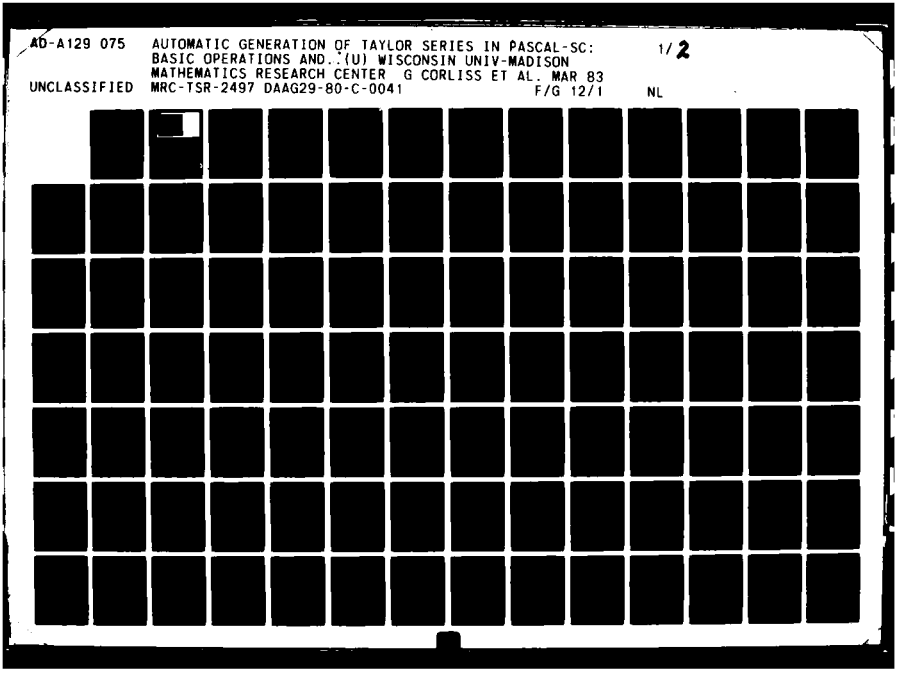
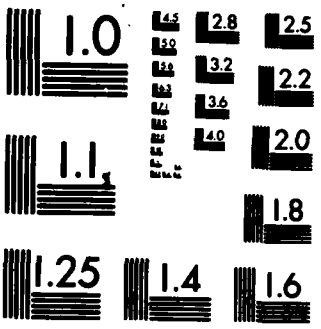
AD-A129 075 AUTOMATIC GENERATION OF TAYLOR SERIES IN PASCAL-SC: BASIC OPERATIONS AND..(U) WISCONSIN UNIV-MADISON MATHEMATICS RESEARCH CENTER G CORLISS ET AL. MAR 83 1/2

UNCLASSIFIED MRC-TSR-2497 DAAG29-80-C-0041 F/G 12/1 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

MRC Technical Summary Report #2497

AUTOMATIC GENERATION OF TAYLOR SERIES IN PASCAL-SC: BASIC OPERATIONS AND APPLICATIONS TO ORDINARY DIFFERENTIAL EQUATIONS

George Corliss and L. B. Rall

Mathematics Research Center
University of Wisconsin—Madison
610 Walnut Street
Madison, Wisconsin 53706

March 1983

(Received March 9, 1983)

Sponsored by

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park
North Carolina 27709

UNIVERSITY OF WISCONSIN-MADISON
MATHEMATICS RESEARCH CENTER

AUTOMATIC GENERATION OF TAYLOR SERIES IN PASCAL-SC: BASIC OPERATIONS AND APPLICATIONS TO ORDINARY DIFFERENTIAL EQUATIONS

George Corliss* and L. B. Rall**

Technical Summary Report #2497

March 1983

ABSTRACT

Taylor series have a long history of usefulness in numerical analysis, especially for the numerical solution of the initial value problem for systems of ordinary differential equations. Since recurrence relations for coefficients of Taylor series are well known, it is possible to automate the computation of arithmetic operations and various standard functions with arguments which are themselves series. If the language used for scientific computation supports user defined operators and data types, then the facilities built into the language compiler itself can be used to generate machine code for the evaluation of Taylor coefficients. Examples of such languages are Pascal-SC, Algol 68, and ADA (a trademark of the U. S. Department of Defense). Pascal-SC (Pascal for Scientific Computation) offers the user highly accurate floating-point and interval arithmetic, the latter being useful for automatic computation of guaranteed error bounds. In this language, series with real coefficients are introduced as type TAYLOR, and the corresponding series with interval coefficients as type ITAYLOR. Source code is given for the operators +, -, *, /, ** and the functions SQR, SQRT, EXP, SIN, COS, ARCTAN, and LN with arguments of these types and some other useful functions and procedures. Integer, real, and interval constants are also allowed in TAYLOR or ITAYLOR expressions. Suggestions for the implementation of additional operators or functions are given. An application of Taylor series and the methods of interval analysis to the solution of the initial value problem for ordinary differential equations is made using types TAYLOR and ITAYLOR. An analysis of the stability of this method is made, which shows that the recurrence relations for generation of the Taylor series for the solution exhibit a mild instability which has no significant effect on the values of the solution computed by analytic continuation.

AMS (MOS) Subject Classifications: 65-04, 65G10, 65L05, 65L07, 65V05

Key words and phrases: Taylor series, recurrence relations for Taylor coefficients, automatic differentiation, numerical solution of ordinary differential equations, stability, error analysis, interval arithmetic

Work unit number 3 - Numerical Analysis

---

*Department of Mathematics, Statistics, & Computer Science, Marquette University, Milwaukee, Wisconsin.

**Research sponsored in part by the United States Army under Contract No. DAAG29-80-C-0041.

SIGNIFICANCE AND EXPLANATION

The reliable numerical solution of the initial value problem for systems of ordinary differential equations is one of the fundamental problems of scientific computation. Taylor series methods for this purpose have long been recognized to be of theoretical importance, but their use in practice has been hampered in the past by the need to differentiate the functions defining the system in order to obtain coefficients of the series. Consequently, methods are commonly used which take linear combinations of function evaluations to be "as good as" Taylor polynomials of some degree as approximate solutions. However, since recurrence relations are well known for the Taylor series coefficients of functions resulting from arithmetic operations and standard functions involving series arguments, it is possible to generate the required coefficients automatically with a computer, and the user need only supply code for the functions defining the differential equations. Many modern compilers allow user defined data types and "overloading" the standard operators and functions so that these series operations can be applied directly to vectors consisting of the coefficients of the Taylor polynomials involved. Examples of such languages are Pascal-SC, Algol 68, and ADA™ (a trademark of the Department of Defense). Pascal-SC offers many advantages for numerical computation, since it is based on accurate real and interval arithmetic for vectors as well as scalars. In this report, source code is given for generation of both real and interval valued Taylor series, and simple programs illustrate the application of each to the numerical solution of ordinary differential equations. In these programs, both the order of the method being used and the step size of the integration are under the control of the user. Advantages of interval computation include the ability to study the range of values of solutions depending on ranges of initial conditions and parameters in the equations, and to obtain rigorous bounds for solutions in applications such as aerospace in which reliability is important. In this report, interval calculations are used to investigate the stability of the process of generating the Taylor coefficients. For rapidly converging Taylor series, instability in the generation of the coefficients has little effect on the computed values of the solution because as the relative error increases for successive terms, the terms themselves become smaller. Hence, their contribution to the error of the sum is insignificant.

The responsibility for the wording and views expressed in this descriptive summary lies with MRC, and not with the authors of this report.

AUTOMATIC GENERATION OF TAYLOR SERIES IN PASCAL-SC: BASIC OPERATIONS AND APPLICATIONS TO ORDINARY DIFFERENTIAL EQUATIONS

George Corliss* and L. B. Rall**

1. Taylor series, polynomials, and forms. A fundamental tool of numerical analysis is the expansion of a real function f of a real variable x into a Taylor series at $x = x_0$, which gives the expression

$$f(x) = \sum_{i=1}^{\infty} f^{(i-1)}(x_0)(x - x_0)^{(i-1)}/(i-1)!, \tag{1.1}$$

valid for $|x - x_0| < \rho$, where ρ is the radius of convergence of the infinite series on the right-hand side of (1.1). Of course, in actual numerical computation, the Taylor polynomial

$$f_n(x) = \sum_{i=1}^{n} f^{(i-1)}(x_0)(x - x_0)^{(i-1)}/(i-1)!, \tag{1.2}$$

is used in place of the infinite series. This results in the truncation error

$$R_n(f,x,x_0) = f(x) - f_n(x) = f^{(n)}(\xi)(x - x_0)^n/n!, \quad \xi \in X, \tag{1.3}$$

where X denotes the interval $X = [\min\{x,x_0\},\max\{x,x_0\}]$, and the remainder term $R_n(f,x_0;x)$ is expressed in Lagrange form. This approximation of f(x) by $f_n(x)$ gives rise to a problem

---

*Department of Mathematics, Statistics, & Computer Science, Marquette University, Milwaukee, Wisconsin.

**Research sponsored in part by the United States Army under Contract No. DAAG29-80-C-0041.

of error estimation which can be solved by the methods of interval analysis. If $F^{(n)}$ is an interval inclusion of the real function $f^{(n)}$, then

$$f(x) - f_n(x) = R_n(f,x_0;x) \in F^{(n)}(X)(x - x_0)^n/n!; \tag{1.4}$$

this allows automatic computation of guaranteed error bounds by the use of interval arithmetic [12], [13].

In order for Taylor series methods to be useful in scientific computation, it must be possible to automate the calculation of the normalized real Taylor coefficients

$$c(i + 1) = f^{(i)}(x_0)(x - x_0)^i/i!, \quad i = 1,\ldots,n-1, \tag{1.5}$$

and the corresponding interval quantities

$$C(i + 1) = F^{(i)}(X)(x - x_0)^i/i!, \quad i = 1,\ldots,n-1. \tag{1.6}$$

These calculations can be carried out by means of well-known recurrence relations [1], [12], [13], [16] for functions defined by subroutines or expressions involving arithmetic operations and a variety of standard functions for which library subroutines are available. A very important application of automated generation of Taylor series by recursion is the numerical solution of the initial-value problem for ordinary differential equations. That is, it is required to find $y = y(x) = (y_1(x),y_2(x),\ldots,y_m(x))$ such that

$$y_i' = f_i(x,y), \quad y_i(x_0) = y_{i0}, \quad i = 1,\ldots,m, \tag{1.7}$$

for values of x in an interval containing x_0 [1], [3], [6], [12], [13].

Another application of the methods in this paper is to the automatic generation of interval inclusions of real functions by means of their interval mean-value and Taylor forms [12], [13], [19]. Suppose, for example, that $f(x)$ is a real function, such as

(1.8) $$f(x) = (x + 3)/(x^2 + 2),$$

which can be evaluated by the corresponding expression

(1.9)
```
f := (x + 3)/(x**2 + 2);
```

in a Pascal-SC program. An interval inclusion F of f on an interval X, for which

(1.10) $$f(X) = \{f(x) \mid x \in X\} \subset F(X)$$

can be obtained simply by declaring the variables F and X to be of type INTERVAL, and then evaluating the expression corresponding to (1.9),

(1.11)
```
F := (X + 3)/(X**2 + 2)
```

using interval arithmetic, a standard feature of Pascal-SC [23]. An inclusion obtained in this way may be too coarse in the sense that F(X) is a much larger interval than needed to contain f(X). In this case, an interval inclusion provided by the mean-value form

(1.12) $$F_1(X) = f(x) + F'(X)(X - x), \quad x \in X,$$

can be better, particularly if the width of X is not large [12], [13], [19]. In (1.12), F' denotes an interval inclusion of the derivative f' of f; F'(X) is obtained automatically by evaluating (1.11) with F and X of type ITAYLOR, as will be explained below. Interval inclusions of f are also provided by Taylor forms of higher order [19], in general,

(1.13) $$F_n(X) = \sum_{i=0}^{n-1} f^{(i)}(x)(X - x)^i/i! + F^{(n)}(X)(X - x)^n/n!, \quad x \in X.$$

These forms can be generated automatically from the expressions (1.9) and (1.11) by the use of types TAYLOR and ITAYLOR, respectively. Recursive generation of real and interval Taylor coefficients makes possible an adaptive method for calculation of interval inclusions of real functions, in which n is increased until $F_n(X)$ includes $F_{n-1}(X)$. It is also possible to reduce the width of computed inclusions by making use of the fact that the intersection of interval inclusions is likewise an interval inclusion.

Previous implementations of automatic generation of Taylor coefficients in computer languages such as FORTRAN have used interpretation [20] or pre-compilation [10] to activate the necessary subroutines [16]. In more modern languages, the compiler itself can be used to produce the necessary routines, leading to a saving of programming effort and an increase in clarity of the source code. The use of Pascal-SC, a language of this type, will be explained in the next section.

2. Pascal-SC. The method for automatic generation of Taylor series given in this report is based on computation with the coefficients of Taylor polynomials of arbitrary length, considered as specific mathematical entities. This requires that the language support i) user defined data types, as do descendents of ALGOL-60 such as Pascal and ADA (ADA is a trademark of the U.S. Department of Defense); and ii) user defined operators, as do ALGOL-68 and ADA.

Pascal-SC [2] is an extension of Pascal which provides both user-defined data types and user-defined operators. This paper assumes a modest familiarity with standard Pascal [8]. For the remainder of this Section, we outline some of the extensions which make Pascal-SC well suited to the applications in this paper. The reader who wishes to omit the discussion of programming language issues may proceed directly to the definition of the data types TAYLOR and ITAYLOR in Section 3.

Pascal-SC was developed with the needs of scientific computation in mind. It is an implementation of Jensen and Wirth Pascal [8] which also provides intervals, complex numbers, complex intervals, as built-in elementary scalar data types [23]. A full range of standard operators is provided to manipulate the elementary scalar data types, as well as

vectors and matrices built of these types [23].

Standard Pascal supports user-defined data types built from elementary data types. This feature will be used to define variables of type TAYLOR and ITAYLOR (interval Taylor) in Section 3.

Pascal-SC allows the user to define operators. Most computer languages allow programmers to define functions, subroutines, or procedures, but except for APL, the languages most often used for scientific computation require that such user-defined functions be called using a prefix notation (eg. SIN (X)), while built-in operators are called using an infix notation (eg. A + B). Programmers can define operators in Pascal-SC to extend the language in a uniform way, retaining the familiar infix notation for operators whose operands are variables of user-defined types (eg. A + B, where A and B are variables of type TAYLOR).

Operators, functions, and procedures in Pascal-SC can be overloaded. That is, the name of an operator, a function, or a procedure can have different meanings, depending on the type or number of its operands. For example, the standard Pascal or FORTRAN operator "+" is said to be overloaded because "A + B" for integer variables A and B has a different meaning from "A + B" for real variables A and B. The support of Pascal-SC for overloading of user defined operators is essential to the uniform extension of the language because we wish to define the meaning of "A + B" for variables which represent Taylor series with real or with interval coefficients.

The support of Pascal-SC for user-defined operators and for overloading is very similar to that provided by ADA. ADA's PACKAGE concept would allow a more secure implementation of data abstractions [9] for real and interval valued Taylor series. The operations on intervals, however, also require support for directed rounding of floating-point results in order to guarantee that the desired answer is contained in the interval computed. The early implementations of ADA do not provide an accuracy of floating-point computations which can compete with Pascal-SC.

Pascal-SC features a highly accurate arithmetic based on a general theory [11] for real and complex numbers, real and complex intervals, and vectors and matrices over these

types. Operations on floating-point numbers are rounded to the closest floating-point number to the true result, or upward or downward to the closest neighboring floating-point number under the control of the user. This accuracy meets the proposed IEEE standard for floating-point arithmetic [14]. In addition, scalar products of vectors

$$\text{SCALP(A,B,ROUND)} = \sum_{i=1}^{N} A_i * B_i \tag{2.1}$$

are calculated with the same accuracy (to the closest floating-point number), and with the same options for rounding [23]. A sufficiently long accumulator is used to store intermediate results in the evaluation of the scalar (or inner) product. This capability can also be used to obtain results of the same high accuracy in evaluation of a given arithmetic expression so that 1.0E+99 + 1.0E-99 - 1.0E+99 yields 1.0E-99.

3. Types TAYLOR and ITAYLOR. We wish to provide the developer of scientific software with a set of tools with which Taylor series methods can be implemented easily for a variety of numerical problems. The ability of the computer to perform formula translation is used. Compilers since the first FORTRAN compiler have produced machine code for the evaluation of an expression such as

```
F := (X*Y + SIN(X) + 4.0) * (3.0 * (Y**2) + 6.0).          (3.1)
```

This is done by analysis of the expression and application of the rules for evaluation of formulas. If the rules for differentiation or recursive generation of Taylor coefficients are applied in the same way, then code for the evaluation of the corresponding quantities results [16]. In this way, fast and inexpensive operations performed by the compiler avoid the overhead involved in invoking symbolic differentiation software. This leads to a more efficient implementation of Taylor series generation all the way from initial coding through program execution.

The normalised Taylor coefficients of a function f(x) expanded at $x = x_0$ are

defined by

$$f.TC[K+1] = f^{(K)}(x_0)t^K/K!, \quad t = x - x_0, \quad K = 0,1,2,\ldots . \tag{3.2}$$

Then

$$f_{DIM}(x) = \sum_{K=1}^{DIM} f.TC[K], \tag{3.3}$$

where DIM is the length of the truncated series which is actually stored. This real or interval vector of normalized Taylor coefficients is the basis for the data types TAYLOR and ITAYLOR. For the remainder of this paper, the term "series" is used to refer to the Taylor polynomial given by equation (3.3) or its interval analog.

In what follows, the general rule will be adopted that all variables or expressions of the scalar types INTEGER, REAL, or INTERVAL are treated as constants for the purposes of differentiation.

To form the real data type TAYLOR, the DIM normalized Taylor coefficients in (3.3) are stored as a vector of floating-point numbers. The appropriate declarations in Pascal-SC are:

```
CONST DIM = n;                          { User supplies n }
TYPE DIMTYPE  = 1..DIM;
     RVECTOR  = ARRAY[DIMTYPE] OF REAL;
     TAYLOR   = RECORD  LENGTH : DIMTYPE;
                        T      : REAL;
                        TC     : RVECTOR  END;
```
(3.4)

These declarations are the same as those given in [18], except for the field named LENGTH. Let F be a variable of type TAYLOR (declared by: VAR F: TAYLOR), then F.LENGTH denotes the actual length of the truncated series ($1 \leq$ F.LENGTH $\leq$ DIM). It may happen that

F.LENGTH < DIM if F is being built up recursively, if F has been defined by term-by-term differentiation of another series, or if F has been defined as a quotient of two series both of whose leading terms are zero (see Section 4.2). This field has been added to the record for type TAYLOR given in [18] for internal documentation and so that i) only series terms actually used need to be processed; and ii) l'Hospital's rule can be applied to certain indeterminant forms 0/0 which may appear.

The normalized Taylor coefficients themselves are stored in the array of real numbers named TC, that is,

$$\text{(3.5)} \qquad \text{F.TC[K]} = F^{(K-1)}(X_0)(X - X_0)^{(K-1)}/K!, \quad K = 1,\ldots,\text{DIM}.$$

The size of the step being used for expansion is F.T = X - X_0. Series are generated using a fixed stepsize for which the series might even be divergent. The series for F at a different point Z is readily computed at a cost proportional to DIM:

$$\text{(3.6)} \qquad \text{F.TC[K]} := \text{F.TC[K]}*((Z - X_0)/\text{F.T}))^{**}(K-1); \quad K = 2,\ldots,\text{DIM},$$

while the cost of series generation is usually proportional to DIM^2. The presence of the stepsize in the record also makes it possible to check that an operation is not being performed on two series with different stepsizes.

One of the important problems to which interval analysis has been applied since its beginnings is the problem of controlling the truncation error of Taylor series methods [12]. Hence it is natural to support Taylor series whose normalized coefficients are intervals. The appropriate declarations in Pascal-SC are

```
CONST DIM = n;                          { User supplies n }
TYPE DIMTYPE  = 1..DIM;
     INTERVAL = RECORD INF, SUP : REAL END;
     IVECTOR  = ARRAY[DIMTYPE] OF INTERVAL;
     ITAYLOR  = RECORD  LENGTH : DIMTYPE;
                        T      : REAL;
                        TC     : IVECTOR  END;
```

(3.7)

The types ITAYLOR and TAYLOR are the same, except that the normalized coefficients of the former are intervals. The same recurrence relations are used to generate series of each type.

The stepsize T remains real. This corresponds to bounding the range of values of a function f at one real number x. There are some applications for which it is necessary to bound the range of f on an interval, as in (1.13). In this case, one can take T = 1 and form the normalized coefficients by computing the needed powers of (X - x) by interval arithmetic, or else introduce a new data type in which T is of type INTERVAL, and a set of operators corresponding to those given here.

The declarations (3.6) and (3.7) of types TAYLOR and ITAYLOR, respectively, are basic to the discussion of operators in the next section.

4. Implementation of operators and functions for types TAYLOR and ITAYLOR. As indicated above, the ability of a compiler to perform formula translation can also be used to produce machine code for the evaluation of the normalized Taylor coefficients [1], [3], [10], [12], [13], [15], [16], [18]. If the value of function f is obtained by a composition

(4.1) $$f = f_1 \circ f_2 \circ \ldots \circ f_m$$

of a finite number of elementary functions, then derivatives of f can be computed by the

chain rule from the derivatives of $f_1,\ldots f_n$. This is a tedious and error-prone calculation to do by hand, but the computer does it not only rapidly, but also accurately.

Recurrence relations for calculating the normalized Taylor coefficients for the basic arithmetic operations and for the elementary functions are well known (see [16], for example). Hence machine code can be generated to expand the Taylor series for f at any point $x = x_0$ at which f is analytic. These recurrence relations are both more efficient and more accurate than numerical differentiation [17]. Recursive generation of the series may be *mildly unstable* [6], but the interval-valued Taylor series introduced in Section 3 can give guaranteed bounds for the effect of any such instability. In Section 6, we show that any instability in the series generation has no significant effect on the series sum.

Rall [18] outlines an approach to abstract data types for real and interval-valued Taylor series. Our implementation generally follows that outline. This report discusses extensions and some of the implementation details. The code is included as Appendix C of this report. First of all, in order for expressions to be evaluated correctly when they include variables of type TAYLOR or ITAYLOR, the arithmetic operations and the standard functions must be defined in a manner which incorporates the appropriate recurrence relations for the generation of the normalized Taylor coefficients. Our implementation in Pascal-SC attempts to follow the principles of uniformity, compactness, locality, and linearity for a good programming language design [22]. Next we attempt to justify significant departures from two of these principles.

The principle of uniformity in programming language design says that the same things should be done the same way whenever they occur. Thus "A + B" means "add", regardless of the types of the variables A and B. The other arithmetic operators enjoy the same uniformity, but the standard functions do not. For example, exp(x) is EXP(X) if X is REAL, IEXP(X) if X is INTERVAL, TEXP(X) if X is TAYLOR, and ITEXP(X) if X is ITAYLOR. EXP and IEXP are built-in functions which were designed to suggest the type of their operand and result as an aid to reading the code. That is especially useful since Pascal tends to violate the principle of locality by placing the declaration of a variable far from its

use. We chose to maintain uniformity of our extensions with the built-in functions. It is important to be able to determine the type of a variable, and it would be quite non-uniform if IEXP were the only function in this family which requires a prefix.

The principle of locality suggests that all relevant parts of the program are found in the same place. We attempt to follow this principle in each of our program units, but the use of the global constant DIM and the global types RVECTOR, IVECTOR, TAYLOR, and ITAYLOR is a violation. The use of such global types needed in the headings of the operators and functions is very difficult to avoid. Their use has the advantage that all of the information about the length of the series to be used is located in only one place, CONST DIM = n, so it is easy to change.

In roughly their order of importance, the goals of this implementation are:

- Consistent set of software tools.
- Correct answer whenever possible.
- Useful error messages when no correct answer is possible.
- Readable code for future adaptations.
- Efficient execution.
- Compact code.

For example, this implies that although efficient, compact code is sought, efficiency and compactness are sometimes sacrificed for higher goals. In particular, it is important that other programmers be able to read the code, perhaps in order to improve its efficiency.

Binary operations with one operand of type TAYLOR may appear with the other operand of type INTEGER, REAL, or TAYLOR; and the two operands may appear in either order. Similarly, binary operations with one operand of type ITAYLOR may have a second operand of type INTEGER, INTERVAL, or ITAYLOR. The operators built into Pascal-SC do not support the mixing of REAL and INTERVAL operands because real numbers are viewed as being potentially inexact [23]. Our extensions of the arithmetic operators to interval valued Taylor series

maintain uniformity with this convention. This is recognised, but not explicitly stated in [18]. If a programmer is certain that a real number X is exact so that it may safely be mixed with an interval, INTPT (X) converts X into the interval [X, X].

The library of subroutines to support computations with types TAYLOR and ITAYLOR includes operators (+,-,*,/,**), special power functions (sqr, sqrt, exp), standard functions (sin, cos, ln, arctan), and additional functions (tan and the Runge function $f(x) = 1/(1 + x^2)$, to which the user can add more functions and procedures as desired. The analytic operations of term-by-term differentiation of real and interval series, as well as term-by-term differentiation of interval series are also provided by means of functions for the given purpose. There is also a set of utility functions and procedures to perform frequently needed tasks, such as reading and writing real and interval series, taking the midpoints of the coefficients of an interval series to obtain a real series, and so on.

The following abbreviations are used in the code to make it as easy as possible to locate a desired operation with any text editor:

K	INTEGER
R	REAL
I	INTERVAL
T	TAYLOR
IT	ITAYLOR.

Using these abbreviations to distinguish between instances of overloading, the operators which are needed to support variables of type TAYLOR and ITAYLOR are:

Addition (Section 4.1):

+ T, K + T, T + K, R + T, T + R, T + T

+ IT, K + IT, IT + K, I + IT, IT + I, IT + IT

Subtraction (Section 4.1):

- T, K - T, T - K, R - T, T - R, T - T

- IT, K - IT, IT - K, R - IT, IT - R, IT - IT

Multiplication (Section 4.2):

K * T, T * K, R * T, T * R, T * T

K * IT, IT * K, I * IT, IT * I, IT * IT

Division (Section 4.2):

K / T, T / K, R / T, T / R, T / T

K / IT, IT / K, I / IT, IT / I, IT / IT

Power (Section 4.3):

K ** K, R ** K, K ** R, R ** R

I ** K, K ** I, I ** I

K ** T, T ** K, R ** T, T ** R, T ** T

K ** IT, IT ** K, I ** IT, IT ** I, IT ** IT

Implementation details of each operator are discussed in the Sections shown. Pascal-SC provides no power operator, so ** must be defined for the scalar types before it can be extended to types TAYLOR and ITAYLOR. The discussion of ** is postponed to follow the introduction in Section 4.3.1 of special cases of exponentiation: sqr, sqrt, and exp.

The priorities of the operators given in this Section are:

Highest: Unary addition and subtraction, functions;

Multiplication, division, and powers: *, /, **

Lowest: Binary addition and subtraction: +, -

In particular, note that the priority of ** relative to * and / is different than in FORTRAN.

For types TAYLOR and ITAYLOR, implementation has been provided for the standard functions which are supported in Pascal-SC for types INTEGER, REAL, and INTERVAL. They are:

Special powers (Section 4.3.1):

TSQR (T), TSQRT (T), TEXP (T)

ITSQR (IT), ITSQRT (IT), ITEXP (IT)

Standard functions (Section 4.4):

TSIN (T), TCOS (T), TLN (T), TARCTAN (T)

ITSIN (IT), ITCOS (IT), ITLN (IT), ITARCTAN (IT)

Additional functions (Section 4.5):

TRUNGE (T), TTAN(T)

ITRUNGE (IT), ITTAN(IT)

Differentiation and integration (Section 4.6):

TDIFF(T), TINTGRL(T)

ITDIFF(IT), ITINTGRL(IT)

Miscellaneous utilities (Section 4.7):

VRNULL, T_IDENT_ZERO(T), T_IDENT_CONSTANT(T), ITMIDPT(IT),

IVRNULL, IT_IDENT_ZERO(IT), IT_IDENT_CONSTANT(IT), WRITE_SERIES(T),

READ_INTERVAL SERIES(IT), WRITE_INTERVAL(I), WRITE_INTERVAL_SERIES(IT)

A brief description of the method for introduction of user-defined functions will be given in Section 4.5. Some implementation details of the operators and functions will now be discussed. The recurrence relations are taken from [16]. In following the conventions of Pascal-SC, minor differences from the indices found there are due to our starting the series indices at 1 instead of starting at 0. In each Section, operations involving the scalar types are discussed before turning to types TAYLOR and ITAYLOR. The Pascal-SC source code for each of these operators and functions, as well as for utilities which they require, is included as Appendix C to this report.

4.1. Addition and subtraction. The ten addition and ten subtraction operators are quite straightforward.

Addition:

+ T, K + T, T + K, R + T, T + R, T + T

+ IT, K + IT, IT + K, I + IT, IT + I, IT + IT

Subtraction:

- T, K - T, T - K, R - T, T - R, T - T

- IT, K - IT, IT - K, I - IT, IT - I, IT - IT

Addition and subtraction of a constant alters only the value of a variable, not the values of any of its derivatives. Interval constants only require that the appropriate built-in interval operator be used. Otherwise, addition or subtraction of series is done term-by-term.

If U := F $\pm$ G, then

(4.1.1) U.TC[K] := F.TC[K] $\pm$ G.TC[K], K = 1,...,DIM.

4.2. Multiplication and Division.

Multiplication:

K * T, T * K, R * T, T * R, T * T

K * IT, IT * K, I * IT, IT * I, IT * IT

Multiplication and division of two Taylor series is done by the well-known Leibnis rule for the Taylor coefficients of a product [16].

If U = F * G, then

$$U.TC[K] = \sum_{I=1}^{K} F.TC[I]*G.TC[K-I+1], \quad K = 1,\ldots,DIM. \tag{4.2.1}$$

The scalar product of two vectors is evaluated in Pascal-SC by the standard function SCALP to the closest floating point number. Fast series multiplication techniques were not used here because

- In many applications of *, the series for U is being generated recursively. That is, the variables F or G involve U itself.
- The accuracy of SCALP would not be available.
- The speed of SCALP, especially when some terms are zero, makes these techniques less attractive.

Multiplication or division of a series by a constant is done term-by-term. Division of a constant by a series is done by generation of the series for C/F(x).

Division:

R / T, T / R, R / T, T / R, T / T

R / IT, IT / R, I / IT, IT / I, IT / IT

If U := F / G, then U * G = F, and Leibnis' rule applies:

U.TC[1] = F.TC[1] / G.TC[1];

for K = 2,...,DIM

$$U.TC[K] = \left(\sum_{I=1}^{K-1} U.TC[I]*G.TC[K-I+1] \right) / G.TC[1], \tag{4.2.2}$$

If $G(x_0)$ = G.TC[1] = 0, then we attempt return the correct answer whenever possible. If $F(x_0)$ = F.TC[1] is also 0, then we can apply l'Hospital's rule because the series for both F and G are known. U.TC[1] = $F'(x_0)$ / $G'(x_0)$ = F.TC[2] / G.TC[2], if this quotient exists, but U ≠ F' / G' as functions.

If U := F / G, and $F(x_0) = G(x_0) = 0$, then let

(4.2.3) V.TC[K] := F.TC[K+1]; W.TC[K] := G.TC[K+1]; K = 1,...,DIM-1.

Then,

(4.2.4) U := V / W.

Thus, l'Hospital's rule is implemented as a recursive call to the division operator with operands whose series length has been reduced by one. This approach would not be possible in a language which does not support recursion. Further, cases in which the series for both f and g have several leading zeros are handled automatically by the language.

L'Hospital's rule is applied in a similar manner when a constant quotient or divisor is equal to zero.

4.3. Power Operators. The power operator ** defined by F ** G = F^G is not standard in Pascal or Pascal-SC, but can be implemented in the latter for data types for which it is meaningful by the use of the operator concept. Coding of ** is simplified by the introduction of a set of basic power functions. These are implemented separately

- for uniformity with Pascal-SC which provides these functions for standard data types,
- to provide tighter bounds for interval operands, and
- for efficiency.

4.3.1. Special Power Functions This set of functions consists of the square, square root, and natural exponential function of variables of types TAYLOR and ITAYLOR:

TSQR (T), TSQRT (T), TEXP (T)

ITSQR (IT), ITSQRT (IT), ITEXP (IT)

These functions are called by the operator ** when appropriate. For example, if X is of type ITAYLOR, then both X ** 2 and X ** INTPT(2.0) are actually performed by a call to ITSQR (X). The use of this function rather than X * X is important in interval computations, since, for example, $[-1,-1]^2 = [0,1]$ while $[-1,-1] * [-1,-1] = [-1,+1]$. Further, the squaring functions TSQR and ITSQR are twice as fast as the multiplication Y * Y for variables of the corresponding types.

The recurrence relations to generate the series terms for these functions can be derived easily using Leibniz' rule. The squares of real and interval Taylor series are computed as follows.

If U = sqr(F), then Leibniz' rule for a product can be shortened to:

For K = 1, ..., DIM,

$$U.TC[K] = \sum_{I=1}^{K \text{ DIV } 2} F.TC[I]*F.TC[K-I+1]; \tag{4.3.1}$$

if K is odd, then U.TC[K] = U.TC[K] + SQR (F.TC[(K+1)/2]).

The inner product contains only TRUNC(K/2) terms. If F is of type ITAYLOR and includes negative numbers, then ITSQR (F) provides tighter bounds than does F * F. The SQR functions are named TSQR and ITSQR to indicate the type of operand accepted and value returned.

A similar function was written for CUBE. Its summations had length TRUNC(K/3) but they were nested to yield a cost proportional to DIM^3. CUBE is not included in the library because F * SQR (F) is faster.

The functions in the next set calculate square roots of real and interval Taylor variables.

If U := SQRT (F), then U * U = F. The algorithm runs as follows:

```
U.TC[1] := SQRT (F.TC[1]);
U.TC[2] := F.TC[2] / (2 * U.TC[1]);
for K = 3, ..., DIM,
```

(4.3.2)

$$PROD := \sum_{I=2}^{K \text{ DIV } 2} U.TC[I]*U.TC[K-I+1];$$

```
if K is odd, then PROD := PROD + SQR (U.TC[(K+1)/2]);
U.TC[K] := (F.TC[K] - PROD) / (2 * U.TC[1]).
```

If $F(x_0) = 0$, and F is not a constant series, then SQRT (F) cannot be computed unless F is the constant 0, because $F'(x_0)$ is not defined. The SQRT functions are named TSQRT and ITSQRT to indicate the type of operand accepted and value returned.

The natural expontial functions (base = e) are now defined for types TAYLOR and ITAYLOR.

If U := EXP (F), then U' = U * F'. This gives the algorithm:

```
U.TC[1] := EXP (F.TC[1]);
for K = 2,...,DIM,
```

(4.3.3)

$$U.TC[K] := \left(\sum_{I=1}^{K-1} U.TC[I]*F.TC[K-I+1]*(K-I) \right) / (K-1);$$

Although this formula would appear slightly simpler with the change of index J = K - I, it

was implemented in this way so that the U.TC terms remain stationary in the inner product as K increases. Thus, only the vector F.TC needs to be "reversed". The EXP functions are named TEXP and ITEXP to indicate the type of operand accepted and value returned.

4.3.2. The Operator **. The family of power operators ** seems to be the most difficult to implement as suggested by [5]. None of the operators are especially difficult, but there are many minor details to be considered. The implementation of ** for types TAYLOR and ITAYLOR is based on the standard power functions above, and the power operators ** for the scalar types INTEGER (K), REAL (R), and INTERVAL (I).

Scalar Powers:

K ** K, R ** K, K ** R, R ** R

I ** K, K ** I, I ** I

Integer powers are implemented using repeated squaring. Real and interval powers which fit no special case are computed by F ** G = EXP (G * LN(F)). We have not attempted optimal implementations of the scalar power operators because it is hoped that they will be provided as standard operators in a later release of Pascal-SC, an approach that is especially attractive for interval operands because the interpreter hides information from programmers which can be used for correctly directed roundings.

Real and Interval Taylor Powers:

K ** T, T ** K, R ** T, T ** R, T ** T

K ** IT, IT ** K, I ** IT, IT ** I, IT ** IT

The power operators for a constant to a variable power follow the pattern of TEXP or ITEXP, as appropriate. A series which represents a constant (only its first term is non-zero) is handled as a special case for accuracy (especially for interval series) and for efficiency.

The operators T ** K and IT ** K take care to return the correct answer whenever that is possible and to produce an appropriate error message when it is not possible. The resolution of various cases is shown in Table 4.1.

Exponent:	0	1	2	> 2	< 0
Base.TC = 0	Undef.	= 0	= 0	= 0	Undef.
Base.TC[1] = 0	1	= BASE	SQR (BASE)	By mult.	Undef.
Base.TC[1] <> 0	1	= BASE	SQR (BASE)	By recurrence	

Table 4.1. Resolution of Cases for **.

Consider a series whose first term is zero, but which has other terms which are non-zero. Raising such a series to a negative power is undefined because it is equivalent to dividing by zero, but raising such a series to the power 0 defines a function which is identically equal to 1, except for a removable singularity at $x = x_0$. Hence it is appropriate to give 1 as the answer. Raising the series to a positive integer power is implemented by repeated squaring because the recurrence relation which is most often used [20] requires division by BASE.TC[1], which is zero.

The special cases of an exponent equal to 1 or 2 are singled out for individual treatment in order to achieve the maximum possible accuracy (especially when the base is an interval series) and for efficiency.

Except in the special cases shown in the table, if U = F ** E, where E is of type INTEGER, then F * U' = E * U * F'. This gives the algorithm

U.TC := 0;

U.TC[1] := F.TC[1] ** E;

(4.3.4) For K := 2 to DIM,

$$U.TC[K] := \left(\sum_{I=1}^{K-1} (E*(K-I) - I + 1) * U.TC[I] * F.TC[K-I+1] \right) / ((K-1)*F.TC[I]).$$

The integer exponent appears in the recurrence only as a multiplier. Hence the speed of this algorithm is nearly independent of the size of the exponent. That is why this algorithm is preferred to repeated squaring.

The operators T ** R and IT ** I are similar to T ** K and IT ** K, respectively, except that the additional special cases of an exponent equal to 1/2 or to an integer are handled.

The operators T ** T and IT ** IT are included primarily for completeness; the authors have never seen a differential equation containing a variable to a variable power, for example. Perhaps any such problems which arise are at once simplified by logarithmic differentiation. With the tools described here, it may be advantagious to attack the problem in its original form.

Within the operators T ** T and IT ** IT, the cases in which either the base or the exponent series represent a constant function are treated as special for reasons of accuracy and efficiency. Otherwise, F ** G = EXP (G * LN (F)), using TEXP and TLN or ITEXP and ITLN, as appropriate.

4.4. Standard Functions. There are many useful library functions which could be provided. We have chosen to implement the functions which are built into Pascal-SC for the standard scalar data types, and a few others. Additional functions can be added as they are needed by following the models provided by this report. In addition to the standard power functions of Section 4.3.1 (which include EXP and IEXP), other standard functions implemented for types TAYLOR and ITAYLOR are

TSIN (T), TCOS (T), TLN (T), TARCTAN (T)

ITSIN (IT), ITCOS (IT), ITLN (IT), ITARCTAN (IT)

If U = sin (F) and V = cos (F), then U' = V * F' and V' = - U * F'.

```
U.TC[1] := SIN (F.TC[1]);  V.TC[1] := COS (F.TC[1]);
for K := 2,...,DIM,
```

(4.4.1)

$$U.TC[K] := \Big(\sum_{I=2}^{K} V.TC[I] * F.TC[K-I+1] * (K-I) \Big) / (K-1);$$

$$V.TC[K] := -\Big(\sum_{I=2}^{K} U.TC[I] * F.TC[K-I+1] * (K-I) \Big) / (K-1).$$

The SIN and COS functions are named TSIN and TCOS or ITSIN and ITCOS to indicate the type of operand which they accept and value they return. Since the series for SIN and COS are always computed together, the library also contains procedures T_SIN_COS and IT_SIN_COS which return both the SIN and COS of variables of type TAYLOR and ITAYLOR, respectively, in the same call.

If U := ln (F), then U' * F = F'.

```
U.TC[1] := LN (F.TC[1]);
for K := 2,...,DIM,
```

(4.4.2)

$$U.TC[K] := \Big(F.TC[K] - \Big(\sum_{I=2}^{K-1} U.TC[I] * F.TC[K-I+1] * (I-1) \Big) / (K-1)\Big) / F.TC[1].$$

There is a misprint in this recurrence relation in ([16], p. 42), but its implementation is straightforward.

If U := arctan (F) and V := 1 / (1 + F^2), then U' = V * F'.

```
V := 1 / ( 1 + SQR (F) );
U.TC[1] := ARCTAN ( F.TC[1] );
```

(4.4.3) for K := 2,...,DIM,

$$U.TC[K] := \Big(\sum_{I=2}^{K} V.TC[I] * F.TC[K-I+1] * (K-I) \Big) / (K-1).$$

Since the series for the Runge function V(F) ([7], p. 78) is required to compute the series for arctan(F), functions TRUNGE and ITRUNGE are included in the library along with the functions TARCTAN and ITARCTAN.

4.5. User Defined Functions. If a programmer requires an operation or a function which is not included in this report, the requirement can be met either by a composition of operators and functions which are already provided, or by a careful derivation of the necessary recurrence relations following the models in this report. For example, the tangent functions TTAN(T) and ITTAN(IT) are implemented essentially by

(4.5.1) TTAN(T) := TSIN(T) / TCOS(T), ITTAN(IT) := ITSIN(IT) / ITCOS(IT),

respectively in the set of additional functions provided in the library. The tangent functions can also be implemented directly by recurrence relations, using the fact that y = tan(x) satisfies the differential equation

(4.5.2) $$y' = 1 + y^2, \quad y(x_0) = \tan(x_0),$$

[12], [13]. Thus, for U := TTAN(T), for example,

(4.5.3) U.TC[1] := TAN(T.TC[1]), U.TC[2] := (1 + SQR(T.TC[1])) * T.T.

The succeeding coefficients can be obtained in a simple way from the recurrence relation (4.3.1) for TSQR(T), and ITTAN(IT) is computed analogously.

The Runge function $f(x) = 1 / (1 + x^2)$, which is an auxiliary function for the series expansion of the arctangent, is implemented in the library by

```
                    U := TSQR(IT);
(4.5.4)             U.TC[1] := 1 + U.TC[1];
                    TRUNGE := 1 / U.
```

ITRUNGE is computed similarly.

4.6. Differentiation and Integration. Functions which return the results of term-by-term differentiation and integration of TAYLOR and ITAYLOR series are also provided. For series with $1 < \text{LENGTH} \leq \text{DIM}$, differentiation decreases the length of the series by one:

(4.6.1) U.TC[K] := T.TC[K + 1] * RATIO / (K + 1), K = 1,...,T.LENGTH - 1,

where RATIO = 1 / T.T if U = TDIFF (T), and RATIO = 1 / INTPT (T.T) if U = ITDIFF (T). Integration results in a series with its first coefficient set to 0 and its length increased by one:

(4.6.2) U.TC[K] := T.TC[K - 1] * RATIO / (K - 1), K = 2,...,T.LENGTH,

with U.TC[1] = 0 and RATIO = T.T for U = TINTGRL(T), while U.TC[1] = INTPT(0) and RATIO = INTPT(T.T) for U = ITINTGRL(T). The result of integration of a series of length DIM will be truncated to length DIM.

4.7. Miscellaneous Utilities. Some useful functions and procedures are provided for convenience. These are the transfer function TMIDPT (IT), the special functions VRNULL, IVRNULL, the comparison functions T_IDENT_ZERO, T_IDENT_CONSTANT, IT_IDENT_ZERO, IT_IDENT_CONSTANT, and the input/output procedures WRITE_INTERVAL (I), WRITE_SERIES (T), READ_INTERVAL_SERIES (IT), WRITE_INTERVAL_SERIES (IT). The purposes of most of these utilities are indicated by their names.

The transfer function TMIDPT (IT) forms a TAYLOR series from a series of type ITAYLOR. The coefficients of the result series are the midpoints of the corresponding coefficients of the interval series.

The parameterless functions VRNULL, IVRNULL yield zero real and interval vectors, respectively, of length DIM. They are standard Pascal-SC functions [23].

The comparison functions yield the BOOLEAN value TRUE if their argument satisfies the stated condition (the series is identically equal to zero or a constant), otherwise, FALSE.

The input/output procedures are also self-explanatory. The procedure WRITE_INTERVAL is included, since the standard Pascal-SC procedure IWRITE only prints the digits of the lower and upper endpoints of intervals which agree up to the last [23]. WRITE_INTERVAL, however, prints all digits of each endpoint.

5. The initial-value problem for ordinary differential equations. Taylor series methods for the numerical solution of initial-value problems for systems of ordinary differential equations have been studied by many authors (see [6] or [13] for summaries), and have been used for applications such as dynamics and parameter identification. Each component of the solution of

$$y_i' = f_i(x,y), \quad y_i(x_0) = y_{i0}, \quad i = 1,\ldots,m, \tag{5.1}$$

is expressed as a Taylor series expanded at $x = x_0$ using recurrence relations derived from the differential equation. Various error control strategies have been employed. The strategy of analyzing the radius of convergence of each component series has the desirable

side effect of producing such analytic information as the location and orders of the singularities in the solution. Once the radius of convergence is known, a stepsize can be chosen which is as large as possible subject to error control and stability constraints. Then each component of the solution is extended by analytic continuation and the process is repeated at the next integration step. This algorithm is discussed in greater detail in [6].

A program RDEQ_SOLV for solving equation (5.1) is given as Appendix A of this report. The program is written for the case m = 1, but can be modified easily to handle systems of several equations. The variables Y and YPRIME are declared to be of type TAYLOR, and the equation is written in a natural way. To solve a different equation, it is only necessary

- o to change the line in RDEQ_SOLV which contains the differential equation;
- o to copy from the library into the source program any operators or functions required by the new differential equation.

Because the differential equation is written using the types and operators discussed in the preceding Sections, the needed recurrence relations are implemented by the Pascal-SC compiler and need not be derived explicitly by the user.

The program prints the series terms, extends the solution by analytic continuation to compute the initial condition at the next step, and repeats the process. The program RDEQ_SOLV in Appendix A is intentionally simple to illustrate the use of the Taylor operators and to explore the stability of the series generation. It would require an error control mechanism in order to be of practical value for the solution of initial value problems. Either scalar [6] or interval [12] error control techniques can be used.

The program IDEQ_SOLVE listed as Appendix B of this report computes interval-valued approximate solutions to equation (5.1) for the case m = 1, but can be modified for systems of several equations. It differs from the program RDEQ_SOLV only in that

i) the variables Y and YPRIME are of type ITAYLOR instead of type TAYLOR, and

ii) additional code has been added to monitor the relative error introduced by instability in the series generation process.

These two programs are designed to serve as examples of one way in which the tools of this report can be used. They are simple, menu-driven programs which allow direct user intervention at each integration step. By observation of the outcome of each step, the user can experiment with error control strategies. A User Manual containing more detailed instructions for using these programs is included as Appendix D of this report.

The bounds computed by IDEQ_SOLV are for the interval-valued Taylor polynomial (3.3). They are not global error bounds for the solution of the differential equation. Global error bounds are readily computable using interval remainder terms (see [12]), but, for simplicity, the programs given here contain no error bounding or control strategy.

6. An application: Stability of series generation. In this section, we present an example which uses the Taylor and interval Taylor operators. This example was chosen because it illustrates the uses of these operators and because it addresses the issue of stability in the generation of the series. The latter issue is central in showing that Taylor series methods are reliable for practical computations.

A numerical computation is said to be unstable if its relative error grows without bound as the computation proceeds. It is possible that the recurrence relations being used might be unstable, although instability has never been observed in practice. This example uses the Taylor and interval Taylor operators to explore the stability of the recurrence relations in one application. In this example, there is instability in the generation of the terms of the series, but that does not seriously affect the accuracy of the series summation. The stability of the recurrence relations in other applications can be handled similarly.

Consider the initial value problem

(6.1) $$y' = y^2, \quad y(0) = 1,$$

whose solution is $y(x) = 1 / (1 - x)$. A program (RDEQ_SOLV) for solving equation (6.1) using the Taylor function TSQR is given as Appendix A of this report. The effect of program RDEQ_SOLV is to generate the normalized Taylor coefficients (4.3.1) of the solution recursively. This recurrence is accomplished automatically by the Taylor function TSQR in the statement YPRIME := TSQR (Y). In this case, the same solution is obtained if Y ** 2 or Y * Y is used instead of TSQR (Y); however, the use of Y ** 2 requires the compilation of much more code, while Y * Y is not as fast as TSQR (Y).

We wish to explore the stability of the recurrence relation (6.2). This issue is separate from the issue of the stability of Taylor series methods for solving initial value problems. If an infinite Taylor series were used, the method would be A-stable, but when a truncated series is used, the region of stability is bounded. Stetter [21] showed that the region of stability for truncated Taylor series methods is the same as that for related Runge-Kutta methods. The real interval of stability is relatively large here because long series are used. For example, the real intervals of stability are [-8.85, 0] and [-16.29, 0], respectively, if DIM = 20 and 40 terms of the series are used.

We will outline the theoretical analysis of the stability of recurrence (6.2). A more complete discussion appears in [6]. Let U(K) denote the actual and Y(K) denote the computed normalized Taylor coefficients. Let $Y(1) = U(1) (1 + \varepsilon) = (1 + \varepsilon)$ from (6.1). Then $U(K) = h^{K-1}$, and

(6.3) $$Y(K) = U(K) (1 + \varepsilon)^K,$$

so the series generation is unstable. However, the summation of the series is unaffected by this instability since

$$y(x_1) = \sum_{K=1}^{DIM} Y(K) = (1 + \varepsilon) \sum_{K=1}^{DIM} (h(1 + \varepsilon))^{K-1}$$

(6.3)

$$= (1 + \varepsilon)\, y(h(1 + \varepsilon)) + O(h^{DIM})$$

$$= (1 + \varepsilon)(y(h) + y'(\xi)h) + O(h^{DIM}), \quad h < \xi < h(1 + \varepsilon).$$

That is, the instability in the series generation is equivalent to a small error in the point at which the series is evaluated. This is because $\Sigma_K Y(K)$ is a convergent series, so the terms for which instability causes the relative error to be largest are themselves very small.

This suggests using interval arithmetic to keep track of the potential growth in the series. The program IDEQ_SOLV listed as Appendix B of this report does so.

By declaring Y and YPRIME to be of type ITAYLOR, the statement YPRIME := ITSQR(Y) invokes the function ITSQR for interval valued series to generate interval normalized Taylor coefficients according the recurrence relation (4.3.1). The lengths of successive coefficients measure the stability of the recurrence. Table 6.1 shows the interval valued series solution of equation (6.1) for DIM = 15, $y(x_0) = y(0) = [0.99\ ,\ 1.01]$ ($\varepsilon = \pm\ 0.01$), and h = 0.5. The computed instability is equal to

$$E_{Computed} = \frac{\text{length (Y.TC[K])}}{\text{midpoint (Y.TC[K])}}, \tag{6.5}$$

a measure of the relative error in Y.TC(K) which appears to grow as K increases. The theoretical instability is equal to

$$E_{Theoretical} = (1 + \varepsilon)^K. \tag{6.6}$$

Table 6.1 shows that these two values are very close, and that the theoretical bound is slightly larger than the actual bound, as it should be. The interval estimate for y(0.5) agrees well with the interval [y(0.46), y(0.54)] =[1.8518, 2.1739].

```
READ INTERVAL INITIAL CONDITIONS X0, Y(X0):


INITIAL CONDITIONS AT X0 = [ 0.00000E+00,  0.00000E+00],
                       Y0 = [ 9.90000E-01,  1.01000E+00].
ENTER STEPSIZE X - X0:  0.5
Computing series terms ...
```

Step	Left Endpoint	Right Endpoint	Computed Instability	Theoretical Instability
1	[9.90000E-01,	1.01000E+00]	1.010E+00	1.010E+00
2	[4.90050E-01,	5.10050E-01]	1.020E+00	1.020E+00
3	[2.42575E-01,	2.57575E-01]	1.030E+00	1.030E+00
4	[1.20075E-01,	1.30076E-01]	1.040E+00	1.041E+00
5	[5.94369E-02,	6.56881E-02]	1.050E+00	1.051E+00
6	[2.94213E-02,	3.31725E-02]	1.060E+00	1.062E+00
7	[1.45635E-02,	1.67521E-02]	1.070E+00	1.072E+00
8	[7.20894E-03,	8.45982E-03]	1.080E+00	1.083E+00
9	[3.56843E-03,	4.27221E-03]	1.090E+00	1.094E+00
10	[1.76637E-03,	2.15747E-03]	1.100E+00	1.105E+00
11	[8.74354E-04,	1.08952E-03]	1.110E+00	1.116E+00
12	[4.32805E-04,	5.50208E-04]	1.119E+00	1.127E+00
13	[2.14239E-04,	2.77855E-04]	1.129E+00	1.138E+00
14	[1.06048E-04,	1.40317E-04]	1.139E+00	1.149E+00
15	[5.24938E-05,	7.08599E-05]	1.149E+00	1.161E+00

```
THE VALUE AT X = [ 5.00000E-01,  5.00000E-01]
         IS Y = [ 1.96034E+00,  2.04033E+00].
```

Table 6.1. Interval bounds for instability.

7. Implementation details. The software described in this report was created and tested using the Pascal-SC compiler developed at the University of Karlsruhe for the Zilog MCE-1 computer with the RIO 2.06 operating system. No other claims of correctness or usability are made.

REFERENCES

1. D. Barton, I. M. Willers, and R. V. M. Zahar. Taylor series methods for ordinary differential equations - An evaluation. In Mathematical Software, John Rice (Ed.). Academic Press, New York, 1971, 369-390.

2. G. Bohlender, K. Gruner, E. Kaucher, R. Klatte, W. Kramer, U. W. Kulisch, S. M. Rump, Ch. Ullrich, J. Wolff von Gudenberg, and W. L. Miranker. Pascal-SC: A Pascal for contemporary scientific computation. Research Report RC 9009, IBM Thomas J. Watson Research Center, Yorktown Heights, N. Y., 1981.

3. Y. F. Chang. Automatic solution of differential equations. In Constructive and Computational Methods for Differential and Integral Equations, D. L. Colton and R. P. Gilbert (Eds.). Lecture Notes in Mathematics, Vol 430, Springer-Verlag, Berlin-Heidelberg-New York, 1974, 61-94.

4. Y. F. Chang, M. Tabor, J. Weiss, and G. F. Corliss. On the structure of the Henon Heiles system. Phys. Lett. A 85A (1981), 211-213.

5. W. J. Cody and W. Waite. Software Manual for the Elementary Functions. Prentice-Hall, Englewood Cliffs, N. J., 1980.

6. G. F. Corliss and Y. F. Chang. Solving ordinary differential equations using Taylor series. ACM Trans. Math. Soft. 8 (1982), 114-144.

7. Philip J. Davis. Interpolation and Approximation. Blaisdell, New York, 1963.

8. K. Jensen and N. Wirth. Pascal User Manual and Report, 2nd Ed. Springer-Verlag, Berlin-Heidelberg-New York, 1974.

9. Carlo Ghezzi and Mehdi Jazayeri. Programming Language Concepts. Wiley, New York, 1982.

10. G. Kedem. Automatic differentiation of computer programs. ACM Trans. Math. Soft. 6 (1980), 150-165.

11. U. W. Kulisch and W. L Miranker. Computer Arithmetic in Theory and Practice. Academic Press, New York, 1981.

12. R. E. Moore. Interval Analysis. Prentice-Hall, Englewood Cliffs, N. J., 1966.

13. R. E. Moore. Methods and Applications of Interval Analysis. SIAM Studies in Applied Mathematics, 2, Philadelphia, 1979.

14. J. F. Palmer. VSLI and the revolution in numeric computation. Proceedings of the 10th IMACS World Congress on System Simulation and Scientific Computation, Vol. 1, pp. 339-341. Montreal, 1982.

15. L. B. Rall. Applications of software for automatic differentiation in numerical computation. Computing, Suppl. 2 (1980), 141-156.

16. L. B. Rall. Automatic Differentiation: Techniques and Applications. Lecture Notes in Computer Science No. 120, Springer-Verlag, Berlin-Heidelberg-New York, 1981.

17. L. B. Rall. Differentiation in Pascal-SC: Type GRADIENT. Technical Summary Report No. 2400, Mathematics Research Center, University of Wisconsin-Madison, 1982.

18. L. B. Rall. Differentiation and generation of Taylor coefficients in Pascal-SC. Technical Summary Report No. 2452, Mathematics Research Center, University of Wisconsin-Madison, 1982.

19. L. B. Rall. Mean-value and Taylor forms in interval analysis. SIAM J. Math. Anal. 14 (1983), no. 2, 223-238.

20. Allen Reiter. Automatic generation of Taylor coefficients (TAYLOR) for the CDC 1604. Technical Summary Report No. 830, Mathematics Research Center, University of Wisconsin-Madison, 1967.

21. H. J. Stetter. Analysis of Discretization Methods for Ordinary Differential Equations. Springer-Verlag, Berlin-Heidelberg-New York, 1973.

22. G. Weinberg. The Psychology of Computer Programming. Van Nostrand Reinhold, New York, 1971.

23. J. Wolff von Gudenberg. Gesamte Arithmetik des PASCAL-SC Rechners: Benutzerhandbuch. Institute for Applied Mathematics, University of Karlsruhe, 1981.

APPENDIX A

```
PROGRAM RDEQ_SOLVE (INPUT, DATA, OUTPUT);

(*  SOLVE A FIRST ORDER DIFFERENTIAL EQUATION:  Y' = SQR (Y)  *)

CONST DIM = 30;
TYPE DIMTYPE  = 1..DIM;
     RVECTOR  = ARRAY[DIMTYPE] OF REAL;
     TAYLOR   = RECORD  LENGTH : DIMTYPE;
                        T      : REAL;
                        TC     : RVECTOR  END;
     CHOICE   = 1..3;

VAR      FLAG              : CHOICE;
         I, IM1            : DIMTYPE;
         X, Y, YPRIME      : TAYLOR;
         DATA              : TEXT;
FUNCTION VRNULL : RVECTOR;
    VAR I: DIMTYPE; U: RVECTOR;
    BEGIN
    FOR I := 1 TO DIM DO U[I] := 0.0;
    VRNULL := U
    END;  (*  FUNCTION VRNULL  *)
FUNCTION TSQR (T: TAYLOR) : TAYLOR;                          (*  TSQR(T)  *)
    (*  Requires:  VRNULL, SCALP, SQR  *)
    VAR I, J, K, HALF: DIMTYPE;
        X, Y: RVECTOR;
        U   : TAYLOR;
    BEGIN
    X := VRNULL;  Y := VRNULL;
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    U.TC := VRNULL;
    U.TC[1] := SQR (T.TC[1]);
    X[1] := T.TC[1];
    FOR K := 2 TO U.LENGTH DO
        BEGIN
        X[K] := T.TC[K];
        HALF := K DIV 2;
        FOR J := 1 TO HALF DO
            BEGIN
            I := K - J + 1;
            Y[J] := T.TC[I]
            END;  (* FOR J *)
        U.TC[K] := 2.0 * SCALP ( X, Y, 0);
        IF K MOD 2 = 1 THEN
            BEGIN
            HALF := HALF + 1;
            U.TC[K]:= U.TC[K] + SQR (T.TC[HALF])
            END  (* IF *)
        END;  (* FOR K *)
    TSQR := U
    END;  (* FUNCTION TSQR (TAYLOR)  *)
```

```
FUNCTION MENU_CHOICE : CHOICE;
    VAR I: INTEGER;
    BEGIN
    WRITELN;
    WRITELN ('ENTER:  1  -  GIVE NEW INITIAL CONDITIONS');
    WRITELN ('        2  -  CONTINUE EXTENDING THE SOLUTION');
    WRITELN ('        3  -  STOP');
    READ (I);
    IF ((I >= 3) OR (I <= 0)) THEN I := 3;
    MENU_CHOICE := I
    END;  (*  FUNCTION MENU_CHOICE  *)
PROCEDURE PRNT_TAY_COEF (Y: TAYLOR; INDEX: DIMTYPE);
    BEGIN
    WRITELN   ('Y(', INDEX:5, ') = ', Y.TC[INDEX])
    END;  (*  PROCEDURE PRNT_TAY_COEF  *)
FUNCTION SUM (VAR A: RVECTOR; DIM, ROUND: INTEGER) : REAL;
  EXTERNAL 480;

BEGIN  (*MAIN PROGRAM RDEQ_SOLVE*)
(*.........  INITIALIZE *)
FLAG := 2;
X.LENGTH := DIM;
Y.LENGTH := DIM;

RESET (DATA);
WHILE FLAG <= 2 DO              (* LOOP FOR NEW INITIAL CONDITIONS *)
    BEGIN
    FLAG := 2;
X.TC := VRNULL;
Y.TC := VRNULL;

    WRITELN ('READ REAL INITIAL CONDITIONS X0, Y(X0):');
    READ (DATA, X.TC[1]); READ (DATA, Y.TC[1]);
    WRITELN;  WRITELN;
    WRITELN ('INITIAL CONDITIONS AT X0 = ', X.TC[1], ',');
    WRITELN ('                     Y0 = ', Y.TC[1], '.');
    WHILE FLAG = 2 DO           (* LOOP FOR ANALYTIC CONTINUATION *)
        BEGIN
        (*.........  READ STEP SIZE *)
        WRITELN ('ENTER STEPSIZE X - X0:  ');
        READ (X.T);
        Y.T := X.T;
        WRITELN ('Computing series terms ...');
        FOR I := 2 TO DIM DO            (* LOOP FOR SERIES GENERATION *)
            BEGIN

(*      YOUR FIRST ORDER DIFFERENTIAL EQUATION GOES HERE:          *)
    YPRIME := TSQR (Y);

            IM1 := I - 1;
            Y.TC[I] := YPRIME.TC[IM1] * Y.T / IM1;
            END;  (*FOR*)
        (*.........  PRINT TABLE *)
        WRITELN;  WRITELN;
        WRITELN ('THE TAYLOR COEFFICIENTS OF Y ARE:');
        FOR I := 1 TO DIM DO PRNT_TAY_COEF (Y, I);
```

```
        (*.........  PERFORM THE ANALYTIC CONTINUATION *)

        Y.TC[1] := SUM (Y.TC, DIM, 0);
        FOR I:= 2 TO DIM DO Y.TC[I] := 0.0;
        X.TC[1] := X.TC[1] + X.T;
        WRITELN;
        WRITELN ('THE VALUE AT X = ', X.TC[1]);
        WRITELN ('          IS Y = ', Y.TC[1], '.');
        FLAG := MENU_CHOICE
        END  (*WHILE*)

    END  (*WHILE*)

END.  (*  MAIN PROGRAM RDEQ_SOLVE  *)
```

APPENDIX B

```
PROGRAM IDEQ_SOLVE (INPUT, DATA, OUTPUT);

(*  SOLVE A FIRST ORDER DIFFERENTIAL EQUATION  *)
(*  Y' = SQR (Y)  *)
(*  SOLUTION IS IN INTERVAL FORM  *)

CONST DIM = 15;

TYPE DIMTYPE  = 1..DIM;
     INTERVAL = RECORD INF, SUP : REAL END;
     IVECTOR  = ARRAY[DIMTYPE] OF INTERVAL;
     ITAYLOR  = RECORD  LENGTH : DIMTYPE;
                        T      : REAL;
                        TC     : IVECTOR  END;
     CHOICE   = 1..3;

VAR      FLAG              : CHOICE;
         I, IM1            : DIMTYPE;
         X, Y, YPRIME      : ITAYLOR;
         DATA              : TEXT;
         EPSILON,
         COMPOUND          : REAL;

       (*  Transfer Functions  *)

FUNCTION INTPT ( RA:REAL ) : INTERVAL;
  EXTERNAL 41;
FUNCTION INTVAL ( RA,RB: REAL ) : INTERVAL;
  EXTERNAL 42;
FUNCTION IINF ( A: INTERVAL) : REAL;
  EXTERNAL 43;
FUNCTION ISUP ( A: INTERVAL) : REAL;
  EXTERNAL 44;

        (* Comparisons *)

OPERATOR <= (A,B: INTERVAL ) RES: BOOLEAN ;
  EXTERNAL 48;
OPERATOR >= (A,B: INTERVAL ) RES: BOOLEAN ;
  EXTERNAL 50;
OPERATOR IN (RA:REAL; B: INTERVAL) RES: BOOLEAN;
  EXTERNAL 47;
OPERATOR IN (KA: INTEGER; B: INTERVAL) RES: BOOLEAN;
  EXTERNAL 46;
OPERATOR >< (A,B: INTERVAL ) RES: BOOLEAN ;
  EXTERNAL 52;

        (* Lattice Operators *)

OPERATOR ++ (A,B: INTERVAL) RES: INTERVAL;
  EXTERNAL 63;
OPERATOR ** (A,B: INTERVAL) RES: INTERVAL;
  EXTERNAL 60;
```

```
        (* Arithmetic Operators *)

OPERATOR + ( A: INTERVAL ) RES: INTERVAL;
  EXTERNAL 67;
OPERATOR - ( A: INTERVAL ) RES: INTERVAL;
  EXTERNAL 66;
OPERATOR + ( A,B: INTERVAL ) RES: INTERVAL;
  EXTERNAL 68;
OPERATOR + ( KA: INTEGER; B: INTERVAL ) RES: INTERVAL;
  EXTERNAL 69;
OPERATOR + ( A: INTERVAL; KB: INTEGER ) RES: INTERVAL;
  EXTERNAL 70;
OPERATOR - ( A,B: INTERVAL ) RES: INTERVAL;
  EXTERNAL 73;
OPERATOR - ( KA: INTEGER; B: INTERVAL ) RES: INTERVAL;
  EXTERNAL 75;
OPERATOR - ( A: INTERVAL; KB: INTEGER ) RES: INTERVAL;
  EXTERNAL 74;
OPERATOR * ( A,B: INTERVAL ) RES: INTERVAL;
  EXTERNAL 78;
OPERATOR * ( KA: INTEGER; B: INTERVAL ) RES: INTERVAL;
  EXTERNAL 79;
OPERATOR * ( A: INTERVAL; KB: INTEGER ) RES: INTERVAL;
  EXTERNAL 80;
OPERATOR / ( A,B: INTERVAL ) RES: INTERVAL;
  EXTERNAL 85;
OPERATOR / ( KA: INTEGER; B: INTERVAL ) RES: INTERVAL;
  EXTERNAL 83;
OPERATOR / ( A: INTERVAL; KB: INTEGER ) RES: INTERVAL;
  EXTERNAL 86;

FUNCTION ISCALP (VAR A, B: IVECTOR; AKDIM : INTEGER) : INTERVAL;
  EXTERNAL 88;

      (*  Standard Functions  *)

FUNCTION IABS ( Y: INTERVAL ) : REAL;
  EXTERNAL 101;
FUNCTION ISQR ( Y: INTERVAL ) : INTERVAL;
  EXTERNAL 102;
FUNCTION ISQRT ( Y: INTERVAL ) : INTERVAL;
  EXTERNAL 105;
FUNCTION IEXP ( Y: INTERVAL ) : INTERVAL;
  EXTERNAL 106;
FUNCTION ILN ( Y: INTERVAL ) : INTERVAL;
  EXTERNAL 107;
FUNCTION IARCTAN ( Y: INTERVAL ) : INTERVAL;
  EXTERNAL 108;
FUNCTION ISIN ( Y: INTERVAL ) : INTERVAL;
  EXTERNAL 109;
FUNCTION ICOS ( Y: INTERVAL ) : INTERVAL;
  EXTERNAL 110;
```

```
      (*  Input / Output  *)

PROCEDURE IREAD ( VAR F:TEXT; VAR A: INTERVAL );
  EXTERNAL 92;
PROCEDURE IWRITE ( VAR F: TEXT; A: INTERVAL );
  EXTERNAL 91;

FUNCTION ISUM (A: IVECTOR; DIM: DIMTYPE) : INTERVAL;
    VAR B: IVECTOR;
        I: DIMTYPE;
    BEGIN
    FOR I := 1 TO DIM DO B[I] := INTPT (1.0);
    ISUM := ISCALP (A, B, DIM)
    END;  (*  FUNCTION ISUM  *)

FUNCTION IVRNULL : IVECTOR;
    VAR I: DIMTYPE; U: IVECTOR;
    BEGIN
    FOR I := 1 TO DIM DO U[I] := INTPT (0.0);
    IVRNULL := U
    END;  (*  FUNCTION IVRNULL  *)

FUNCTION ITSQR (T: ITAYLOR) : ITAYLOR;                    (*  ITSQR(IT)  *)
    (*  Requires:  IVRNULL, ISCALP, ISQR  *)
    VAR I, J, K, HALF: DIMTYPE;
        X, Y: IVECTOR;
        U   : ITAYLOR;

    BEGIN
    X := IVRNULL;  Y := IVRNULL;
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    U.TC := IVRNULL;

    U.TC[1] := ISQR (T.TC[1]);
    X[1] := T.TC[1];

    FOR K := 2 TO U.LENGTH DO
        BEGIN
        X[K] := T.TC[K];
        HALF := K DIV 2;
        FOR J := 1 TO HALF DO
            BEGIN
            I := K - J + 1;
            Y[J] := T.TC[I]
            END;  (* FOR J *)
        U.TC[K] := 2 * ISCALP ( X, Y, HALF);
        IF K MOD 2 = 1 THEN
            BEGIN
            HALF := HALF + 1;
            U.TC[K]:= U.TC[K] + ISQR (T.TC[HALF])
            END  (* IF *)
        END;  (* FOR K *)
    ITSQR := U
    END;  (* FUNCTION ITSQR (ITAYLOR)  *)
```

```
FUNCTION MENU_CHOICE : CHOICE;
    VAR I: INTEGER;
    BEGIN
    WRITELN;
    WRITELN ('ENTER:  1  -  GIVE NEW INITIAL CONDITIONS');
    WRITELN ('        2  -  CONTINUE EXTENDING THE SOLUTION');
    WRITELN ('        3  -  STOP');
    READ (I);
    IF ((I >= 3) OR (I <= 0)) THEN I := 3;
    MENU_CHOICE := I
    END;  (*  FUNCTION MENU_CHOICE  *)


PROCEDURE WRITE_INTERVAL (INT: INTERVAL);
    BEGIN
    WRITE ('[', INT.INF:12, ', ', INT.SUP:12, ']');
    END;  (*  PROCEDURE WRITE_INTERVAL  *)


PROCEDURE PRNT_ITAY_COEF (Y: ITAYLOR; INDEX: DIMTYPE);
    BEGIN
    WRITE ('Y(', INDEX:5, ') = ');
    WRITE_INTERVAL (Y.TC[INDEX]);
    WRITELN
    END;  (*  PROCEDURE PRNT_ITAY_COEF  *)


FUNCTION INTERVAL_LENGTH (INT: INTERVAL) : REAL;
    BEGIN
    INTERVAL_LENGTH := INT.SUP - INT.INF
    END;  (*  FUNCTION INTERVAL_LENGTH  *)


FUNCTION RELATIVE_LENGTH (INT: INTERVAL) : REAL;
    BEGIN
    RELATIVE_LENGTH := 2.0 * (INT.SUP - INT.INF)
                           / (INT.SUP + INT.INF)
    END;  (*  FUNCTION RELATIVE_LENGTH  *)


FUNCTION RELATIVE_ERROR (INT: INTERVAL) : REAL;
    BEGIN
    RELATIVE_ERROR := 2.0 * INT.SUP / (INT.SUP + INT.INF)
    END;  (*  FUNCTION RELATIVE_ERROR  *)



BEGIN  (*  MAIN PROGRAM IDEQ_SOLVE  *)

(*.....  INITIALIZE *)
FLAG := 2;
X.LENGTH := DIM;
Y.LENGTH := DIM;
RESET (DATA);
```

```
WHILE FLAG <= 2 DO                (* LOOP FOR NEW INITIAL CONDITIONS *)
    BEGIN
    FLAG := 2;
X.TC := IVRNULL;
Y.TC := IVRNULL;
    WRITELN ('READ INTERVAL INITIAL CONDITIONS X0, Y(X0):');
    IREAD (DATA, X.TC[1]); IREAD (DATA, Y.TC[1]);
    WRITELN;  WRITELN;
    WRITE ('INITIAL CONDITIONS AT X0 = ');
    WRITE_INTERVAL (X.TC[1]);  WRITELN (',');
    WRITE ('                     Y0 = ');
    WRITE_INTERVAL (Y.TC[1]);  WRITELN ('.');
    WHILE FLAG = 2 DO             (* LOOP FOR ANALYTIC CONTINUATION *)
        BEGIN
        (*.........  READ STEP SIZE *)
        WRITELN ('ENTER STEPSIZE X - X0:  ');
        READ (X.T);
        Y.T := X.T;
        WRITELN ('Computing series terms ...');
        FOR I := 2 TO DIM DO              (* LOOP FOR SERIES GENERATION *)
            BEGIN
(*      YOUR FIRST ORDER DIFFERENTIAL EQUATION GOES HERE:          *)
    YPRIME := ITSQR (Y);

            IM1 := I - 1;
            Y.TC[I] := YPRIME.TC[IM1] * INTPT (Y.T / IM1);
            END;  (*FOR*)

        (*.........  PRINT TABLE *)
        EPSILON := 0.5 * RELATIVE_LENGTH (Y.TC[1]);
        COMPOUND := 1.0;
        WRITELN;  WRITELN;
        WRITELN ('Step      Left          Right          Computed
Theoretical');
        WRITELN ('        Endpoint      Endpoint      Instability
Instability');
        WRITELN;
        FOR I := 1 TO DIM DO              (* LOOP FOR ERROR MEASUREMENT *)
            BEGIN
            COMPOUND := COMPOUND * (1.0 + EPSILON);
            WRITE (I:3);  WRITE ('  ');
            WRITE_INTERVAL (Y.TC[I]);
            WRITE ('    ', RELATIVE_ERROR (Y.TC[I]):10);
            WRITE ('      ', COMPOUND:10);  WRITELN
            END;  (*FOR*)
        (*.........  PERFORM THE ANALYTIC CONTINUATION *)
        Y.TC[1] := ISUM (Y.TC, DIM);
        FOR I :=2 TO DIM DO Y.TC[I] := INTPT (0.0);
        X.TC[1] := X.TC[1] + INTPT (X.T);
        WRITELN;
        WRITE ('THE VALUE AT X = ');  WRITE_INTERVAL (X.TC[1]);  WRITELN;
        WRITE ('          IS Y = ');  WRITE_INTERVAL (Y.TC[1]);  WRITELN ('.');
        FLAG := MENU_CHOICE
        END  (*WHILE*)
    END  (*WHILE*)
END.  (*MAIN PROGRAM IDEQ_SOLVE*)
```

APPENDIX C

Source Code for TAYLOR and ITAYLOR Operators, Functions, and Procedures

The Pascal-SC source code for the operators, functions, and procedures described in this report is contained in seven files:

1. Real and interval Taylor addition and subtraction operators (RIT_ADD.LIB);
2. Real and interval Taylor multiplication operators including TSQR and ITSQR (RIT_MUL.LIB);
3. Real and interval Taylor division operators (RIT_DIV.LIB);
4. Real Taylor power operators and functions (RT_POW.LIB);
5. Interval Taylor power operators and functions (IT_POW.LIB);
6. Real and interval Taylor functions (RIT_FNS.LIB);
7. Utility functions and procedures (UTIL.LIB).

Each file is headed by a table of contents to assist in the location of the needed routines. In addition, a complete table of contents for all files of TAYLOR and ITAYLOR operators, functions, and procedures is given in the file CONTENTS.LIB. The first line of code for each subroutine contains a short identification in comments brackets, for example, (* K + T *), to assist in locating it with the help of a text editor.

C.1. Real and Interval Taylor Addition and Subtraction Operators.

```
(*  RIT_ADD.LIB  -  REAL AND INTERVAL TAYLOR ADD AND SUBTRACT    <<<<<<

Contents:

    + T          OPERATOR + (T: TAYLOR) RES : TAYLOR;
    K + T        OPERATOR + (K: INTEGER; T: TAYLOR) RES : TAYLOR;
    T + K        OPERATOR + (T: TAYLOR; K: INTEGER) RES : TAYLOR;
    R + T        OPERATOR + (R: REAL; T: TAYLOR) RES : TAYLOR;
    T + R        OPERATOR + (T: TAYLOR; R: REAL) RES : TAYLOR;
    T + T        OPERATOR + (TA, TB: TAYLOR) RES : TAYLOR;
    + IT         OPERATOR + (T: ITAYLOR) RES : ITAYLOR;
    K + IT       OPERATOR + (K: INTEGER; T: ITAYLOR) RES : ITAYLOR;
    IT + K       OPERATOR + (T: ITAYLOR; K: INTEGER) RES : ITAYLOR;
    I + IT       OPERATOR + (K: INTERVAL; T: ITAYLOR) RES : ITAYLOR;
    IT + I       OPERATOR + (T: ITAYLOR; K: INTERVAL) RES : ITAYLOR;
    IT + IT      OPERATOR + (TA, TB: ITAYLOR) RES : ITAYLOR;

    - T          OPERATOR - (T: TAYLOR) RES : TAYLOR;
    K - T        OPERATOR - (K: INTEGER; T: TAYLOR) RES : TAYLOR;
    T - K        OPERATOR - (T: TAYLOR; K: INTEGER) RES : TAYLOR;
    R - T        OPERATOR - (R: REAL; T: TAYLOR) RES : TAYLOR;
    T - R        OPERATOR - (T: TAYLOR; R: REAL) RES : TAYLOR;
    T - T        OPERATOR - (TA, TB: TAYLOR) RES : TAYLOR;
    - IT         OPERATOR - (T: ITAYLOR) RES : ITAYLOR;
    K - IT       OPERATOR - (K: INTEGER; T: ITAYLOR) RES : ITAYLOR;
    IT - K       OPERATOR - (T: ITAYLOR; K: INTEGER) RES : ITAYLOR;
    I - IT       OPERATOR - (K: INTERVAL; T: ITAYLOR) RES : ITAYLOR;
    IT - I       OPERATOR - (T: ITAYLOR; K: INTERVAL) RES : ITAYLOR;
    IT - IT      OPERATOR - (TA, TB: ITAYLOR) RES : ITAYLOR;

================================================================== *)

OPERATOR + (T: TAYLOR) RES : TAYLOR;                              (*  + T  *)
    BEGIN  RES := T  END;  (*  + TAYLOR  *)


OPERATOR + (K: INTEGER; T: TAYLOR) RES : TAYLOR;                  (*  K + T  *)
    VAR U: TAYLOR;
    BEGIN
    U := T;
    U.TC[1] := K + T.TC[1];
    RES := U
    END;  (*  INTEGER + TAYLOR  *)


OPERATOR + (T: TAYLOR; K: INTEGER) RES : TAYLOR;                  (*  T + K  *)
    VAR U: TAYLOR;
    BEGIN
    U := T;
    U.TC[1] := T.TC[1] + K;
    RES := U
    END;  (*  TAYLOR + INTEGER  *)
```

```
OPERATOR + (R: REAL; T: TAYLOR) RES : TAYLOR;                    (*  R + T  *)
    VAR U: TAYLOR;
    BEGIN
    U := T;
    U.TC[1] := R + U.TC[1];
    RES := U
    END;  (*  REAL + TAYLOR  *)


OPERATOR + (T: TAYLOR; R: REAL) RES : TAYLOR;                    (*  T + R  *)
    VAR U: TAYLOR;
    BEGIN
    U := T;
    U.TC[1] := U.TC[1] + R;
    RES := U
    END;  (*  TAYLOR + REAL  *)


OPERATOR + (TA, TB: TAYLOR) RES : TAYLOR;                        (*  T + T  *)
    VAR U: TAYLOR; I: DIMTYPE;
    BEGIN
    IF TA.T <> TB.T
        THEN BEGIN
            WRITE   ('ERROR:  ADDITION OF TAYLOR VARIABLES');
            WRITELN (' WITH UNEQUAL SCALE FACTORS');
            SVR (0)  END                (* RETURN TO OPERATING SYSTEM*)
        ELSE BEGIN
            U.LENGTH := TA.LENGTH;
            IF U.LENGTH > TB.LENGTH THEN U.LENGTH := TB.LENGTH;
            U.T := TA.T;
            FOR I := 1 TO U.LENGTH DO U.TC[I] := TA.TC[I] + TB.TC[I];
            RES := U
        END     (* ELSE *)
    END;  (*  TAYLOR + TAYLOR  *)


OPERATOR + (T: ITAYLOR) RES : ITAYLOR;                           (*  + IT  *)
    BEGIN  RES := T  END;  (*  + ITAYLOR  *)


OPERATOR + (K: INTEGER; T: ITAYLOR) RES : ITAYLOR;               (*  K + IT  *)
    VAR U: ITAYLOR;
    BEGIN
    U := T;
    U.TC[1] := K + T.TC[1];
    RES := U
    END;  (*  INTEGER + ITAYLOR  *)


OPERATOR + (T: ITAYLOR; K: INTEGER) RES : ITAYLOR;               (*  IT + K  *)
    VAR U: ITAYLOR;
    BEGIN
    U := T;
    U.TC[1] := T.TC[1] + K;
    RES := U
    END;  (*  ITAYLOR + INTEGER  *)
```

```
OPERATOR + (K: INTERVAL; T: ITAYLOR) RES : ITAYLOR;              (*  I + IT  *)
    VAR U: ITAYLOR;
    BEGIN
    U := T;
    U.TC[1] := K + T.TC[1];
    RES := U
    END;  (*  INTERVAL + ITAYLOR  *)


OPERATOR + (T: ITAYLOR; K: INTERVAL) RES : ITAYLOR;              (*  IT + I  *)
    VAR U: ITAYLOR;
    BEGIN
    U := T;
    U.TC[1] := T.TC[1] + K;
    RES := U
    END;  (*  ITAYLOR + INTERVAL  *)


OPERATOR + (TA, TB: ITAYLOR) RES : ITAYLOR;                      (*  IT + IT  *)
    VAR U: ITAYLOR; I: DIMTYPE;
    BEGIN
    IF TA.T <> TB.T
        THEN BEGIN
            WRITE   ('ERROR:  ADDITION OF ITAYLOR VARIABLES');
            WRITELN (' WITH UNEQUAL SCALE FACTORS');
            SVR (0)  END                (* RETURN TO OPERATING SYSTEM*)
        ELSE BEGIN
            U.LENGTH := TA.LENGTH;
            IF U.LENGTH > TB.LENGTH THEN U.LENGTH := TB.LENGTH;
            U.T := TA.T;
            FOR I := 1 TO U.LENGTH DO U.TC[I] := TA.TC[I] + TB.TC[I];
            RES := U
        END     (* ELSE *)
    END;  (*  ITAYLOR + ITAYLOR  *)


OPERATOR - (T: TAYLOR) RES : TAYLOR;                             (*  - T  *)
    VAR U: TAYLOR; I: DIMTYPE;
    BEGIN
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    FOR I := 1 TO U.LENGTH DO U.TC[I] := - T.TC[I];
    RES := U
    END;  (*  - TAYLOR  *)


OPERATOR - (K: INTEGER; T: TAYLOR) RES : TAYLOR;                 (*  K - T  *)
    VAR U:TAYLOR; I: DIMTYPE;
    BEGIN
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    U.TC[1] := K - T.TC[1];
    FOR I:= 2 TO U.LENGTH DO U.TC[I] := - T.TC[I];
    RES := U
    END;  (*  INTEGER - TAYLOR  *)
```

```
OPERATOR - (T: TAYLOR; K: INTEGER) RES : TAYLOR;                    (*  T - K  *)
    VAR U: TAYLOR;
    BEGIN
    U := T;
    U.TC[1] := T.TC[1] - K;
    RES := U
    END;  (*  TAYLOR - INTEGER  *)


OPERATOR - (R: REAL; T: TAYLOR) RES : TAYLOR;                       (*  R - T  *)
    VAR U: TAYLOR; I: DIMTYPE;
    BEGIN
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    U.TC[1] := R - T.TC[1];
    FOR I := 2 TO U.LENGTH DO U.TC[I] := - T.TC[I];
    RES := U
    END;  (*  REAL - TAYLOR  *)


OPERATOR - (T: TAYLOR; R: REAL) RES : TAYLOR;                       (*  T - R  *)
    VAR U: TAYLOR;
    BEGIN
    U := T;
    U.TC[1]:=T.TC[1] - R;
    RES := U
    END;  (*  TAYLOR - REAL  *)


OPERATOR - (TA, TB: TAYLOR) RES : TAYLOR;                           (*  T - T  *)
    VAR U: TAYLOR; I: DIMTYPE;
    BEGIN
    IF TA.T <> TB.T
        THEN BEGIN
            WRITE   ('ERROR:  SUBTRACTION OF TAYLOR VARIABLES');
            WRITELN (' WITH UNEQUAL SCALE FACTORS');
            SVR (0)  END                (* RETURN TO OPERATING SYSTEM*)
        ELSE BEGIN
            U.LENGTH := TA.LENGTH;
            IF U.LENGTH > TB.LENGTH THEN U.LENGTH := TB.LENGTH;
            U.T := TA.T;
            FOR I := 1 TO U.LENGTH DO U.TC[I] := TA.TC[I] - TB.TC[I];
            RES := U
        END  (* ELSE *)
    END;  (*  TAYLOR - TAYLOR  *)


OPERATOR - (T: ITAYLOR) RES : ITAYLOR;                              (*  - IT  *)
    VAR U: ITAYLOR; I: DIMTYPE;
    BEGIN
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    FOR I := 1 TO U.LENGTH DO U.TC[I] := - T.TC[I];
    RES := U
    END;  (*  - ITAYLOR  *)
```

```
OPERATOR - (K: INTEGER; T: ITAYLOR) RES : ITAYLOR;            (*  K - IT  *)
    VAR U:ITAYLOR; I: DIMTYPE;
    BEGIN
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    U.TC[1] := K - T.TC[1];
    FOR I:= 2 TO U.LENGTH DO U.TC[I] := - T.TC[I];
    RES := U
    END;  (*  INTEGER - ITAYLOR  *)

OPERATOR - (T: ITAYLOR; K: INTEGER) RES : ITAYLOR;            (*  IT - K  *)
    VAR U: ITAYLOR;
    BEGIN
    U := T;
    U.TC[1] := T.TC[1] - K;
    RES := U
    END;  (*  ITAYLOR - INTEGER  *)

OPERATOR - (K: INTERVAL; T: ITAYLOR) RES : ITAYLOR;           (*  I - IT  *)
    VAR U:ITAYLOR; I: DIMTYPE;
    BEGIN
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    U.TC[1] := K - T.TC[1];
    FOR I:= 2 TO U.LENGTH DO U.TC[I] := - T.TC[I];
    RES := U
    END;  (*  INTERVAL - ITAYLOR  *)

OPERATOR - (T: ITAYLOR; K: INTERVAL) RES : ITAYLOR;           (*  IT - I  *)
    VAR U: ITAYLOR;
    BEGIN
    U := T;
    U.TC[1] := T.TC[1] - K;
    RES := U
    END;  (*  ITAYLOR - INTERVAL  *)

OPERATOR - (TA, TB: ITAYLOR) RES : ITAYLOR;                   (*  IT - IT  *)
    VAR U: ITAYLOR; I: DIMTYPE;
    BEGIN
    IF TA.T <> TB.T
        THEN BEGIN
            WRITE   ('ERROR:  SUBTRACTION OF ITAYLOR VARIABLES');
            WRITELN (' WITH UNEQUAL SCALE FACTORS');
            SVR (0)  END              (* RETURN TO OPERATING SYSTEM*)
        ELSE BEGIN
            U.LENGTH := TA.LENGTH;
            IF U.LENGTH > TB.LENGTH THEN U.LENGTH := TB.LENGTH;
            U.T := TA.T;
            FOR I := 1 TO U.LENGTH DO U.TC[I] := TA.TC[I] - TB.TC[I];
            RES := U
        END  (* ELSE *)
    END;  (*  ITAYLOR - ITAYLOR  *)


(*  END OF RIT_ADD.LIB       >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> *)
```

C.2. Real and Interval Taylor Multiplication Operators, including TSQR and ITSQR.

```
(*  RIT_MUL.LIB  -  REAL AND INTERVAL TAYLOR MULTIPLY      <<<<<<<<<<<<<

Contents:

    K * T       OPERATOR * (K: INTEGER; T: TAYLOR) RES : TAYLOR;
    T * K       OPERATOR * (T: TAYLOR; K: INTEGER) RES : TAYLOR;
    R * T       OPERATOR * (R: REAL; T: TAYLOR) RES : TAYLOR;
    T * R       OPERATOR * (T: TAYLOR; R: REAL) RES : TAYLOR;
    T * T       OPERATOR * (TA, TB: TAYLOR) RES : TAYLOR;
    TSQR(T)     FUNCTION TSQR (T: TAYLOR) : TAYLOR;
    K * IT      OPERATOR * (K: INTEGER; T: ITAYLOR) RES : ITAYLOR;
    IT * K      OPERATOR * (T: ITAYLOR; K: INTEGER) RES : ITAYLOR;
    I * IT      OPERATOR * (K: INTERVAL; T: ITAYLOR) RES : ITAYLOR;
    IT * I      OPERATOR * (T: ITAYLOR; K: INTERVAL) RES : ITAYLOR;
    IT * IT     OPERATOR * (TA, TB: ITAYLOR) RES : ITAYLOR;
    ITSQR(IT)   FUNCTION ITSQR (T: ITAYLOR) : ITAYLOR;

================================================================== *)

OPERATOR * (K: INTEGER; T: TAYLOR) RES : TAYLOR;                (*  K * T  *)
    VAR I:DIMTYPE; U: TAYLOR;
    BEGIN
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    FOR I := 1 TO U.LENGTH DO U.TC[I] := K * T.TC[I];
    RES := U
    END;  (*  INTEGER * TAYLOR  *)


OPERATOR * (T: TAYLOR; K: INTEGER) RES : TAYLOR;                (*  T * K  *)
    VAR I:DIMTYPE; U: TAYLOR;
    BEGIN
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    FOR I := 1 TO U.LENGTH DO U.TC[I] := T.TC[I] * K;
    RES := U
    END;  (*  TAYLOR * INTEGER  *)


OPERATOR * (R: REAL; T: TAYLOR) RES : TAYLOR;                   (*  R * T  *)
    VAR I: DIMTYPE; U: TAYLOR;
    BEGIN
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    FOR I := 1 TO U.LENGTH DO U.TC[I] := R * T.TC[I];
    RES := U
    END;  (*  REAL * TAYLOR  *)
```

```
OPERATOR * (T: TAYLOR; R: REAL) RES : TAYLOR;                    (*  T * R  *)
    VAR I: DIMTYPE; U: TAYLOR;

    BEGIN
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    FOR I := 1 TO U.LENGTH DO U.TC[I] := T.TC[I] * R;

    RES := U
    END;  (*  TAYLOR * REAL  *)



OPERATOR * (TA, TB: TAYLOR) RES : TAYLOR;                        (*  T * T  *)
    (*  REQUIRES:  VRNULL, SCALP  *)
    VAR I, J, K : DIMTYPE;
        X, Y    : RVECTOR;
        U       : TAYLOR;

    BEGIN
    IF TA.T <> TB.T

        THEN BEGIN
            WRITE   ('ERROR:  MULTIPLICATION OF TAYLOR VARIABLES');
            WRITELN (' WITH UNEQUAL SCALE FACTORS');
            SVR (0)  END                (* RETURN TO OPERATING SYSTEM *)

        ELSE BEGIN
            U.LENGTH := TA.LENGTH;
            IF U.LENGTH > TB.LENGTH THEN U.LENGTH := TB.LENGTH;
            U.T := TA.T;
            X := VRNULL;  Y := VRNULL;
            FOR I := 1 TO U.LENGTH DO BEGIN
                X[I] := TA.TC[I];
                FOR J := 1 TO I DO BEGIN
                    K := I - J + 1;
                    Y[J]:= TB.TC[K]
                    END;  (* FOR J LOOP *)
                U.TC[I] := SCALP (X, Y, 0)
                END  (* FOR I LOOP *)
        END;   (* ELSE *)

        RES := U
    END;  (*  TAYLOR * TAYLOR  *)
```

```
FUNCTION TSQR (T: TAYLOR) : TAYLOR;                                    (*  TSQR(T)  *)
    (*  Requires:  VRNULL, SCALP, SQR  *)
    VAR I, J, K, HALF: DIMTYPE;
        X, Y: RVECTOR;
        U   : TAYLOR;

    BEGIN
    X := VRNULL;  Y := VRNULL;
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    U.TC := VRNULL;

    U.TC[1] := SQR (T.TC[1]);
    X[1] := T.TC[1];

    FOR K := 2 TO U.LENGTH DO
        BEGIN
        X[K] := T.TC[K];
        HALF := K DIV 2;
        FOR J := 1 TO HALF DO
            BEGIN
            I := K - J + 1;
            Y[J] := T.TC[I]
            END;  (* FOR J *)
        U.TC[K] := 2.0 * SCALP ( X, Y, 0);
        IF K MOD 2 = 1 THEN
            BEGIN
            HALF := HALF + 1;
            U.TC[K]:= U.TC[K] + SQR (T.TC[HALF])
            END  (* IF *)
        END;  (* FOR K *)
    TSQR := U
    END;  (* FUNCTION TSQR (TAYLOR)  *)


OPERATOR * (K: INTEGER; T: ITAYLOR) RES : ITAYLOR;                     (*  K * IT  *)
    VAR I:DIMTYPE; U: ITAYLOR;
    BEGIN
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    FOR I := 1 TO U.LENGTH DO U.TC[I] := K * T.TC[I];
    RES := U
    END;  (*  INTEGER * ITAYLOR  *)


OPERATOR * (T: ITAYLOR; K: INTEGER) RES : ITAYLOR;                     (*  IT * K  *)
    VAR I:DIMTYPE; U: ITAYLOR;
    BEGIN
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    FOR I := 1 TO U.LENGTH DO U.TC[I] := T.TC[I] * K;
    RES := U
    END;  (*  ITAYLOR * INTEGER  *)
```

```
OPERATOR * (K: INTERVAL; T: ITAYLOR) RES : ITAYLOR;              (*  I * IT  *)
    VAR I:DIMTYPE; U: ITAYLOR;
    BEGIN
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    FOR I := 1 TO U.LENGTH DO U.TC[I] := K * T.TC[I];
    RES := U
    END;  (*  INTERVAL * ITAYLOR  *)


OPERATOR * (T: ITAYLOR; K: INTERVAL) RES : ITAYLOR;              (*  IT * I  *)
    VAR I:DIMTYPE; U: ITAYLOR;
    BEGIN
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    FOR I := 1 TO U.LENGTH DO U.TC[I] := T.TC[I] * K;
    RES := U
    END;  (*  ITAYLOR * INTERVAL  *)


OPERATOR * (TA, TB: ITAYLOR) RES : ITAYLOR;                      (*  IT * IT  *)
    (*  Requires:  IVRNULL, ISCALP  *)
    VAR I, J, K : DIMTYPE;
        X, Y    : IVECTOR;
        U       : ITAYLOR;
    BEGIN
    IF TA.T <> TB.T
        THEN BEGIN
            WRITE   ('ERROR:  MULTIPLICATION OF ITAYLOR VARIABLES');
            WRITELN (' WITH UNEQUAL SCALE FACTORS');
            SVR (0)  END                (* RETURN TO OPERATING SYSTEM *)
        ELSE BEGIN
            X := IVRNULL;  Y := IVRNULL;
            U.LENGTH := TA.LENGTH;
            IF U.LENGTH > TB.LENGTH THEN U.LENGTH := TB.LENGTH;
            U.T := TA.T;
            FOR I := 1 TO U.LENGTH DO BEGIN
                X[I] := TA.TC[I];
                FOR J := 1 TO I DO BEGIN
                    K := I - J + 1;
                    Y[J]:= TB.TC[K]
                    END;  (* FOR J LOOP *)
                U.TC[I] := ISCALP (X, Y, I)
                END  (* FOR I LOOP *)
        END;  (* ELSE *)

        RES := U
    END;  (*  ITAYLOR * ITAYLOR  *)
```

```
FUNCTION ITSQR (T: ITAYLOR) : ITAYLOR;                          (*  ITSQR(IT)  *)
    (*  Requires:  IVRNULL, ISCALP, ISQR  *)
    VAR I, J, K, HALF: DIMTYPE;
        X, Y: IVECTOR;
        U   : ITAYLOR;

    BEGIN
    X := IVRNULL;  Y := IVRNULL;
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    U.TC := IVRNULL;

    U.TC[1] := ISQR (T.TC[1]);
    X[1] := T.TC[1];

    FOR K := 2 TO U.LENGTH DO
        BEGIN
        X[K] := T.TC[K];
        HALF := K DIV 2;
        FOR J := 1 TO HALF DO
            BEGIN
            I := K - J + 1;
            Y[J] := T.TC[I]
            END;  (* FOR J *)
        U.TC[K] := 2 * ISCALP ( X, Y, HALF);
        IF K MOD 2 = 1 THEN
            BEGIN
            HALF := HALF + 1;
            U.TC[K]:= U.TC[K] + ISQR (T.TC[HALF])
            END  (* IF *)
        END;  (* FOR K *)
    ITSQR := U
    END;  (* FUNCTION ITSQR (ITAYLOR)  *)

(*  END OF RIT_MUL.LIB       >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> *)
```

C.3. Real and Interval Taylor Division Operators.

```
(*  RIT_DIV.LIB  -  REAL AND INTERVAL TAYLOR DIVIDE      <<<<<<<<<<<<<<<

Contents:

    T / K       OPERATOR / (K: INTEGER; T: TAYLOR) RES : TAYLOR;
    K / T       OPERATOR / (T: TAYLOR; K: INTEGER) RES : TAYLOR;
    R / T       OPERATOR / (R: REAL; T: TAYLOR) RES : TAYLOR;
    T / R       OPERATOR / (T: TAYLOR; R: REAL) RES : TAYLOR;
    T / T       OPERATOR / (TA, TB: TAYLOR) RES : TAYLOR;
    K / IT      OPERATOR / (K: INTEGER; T: ITAYLOR) RES : ITAYLOR;
    IT / K      OPERATOR / (T: ITAYLOR; K: INTEGER) RES : ITAYLOR;
    I / IT      OPERATOR / (K: INTERVAL; T: ITAYLOR) RES : ITAYLOR;
    IT / I      OPERATOR / (T: ITAYLOR; K: INTERVAL) RES : ITAYLOR;
    IT / IT     OPERATOR / (TA, TB: ITAYLOR) RES : ITAYLOR;

================================================================== *)


OPERATOR / (T: TAYLOR; K: INTEGER) RES : TAYLOR;                (*  T / K  *)
    VAR U: TAYLOR; I: DIMTYPE;

    BEGIN
    U.LENGTH := T.LENGTH;
    U.T := T.T;

    IF K = 0
        THEN BEGIN
            WRITELN ('ERROR:  DIVISION OF TAYLOR VARIABLE BY ZERO');
            SVR (0)  END                    (* RETURN TO OPERATING SYSTEM *)

        ELSE BEGIN
            FOR I := 1 TO U.LENGTH DO U.TC[I] := T.TC[I] / K
        END;  (* ELSE *)

        RES := U
    END;  (*  TAYLOR / INTEGER  *)
```

```
OPERATOR / (K: INTEGER; T: TAYLOR) RES : TAYLOR;                (*  K / T  *)
    (*  Requires:  VRNULL  *)
    VAR U, W: TAYLOR;
        I, J, IP1, REV: DIMTYPE;
        RATIO: REAL;

    BEGIN
    U.LENGTH := T.LENGTH;
    U.T:= T.T;

    U.TC := VRNULL; W.TC := VRNULL;      (* ZERO RESULT AND WORK VECTORS *)

    IF T.TC[1] <> 0.0
        THEN BEGIN                       (* NORMAL DIVIDE *)
            RATIO := 1.0 / T.TC[1];
            U.TC[1] := K * RATIO;
            FOR I := 2 TO U.LENGTH DO
                BEGIN
                FOR J := 1 TO (I-1) DO
                    BEGIN
                    REV := I - J + 1;
                    W.TC[J] := T.TC[REV]
                    END;  (* FOR J *)
                U.TC[I] := - SCALP (U.TC, W.TC, 0) * RATIO
                END  (* FOR I *)
            END  (* THEN *)
    ELSE IF K <> 0
        THEN BEGIN
            WRITE   ('ERROR:  DIVISION BY TAYLOR VARIABLE');
            WRITELN (' EQUAL TO ZERO');
            SVR (0)              (* RETURN TO OPERATING SYSTEM *)
            END  (* THEN *)
    ELSE BEGIN                   (* APPLY L'HOSPITAL'S RULE *)
        IF U.LENGTH = 1
            THEN BEGIN
                WRITE   ('ERROR:  DIVISION WITH TAYLOR VARIABLE');
                WRITELN (' OF LENGTH ONE.');
                SVR (0)                      (* RETURN TO OPERATING SYSTEM *)
                END;  (* IF *)

        U.LENGTH := T.LENGTH - 1;
        FOR I :=1 TO U.LENGTH DO
            BEGIN
            IP1 := I + 1;
            U.TC[I] := T.TC[IP1]
            END;  (* FOR I *)
        U := 0 / U               (* THIS IS A RECURSIVE CALL *)
        END;  (* IF *)
    RES := U
    END;  (*  INTEGER / TAYLOR  *)
```

```
OPERATOR / (R: REAL; T: TAYLOR) RES : TAYLOR;                    (*  R / T  *)
    (*  Requires:  VRNULL  *)
    VAR U, W: TAYLOR;
        I, J, IP1, REV: DIMTYPE;
        RATIO: REAL;

    BEGIN
    U.LENGTH := T.LENGTH;
    U.T:= T.T;

    U.TC := VRNULL; W.TC := VRNULL;      (* ZERO RESULT AND WORK VECTORS *)

    IF T.TC[1] <> 0.0
        THEN BEGIN                       (* NORMAL DIVIDE *)
            RATIO := 1.0 / T.TC[1];
            U.TC[1] := R * RATIO;
            FOR I := 2 TO U.LENGTH DO
                BEGIN
                FOR J := 1 TO (I-1) DO
                    BEGIN
                    REV := I - J + 1;
                    W.TC[J] := T.TC[REV]
                    END;  (* FOR J *)
                U.TC[I] := - SCALP (U.TC, W.TC, 0) * RATIO
                END  (* FOR I *)
            END  (* THEN *)
    ELSE IF R <> 0
        THEN BEGIN
            WRITE   ('ERROR:  DIVISION BY TAYLOR VARIABLE');
            WRITELN (' EQUAL TO ZERO');
            SVR (0)              (* RETURN TO OPERATING SYSTEM *)
            END  (* THEN *)
    ELSE BEGIN                   (* APPLY L'HOSPITAL'S RULE *)
        IF U.LENGTH = 1
            THEN BEGIN
                WRITE   ('ERROR:  DIVISION WITH TAYLOR VARIABLE');
                WRITELN (' OF LENGTH ONE.');
                SVR (0)                      (* RETURN TO OPERATING SYSTEM *)
                END;  (* IF *)

        U.LENGTH := T.LENGTH - 1;
        FOR I :=1 TO U.LENGTH DO
            BEGIN
            IP1 := I + 1;
            U.TC[I] := T.TC[IP1]
            END;  (* FOR *)
        U := 0.0 / U             (* THIS IS A RECURSIVE CALL *)
        END;  (* IF *)
    RES := U
    END;  (*  REAL / TAYLOR  *)
```

```
OPERATOR / (T: TAYLOR; R: REAL) RES : TAYLOR;                    (*  T / R  *)
    VAR U: TAYLOR; I: DIMTYPE;
    BEGIN
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    IF R = 0
        THEN BEGIN
            WRITELN ('ERROR:  DIVISION OF TAYLOR VARIABLE BY ZERO');
            SVR (0)  END                (* RETURN TO OPERATING SYSTEM *)
        ELSE BEGIN
            FOR I := 1 TO U.LENGTH DO U.TC[I] := T.TC[I] / R
        END;  (* ELSE *)
        RES := U
    END;  (*  TAYLOR / REAL  *)



OPERATOR / (TA, TB: TAYLOR) RES : TAYLOR;                        (*  T / T  *)
    (*  Requires:  VRNULL, SCALP  *)

(*  THIS OPERATOR ATTEMPTS TO HANDLE  0 / 0  SITUATIONS USING L'HOSPITAL'S RULE

                TA.TC[1] =      0.0             <> 0.0
                                ---------------------------------
                                |   CASE 1          CASE 2
                    = 0.0       | TA' / TB'         ERROR
        TB.TC[1]                |
                    <> 0.0      |   CASE 3          CASE 3
                                |   TA / TB         TA / TB

   CASE 1 IS HANDLED BY RECURSION UNLESS THE SERIES LENGTH IS ONLY 1  *)

    VAR U, W : TAYLOR;
        I, J, IP1, REV: DIMTYPE;
        RATIO: REAL;

    BEGIN
    IF TA.T <> TB.T             (* UNEQUAL SCALE FACTORS *)
        THEN BEGIN
            WRITE   ('ERROR:  DIVISION OF TAYLOR VARIABLES');
            WRITELN (' WITH UNEQUAL SCALE FACTORS');
            SVR (0)             (* RETURN TO OPERATING SYSTEM *)
        END;  (* IF *)

    U.LENGTH := TA.LENGTH;
    IF U.LENGTH > TB.LENGTH THEN U.LENGTH := TB.LENGTH;
    U.T := TA.T;

    U.TC := VRNULL; W.TC := VRNULL;     (* ZERO RESULT AND WORK VECTORS *)
```

```
    IF TB.TC[1] <> 0.0
        THEN BEGIN                  (* CASE 3: USUAL DIVIDE *)
            RATIO := 1.0 / TB.TC[1];
            U.TC[1] := TA.TC[1] * RATIO;
            FOR I := 2 TO U.LENGTH DO
                BEGIN
                FOR J := 1 TO (I-1) DO
                    BEGIN
                    REV := I - J + 1;
                    W.TC[J] := TB.TC[REV]
                    END;  (* FOR J *)
                U.TC[I] := (TA.TC[I] - SCALP (U.TC, W.TC, 0)) * RATIO
                END  (* FOR I *)
            END  (* THEN *)
    ELSE IF (TA.TC[1] <> 0.0) OR (U.LENGTH = 1)
        THEN BEGIN                  (* CASE 1: BOTH SERIES IDENTICALLY ZERO,
                                       OR CASE 2: TA / 0 *)
            WRITE   ('ERROR:  DIVISION BY TAYLOR VARIABLE');
            WRITELN (' EQUAL TO ZERO');
            SVR (0)                 (* RETURN TO OPERATING SYSTEM *)
            END  (* THEN *)
    ELSE BEGIN                      (* APPLY L'HOSPITAL'S RULE *)
        IF U.LENGTH = 1
            THEN BEGIN
                WRITE   ('ERROR:  DIVISION WITH TAYLOR VARIABLE');
                WRITELN (' OF LENGTH ONE.');
                SVR (0)                         (* RETURN TO OPERATING SYSTEM *)
                END;  (* IF *)

        U.LENGTH := TA.LENGTH - 1;
        W.LENGTH := U.LENGTH;
        W.T := U.T;
        FOR I := 1 TO U.LENGTH DO
            BEGIN
            IP1 := I + 1;
            U.TC[I] := TA.TC[IP1];
            W.TC[I] := TB.TC[IP1]
            END;  (* FOR I *)
        U := U / W                  (* THIS IS A RECURSIVE CALL *)
        END;  (* IF *)
    RES := U
    END;  (*  TAYLOR / TAYLOR  *)


OPERATOR / (K: INTEGER; T: ITAYLOR) RES : ITAYLOR;              (*  K / IT  *)
    (*  Requires:  IVRNULL, ISCALP  *)
    VAR U, W: ITAYLOR;
        I, J, IP1, REV: DIMTYPE;
        RATIO: INTERVAL;

    BEGIN
    U.LENGTH := T.LENGTH;
    U.T:= T.T;

    U.TC := IVRNULL; W.TC := IVRNULL;   (* ZERO RESULT AND WORK VECTORS *)
```

```
    IF NOT (0.0 IN T.TC[1])
        THEN BEGIN                          (* NORMAL DIVIDE *)
            RATIO := 1 / T.TC[1];
            U.TC[1] := K * RATIO;
            FOR I := 2 TO U.LENGTH DO
                BEGIN
                FOR J := 1 TO (I-1) DO
                    BEGIN
                    REV := I - J + 1;
                    W.TC[J] := T.TC[REV]
                    END;  (* FOR J *)
                U.TC[I] := - ISCALP (U.TC, W.TC, I) * RATIO
                END  (* FOR I *)
            END  (* THEN *)
    ELSE IF (K <> 0) OR (T.TC[1] <> INTPT (0.0))
        THEN BEGIN
            WRITE   ('ERROR:  DIVISION BY ITAYLOR VARIABLE');
            WRITELN (' EQUAL TO ZERO');
            SVR (0)              (* RETURN TO OPERATING SYSTEM *)
            END  (* THEN *)
    ELSE BEGIN                   (* APPLY L'HOSPITAL'S RULE *)
        IF U.LENGTH = 1
            THEN BEGIN
                WRITE   ('ERROR:  DIVISION WITH ITAYLOR VARIABLE');
                WRITELN (' OF LENGTH ONE.');
                SVR (0)                      (* RETURN TO OPERATING SYSTEM *)
                END;  (* IF *)

        U.LENGTH := T.LENGTH - 1;
        FOR I :=1 TO U.LENGTH DO
            BEGIN
            IP1 := I + 1;
            U.TC[I] := T.TC[IP1]
            END;  (* FOR I *)
        U := 0 / U               (* THIS IS A RECURSIVE CALL *)
        END;  (* IF *)
    RES := U
    END;  (*  INTEGER / ITAYLOR  *)


OPERATOR / (T: ITAYLOR; K: INTEGER) RES : ITAYLOR;              (*  IT / K  *)
    VAR U: ITAYLOR; I: DIMTYPE;
    BEGIN
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    IF K = 0
        THEN BEGIN
            WRITELN ('ERROR:  DIVISION OF ITAYLOR VARIABLE BY ZERO');
            SVR (0)  END                (* RETURN TO OPERATING SYSTEM *)
    ELSE FOR I := 1 TO U.LENGTH DO U.TC[I] := T.TC[I] / K;

    RES := U
    END;  (*  ITAYLOR / INTEGER  *)
```

```
OPERATOR / (K: INTERVAL; T: ITAYLOR) RES : ITAYLOR;              (*  I / IT  *)
    (*  Requires:  IVRNULL, ISCALP  *)
    VAR U, W: ITAYLOR;
        I, J, IP1, REV: DINTYPE;
        RATIO: INTERVAL;

    BEGIN
    U.LENGTH := T.LENGTH;
    U.T:= T.T;

    U.TC := IVRNULL; W.TC := IVRNULL;   (* ZERO RESULT AND WORK VECTORS *)

    IF NOT (0.0 IN T.TC[1])
        THEN BEGIN                          (* NORMAL DIVIDE *)
            RATIO := 1 / T.TC[1];
            U.TC[1] := K * RATIO;
            FOR I := 2 TO U.LENGTH DO
                BEGIN
                FOR J := 1 TO (I-1) DO
                    BEGIN
                    REV := I - J + 1;
                    W.TC[J] := T.TC[REV]
                    END;  (* FOR J *)
                U.TC[I] := - ISCALP (U.TC, W.TC, I) * RATIO
                END  (* FOR I *)
            END  (* THEN *)
    ELSE IF (K <> INTPT (0.0)) OR (T.TC[1] <> INTPT (0.0))
        THEN BEGIN
            WRITE   ('ERROR:  DIVISION BY ITAYLOR VARIABLE');
            WRITELN (' EQUAL TO ZERO');
            SVR (0)             (* RETURN TO OPERATING SYSTEM *)
            END  (* THEN *)
    ELSE BEGIN                  (* APPLY L'HOSPITAL'S RULE *)
        IF U.LENGTH = 1
            THEN BEGIN
                WRITE   ('ERROR:  DIVISION WITH ITAYLOR VARIABLE');
                WRITELN (' OF LENGTH ONE.');
                SVR (0)                     (* RETURN TO OPERATING SYSTEM *)
                END;  (* IF *)

        U.LENGTH := T.LENGTH - 1;
        FOR I :=1 TO U.LENGTH DO
            BEGIN
            IP1 := I + 1;
            U.TC[I] := T.TC[IP1]
            END;  (* FOR *)
        U := 0 / U              (* THIS IS A RECURSIVE CALL *)
        END;  (* IF *)
    RES := U
    END;  (*  INTERVAL / ITAYLOR  *)
```

```
OPERATOR / (T: ITAYLOR; K: INTERVAL) RES : ITAYLOR;              (*  IT / I  *)
    VAR U: ITAYLOR; I: DIMTYPE;
    BEGIN
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    IF 0.0 IN K
        THEN BEGIN
            WRITELN ('ERROR:  DIVISION OF ITAYLOR VARIABLE BY ZERO');
            SVR (0)  END                  (* RETURN TO OPERATING SYSTEM *)
    ELSE FOR I := 1 TO U.LENGTH DO U.TC[I] := T.TC[I] / K;

    RES := U
    END;  (*  ITAYLOR / INTERVAL  *)



OPERATOR / (TA, TB: ITAYLOR) RES : ITAYLOR;                      (*  IT / IT  *)
    (*  Requires:  IVRNULL, ISCALP  *)

(*  THIS OPERATOR ATTEMPTS TO HANDLE  0 / 0  SITUATIONS USING L'HOSPITAL'S RULE

                TA.TC[1] =      0.0             <> 0.0
                            -------------------------------
                           |    CASE 1          CASE 2
                    = 0.0  |  TA' / TB'         ERROR
        TB.TC[1]           |
                   <> 0.0  |    CASE 3          CASE 3
                           |    TA / TB         TA / TB

   CASE 1 IS HANDLED BY RECURSION UNLESS THE SERIES LENGTH IS ONLY 1  *)

    VAR U, W : ITAYLOR;
        I, J, IP1, REV: DIMTYPE;
        RATIO: INTERVAL;

    BEGIN
    IF TA.T <> TB.T               (* UNEQUAL SCALE FACTORS *)
        THEN BEGIN
            WRITE   ('ERROR:  DIVISION OF ITAYLOR VARIABLES');
            WRITELN (' WITH UNEQUAL SCALE FACTORS');
            SVR (0)               (* RETURN TO OPERATING SYSTEM *)
        END;  (* IF *)

    U.LENGTH := TA.LENGTH;
    IF U.LENGTH > TB.LENGTH THEN U.LENGTH := TB.LENGTH;
    U.T := TA.T;

    U.TC := IVRNULL; W.TC := IVRNULL;   (* ZERO RESULT AND WORK VECTORS *)
```

```
    IF NOT (0.0 IN TB.TC[1])
        THEN BEGIN                  (* CASE 3: USUAL DIVIDE *)
            RATIO := 1 / TB.TC[1];
            U.TC[1] := TA.TC[1] * RATIO;
            FOR I := 2 TO U.LENGTH DO
                BEGIN
                FOR J := 1 TO (I-1) DO
                    BEGIN
                    REV := I - J + 1;
                    W.TC[J] := TB.TC[REV]
                    END;  (* FOR J *)
                U.TC[I] := (TA.TC[I] - ISCALP (U.TC, W.TC, I)) * RATIO
                END  (* FOR I *)
            END  (* THEN *)
    ELSE IF (U.LENGTH = 1) OR
            (NOT (0.0 IN TA.TC[1])) OR
            (TA.TC[1] <> INTPT (0.0)) OR
            (TB.TC[1] <> INTPT (0.0))
        THEN BEGIN                  (* CASE 1: BOTH SERIES IDENTICALLY ZERO,
                                       OR CASE 2: TA / 0 *)
            WRITE   ('ERROR:  DIVISION BY ITAYLOR VARIABLE');
            WRITELN (' EQUAL TO ZERO');
            SVR (0)                 (* RETURN TO OPERATING SYSTEM *)
            END  (* THEN *)
    ELSE BEGIN                      (* APPLY L'HOSPITAL'S RULE *)
                                    (* ONLY IF INF = SUP = 0 *)
        IF U.LENGTH = 1
            THEN BEGIN
                WRITE   ('ERROR:  DIVISION WITH ITAYLOR VARIABLE');
                WRITELN (' OF LENGTH ONE.');
                SVR (0)                         (* RETURN TO OPERATING SYSTEM *)
                END;  (* IF *)

        U.LENGTH := TA.LENGTH - 1;
        W.LENGTH := U.LENGTH;
        W.T := U.T;
        FOR I := 1 TO U.LENGTH DO
            BEGIN
            IP1 := I + 1;
            U.TC[I] := TA.TC[IP1];
            W.TC[I] := TB.TC[IP1]
            END;  (* FOR I *)
        U := U / W                  (* THIS IS A RECURSIVE CALL *)
        END;  (* IF *)
    RES := U
    END;  (*  ITAYLOR / ITAYLOR  *)


(*  END OF RIT_DIV.LIB       >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> *)
```

C.4. Real Taylor Power Operators and Functions.

```
(*  RT_POW.LIB   -  REAL TAYLOR POWERS          <<<<<<<<<<<<<<<<<<<<<<<<<<<

Contents:

    TSQR(T)      FUNCTION TSQR (T: TAYLOR) : TAYLOR;
    TSQRT(T)     FUNCTION TSQRT (T: TAYLOR) : TAYLOR;
    TEXP(T)      FUNCTION TEXP (T: TAYLOR) : TAYLOR;

    K ** K       OPERATOR ** (BASE, EXPONENT : INTEGER) RES : INTEGER;
    R ** K       OPERATOR ** (BASE: REAL; EXPONENT: INTEGER) RES : REAL;
    R ** R       OPERATOR ** (BASE, EXPONENT: REAL) RES : REAL;
    K ** R       OPERATOR ** (BASE: INTEGER; EXPONENT: REAL) RES : REAL;
    T ** K       OPERATOR ** (BASE: TAYLOR; EXPONENT: INTEGER) RES : TAYLOR;
    T ** R       OPERATOR ** (BASE: TAYLOR; EXPONENT: REAL) RES : TAYLOR;
    R ** T       OPERATOR ** (BASE: REAL; EXPONENT: TAYLOR) RES : TAYLOR;
    K ** T       OPERATOR ** (BASE: INTEGER; EXPONENT: TAYLOR) RES : TAYLOR;
    T ** T       OPERATOR ** (BASE, EXPONENT: TAYLOR) RES : TAYLOR;

================================================================== *)

FUNCTION TSQR (T: TAYLOR) : TAYLOR;                          (*  TSQR(T)  *)
    (*  Requires:  VRNULL, SCALP, SQR  *)
    VAR I, J, K, HALF: DIMTYPE;
        X, Y: RVECTOR;
        U   : TAYLOR;
    BEGIN
    X := VRNULL;  Y := VRNULL;
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    U.TC := VRNULL;
    U.TC[1] := SQR (T.TC[1]);
    X[1] := T.TC[1];
    FOR K := 2 TO U.LENGTH DO
        BEGIN
        X[K] := T.TC[K];
        HALF := K DIV 2;
        FOR J := 1 TO HALF DO
            BEGIN
            I := K - J + 1;
            Y[J] := T.TC[I]
            END;  (* FOR J *)
        U.TC[K] := 2.0 * SCALP ( X, Y, 0);
        IF K MOD 2 = 1 THEN
            BEGIN
            HALF := HALF + 1;
            U.TC[K]:= U.TC[K] + SQR (T.TC[HALF])
            END  (* IF *)
        END;  (* FOR K *)
    TSQR := U
    END;  (* FUNCTION TSQR (TAYLOR)  *)
```

```
FUNCTION TSQRT (T: TAYLOR) : TAYLOR;                          (*  TSQRT(T)  *)
    (*  Requires:  VRNULL, SQRT, SQR, SCALP  *)
    VAR I, K, INDEX, INDX2: DIMTYPE;
        RATIO: REAL;
        X, Y : RVECTOR;
        U    : TAYLOR;

    BEGIN
    IF (T.TC[1] < 0.0) OR
       ((T.TC[1] = 0.0) AND (T.LENGTH >= 2))
        THEN BEGIN
            WRITELN;
            WRITELN ('ERROR: SQUARE ROOT OF TAYLOR VARIABLE <= ZERO.');
            SVR (0)  END;                (* RETURN TO OPERATING SYSTEM *)
        (* ELSE *)

    U.LENGTH := T.LENGTH;
    U.T := T.T;
    X := VRNULL;  Y := VRNULL;
    U.TC := VRNULL;

    U.TC[1] := SQRT (T.TC[1]);           (* REAL *)

    IF U.LENGTH >= 2 THEN BEGIN
    RATIO := 1.0 / (2.0 * U.TC[1]);
    U.TC[2] := T.TC[2] * RATIO;

    IF U.LENGTH >= 3 THEN BEGIN
    U.TC[3] := (T.TC[3] - SQR (U.TC[2])) * RATIO;

    FOR K := 4 TO U.LENGTH DO
        BEGIN
        IF K MOD 2 = 0
            THEN BEGIN
                INDEX := K DIV 2;
                INDX2 := INDEX - 1;
                X[INDX2] := U.TC[INDEX]
                END  (* THEN *)
            ELSE BEGIN
                INDEX := (K + 1) DIV 2;
                U.TC[K] := SQR (U.TC[INDEX])
            END;  (* IF *)
        FOR I := 1 TO INDX2 DO
            BEGIN
            INDEX := K - I;
            Y[I] := U.TC[INDEX];
            END;  (* FOR I *)
        U.TC[K] := (T.TC[K] - U.TC[K] - 2.0 * SCALP (X, Y, 0)) * RATIO
    END   (* FOR K *)
    END   (* IF U.LENGTH >= 3 *)
    END;  (* IF U.LENGTH >= 2 *)

    TSQRT := U
    END;  (* FUNCTION TSQRT (TAYLOR)  *)
```

```
FUNCTION TEXP (T: TAYLOR) : TAYLOR;                          (*  TEXP(T)  *)
    (*  Requires:  VRNULL, SCALP  *)
    VAR J, K, INDEX: DIMTYPE;
        RATIO: REAL;
        X    : RVECTOR;
        U    : TAYLOR;

    BEGIN
    X        := VRNULL;
    U.LENGTH := T.LENGTH;
    U.T      := T.T;
    U.TC     := VRNULL;
    U.TC[1]  := EXP (T.TC[1]);

    FOR K := 2 TO U.LENGTH DO
        BEGIN
        RATIO := 1.0 / (K - 1);
        FOR J := 2 TO K DO
            BEGIN
            INDEX := K - J + 1;
            X[INDEX] := T.TC[J] * (J - 1) * RATIO
            END;  (* FOR J *)
        U.TC[K] := SCALP (U.TC, X, 0)
        END;  (* FOR K *)
    TEXP := U
    END;  (*  FUNCTION TEXP (TAYLOR)  *)


OPERATOR ** (BASE, EXPONENT : INTEGER) RES : INTEGER;        (*  K ** K  *)
    VAR U: INTEGER;

    BEGIN
    IF EXPONENT < 0
        THEN BEGIN
            WRITELN ('ERROR:  INTEGER ** NEGATIVE INTEGER');
            SVR (0)                    (* RETURN TO OPERATING SYSTEM *)
        END;  (* IF *)
    IF BASE = 0
        THEN IF EXPONENT = 0
                THEN BEGIN
                    WRITELN ('ERROR:  ZERO ** ZERO');
                    SVR (0)            (* RETURN TO OPERATING SYSTEM *)
                    END  (* THEN *)
                ELSE U := 0

    ELSE BEGIN                         (* NORMAL BRANCH *)
        U := 1;
        WHILE EXPONENT > 0 DO
            BEGIN
            IF EXPONENT MOD 2 = 1 THEN U := U * BASE;
            EXPONENT := EXPONENT DIV 2;
            IF EXPONENT > 0 THEN BASE := SQR (BASE)
            END  (* WHILE *)
        END;  (* ELSE *)
    RES := U
    END;  (*  INTEGER ** INTEGER  *)
```

```
OPERATOR ** (BASE: REAL; EXPONENT: INTEGER) RES : REAL;          (*  R ** K  *)
    VAR U: REAL;
        NEG_EXP: BOOLEAN;

    BEGIN
    IF BASE = 0.0
        THEN IF EXPONENT = 0
                THEN BEGIN
                    WRITELN;
                    WRITELN ('ERROR:  ZERO ** ZERO');
                    SVR (0)              (* RETURN TO OPERATING SYSTEM *)
                    END  (* THEN *)
                ELSE IF EXPONENT < 0
                        THEN BEGIN
                            WRITELN;
                            WRITELN ('ERROR: ZERO ** NEGATIVE');
                            SVR (0)      (* RETURN TO OPERATING SYSTEM *)
                        END  (* THEN *)
                ELSE U := 0              (* ZERO ** POSITIVE *)

    ELSE IF BASE = 1.0 THEN U := 1.0
    ELSE BEGIN                       (* BASE <> 0 OR 1.  USUAL BRANCH *)
        NEG_EXP := EXPONENT < 0;
        EXPONENT := ABS (EXPONENT);
        IF EXPONENT = 1      THEN U := BASE
        ELSE IF EXPONENT = 2 THEN U := SQR (BASE)
        ELSE BEGIN
            U := 1;
            WHILE EXPONENT > 0 DO          (* EXPONENT = 0 falls through *)
                BEGIN
                IF EXPONENT MOD 2 = 1 THEN U := U * BASE;
                EXPONENT := EXPONENT DIV 2;
                IF EXPONENT > 0 THEN BASE := SQR (BASE)
                END  (* WHILE *)
            END;  (* ELSE *)
        IF NEG_EXP THEN U := 1.0 / U
        END;  (* ELSE*)

    RES := U
    END;  (*  REAL ** INTEGER*)
```

```
OPERATOR ** (BASE, EXPONENT: REAL) RES : REAL;                    (*  R ** R  *)
    (*  Requires:  R**K, SQR, EXP, LN  *)
    VAR U: REAL;
        K: INTEGER;

    BEGIN
    IF BASE = 0.0
    THEN IF EXPONENT = 0.0
            THEN BEGIN
                WRITELN;
                WRITELN ('ERROR:  ZERO ** ZERO');
                SVR (0)                  (* RETURN TO OPERATING SYSTEM *)
                END  (* THEN *)
            ELSE IF EXPONENT < 0.0
                    THEN BEGIN
                        WRITELN;
                        WRITELN ('ERROR: ZERO ** NEGATIVE');
                        SVR (0)          (* RETURN TO OPERATING SYSTEM *)
                        END  (* THEN *)
            ELSE U := 0.0                (* ZERO ** POSITIVE *)

    ELSE IF BASE = 1.0 THEN U := 1.0

    ELSE BEGIN                   (* BASE <> 0 OR 1.  USUAL BRANCH *)
        K := TRUNC (EXPONENT);
        U := BASE ** K;          (* USE REAL ** INTEGER *)
        IF EXPONENT <> K
            THEN IF BASE < 0.0
                    THEN BEGIN
                        WRITELN;
                        WRITELN ('ERROR:  NEGATIVE ** REAL');
                        SVR (0) (* RETURN TO OPERATING SYSTEM *)
                        END  (* THEN *)

                            (* USUAL BRANCH
                              ASSERT:  BASE > 0, -1 < EXPONENT - K < 1 *)

                    ELSE U := U * EXP ((EXPONENT - K ) * LN (BASE))
        END;  (* ELSE *)
    RES := U
    END;  (*  REAL ** REAL  *)



OPERATOR ** (BASE: INTEGER; EXPONENT: REAL) RES : REAL;           (*  K ** R  *)
    (*  Requires:  R**R, R**K, SQR, EXP, LN  *)
    VAR U: REAL;
    BEGIN
    U := BASE;
    RES := U ** EXPONENT                 (* USE REAL ** REAL *)
    END;  (*  INTEGER ** REAL  *)
```

```
OPERATOR ** (BASE: TAYLOR; EXPONENT: INTEGER) RES : TAYLOR;      (*  T ** K  *)
    (*  Requires:  VRNULL, R**K, TSQR, T*T, SCALP, T_IDENT_ZERO(T)  *)

    (* DECISION TABLE for special cases:

        EXPONENT:          0          1          2          > 2        < 0
                      +----------+----------+----------+----------+----------+
                      | Box 1                                                 |
                      |                                                       |
        BASE.TC  = 0  | Undef.     = BASE     SQR (BASE)    == 0       Undef. |
                      +----------+----------+----------+----------+----------+
                      | Box 2    | Box 3    | Box 4    | Box 5               |
                      |          |          |          |                     |
       BASE.TC[1] = 0 |    1     | = BASE   | SQR (BASE) | By mult.   Undef. |
                      |          |          |          +---------------------+
                      |          |          |          | Box 6               |
                      |          |          |          |                     |
            [1] <> 0  |    1     | = BASE   | SQR (BASE) |  By  recurrence   |
                      +----------+----------+----------+---------------------+

                Box 6 is the usual method.                                  *)


    VAR U       : TAYLOR;
        V       : RVECTOR;
        RATIO   : REAL;
        K, J, REV       : DIMTYPE;

    BEGIN
    U.LENGTH := BASE.LENGTH;
    U.T      := BASE.T;
                                (* ZERO RESULT AND WORK VECTORS *)
    U.TC := VRNULL;  V := VRNULL;

    IF T_IDENT_ZERO (BASE)                              (* Decision table Box 1
*)
        THEN IF EXPONENT <= 0
                THEN BEGIN
                    WRITELN;
                    WRITELN ('ERROR:  ZERO ** ZERO');
                    SVR (0)                     (* RETURN TO OPERATING SYSTEM *)
                    END  (* THEN *)
                ELSE  (*  U.TC is already == 0  *)

    ELSE IF EXPONENT = 0                                (* Decision table Box 2 *)
        THEN U.TC[1] := 1.0

    ELSE IF EXPONENT = 1                                (* Decision table Box 3 *)
        THEN U := BASE

    ELSE IF EXPONENT = 2                                (* Decision table Box 4 *)
        THEN U := TSQR (BASE)
```

```
    ELSE IF BASE.TC[1] = 0                              (* Decision table Box 5 *)
        THEN IF EXPONENT < 0
                THEN BEGIN
                    WRITELN;
                    WRITELN ('ERROR:  ZERO TAYLOR VARIABLE ** NEGATIVE');
                    SVR (0)                        (* RETURN TO OPERATING SYSTEM *)
                    END  (* THEN *)
                ELSE BEGIN          (* Use repeated multiplications of series *)
                    U.TC[1] := 1.0;
                    WHILE EXPONENT > 0 DO
                        BEGIN
                        IF EXPONENT MOD 2 = 1 THEN U := U * BASE;
                        EXPONENT := EXPONENT DIV 2;
                        IF EXPONENT > 0 THEN BASE := TSQR (BASE)
                        END  (* WHILE *)
                END  (* ELSE *)
    ELSE BEGIN                          (* Decision table Box 6.  Usual branch *)
        U.TC[1]  := BASE.TC[1] ** EXPONENT;              (*  REAL ** INTEGER *)
        RATIO    := 1.0 / BASE.TC[1];
        IF U.LENGTH >= 2 THEN BEGIN
        U.TC[2]  := EXPONENT * U.TC[1] * BASE.TC[2] * RATIO;
        FOR K := 3 TO U.LENGTH DO
            BEGIN
            FOR J := 1 TO K - 1 DO
                BEGIN
                REV  := K - J + 1;
                V[J] := BASE.TC[REV] * (EXPONENT * (K - J) - J + 1);
                END;  (* FOR J *)
            U.TC[K] := RATIO * (EXPONENT * SCALP (U.TC, V, 0)
            END  (* FOR K *)
        END  (* IF U.LENGTH >= 2 *)
        END;  (* ELSE *)
    RES := U
    END;  (*  TAYLOR ** INTEGER  *)

                                                              (*  T ** R  *)
OPERATOR ** (BASE: TAYLOR; EXPONENT: REAL) RES : TAYLOR;
    (*  Requires:  VRNULL, T**K, R**R, R**K, T*T, TSQR, TSQRT, SCALP,
                T_IDENT_ZERO, T_IDENT_CONSTANT  *)

    (* DECISION TABLE for special cases:
```

EXPONENT:	0	1	2 or 1/2	Int. > 2	Not Int. or < 0
	Box 1	Box 2	Box 3	Box 4	Box 5
BASE.TC = 0	Undef.	= BASE	SQR or SQRT	== 0	Undef.
BASE.TC[1] <= 0	1	= BASE	SQR or SQRT	T ** K	Undef.
					Box 6
BASE.TC[1] > 0	1	= BASE	SQR or SQRT	T ** K	recur.

```
            Boxes 4 or 6 are the usual method.                        *)
```

```
VAR U        : TAYLOR;
    V        : EVECTOR;
    RATIO    : REAL;
    I        : INTEGER;
    K, J, REV        : DIMTYPE;

BEGIN
U.LENGTH := BASE.LENGTH;
U.T      := BASE.T;
                              (* ZERO RESULT AND WORK VECTORS *)
U.TC := VRNULL;  V := VRNULL;
IF EXPONENT = 0.0             (* Decision table Box 1 *)
    THEN IF T_IDENT_ZERO (BASE)
            THEN BEGIN
                WRITELN;
                WRITELN ('ERROR:  ZERO ** ZERO');
                SVR (0)                    (* RETURN TO OPERATING SYSTEM *)
                END  (* THEN *)
            ELSE U.TC[1] := 1.0
ELSE IF EXPONENT = 1.0        (* Decision table Box 2 *)
    THEN U := BASE
ELSE IF EXPONENT = 2.0        (* Decision table Box 3 *)
    THEN U := TSQR (BASE)
ELSE IF EXPONENT = 0.5        (* Decision table Box 3 *)
    THEN U := TSQRT (BASE)          (*  Error if BASE.TC[1] <= 0.0  *)
ELSE BEGIN
    I := TRUNC (EXPONENT);          (* Decision table Box 4 *)
    IF (EXPONENT = I) AND (I > 2)
        THEN U := BASE ** I         (* TAYLOR ** INTEGER *)
    ELSE IF BASE.TC[1] <= 0.0
        THEN BEGIN                  (* Decision table Box 5 *)
            WRITELN;
            WRITE   ('ERROR:  ZERO TAYLOR VARIABLE ** NEGATIVE');
            WRITELN (' OR NON-INTEGER');
            SVR (0)                       (* RETURN TO OPERATING SYSTEM *)
            END  (* THEN *)
    ELSE BEGIN                    (* Decision table Box 6.  Usual branch *)
        U.TC[1]   := BASE.TC[1] ** EXPONENT;            (* REAL ** REAL *)
        RATIO     := 1.0 / BASE.TC[1];
        IF U.LENGTH >= 2 THEN BEGIN
        U.TC[2]   := EXPONENT * U.TC[1] * BASE.TC[2] * RATIO;
        FOR K := 3 TO U.LENGTH DO
            BEGIN
            FOR J := 1 TO K - 1 DO
                BEGIN
                REV  := K - J + 1;
                V[J] := BASE.TC[REV] * (EXPONENT * (K - J) - J + 1);
                END;  (* FOR J *)
            U.TC[K] := RATIO * SCALP (U.TC, V, 0) / (K - 1)
            END  (* FOR K *)
        END  (* IF U.LENGTH >= 2 *)
        END  (* ELSE *)
    END;  (* ELSE *)
RES := U
END;  (*  TAYLOR ** REAL  *)
```

```
OPERATOR ** (BASE: REAL; EXPONENT: TAYLOR) RES : TAYLOR;        (*  R ** T  *)
    (*  Requires:  VRNULL, R**R, R**K, SQR, LN, SCALP  *)
    VAR J, K, INDEX: DINTYPE;
        RATIO, LOG_BASE: REAL;
        X    : RVECTOR;
        U    : TAYLOR;

    BEGIN
    X         := VRNULL;
    U.LENGTH  := EXPONENT.LENGTH;
    U.T       := EXPONENT.T;
    U.TC      := VRNULL;
    U.TC[1]   := BASE ** EXPONENT.TC[1];
                                  (*  REAL ** REAL  -   may generate errors  *)

    IF NOT T_IDENT_CONSTANT (EXPONENT) THEN BEGIN
        IF BASE <= 0.0
            THEN BEGIN
                WRITELN;
                WRITELN ('ERROR IN REAL ** TAYLOR:  BASE <= ZERO');
                SVR (0)                  (* RETURN TO OPERATING SYSTEM *)
                END  (* THEN *)
        ELSE BEGIN
            LOG_BASE := LN (BASE);
            FOR K := 2 TO U.LENGTH DO
                BEGIN
                RATIO := 1.0 / (K - 1);
                FOR J := 2 TO K DO
                    BEGIN
                    INDEX := K - J + 1;
                    X[INDEX] := EXPONENT.TC[J] * (J - 1) * RATIO
                    END;  (* FOR J *)
                U.TC[K] := LOG_BASE * SCALP (U.TC, X, 0)
                END  (* FOR K *)
            END  (* ELSE *)
        END;  (* IF *)

    RES := U
    END;  (*  REAL ** TAYLOR  *)



OPERATOR ** (BASE: INTEGER; EXPONENT: TAYLOR) RES : TAYLOR;     (*  K ** T  *)
    (*  Requires:  VRNULL, R**T, R**R, R**K, SQR, LN, SCALP  *)
    VAR U: REAL;
    BEGIN
    U := BASE;
    RES := U ** EXPONENT                 (*  USE REAL ** TAYLOR  *)
    END;  (*  INTEGER ** TAYLOR  *)
```

```
OPERATOR ** (BASE, EXPONENT: TAYLOR) RES : TAYLOR;            (*  T ** T  *)
    (*  Requires: T**R, R**T, T**K, T*T, TLN, TEXP,
                  VRMULL, SCALP, T_IDENT_CONSTANT  *)

    VAR U: TAYLOR;

    BEGIN
    IF T_IDENT_CONSTANT (EXPONENT)
        THEN U := BASE ** EXPONENT.TC[1]        (*  USE T ** R  *)

    ELSE IF T_IDENT_CONSTANT (BASE)
        THEN U := BASE.TC[1] ** EXPONENT        (*  USE R ** T  *)

    ELSE IF BASE.TC[1] <= 0.0
        THEN BEGIN
            WRITELN;
            WRITELN ('ERROR:  TAYLOR BASE = ZERO');
            SVR (0)                      (* RETURN TO OPERATING SYSTEM *)
            END  (* THEN *)

    ELSE BEGIN
        IF BASE.T <> EXPONENT.T
            THEN BEGIN
                WRITELN;
                WRITE   (ERROR:  ** OF TAYLOR VARIABLES WITH');
                WRITELN (' UNEQUAL SCALE FACTORS');
                SVR (0)                  (* RETURN TO OPERATING SYSTEM *)
                END;  (* IF *)

        U := EXPONENT * TLN (BASE);
        U := TEXP (U);
        END;  (* ELSE *)

    RES := U
    END;  (*  TAYLOR ** TAYLOR  *)


(*  END OF RIT_POW.LIB     >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> *)
```

C.5. Interval Taylor Power Operators and Functions.

```
(*  IT_POW.LIB   -  INTERVAL TAYLOR POWERS        <<<<<<<<<<<<<<<<<<<<<<<<

Contents:

    ITSQR(IT)    FUNCTION ITSQR (T: ITAYLOR) : ITAYLOR;
    ITSQRT(IT)   FUNCTION ITSQRT (T: ITAYLOR) : ITAYLOR;
    ITEXP(IT)    FUNCTION ITEXP (T: ITAYLOR) : ITAYLOR;

    I ** K       OPERATOR ** (BASE: INTERVAL; EXPONENT: INTEGER) RES : INTERVAL;
    K ** I       OPERATOR ** (BASE: INTEGER; EXPONENT: INTERVAL) RES : INTERVAL;
    I ** I       OPERATOR ** (BASE: INTERVAL; EXPONENT: INTERVAL) RES : INTERVAL;
    IT ** K      OPERATOR ** (BASE: ITAYLOR; EXPONENT: INTEGER) RES : ITAYLOR;
    IT ** I      OPERATOR ** (BASE: ITAYLOR; EXPONENT: INTERVAL) RES : ITAYLOR;
    K ** IT      OPERATOR ** (BASE: INTEGER; EXPONENT: ITAYLOR) RES : ITAYLOR;
    I ** IT      OPERATOR ** (BASE: INTERVAL; EXPONENT: ITAYLOR) RES : ITAYLOR;
    IT ** IT     OPERATOR ** (BASE: ITAYLOR; EXPONENT: ITAYLOR) RES : ITAYLOR;

================================================================================ *)

FUNCTION ITSQR (T: ITAYLOR) : ITAYLOR;                          (*  ITSQR(IT)  *)
    (*  Requires:  IVRNULL, ISCALP, IT_IDENT_CONSTANT(IT), ISQR  *)
    VAR I, J, K, HALF: DIMTYPE;
        X, Y: IVECTOR;
        U   : ITAYLOR;
    BEGIN
    X := IVRNULL;  Y := IVRNULL;
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    U.TC := IVRNULL;
    U.TC[1] := ISQR (T.TC[1]);
    IF NOT IT_IDENT_CONSTANT (T) THEN BEGIN
    X[1] := T.TC[1];
    FOR K := 2 TO U.LENGTH DO
        BEGIN
        X[K] := T.TC[K];
        HALF := K DIV 2;
        FOR J := 1 TO HALF DO
            BEGIN
            I := K - J + 1;
            Y[J] := T.TC[I]
            END;  (* FOR J *)
        U.TC[K] := 2 * ISCALP ( X, Y, HALF);
        IF K MOD 2 = 1 THEN
            BEGIN
            HALF := HALF + 1;
            U.TC[K]:= U.TC[K] + ISQR (T.TC[HALF])
            END  (* IF *)
        END  (* FOR K *)
    END;  (* IF NOT IT_IDENT_CONSTANT (T) *)
    ITSQR := U
    END;  (* FUNCTION ITSQR (ITAYLOR)  *)
```

```
FUNCTION ITSQRT (T: ITAYLOR) : ITAYLOR;                    (*  ITSQRT(IT)  *
    (*  Requires:  IVRNULL; ISQRT, IT_IDENT_CONSTANT(IT),
                   ISQR, ISCALP  *)
    VAR I, K, INDEX, INDX2: DIMTYPE;
        RATIO: INTERVAL;
        X, Y : IVECTOR;
        U    : ITAYLOR;

    BEGIN
    IF (IINF (T.TC[1]) < 0.0) OR
       ((IINF (T.TC[1]) = 0.0) AND (T.LENGTH >= 2))
        THEN BEGIN
            WRITELN;
            WRITELN ('ERROR: SQUARE ROOT OF ITAYLOR VARIABLE <= ZERO.');
            SVR (0)  END;                  (* RETURN TO OPERATING SYSTEM *)
        (* ELSE *)
    X := IVRNULL;  Y := IVRNULL;
    U.LENGTH := T.LENGTH;
    U.T      := T.T;
    U.TC     := IVRNULL;
    U.TC[1] := ISQRT (T.TC[1]);            (* INTERVAL *)

    IF NOT IT_IDENT_CONSTANT (T) THEN BEGIN
    RATIO := 1 / (2 * U.TC[1]);

    IF U.LENGTH >= 2 THEN BEGIN
    U.TC[2] := T.TC[2] * RATIO;

    IF U.LENGTH >= 3 THEN BEGIN
    U.TC[3] := (T.TC[3] - ISQR (U.TC[2])) * RATIO;

    FOR K := 4 TO U.LENGTH DO
        BEGIN
        IF K MOD 2 = 0
            THEN BEGIN
                INDEX := K DIV 2;
                INDX2 := INDEX - 1;
                X[INDX2] := U.TC[INDEX]
                END  (* THEN *)
            ELSE BEGIN
                INDEX := (K + 1) DIV 2;
                U.TC[K] := ISQR (U.TC[INDEX])
            END;  (* IF *)
        FOR I := 1 TO INDX2 DO
            BEGIN
            INDEX := K - I;
            Y[I] := U.TC[INDEX];
            END;  (* FOR I *)
        U.TC[K] := (U.TC[K] - 2 * ISCALP (X, Y, INDX2)) * RATIO
    END  (* FOR K *)
    END  (* IF U.LENGTH >= 3 *)
    END  (* IF U.LENGTH >= 2 *)
    END;  (* IF NOT IT_IDENT_CONSTANT (T) *)

    ITSQRT := U
    END;  (* FUNCTION ITSQRT (ITAYLOR)  *)
```

```
FUNCTION ITEXP (T: ITAYLOR) : ITAYLOR;                          (*  ITEXP(IT)  *)
    (*  Requires:  IVRNULL, IT_IDENT_CONSTANT(IT), ISCALP  *)
    VAR J, K, INDEX: DIMTYPE;
        RATIO: INTERVAL;
        X    : IVECTOR;
        U    : ITAYLOR;

    BEGIN
    X         := IVRNULL;
    U.LENGTH  := T.LENGTH;
    U.T       := T.T;
    U.TC      := IVRNULL;
    U.TC[1]   := IEXP (T.TC[1]);

    IF NOT IT_IDENT_CONSTANT (T) THEN BEGIN
    FOR K := 2 TO U.LENGTH DO
        BEGIN
        RATIO := 1 / INTPT (K - 1));
        FOR J := 2 TO K DO
            BEGIN
            INDEX := K - J + 1;
            X[INDEX] := T.TC[J] * (J - 1) * RATIO
            END;  (* FOR J *)

        U.TC[K] := ISCALP (U.TC, X, K)
        END  (* FOR K *)

    END;  (* IF NOT IT_IDENT_CONSTANT (T) *)

    ITEXP := U
    END;  (*  FUNCTION ITEXP (ITAYLOR)  *)



                                                                (*  I ** K  *)
OPERATOR ** (BASE: INTERVAL; EXPONENT: INTEGER) RES : INTERVAL;
    (*  Requires:  K**K  *)
    VAR U: INTERVAL;
        NEG_EXP: BOOLEAN;

    BEGIN

            (*  BASE == INTEGER and EXPONENT >= 0.  Use INTEGER ** INTEGER  *)
    IF (BASE = INTPT (TRUNC (ISUP (BASE)))) AND
                (EXPONENT >= 0)
        THEN U := INTPT (TRUNC (ISUP (BASE)) ** EXPONENT)
    ELSE IF (0.0 IN BASE) AND (EXPONENT <= 0)
        THEN BEGIN
            WRITELN;
            WRITELN ('ERROR IN INTERVAL ** INTEGER: ZERO ** <= ZERO');
            SVR (0)                              (* RETURN TO OPERATING SYSTEM *)
            END  (* THEN *)
```

```
    ELSE BEGIN                                              (*  Usual branch  *)
        NEG_EXP := EXPONENT < 0;
        EXPONENT := ABS (EXPONENT);
        IF EXPONENT = 1       THEN U := BASE
        ELSE IF EXPONENT = 2 THEN U := ISQR (BASE)
        ELSE BEGIN
            U := INTPT (1.0);
            WHILE EXPONENT > 0 DO           (*  EXPONENT = 0 falls through  *)
                BEGIN
                IF EXPONENT MOD 2 = 1 THEN U := U * BASE;
                EXPONENT := EXPONENT DIV 2;
                IF EXPONENT > 0 THEN BASE := ISQR (BASE)
                END  (* WHILE *)
            END;  (* ELSE *)
        IF NEG_EXP THEN U := 1 / U
        END;  (* ELSE*)
    RES := U
    END;  (*  INTERVAL ** INTEGER  *)



                                                            (*  K ** I  *)
OPERATOR ** (BASE: INTEGER; EXPONENT: INTERVAL) RES : INTERVAL;
    (*  Requires:  K**K  *)
    VAR U: INTERVAL;

    BEGIN
    IF BASE = 1 THEN U := INTPT (1.0)                       (*  BASE == 1  *)

                    (*  EXPONENT == INTEGER >= 0.  Use INTEGER ** INTEGER  *)
    ELSE IF (EXPONENT = INTPT (TRUNC (ISUP (EXPONENT)))) AND
                        (ISUP (EXPONENT) >= 0.0)
        THEN U := INTPT (BASE ** TRUNC (ISUP (EXPONENT)))

    ELSE IF BASE <= 0       (*  EXPONENT contains at least one real number  *)
        THEN BEGIN
            WRITELN;
            WRITELN ('ERROR IN INTEGER ** INTERVAL: BASE <= ZERO');
            SVR (0)                         (* RETURN TO OPERATING SYSTEM *)
            END  (* THEN *)

    ELSE U := IEXP (EXPONENT * ILN (INTPT (BASE)));         (*  Usual branch  *)

    RES := U
    END;  (*  INTEGER ** INTERVAL  *)
```

```
                                                                    (*  I ** I  *)
OPERATOR ** (BASE: INTERVAL; EXPONENT: INTERVAL) RES : INTERVAL;
    (*  Requires:  K**K, I**K, K**I  *)
    VAR U: INTERVAL;

    BEGIN
    IF BASE = INTPT (1.0) THEN U := BASE                        (*  BASE == 1  *)

                          (*  EXPONENT == INTEGER.  Use INTERVAL ** INTEGER  *)
    ELSE IF EXPONENT = INTPT (TRUNC (ISUP (EXPONENT)))
        THEN U := BASE ** TRUNC (ISUP (EXPONENT))

                               (*  BASE == INTEGER.  Use INTEGER ** INTERVAL  *)
    ELSE IF BASE = INTPT (TRUNC (ISUP (BASE)))
        THEN U := TRUNC (ISUP (BASE)) ** EXPONENT

              (*  BASE and EXPONENT each contain at least one real number.  *)
    ELSE IF IINF (BASE) <= 0.0
        THEN BEGIN
            WRITELN;
            WRITELN ('ERROR IN INTERVAL ** INTERVAL: BASE <= ZERO');
            SVR (0)                              (* RETURN TO OPERATING SYSTEM *)
            END  (* THEN *)

    ELSE U := IEXP (EXPONENT * ILN (BASE));                 (*  Usual branch  *)

    RES := U
    END;  (*  INTERVAL ** INTERVAL  *)


                                                                    (*  IT ** K  *)
OPERATOR ** (BASE: ITAYLOR; EXPONENT: INTEGER) RES : ITAYLOR;
    (*  Requires:  IVRNULL, I**K, ITSQR, I*IT, IT*IT, ISCALP,
                   IT_IDENT_ZERO(IT)  *)

    (* DECISION TABLE for special cases:

        EXPONENT:        0          1          2          > 2          < 0
                    +----------+----------+----------+----------+----------+
                    | Box 1                                                 |
                    |                                                       |
       BASE.TC = 0  | Undef.     = BASE    SQR (BASE)    == 0       Undef.  |
                    +----------+----------+----------+----------------------+
                    | Box 2    | Box 3    | Box 4    | Box 5                |
                    |          |          |          |                      |
    0 IN BASE.TC[1] |   1      | = BASE   | SQR (BASE) | By mult.   Undef.  |
                    |          |          |          +----------------------+
                    |          |          |          | Box 6                |
                    |          |          |          |                      |
  BASE.TC[1] <> 0   |   1      | = BASE   | SQR (BASE) |  By  recurrence    |
                    +----------+----------+----------+----------------------+

              Box 6 is the usual method.                                  *)
```

```
VAR U       : ITAYLOR;
    V       : IVECTOR;
    RATIO   : INTERVAL;
    K, J, REV       : DIMTYPE;

BEGIN
U.LENGTH := BASE.LENGTH;
U.T      := BASE.T;
                          (* ZERO RESULT AND WORK VECTORS *)
U.TC := IVRNULL;  V := IVRNULL;

IF IT_IDENT_ZERO (BASE)                              (* Decision table Box 1
*)
    THEN IF EXPONENT <= 0
            THEN BEGIN
                WRITELN;
                WRITELN ('ERROR:  ZERO ** ZERO');
                SVR (0)                    (* RETURN TO OPERATING SYSTEM *)
                END  (* THEN *)
            ELSE  (*  U.TC is already == 0  *)

ELSE IF EXPONENT = 0                                 (* Decision table Box 2 *)
    THEN U.TC[1] := INTPT (1.0)

ELSE IF EXPONENT = 1                                 (* Decision table Box 3 *)
    THEN U := BASE

ELSE IF EXPONENT = 2                                 (* Decision table Box 4 *)
    THEN U := ITSQR (BASE)

ELSE IF 0.0 IN BASE.TC[1]                            (* Decision table Box 5 *)
    THEN IF EXPONENT < 0
            THEN BEGIN
                WRITELN;
                WRITELN ('ERROR:  ZERO ITAYLOR VARIABLE ** NEGATIVE');
                SVR (0)                    (* RETURN TO OPERATING SYSTEM *)
                END  (* THEN *)
            ELSE BEGIN         (* Use repeated multiplications of series *)
                U.TC[1] := INTPT (1.0);
                WHILE EXPONENT > 0 DO
                    BEGIN
                    IF EXPONENT MOD 2 = 1 THEN U := U * BASE;
                    EXPONENT := EXPONENT DIV 2;
                    IF EXPONENT > 0 THEN BASE := ITSQR (BASE)
                    END  (* WHILE *)
            END  (* ELSE *)

 ELSE BEGIN                          (* Decision table Box 6.  Usual branch *)
    U.TC[1]  := BASE.TC[1] ** EXPONENT;          (*  INTERVAL ** INTEGER *)
    RATIO    := 1 / BASE.TC[1];

    IF U.LENGTH >= 2 THEN BEGIN
    U.TC[2]  := (EXPONENT * RATIO) * U.TC[1] * BASE.TC[2];
```

```
        FOR K := 3 TO U.LENGTH DO
            BEGIN
            FOR J := 1 TO K - 1 DO
                BEGIN
                REV  := K - J + 1;
                V[J] := BASE.TC[REV] * (EXPONENT * (K - J) - J + 1);
                END;  (* FOR J *)
            U.TC[K] := RATIO * ISCALP (U.TC, V, K) / (K - 1)
            END  (* FOR K *)
        END  (* IF U.LENGTH >= 2 *)
        END;  (* ELSE *)

    RES := U
    END;  (*  ITAYLOR ** INTEGER  *)




                                                                    (*  IT ** I  *)
OPERATOR ** (BASE: ITAYLOR; EXPONENT: INTERVAL) RES : ITAYLOR;
    (*  Requires:  IVRNULL, I**K, ITSQR, I*IT, IT*IT, IT**K, I**I,
                   K**I, ISCALP, IT_IDENT_ZERO(IT), IT_IDENT_CONSTANT(IT)  *)

    VAR U       : ITAYLOR;
        V       : IVECTOR;
        RATIO   : INTERVAL;
        K, J, REV       : DINTYPE;

    BEGIN
    U.LENGTH := BASE.LENGTH;
    U.T      := BASE.T;
                                (* ZERO RESULT AND WORK VECTORS *)
    U.TC := IVRNULL;  V := IVRNULL;

    IF IT_IDENT_ZERO (BASE)
        THEN IF IINF (EXPONENT) <= 0.0
                THEN BEGIN
                    WRITELN;
                    WRITELN ('ERROR:  ZERO ** ZERO');
                    SVR (0)                     (* RETURN TO OPERATING SYSTEM *)
                    END  (* THEN *)
                ELSE  (*  U.TC is already == 0  *)

                        (*  EXPONENT == INTEGER.  Use ITAYLOR ** INTEGER  *)
    ELSE IF EXPONENT = INTPT (TRUNC (ISUP (EXPONENT)))
        THEN U := BASE ** TRUNC (ISUP (EXPONENT))

    ELSE IF IINF (BASE.TC[1]) <= 0.0
        THEN BEGIN
             WRITELN;
             WRITELN ('ERROR IN ITAYLOR ** INTERVAL:  BASE <= ZERO.');
             SVR (0)                    (* RETURN TO OPERATING SYSTEM *)
             END  (* THEN *)
```

```
    ELSE BEGIN                                    (*  Recursion.  Usual branch  *)
        U.TC[1]  := BASE.TC[1] ** EXPONENT;          (*  INTERVAL ** INTERVAL *)
        IF NOT IT_IDENT_CONSTANT (BASE) THEN
            BEGIN
            RATIO    := 1 / BASE.TC[1];

            IF U.LENGTH >= 2 THEN BEGIN
                U.TC[2]  := (EXPONENT * RATIO) * U.TC[1] * BASE.TC[2];

                FOR K := 3 TO U.LENGTH DO
                    BEGIN
                    FOR J := 1 TO K - 1 DO
                        BEGIN
                        REV  := K - J + 1;
                        V[J] := BASE.TC[REV] * (EXPONENT * (K - J) - J + 1);
                        END;  (* FOR J *)
                    U.TC[K] := RATIO * ISCALP (U.TC, V, 0) / (K - 1)
                    END  (* FOR K *)
                END  (* IF U.LENGTH >= 2 *)

            END  (* IF NOT IT_IDENT_CONSTANT (BASE) *)
        END;  (* ELSE *)

    RES := U
    END;  (*  ITAYLOR ** INTERVAL  *)



                                                                    (*  K ** IT  *)
OPERATOR ** (BASE: INTEGER; EXPONENT: ITAYLOR) RES : ITAYLOR;
    (*  Requires:  IVRNULL, K**I, K**K, ILN,
                   IT_IDENT_CONSTANT(IT), ISCALP  *)
    VAR J, K, INDEX: DIMTYPE;
        RATIO, LOG_BASE: INTERVAL;
        X    : IVECTOR;
        U    : ITAYLOR;

    BEGIN
    X         := IVRNULL;
    U.LENGTH  := EXPONENT.LENGTH;
    U.T       := EXPONENT.T;
    U.TC      := IVRNULL;
    U.TC[1]   := BASE ** EXPONENT.TC[1];
                          (*  INTEGER ** INTERVAL  -  may generate errors  *)

    IF NOT IT_IDENT_CONSTANT (EXPONENT) THEN BEGIN
        IF BASE <= 0
            THEN BEGIN
                WRITELN;
                WRITELN ('ERROR IN INTEGER ** ITAYLOR:  BASE <= ZERO');
                SVR (0)                  (* RETURN TO OPERATING SYSTEM *)
                END  (* THEN *)

        ELSE IF BASE = 1 THEN U.TC[1] := INTPT (1.0)
```

```
        ELSE BEGIN
            LOG_BASE := ILN (INTPT (BASE));
            FOR K := 2 TO U.LENGTH DO
                BEGIN
                RATIO := 1 / INTPT (K - 1);
                FOR J := 2 TO K DO
                    BEGIN
                    INDEX := K - J + 1;
                    X[INDEX] := EXPONENT.TC[J] * (J - 1) * RATIO
                    END;  (* FOR J *)
                U.TC[K] := LOG_BASE * ISCALP (U.TC, X, K)
                END  (* FOR K *)
            END  (* ELSE *)
        END;  (* IF *)

    RES := U
    END;  (*  INTEGER ** ITAYLOR  *)




                                                                    (*  I ** IT  *)
OPERATOR ** (BASE: INTERVAL; EXPONENT: ITAYLOR) RES : ITAYLOR;
    (*  Requires:  IVRNULL, K**I, K**K, ILN, I**I, I**K,
                   IT_IDENT_CONSTANT(IT), ISCALP  *)
    VAR J, K, INDEX: DIMTYPE;
        RATIO, LOG_BASE: INTERVAL;
        X    : IVECTOR;
        U    : ITAYLOR;

    BEGIN
    X        := IVRNULL;
    U.LENGTH := EXPONENT.LENGTH;
    U.T      := EXPONENT.T;
    U.TC     := IVRNULL;
    U.TC[1]  := BASE ** EXPONENT.TC[1];
                        (*  INTERVAL ** INTERVAL  -   may generate errors  *)
                (*  Note that there is neither accuracy or speed advantage to
                    be gained by treating BASE == INTEGER using K**IT
                    because the BASE is first converted to an interval
                    there. *)

    IF NOT IT_IDENT_CONSTANT (EXPONENT) THEN BEGIN
        IF IINF (BASE) <= 0.0
            THEN BEGIN
                WRITELN;
                WRITELN ('ERROR IN INTERVAL ** ITAYLOR:  BASE <= ZERO');
                SVR (0)                  (* RETURN TO OPERATING SYSTEM *)
                END  (* THEN *)

        ELSE IF (IINF (BASE) = ISUP (BASE)) AND              (*  BASE == 1.0  *)
                (IINF (BASE) = 1.0)
            THEN U.TC[1] := INTPT (1.0)
```

```
        ELSE BEGIN
            LOG_BASE := ILN (BASE);
            FOR K := 2 TO U.LENGTH DO
                BEGIN
                RATIO := 1 / INTPT (K - 1);
                FOR J := 2 TO K DO
                    BEGIN
                    INDEX := K - J + 1;
                    X[INDEX] := EXPONENT.TC[J] * (J - 1) * RATIO
                    END;  (* FOR J *)
                U.TC[K] := LOG_BASE * ISCALP (U.TC, X, K)
                END  (* FOR K *)
            END  (* ELSE *)
        END;  (* IF *)

    RES := U
    END;  (*  INTERVAL ** ITAYLOR  *)


OPERATOR ** (BASE, EXPONENT: ITAYLOR) RES : ITAYLOR;          (*  IT ** IT  *)
    (*  Requires: IT**I, I**IT, I**I, K**I, I**K, K**K,
                  ITLN, ITEXP, ITSQR, I*IT, IT*IT,
                  IVRNULL, ISCALP, IT_IDENT_CONSTANT(IT)  *)

    VAR U: ITAYLOR;

    BEGIN
    IF IT_IDENT_CONSTANT (EXPONENT)
        THEN U := BASE ** EXPONENT.TC[1]        (*  Use ITAYLOR ** INTERVAL  *)
    ELSE IF IT_IDENT_CONSTANT (BASE)
        THEN U := BASE.TC[1] ** EXPONENT        (*  Use INTERVAL ** ITAYLOR  *)

    ELSE IF IINF (BASE.TC[1]) <= 0.0
        THEN BEGIN
            WRITELN;
            WRITELN ('ERROR:  ITAYLOR BASE <= ZERO');
            SVR (0)                     (* RETURN TO OPERATING SYSTEM *)
            END  (* THEN *)

    ELSE BEGIN
        IF BASE.T <> EXPONENT.T
            THEN BEGIN
                WRITELN;
                WRITE   ('ERROR:  ** OF ITAYLOR VARIABLES WITH');
                WRITELN (' UNEQUAL SCALE FACTORS');
                SVR (0)                 (* RETURN TO OPERATING SYSTEM *)
                END;  (* IF *)

        U := EXPONENT * ITLN (BASE);
        U := ITEXP (U);
        END;  (* ELSE *)

    RES := U
    END;  (*  ITAYLOR ** ITAYLOR  *)

(*  END OF IT_POW.LIB       >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> *)
```

C.6. Real and Interval Taylor Functions.

```
(*  RIT_FNS.LIB  -  REAL AND INTERVAL TAYLOR FUNCTIONS      <<<<<<<<<<<<

Contents:
    TSQR(T)      FUNCTION TSQR (T: TAYLOR) : TAYLOR;
    TSQRT(T)     FUNCTION TSQRT (T: TAYLOR) : TAYLOR;
    TEXP(T)      FUNCTION TEXP (T: TAYLOR) : TAYLOR;
    TLN(T)       FUNCTION TLN (T: TAYLOR) : TAYLOR;
    T_SIN_COS    PROCEDURE T_SIN_COS (T: TAYLOR; VAR S, C: TAYLOR);
    TSIN(T)      FUNCTION TSIN (T: TAYLOR) : TAYLOR;
    TCOS(T)      FUNCTION TCOS (T: TAYLOR) : TAYLOR;
    TRUNGE(T)    FUNCTION TRUNGE (T: TAYLOR) : TAYLOR;
    TARCTAN(T)   FUNCTION TARCTAN (T: TAYLOR) : TAYLOR;
    TTAN(T)      FUNCTION TTAN (T: TAYLOR) : TAYLOR;
    TDIFF(T)     FUNCTION TDIFF (T: TAYLOR) : TAYLOR;
    TINTGRL(T)   FUNCTION TINTGRL (T: TAYLOR) : TAYLOR;

    ITSQR(IT)    FUNCTION ITSQR (T: ITAYLOR) : ITAYLOR;
    ITSQRT(IT)   FUNCTION ITSQRT (T: ITAYLOR) : ITAYLOR;
    ITEXP(IT)    FUNCTION ITEXP (T: ITAYLOR) : ITAYLOR;
    ITLN(IT)     FUNCTION ITLN (T: ITAYLOR) : ITAYLOR;
    IT_SIN_COS   PROCEDURE IT_SIN_COS (T: ITAYLOR; VAR S, C: ITAYLOR);
    ITSIN(IT)    FUNCTION ITSIN (T: ITAYLOR) : ITAYLOR;
    ITCOS(IT)    FUNCTION ITCOS (T: ITAYLOR) : ITAYLOR;
    ITRUNGE(IT)  FUNCTION ITRUNGE (T: ITAYLOR) : ITAYLOR;
    ITARCTAN(IT)  FUNCTION ITARCTAN (T: ITAYLOR) : ITAYLOR;
    ITTAN(IT)    FUNCTION ITTAN (T: ITAYLOR) : ITAYLOR;
    ITDIFF(T)    FUNCTION ITDIFF (T: ITAYLOR) : ITAYLOR;
    ITINTGRL(T)  FUNCTION ITINTGRL (T: ITAYLOR) : ITAYLOR;

================================================================== *)


FUNCTION TSQR (T: TAYLOR) : TAYLOR;                          (*  TSQR(T)  *)
    (*  Requires:  VRNULL, SCALP, SQR  *)
    VAR I, J, K, HALF: DIMTYPE;
        X, Y: RVECTOR;
        U   : TAYLOR;

    BEGIN
    X := VRNULL;  Y := VRNULL;
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    U.TC := VRNULL;

    U.TC[1] := SQR (T.TC[1]);
    X[1] := T.TC[1];
```

```
    FOR K := 2 TO U.LENGTH DO
        BEGIN
        X[K] := T.TC[K];
        HALF := K DIV 2;
        FOR J := 1 TO HALF DO
            BEGIN
            I := K - J + 1;
            Y[J] := T.TC[I]
            END;  (* FOR J *)
        U.TC[K] := 2.0 * SCALP ( X, Y, 0);
        IF K MOD 2 = 1 THEN
            BEGIN
            HALF := HALF + 1;
            U.TC[K]:= U.TC[K] + SQR (T.TC[HALF])
            END  (* IF *)
        END;  (* FOR K *)
    TSQR := U
    END;  (* FUNCTION TSQR (TAYLOR)  *)



FUNCTION TSQRT (T: TAYLOR) : TAYLOR;                          (*  TSQRT(T)  *)
    (*  Requires:  VRNULL, SQRT, SQR, SCALP  *)
    VAR I, K, INDEX, INDX2: DINTYPE;
        RATIO: REAL;
        X, Y : RVECTOR;
        U    : TAYLOR;

    BEGIN
    IF (T.TC[1] < 0.0) OR
       ((T.TC[1] = 0.0) AND (T.LENGTH >= 2))
        THEN BEGIN
            WRITELN;
            WRITELN ('ERROR: SQUARE ROOT OF TAYLOR VARIABLE <= ZERO.');
            SVR (0)  END;                (* RETURN TO OPERATING SYSTEM *)
        (* ELSE *)

    U.LENGTH := T.LENGTH;
    U.T := T.T;
    X := VRNULL;  Y := VRNULL;
    U.TC := VRNULL;

    U.TC[1] := SQRT (T.TC[1]);          (* REAL *)

    IF U.LENGTH >= 2 THEN BEGIN
    RATIO := 1.0 / (2.0 * U.TC[1]);
    U.TC[2] := T.TC[2] * RATIO;

    IF U.LENGTH >= 3 THEN BEGIN
    U.TC[3] := (T.TC[3] - SQR (U.TC[2])) * RATIO;
```

```
    FOR K := 4 TO U.LENGTH DO
        BEGIN
        IF K MOD 2 = 0
            THEN BEGIN
                INDEX := K DIV 2;
                INDX2 := INDEX - 1;
                X[INDX2] := U.TC[INDEX]
                END  (* THEN *)
            ELSE BEGIN
                INDEX := (K + 1) DIV 2;
                U.TC[K] := SQR (U.TC[INDEX])
            END;  (* IF *)
        FOR I := 1 TO INDX2 DO
            BEGIN
            INDEX := K - I;
            Y[I] := U.TC[INDEX];
            END;  (* FOR I *)
        U.TC[K] := (T.TC[K] - U.TC[K] - 2.0 * SCALP (X, Y, 0)) * RATIO
    END   (* FOR K *)
    END   (* IF U.LENGTH >= 3 *)
    END;  (* IF U.LENGTH >= 2 *)

    TSQRT := U
    END;  (* FUNCTION TSQRT (TAYLOR)  *)


FUNCTION TEXP (T: TAYLOR) : TAYLOR;                          (*  TEXP(T)  *)
    (*  Requires:  VRNULL, SCALP  *)
    VAR J, K, INDEX: DIMTYPE;
        RATIO: REAL;
        X    : RVECTOR;
        U    : TAYLOR;

    BEGIN
    X         := VRNULL;
    U.LENGTH  := T.LENGTH;
    U.T       := T.T;
    U.TC      := VRNULL;
    U.TC[1]   := EXP (T.TC[1]);

    FOR K := 2 TO U.LENGTH DO
        BEGIN
        RATIO := 1.0 / (K - 1);
        FOR J := 2 TO K DO
            BEGIN
            INDEX := K - J + 1;
            X[INDEX] := T.TC[J] * (J - 1) * RATIO
            END;  (* FOR J *)
        U.TC[K] := SCALP (U.TC, X, 0)
        END;  (* FOR K *)
    TEXP := U
    END;  (*  FUNCTION TEXP (TAYLOR)  *)
```

```
FUNCTION TLN (T: TAYLOR) : TAYLOR;                              (*  TLN(T)  *)
    (*  Requires:  VRNULL, LN, SCALP  *)
    VAR J, K, INDEX : DIMTYPE;
        RATIO, RATI2: REAL;
        X     : RVECTOR;
        U     : TAYLOR;

    BEGIN
    IF T.TC[1] <= 0.0
        THEN BEGIN
            WRITELN;
            WRITELN ('ERROR:  LN OF TAYLOR VARIABLE <= ZERO.');
            SVR (0)  END;                (* RETURN TO OPERATING SYSTEM *)
        (* ELSE *)

    X         := VRNULL;
    U.LENGTH  := T.LENGTH;
    U.T       := T.T;
    U.TC      := VRNULL;
    U.TC[1]   := LN (T.TC[1]);
    RATIO     := 1.0 / T.TC[1];

    IF U.LENGTH >= 2 THEN BEGIN
    U.TC[2]   := T.TC[2] * RATIO;

    FOR K := 3 TO U.LENGTH DO
        BEGIN
        RATI2 := 1.0 / (K - 1);
        FOR J := 2 TO K-1 DO
            BEGIN
            INDEX := K - J + 1;
            X[INDEX] := T.TC[J] * (INDEX - 1) * RATI2
            END;  (* FOR J *)
        U.TC[K] := RATIO * (T.TC[K] - SCALP (U.TC, X, 0))
        END  (* FOR K *)
    END;  (* IF U.LENGTH >= 2 *)

    TLN := U
    END;  (*  FUNCTION TLN  *)


PROCEDURE T_SIN_COS (T: TAYLOR; VAR S, C: TAYLOR);          (*  T_SIN_COS  *)
    (*  Requires:  VRNULL, SIN, COS, SCALP  *)
    VAR J, K, INDEX: DIMTYPE;
        RATIO: REAL;
        X     : RVECTOR;
    BEGIN
    X         := VRNULL;
    S.LENGTH  := T.LENGTH;
    S.T       := T.T;
    S.TC      := VRNULL;
    S.TC[1]   := SIN (T.TC[1]);
    C.LENGTH  := T.LENGTH;
    C.T       := T.T;
    C.TC      := VRNULL;
    C.TC[1]   := COS (T.TC[1]);
```

```
    FOR K := 2 TO T.LENGTH DO
        BEGIN
        RATIO := 1.0 / (K - 1);
        FOR J := 2 TO K DO
            BEGIN
            INDEX := K - J + 1;
            X[INDEX] := T.TC[J] * (J - 1) * RATIO;
            END;  (* FOR J *)
        S.TC[K] :=   SCALP (C.TC, X, 0);
        C.TC[K] := - SCALP (S.TC, X, 0)
        END;  (* FOR K *)
    END;  (*  PROCEDURE T_SIN_COS  *)


FUNCTION TSIN (T: TAYLOR) : TAYLOR;                              (*  TSIN(T)  *)
    (*  Requires:  VRNULL, T_SIN_COS, SIN, COS, SCALP  *)
    VAR S, C: TAYLOR;

    BEGIN
    T_SIN_COS (T, S, C);
    TSIN := S
    END;  (*  FUNCTION TSIN (TAYLOR)  *)


FUNCTION TCOS (T: TAYLOR) : TAYLOR;                              (*  TCOS(T)  *)
    (*  Requires:  VRNULL, T_SIN_COS, SIN, COS, SCALP  *)
    VAR S, C: TAYLOR;

    BEGIN
    T_SIN_COS (T, S, C);
    TCOS := C
    END;  (*  FUNCTION TCOS (TAYLOR)  *)


FUNCTION TRUNGE (T: TAYLOR) : TAYLOR;                            (*  TRUNGE(T)  *)
    (*  Requires: K/T, TSQR  *)
    VAR U: TAYLOR;

    BEGIN
    U := TSQR (T);
    U.TC[1] := U.TC[1] + 1.0;
    U := 1 / U;
    TRUNGE := U
    END;  (*  FUNCTION TRUNGE  *)


FUNCTION TARCTAN (T: TAYLOR) : TAYLOR;                           (*  TARCTAN(T)  *)
    (*  Requires:  VRNULL, TRUNGE, K/T, TSQR, ARCTAN, SCALP  *)
    VAR J, K, INDEX: DIMTYPE;
        RATIO: REAL;
        X    : RVECTOR;
        U, V : TAYLOR;
```

```
    BEGIN
    X         := VRNULL;
    U.LENGTH  := T.LENGTH;
    U.T       := T.T;
    U.TC[1]   := ARCTAN (T.TC[1]);

    V := TRUNGE (T);

    FOR K := 2 TO U.LENGTH DO
        BEGIN
        RATIO := 1.0 / (K - 1);
        FOR J := 2 TO K DO
            BEGIN
            INDEX := K - J + 1;
            X[INDEX] := T.TC[J] * (J - 1) * RATIO;
            END;  (* FOR J *)
        U.TC[K] :=   SCALP (V.TC, X, 0)
        END;  (* FOR K *)

    TARCTAN := U
    END;  (*  FUNCTION TARCTAN  *)


FUNCTION TTAN (T: TAYLOR) : TAYLOR;                                    (*  TTAN(T)  *)
    (*  Requires:  VRNULL, T/T, T_SIN_COS, SCALP  *)
    VAR S, C: TAYLOR;
    BEGIN
    T_SIN_COS (T, S, C);
    TTAN := S / C
    END;  (*  FUNCTION TTAN  *)


FUNCTION TDIFF (T: TAYLOR) : TAYLOR;                                   (*  TDIFF(T)  *)
    VAR K, INDEX: DIMTYPE;
        RATIO: REAL;
        U    : TAYLOR;

    BEGIN
    U.T := T.T;
    IF T.LENGTH = 1 THEN
        BEGIN
        U.LENGTH := 1;
        U.TC[1]  := 0.0
        END  (* THEN *)
    ELSE BEGIN
    U.LENGTH := T.LENGTH - 1;
    RATIO    := 1.0 / T.T;
    FOR K := 1 TO U.LENGTH DO
        BEGIN
        INDEX := K + 1;
        U.TC[K] := T.TC[INDEX] * RATIO * K
        END  (* FOR K *)
    END;  (* ELSE *)
    TDIFF := U
    END;  (*  FUNCTION TDIFF  *)
```

```
FUNCTION TINTGRL (T: TAYLOR) : TAYLOR;                    (*  TINTGRL(T)  *)
    VAR K, INDEX: DIMTYPE;
        U     : TAYLOR;

    BEGIN
    IF T.LENGTH < DIM THEN U.LENGTH := T.LENGTH + 1
                      ELSE U.LENGTH := DIM;
    U.T       := T.T;
    U.TC[1]   := 0.0;

    FOR K := 2 TO U.LENGTH DO
        BEGIN
        INDEX := K - 1;
        U.TC[K] := T.TC[INDEX] * T.T / INDEX
        END;  (* FOR K *)

    TINTGRL := U
    END;  (*  FUNCTION TINTGRL  *)



FUNCTION ITSQR (T: ITAYLOR) : ITAYLOR;                    (*  ITSQR(IT)  *)
    (*  Requires:  IVRNULL, ISQR, ISCALP  *)
    VAR I, J, K, HALF: DIMTYPE;
        X, Y: IVECTOR;
        U   : ITAYLOR;
    BEGIN
    U.LENGTH := T.LENGTH;
    U.T := T.T;
    X := IVRNULL;  Y := IVRNULL;
    U.TC := IVRNULL;
    U.TC[1] := ISQR (T.TC[1]);
    X[1] := T.TC[1];
    FOR K := 2 TO U.LENGTH DO
        BEGIN
        X[K] := T.TC[K];
        HALF := K DIV 2;
        FOR J := 1 TO HALF DO
            BEGIN
            I := K - J + 1;
            Y[J] := T.TC[I]
            END;  (* FOR J *)
        U.TC[K] := 2 * ISCALP ( X, Y, HALF);
        IF K MOD 2 = 1 THEN
            BEGIN
            HALF := HALF + 1;
            U.TC[K]:= U.TC[K] + ISQR (T.TC[HALF])
            END  (* IF *)
        END;  (* FOR K *)
    ITSQR := U
    END;  (* FUNCTION ITSQR (ITAYLOR)  *)
```

```
FUNCTION ITSQRT (T: ITAYLOR) : ITAYLOR;                    (*  ITSQRT(IT)  *)
   (*  Requires:  IVRNULL; ISQRT, IT_IDENT_CONSTANT(IT),
                  ISQR, ISCALP  *)
   VAR I, K, INDEX, INDX2: DIMTYPE;
       RATIO: INTERVAL;
       X, Y : IVECTOR;
       U    : ITAYLOR;

   BEGIN
   IF (IINF (T.TC[1] < 0.0) OR
      ((IINF (T.TC[1] = 0.0) AND (T.LENGTH >= 2))
       THEN BEGIN
           WRITELN;
           WRITELN ('ERROR: SQUARE ROOT OF ITAYLOR VARIABLE <= ZERO.');
           SVR (0)  END;                (* RETURN TO OPERATING SYSTEM *)
       (* ELSE *)
   X := IVRNULL;  Y := IVRNULL;
   U.LENGTH := T.LENGTH;
   U.T      := T.T;
   U.TC     := IVRNULL;
   U.TC[1] := ISQRT (T.TC[1]);          (* INTERVAL *)

   IF NOT IT_IDENT_CONSTANT (T) THEN BEGIN
   RATIO := 1 / (2 * U.TC[1]);

   IF U.LENGTH >= 2 THEN BEGIN
   U.TC[2] := T.TC[2] * RATIO;

   IF U.LENGTH >= 3 THEN BEGIN
   U.TC[3] := (T.TC[3] - ISQR (U.TC[2])) * RATIO;

   FOR K := 4 TO U.LENGTH DO
       BEGIN
       IF K MOD 2 = 0
           THEN BEGIN
               INDEX := K DIV 2;
               INDX2 := INDEX - 1;
               X[INDX2] := U.TC[INDEX]
               END  (* THEN *)
           ELSE BEGIN
               INDEX := (K + 1) DIV 2;
               U.TC[K] := ISQR (U.TC[INDEX])
           END;  (* IF *)
       FOR I := 1 TO INDX2 DO
           BEGIN
           INDEX := K - I;
           Y[I] := U.TC[INDEX];
           END;  (* FOR I *)
       U.TC[K] := (U.TC[K] - 2 * ISCALP (X, Y, INDX2)) * RATIO
   END  (* FOR K *)
   END  (* IF U.LENGTH >= 3 *)
   END  (* IF U.LENGTH >= 2 *)
   END;  (* IF NOT IT_IDENT_CONSTANT (T) *)

   ITSQRT := U
   END;  (* FUNCTION ITSQRT (ITAYLOR)  *)
```

```
FUNCTION ITEXP (T: ITAYLOR) : ITAYLOR;                          (*  ITEXP(IT)  *)
    (*  Requires:  IVRNULL, ISCALP  *)
    VAR J, K, INDEX: DIMTYPE;
        RATIO: INTERVAL;
        X    : IVECTOR;
        U    : ITAYLOR;

    BEGIN
    X         := IVRNULL;
    U.LENGTH  := T.LENGTH;
    U.T       := T.T;
    U.TC      := IVRNULL;
    U.TC[1]   := IEXP (T.TC[1]);

    FOR K := 2 TO U.LENGTH DO
        BEGIN
        RATIO := 1 / INTPT (K - 1));
        FOR J := 2 TO K DO
            BEGIN
            INDEX := K - J + 1;
            X[INDEX] := T.TC[J] * (J - 1) * RATIO
            END;  (* FOR J *)
        U.TC[K] := ISCALP (U.TC, X, K)
        END;  (* FOR K *)
    ITEXP := U
    END;  (*  FUNCTION ITEXP (ITAYLOR)  *)



FUNCTION ITLN (T: ITAYLOR) : ITAYLOR;                           (*  ITLN(IT)  *)
    (*  Requires:  IVRNULL, ILN, ISCALP  *)
    VAR J, K, INDEX : DIMTYPE;
        RATIO, RATI2: INTERVAL;
        X    : IVECTOR;
        U    : ITAYLOR;

    BEGIN
    IF IINF (T.TC[1]) <= 0.0
        THEN BEGIN
            WRITELN;
            WRITELN ('ERROR:  LN OF ITAYLOR VARIABLE <= ZERO.');
            SVR (0)  END;                (* RETURN TO OPERATING SYSTEM *)
        (* ELSE *)

    X         := IVRNULL;
    U.LENGTH  := T.LENGTH;
    U.T       := T.T;
    U.TC      := IVRNULL;
    U.TC[1]   := ILN (T.TC[1]);
    RATIO     := 1 / T.TC[1];

    IF U.LENGTH >= 2 THEN BEGIN
    U.TC[2]   := T.TC[2] * RATIO;
```

```
    FOR K := 3 TO U.LENGTH DO
        BEGIN
        RATI2 := 1 / INTPT (K - 1);
        FOR J := 2 TO K-1 DO
            BEGIN
            INDEX := K - J + 1;
            X[INDEX] := T.TC[J] * RATI2 * (INDEX - 1)
            END;  (* FOR J *)
        U.TC[K] := RATIO * (T.TC[K] - ISCALP (U.TC, X, K))
        END  (* FOR K *)
    END;  (* IF U.LENGTH >= 2 *)

    ITLN := U
    END;  (*  FUNCTION ITLN  *)


PROCEDURE IT_SIN_COS (T: ITAYLOR; VAR S, C: ITAYLOR);        (*  IT_SIN_COS  *)
    (*  Requires:  IVRNULL, ISINE, ICOS, ISCALP  *)
    VAR J, K, INDEX: DIMTYPE;
        RATIO: INTERVAL;
        X    : IVECTOR;

    BEGIN
    X        := IVRNULL;
    S.LENGTH := T.LENGTH;
    S.T      := T.T;
    S.TC     := IVRNULL;
    S.TC[1]  := ISINE (T.TC[1]);

    C.LENGTH := T.LENGTH;
    C.T      := T.T;
    C.TC     := IVRNULL;
    C.TC[1]  := ICOS (T.TC[1]);

    FOR K := 2 TO T.LENGTH DO
        BEGIN
        RATIO := 1 / INTPT (K - 1);
        FOR J := 2 TO K DO
            BEGIN
            INDEX := K - J + 1;
            X[INDEX] := T.TC[J] * RATIO * (J - 1);
            END;  (* FOR J *)
        S.TC[K] :=   ISCALP (C.TC, X, K);
        C.TC[K] := - ISCALP (S.TC, X, K)
        END;  (* FOR K *)
    END;  (*  PROCEDURE IT_SIN_COS  *)


FUNCTION ITSIN (T: ITAYLOR) : ITAYLOR;                       (*  ITSIN(IT)  *)
    (*  Requires:  IVRNULL, IT_SIN_COS, ISINE, ICOS, ISCALP  *)
    VAR S, C: ITAYLOR;

    BEGIN
    IT_SIN_COS (T, S, C);
    ITSIN := S
    END;  (*  FUNCTION ITSIN (ITAYLOR)  *)
```

```
FUNCTION ITCOS (T: ITAYLOR) : ITAYLOR;                          (*  ITCOS(IT)  *)
    (*  Requires:  IVRNULL, IT_SIN_COS, ISINE, ICOS, ISCALP  *)
    VAR S, C: ITAYLOR;

    BEGIN
    IT_SIN_COS (T, S, C);
    ITCOS := C
    END;  (*  FUNCTION ITCOS (ITAYLOR)  *)


FUNCTION ITRUNGE (T: ITAYLOR) : ITAYLOR;                        (*  ITRUNGE(IT)  *)
    (*  Requires: K/IT, ITSQR  *)
    VAR U: ITAYLOR;

    BEGIN
    U := ITSQR (T);
    U.TC[1] := U.TC[1] + 1;
    U := 1 / U;
    TRUNGE := U
    END;  (*  FUNCTION ITRUNGE  *)


FUNCTION ITARCTAN (T: ITAYLOR) : ITAYLOR;                       (*  ITARCTAN(IT)  *)
    (*  Requires:  IVRNULL, ITRUNGE, K/IT, ITSQR, IARCTAN, ISCALP  *)
    VAR J, K, INDEX: DIMTYPE;
        RATIO: INTERVAL;
        X    : IVECTOR;
        U, V : ITAYLOR;
    BEGIN
    X        := IVRNULL;
    U.LENGTH := T.LENGTH;
    U.T      := T.T;
    U.TC[1]  := IARCTAN (T.TC[1]);
                                (*  Generates error if ABS (T.TC[1]) > 1.0  *)
    V := ITRUNGE (T);
    FOR K := 2 TO U.LENGTH DO
        BEGIN
        RATIO := 1 / INTPT (K - 1);
        FOR J := 2 TO K DO
            BEGIN
            INDEX := K - J + 1;
            X[INDEX] := T.TC[J] * (J - 1) * RATIO;
            END;  (* FOR J *)
        U.TC[K] :=  ISCALP (V.TC, X, K)
        END;  (* FOR K *)
    ITARCTAN := U
    END;  (*  FUNCTION ITARCTAN  *)


FUNCTION ITTAN (T: ITAYLOR) : ITAYLOR;                          (*  ITTAN(IT)  *)
    (*  Requires:  IVRNULL, IT/IT, IT_SIN_COS, ISCALP  *)
    VAR S, C: ITAYLOR;
    BEGIN
    IT_SIN_COS (T, S, C);
    ITTAN := S / C
    END;  (*  FUNCTION ITTAN  *)
```

```
FUNCTION ITDIFF (T: ITAYLOR) : ITAYLOR;                              (*  ITDIFF(IT)  *)
    VAR K, INDEX: DIMTYPE;
        RATIO: INTERVAL;
        U    : ITAYLOR;

    BEGIN
    U.T       := T.T;
    IF T.LENGTH = 1 THEN
        BEGIN
        U.LENGTH := 1;
        U.TC[1]  := 0.0
        END  (* THEN *)
    ELSE BEGIN
    U.LENGTH := T.LENGTH - 1;
    RATIO    := 1 / INTPT (T.T);

    FOR K := 1 TO U.LENGTH DO
        BEGIN
        INDEX := K + 1;
        U.TC[K] := T.TC[INDEX] * RATIO * K
        END  (* FOR K *)
    END;  (* ELSE *)

    ITDIFF := U
    END;  (*  FUNCTION ITDIFF  *)


FUNCTION ITINTGRL (T: ITAYLOR) : ITAYLOR;                          (*  ITINTGRL(IT)  *)
    VAR K, INDEX: DIMTYPE;
        RATIO: INTERVAL;
        U    : ITAYLOR;

    BEGIN
    IF T.LENGTH < DIM THEN U.LENGTH := T.LENGTH + 1
                      ELSE U.LENGTH := DIM;
    U.T       := T.T;
    U.TC[1]   := INTPT (0.0);
    RATIO     := INTPT (T.T);

    FOR K := 2 TO U.LENGTH DO
        BEGIN
        INDEX := K - 1;
        U.TC[K] := T.TC[INDEX] * RATIO / INDEX
        END;  (* FOR K *)

    ITINTGRL := U
    END;  (*  FUNCTION ITINTGRL  *)


(*  END OF RIT_FNS.LIB       >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>  *)
```

AD-A129 075 AUTOMATIC GENERATION OF TAYLOR SERIES IN PASCAL-SC: BASIC OPERATIONS AND..(U) WISCONSIN UNIV-MADISON MATHEMATICS RESEARCH CENTER G CORLISS ET AL. MAR 83

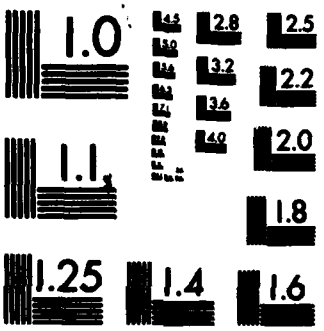
2/2

UNCLASSIFIED MRC-TSR-2497 DAAG29-80-C-0041 F/G 12/1 NL

END

DATE FILMED

DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

C.7. Utility Functions and Procedures.

```
(*  UTIL.LIB     -  UTILITY PROCEDURES & FUNCTIONS     <<<<<<<<<<<<<<<<<

Contents:

        FUNCTION VRNULL : RVECTOR;
        FUNCTION IVRNULL : IVECTOR;
        FUNCTION T_IDENT_ZERO (T: TAYLOR) : BOOLEAN;
        FUNCTION T_IDENT_CONSTANT (T: TAYLOR) : BOOLEAN;
        FUNCTION IT_IDENT_ZERO (T: ITAYLOR) : BOOLEAN;
        FUNCTION IT_IDENT_CONSTANT (T: ITAYLOR) : BOOLEAN;
        PROCEDURE WRITE_INTERVAL (INT: INTERVAL);
        PROCEDURE WRITE_INTERVAL_SERIES (SER: ITAYLOR);
        PROCEDURE READ_INTERVAL_SERIES (VAR F: ITAYLOR);
        PROCEDURE WRITE_SERIES (T: TAYLOR);
        FUNCTION TMIDPT (F : ITAYLOR) : TAYLOR;

================================================================== *)


FUNCTION VRNULL : RVECTOR;
    VAR I: DIMTYPE; U: RVECTOR;
    BEGIN
    FOR I := 1 TO DIM DO U[I] := 0.0;
    VRNULL := U
    END;  (*  FUNCTION VRNULL  *)


FUNCTION IVRNULL : IVECTOR;
    VAR I: DIMTYPE; U: IVECTOR;
    BEGIN
    FOR I := 1 TO DIM DO U[I] := INTPT (0.0);
    IVRNULL := U
    END;  (*  FUNCTION IVRNULL  *)


FUNCTION T_IDENT_ZERO (T: TAYLOR) : BOOLEAN;
        (*  TRUE if the series is identically equal to zero *)
    VAR U: BOOLEAN;
        K: INTEGER;

    BEGIN
    U := TRUE;
    K := 1;
    WHILE (K <= T.LENGTH) AND (U) DO
        BEGIN
        IF T.TC[K] <> 0.0 THEN U := FALSE;
        K := K + 1
        END;  (* WHILE *)
    T_IDENT_ZERO := U
    END;  (*  FUNCTION T_IDENT_ZERO  *)
```

```
FUNCTION T_IDENT_CONSTANT (T: TAYLOR) : BOOLEAN;
        (*  TRUE if the series is identically equal to some constant *)
    VAR U: BOOLEAN;
        K: INTEGER;

    BEGIN
    U := TRUE;
    K := 2;
    WHILE (K <= T.LENGTH) AND (U) DO
        BEGIN
        IF T.TC[K] <> 0.0 THEN U := FALSE;
        K := K + 1
        END;  (* WHILE *)
    T_IDENT_CONSTANT := U
    END;  (*  FUNCTION T_IDENT_CONSTANT  *)


FUNCTION IT_IDENT_ZERO (T: ITAYLOR) : BOOLEAN;
        (*  TRUE if the series is identically equal to zero  *)
    VAR U: BOOLEAN;
        K: INTEGER;

    BEGIN
    U := TRUE;
    K := 1;
    WHILE (K <= T.LENGTH) AND (U) DO
        BEGIN
        IF T.TC[K] <> INTPT (0.0) THEN U := FALSE;
        K := K + 1
        END;  (* WHILE *)
    IT_IDENT_ZERO := U
    END;  (*  FUNCTION IT_IDENT_ZERO (ITAYLOR)  *)


FUNCTION IT_IDENT_CONSTANT (T: ITAYLOR) : BOOLEAN;
        (*  TRUE if the series is identically equal
                                to a constant interval *)
    VAR U: BOOLEAN;
        K: INTEGER;

    BEGIN
    U := TRUE;
    K := 2;
    WHILE (K <= T.LENGTH) AND (U) DO
        BEGIN
        IF T.TC[K] <> INTPT (0.0) THEN U := FALSE;
        K := K + 1
        END;  (* WHILE *)
    IT_IDENT_CONSTANT := U
    END;  (*  FUNCTION IT_IDENT_CONSTANT (ITAYLOR)  *)


PROCEDURE WRITE_INTERVAL (INT: INTERVAL);
    BEGIN
    WRITE ('[', INT.INF, ', ', INT.SUP, ']');
    END;  (*  PROCEDURE WRITE_INTERVAL  *)
```

```
PROCEDURE WRITE_INTERVAL_SERIES (SER: ITAYLOR);
    VAR I: DIMTYPE;
    BEGIN
    FOR I := 1 TO SER.LENGTH DO BEGIN
        WRITE (I:4, '   ');
        WRITE_INTERVAL (SER.TC[I]);
        WRITELN  END
    END;  (*  PROCEDURE WRITE_INTERVAL_SERIES  *)


PROCEDURE READ_INTERVAL_SERIES (VAR F: ITAYLOR);
    VAR I: DIMTYPE;
    BEGIN
    READ (DATA, F.T);
    WRITELN ('SCALE FACTOR:', F.T);
    FOR I := 1 TO F.LENGTH DO IREAD (DATA, F.TC[I]);
    WRITELN ('SERIES FOR F:');
    WRITE_INTERVAL_SERIES (F);
    END;  (*  PROCEDURE READ_INTERVAL_SERIES  *)


PROCEDURE WRITE_SERIES (T: TAYLOR);
    VAR I: DIMTYPE;
    BEGIN
    FOR I := 1 TO T.LENGTH DO
        WRITELN (I:4, '   ', T.TC[I])
    END;  (*  PROCEDURE WRITE_SERIES  *)


FUNCTION TMIDPT (F : ITAYLOR) : TAYLOR;
    VAR I: DIMTYPE;  U: TAYLOR;
    BEGIN
    U.T := F.T;
    U.LENGTH := F.LENGTH;
    FOR I := 1 TO U.LENGTH DO
        U.TC[I] := (IINF (F.TC[I]) + ISUP (F.TC[I])) * 0.5;
    ITMIDPT := U
    END;  (*  FUNCTION ITMIDPT  *)


(*  END OF UTIL.LIB          >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> *)
```

APPENDIX D

User Manual for RDEQ_SOLV and IDEQ_SOLV

This Appendix is addressed to readers with access to Pascal-SC who wish to use the programs RDEQ_SOLV and IDEQ_SOLV to explore the solutions of initial-value problems for ordinary differential equations.

These programs allow direct user intervention in the selection of the integration step size at each step. As such, they are quite useful for hands-on exploration, but they are not intended to serve as general purpose solvers.

The illustrations given here are for the Pascal-SC compiler for the Zilog RIO operating system and its text editor. Users with the Zilog MCZ-1 microcomputer can obtain the software described in this report by sending a hard-sectored, single-sided 8" floppy disk to the second author. The directions given here will apply with only minor modifications to the use of Pascal-SC compilers for other systems.

Step 1. Create the source program.

```
EDIT      Myprob.S
     GET       RDEQ_SOLV.S or IDEQ_SOLV.S
     CHANGE    YPRIME :=  to the expression for the equation to be solved
     Repace the source code for the TEQR (ITEQR) operator by the operators
     and functions needed to evaluate the expression in your equation; they
     are obtained from -.LIB as needed.  Many operators call other
     operators, which must also be included in your source code.  The calls
```

to the basic interval operators are already included in the heading of IDEQ_SOLV.

The process of assembling all the necessary operators in the correct order is rather tedious and error-prone; be patient and careful. A good library manager would be an asset.

Step 2. Compile.

For RDEQ_SOLV, use: CMP Myprob SUM_LIB

For IDEQ_SOLV, use: CMP Myprob I_LIB

Step 3. If the compiler detects errors, for example, an omitted subroutine, then edit Myprob.S again and recompile.

Step 4. If you wish to read input data from a file instead of the console, then this file must be created. The initial values of X and Y are read from an external file which can be named as the user chooses. Myprog.DAT will be used here for illustration. Output can also be directed to an external file instead of the console, for example, to Myprog.OUT.

Step 5. Run.

For input/output using the terminal, use

XQP Myprog $CON $CON $CON

For input from Myprog.DAT and output to the terminal, use

```
XQP   Myprog   $CON  Myprog.DAT  $CON
```

For input from Myprog.DAT and output to Myprog.OUT, use

```
XQP   Myprog   $CON   Myprog.DAT   Myprog.OUT
```

These programs (especially IDEQ_SOLV) can take a few minutes to compute the series solution on the Z80 system, so you should be patient. The authors have found these programs very useful as tools to quantify suspected catastrophic cancellations or instabilities in the generation of Taylor series solutions.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: # 2497	2. GOVT ACCESSION NO.: AD-A129075	3. RECIPIENT'S CATALOG NUMBER:
4. TITLE (and Subtitle): AUTOMATIC GENERATION OF TAYLOR SERIES IN PASCAL-SC: BASIC OPERATIONS AND APPLICATIONS TO ORDINARY DIFFERENTIAL EQUATIONS	5. TYPE OF REPORT & PERIOD COVERED: Summary Report - no specific reporting period
	6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): George Corliss and L. B. Rall	8. CONTRACT OR GRANT NUMBER(s): DAAG29-80-C-0041
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Mathematics Research Center, University of Wisconsin, 610 Walnut Street, Madison, Wisconsin 53706	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: Work Unit Number 3 - Numerical Analysis
11. CONTROLLING OFFICE NAME AND ADDRESS: U. S. Army Research Office, P.O. Box 12211, Research Triangle Park, North Carolina 27709	12. REPORT DATE: March 1983
	13. NUMBER OF PAGES: 99
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Taylor series
recurrence relations for Taylor coefficients
automatic differentiation
numerical solution of ordinary differential equations
stability
error analysis
interval arithmetic

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Taylor series have a long history of usefulness in numerical analysis, especially for the numerical solution of the initial value problem for systems of ordinary differential equations. Since recurrence relations for coefficients of Taylor series are well known, it is possible to automate the computation of arithmetic operations and various standard functions with

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. ABSTRACT - cont'd.

arguments which are themselves series. If the language used for scientific computation supports user defined operators and data types, then the facilities built into the language compiler itself can be used to generate machine code for the evaluation of Taylor coefficients. Examples of such languages are Pascal-SC, Algol 68, and ADA (a trademark of the U. S. Department of Defense). Pascal-SC (Pascal for Scientific Computation) offers the user highly accurate floating-point and interval arithmetic, the latter being useful for automatic computation of guaranteed error bounds. In this language, series with real coefficients are introduced as type TAYLOR, and the corresponding series with interval coefficients as type ITAYLOR. Source code is given for the operators +, -, *, /, ** and the functions SQR, SQRT, EXP, SIN, COS, ARCTAN, and LN with arguments of these types and some other useful functions and procedures. Integer, real, and interval constants are also allowed in TAYLOR or ITAYLOR expressions. Suggestions for the implementation of additional operators or functions are given. An application of Taylor series and the methods of interval analysis to the solution of the initial value problem for ordinary differential equations is made using types TAYLOR and ITAYLOR. An analysis of the stability of this method is made, which shows that the recurrence relations for generation of the Taylor series for the solution exhibit a mild instability which has no significant effect on the values of the solution computed by analytic continuation.