows that the running time is determined by step 2 of the algorithm. Consequently, we restrict our attention to that step. We further restrict our attention to case 2, where $f(x) = x^n \exp(\alpha+i\beta)x$, $\beta \neq 0$.

To facilitate the analysis, we set $N(t) = t^2-2\alpha t+\alpha^2+\beta^2$, $\ell = \deg L(t)$, and assume for convenience that $\ell-2r \geq 3$, where r is the multiplicity of $\alpha \pm i\beta$ as a root of $L(t)$. We assume further that the method in Proposition 2 is used to implement step 2. In addition, we set $q_0 = \deg B_0$, where $A_0(t)$, $B_0(t)$ are polynomials as described in Proposition 2, with $\bar{L} = L/N^r$.

Theorem The time to execute step 2 of the algorithm in case 2 is bounded above by

$$(r+2)\ \underline{D}[\lceil \tfrac{1}{2}\ell \rceil] + \underline{D}[1] + \underline{M}[\ell-2r-2] + 2\log_2(2n+1)\ \underline{M}[(2n+1)(q_0+2)-(\ell-2r)],$$

where $\lceil \gamma \rceil$ denotes the least integer $\geq \gamma$.

Proof Step 2 may be divided into three parts:

a. determine $\bar{L}(t) = L(t)/N(t)^r$, where $N(t)^r \mid L(t)$, $N(t)^{r+1} \nmid L(t)$;

b. calculate $A_0(t)$, $B_0(t)$ such that $A_0\bar{L} + B_0 N = I$;

c. calculate $A_s(t)$, $B_s(t)$ recursively by $A_{s+1} = A_s[I+B_s N^{(2^s)}]$, $B_{s+1} = B_s^2$, until $2^s \geq n+1$.

We proceed to determine the running time for each step.

a. This part involves computing successively $L_1 \equiv L/N$, $L_2 \equiv L_1/N = L/N^2$, $\ldots$, $L_r \equiv L_{r-1}/N = L/N^r$, and terminating with $L_r \div N$, which has a non-zero remainder. The time to compute L_1 is bounded above by $D[\lceil \tfrac{1}{2}\ell \rceil]$. Since $\deg L_1 > \deg L_2 > \deg L_3 > \ldots$, the times to execute the remaining steps are all bounded above by $\underline{D}[\lceil \tfrac{1}{2}\ell \rceil]$ as well so that the running time for part a is bounded by $(r+1)\, \underline{D}[\lceil \tfrac{1}{2}\ell \rceil]$.

b. Since $\ell-2r \geq 3$ by assumption, we have that $\deg \bar{L} \geq 3$, hence $\bar{L} = S_0 N + T_0$, for polynomials S_0, T_0, with $\deg T_0 \leq 1$. If $\deg T_0 = 1$, then $N = S_1 T_0 + T_1$ for polynomials S_1, T_1, with $\deg T_1 = 0$, T_1 necessarily non-zero. It follows that $A_0 \bar{L} + B_0 N = I$, for $A_0 = -S_1/T_1$, $B_0 = (S_1 S_0 + I)/T_1$. Now $\deg S_0 = \ell-2r-2 \geq 1$ and $\deg S_1 = 1$. It then follows that the work to calculate A_0 and B_0 in this situation is bounded above by $\underline{D}[\lceil \tfrac{1}{2}\ell-r \rceil] + \underline{D}[1] + \underline{M}[\ell-2r-2]$.

If $\deg T_0 = 0$, we may similarly obtain upper bounds on the work to calculate and on the degrees of A_0 and B_0. In all cases the bounds do not exceed the corresponding ones obtained above.

c. Set $p_s = \deg A_s$, $q_s = \deg B_s$. It follows readily from the recursive definitions of A_s and B_s that $q_s = 2^s q_0$, $p_s = q_s + 2^{s+1} - (\ell-2r) = 2^s(q_0+2) - (\ell-2r)$. Hence $p_0 \leq p_1 \leq p_2 \leq \ldots \leq p_s$ and $q_0 \leq q_1 \leq q_2 \leq \ldots \leq q_s$, so that the time to perform each step of the recursion is bounded above by the time for the last step.

Suppose now that s is the smallest integer such that $2^s \geq n+1$. Then the time to compute A_s is determined by the time to compute $A_{s-1} B_{s-1} N^{(2^{s-1})}$. To get upper bounds on the degrees of these polynomials, note first that $n+1 \leq 2^s \leq 2n+1$, by minimality of s. It then follows that $p_{s-1} \leq \tfrac{1}{2}(2n+1)(q_0+2) - (\ell-2r)$, $q_{s-1} \leq \tfrac{1}{2}(2n+1)q_0$, $\deg N^{(2^{s-1})} \leq 2n+1$. Adding the bounds, it follows that $p_{s-1} + q_{s-1} + \deg N^{(2^{s-1})} \leq (2n+1)(q_0+2) - (\ell-2r)$. Thus the time to compute $A_{s-1} B_{s-1} N^{(2^{s-1})}$ is bounded above by $2M[(2n+1)(q_0+2) - (\ell-2r)]$, and the total

time to compute A_s is bounded above by $2s\,\underline{M}[(2n+1)(q_0+2) - (\ell-2r)]$
$\leq 2\log_2(2n+1)\underline{M}[(2n+1)(q_0+2) - (\ell-2r)]$, since $2^s \leq 2n+1$.

Having obtained bounds for the execution times of parts a, b, and c of step 2, we obtain the overall bound announced in the theorem by adding these together. Q.E.D.

COMPARISON WITH OTHER METHODS

Linear differential equations of the type considered here are commonly solved by the method of undetermined coefficients or the method of Laplace transforms. The former method requires in general $r+n$ divisions of the polynomial $L(t)$ over the complex numbers, while our method requires $r+2$ divisions over the reals. The latter method requires the factorization of the polynomial $L(t)$ over the rationals, and in general will not give explicit answers unless all factors have degree ≤ 4. See [5] and [4] for details on the two methods.

Both methods have been implemented as programs in the MACSYMA system, the former by Ivie [7] and the latter by Bogen [3]. It would clearly be of interest to compare the methods with ours in a practical sense, by implementing the three in a common system and trying them on a common set of problems.

REFERENCES

1. Aho, A., Hopcroft, J., and Ullman, J. The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading, Mass., 1974.
2. Bahr, K. SHARE FORMAC/FORMAC73 Version A.77. SHARE Program Library Agency.
3. Bogen, R. Automatic computation of direct and inverse Laplace transforms using computer symbolic mathematics. Proc. 10th Hawaii International Conf. on System Sciences. Univ. of Hawaii, 161-169.
4. Boyce, W., and DiPrima, R. Elementary Differential Equations and Boundary Value Problems. John Wiley, New York, 1977.
5. Coddington, E. An Introduction to Ordinary Differential Equations. Prentice Hall, Englewood Cliffs, N.J., 1961.
6. Hanson, J., and Russo, P. Some data conversions for managing the internal and output form of FORMAC constants. SIGSAM 10(May 1976), 21-26.
7. Ivie, J. Some MACSYMA programs for solving recurrence relations. TOMS 4 (March 1978), 24-33.