6-4-1987

Automatic differentiation: Implementation in the Ada programming language

Robert P. Herloski

Rochester Institute of Technology
School of Computer Science and Technology

Automatic Differentiation: Implementation in the Ada® Programming Language

by:

Robert P. Herloski

A thesis, submitted to the Faculty
of the School of Computer Science and Technology,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Approved by:

Stanislaw Radziszowski

Professor Stanislaw Radziszowski

Peter G. Anderson

Professor Peter Anderson

Jack Hollingsworth

Professor Jack Hollingsworth

June 4, 1987

1.2 ABSTRACT

This thesis describes the characteristics of a novel computational technique called automatic differentiation, and gives an implementation of this technique in the Ada® programming language. Automatic differentiation is a technique with which one can directly calculate the value of any partial derivative of nearly any function that has been programmed into a computer (at any point of interest), without requiring the algebraic derivation of the desired partial derivative of the function of interest. It is closely related to the technique of power series manipulation, with which one can find, for example, the sum, product, square root, etc., of given power series.

The result of this work is an Ada package called DIFFERENTIALS, which exports various types and functions that can be included in user programs. The first half of this thesis describes the technique of automatic differentiation, gives the fundamental specification for the DIFFERENTIALS package, and explains the details of the implementation. The second half of this thesis gives specific examples of the use of DIFFERENTIALS in the area of optical system analysis. Presented there are several user programs that use the DIFFERENTIALS package to calculate various partial derivatives of the ray tracing function, and illustrate the proper usage of this package.

1.3 KEYWORDS AND PHRASES

differentiation, derivative, optics, aberrations, power series, Taylor series, Ada, differential

1.4 COMPUTING REVIEW SUBJECT CODES

G.1.4 Mathematics of Computing/Numerical Analysis/Quadrature and Numerical Differentiation

I.1.3 Computing Methodologies/Algebraic Manipulation/Languages and Systems

1.5 TABLE OF CONTENTS

1.6 REPRODUCTION PERMISSION

Title of Thesis Automatic Differentiation: Implementation in Ada

I Robert Herloski hereby grant permission to the Wallace Memorial Library, of RIT, to reproduce all but Chapter 9 of my thesis. Chapter 9 of this thesis may be reproduced only with my written permission on a per-request basis. Any reproduction will not be for commercial use or profit.

Date: 7/30/87

Signed: Robert P. Herloski

Address:

2. INTRODUCTION AND BACKGROUND

2.1 PROBLEM STATEMENT

It is a classic problem in numerical analysis to find the values of various derivatives of a function of interest. Suppose that a function $f(x_1,x_2,\ldots,x_n)$ of n independent variables is specified (f: $D^n \rightarrow D$, where the domain D can be any appropriate domain), and has been programmed into a computer. One can evaluate the function f at any point $A_0=\{a_1,a_2,\ldots,a_n\}$ by direct substitution. To find the value of a particular derivative of f at A_0, though, some additional computational work is required. There are two classic ways to solve this problem:

(a) One can use a finite difference/quotient method, where one evaluates the function at an appropriate number of closely spaced points, then evaluates a series of differences and quotients to derive the value. This technique is based on the definition of a derivative of a function of a single variable:

$$f'(x) = \lim_{\Delta x \rightarrow 0} \frac{f(x+\Delta x) - f(x)}{\Delta x} \tag{1}$$

and its extension to higher order derivatives of multivariate functions. The drawback to this technique is that, because of the ill-conditioned nature of this formula (denominator $\rightarrow$ 0), there is a tradeoff between the *accuracy* of the result and the *precision* of the result; an optimum tradeoff seems to be where the calculated derivative has $\frac{1}{2}$ the bits of precision of the calculated function value.

(b) The second classic technique is to symbolically derive the formula for the derivative of f, and then, by direct substitution, find the appropriate value. The drawback in this method is that the algebraic formulas for the derivatives of a function f tend in general to be much more complex than the formula for f itself, especially for higher order derivatives. Mistakes in derivation can be made, and sometimes it is impractical to do symbolic differentiation.

There are of course solutions to these drawbacks. For the finite difference method, one can go to very high precision calculations, and for the symbolic differentiation case, one can use a symbolic differentiation package such as MACSYMA, MAPLE, or SMP[1] These techniques tend to be computationally intensive, though, and are often not easily integrable into one's own user program, especially when the function is defined by a *program* rather than a *formula*.

However, there exists a third computational technique that can be used to compute derivatives, namely automatic differentiation. It combines the advantages of the two previous techniques, while lacking their disadvantages. It allows one to calculate the value of any derivative of any function that can be differentiated using the standard rules of algebraic differentiation, to the precision with which one can calculate the function value itself, without the need to algebraically differentiate the function.

There are several concepts that are fundamental to any implementation of automatic differentiation:

(a) In traditional computer programming and numerical analysis, the function of interest is coded in some high-level language within some larger program. This coding involves the use of local variable(s) that represent the independent variable(s) or parameters of the

function. These variables are scalar quantities in many cases; the function value at any point in the domain is calculated via substitution into the variables.

(b) Many functions are composed of a sequence of simple operations. These operations form a basic "library" of operators; e.g., a typical library might be {+, −, *, ÷, sin, cos, sqrt, tan, exp, ↑}.

(c) Consider an abstraction of the above concepts, in which instead of scalar quantities being substituted into the function, the function is defined such that it takes some input "vector" quantity, and returns as a result another "vector" quantity. The interpretation of the output vector quantity is such that one of the values is the "value" of the function at the point of interest. The other values correspond to the various derivatives of the function at the point of interest, with respect to the variable(s) of interest.

(d) Since the function is composed from this basic "library" of operators, it is necessary to define the meanings of these operators operating on "vector" rather than scalar quantities.

The ideas presented here are analogous to the concept of "ordered pair" arithmetic[2], as illustrated by *complex* or *rational* arithmetic. For example, in rational arithmetic, one defines an ordered pair of integers (p, q), which represents the rational number $r=p/q$. One can define special +, −, *, and ÷ operations that take two such ordered pairs as input, and generate an ordered pair as output. For example, the rule for multiplication of rational numbers is:

$$(p_1, q_1)(p_2, q_2) = (p_1 p_2, q_1 q_2). \tag{2}$$

It will be shown in §2.4 that this concept of "ordered pair" arithmetic can be extended to differentiation arithmetic, in which one can manipulate, in general, "ordered tuplets" of numbers, each component of which represents a different derivative.

2.2 PREVIOUS WORK

2.2.1 Published Work

A good summary and description of the early work in automatic differentiation was written by L. B. Rall of the University of Wisconsin at Madison in 1981[3]. This early work was carried out in the FORTRAN programming language. Automatic differentiation was implemented with the use of various preprocessing programs. For example, CODEX and SUPER-CODEX were developed to take as input some representation of a formula, and output either machine code or FORTRAN code that could be used to evaluate both function values and function derivatives. Later, the AUGMENT[4] preprocessor was used to process an entire FORTRAN program, in which one had defined such "special" data types as "Taylor" or "Gradient", to generate a new FORTRAN program with appropriate subroutine calls when these special data types were encountered.

More recent work by Rall[5] has involved the use of the PASCAL-SC programming language in implementing automatic differentiation. PASCAL-SC is a scientific computing language that allows one to define new data types, and operators that operate on those data types. These capabilities are quite useful in implementing automatic differentiation.

In the optics community there are several authors that have used the equivalent of automatic differentiation to solve typical analysis problems in lens design. T. B. Anderson has published several papers[6-10] in which he uses a FORTRAN implementation of power series manipulation to calculate optical aberration coefficients and derivatives of the aberration coefficients with respect to lens system constructional parameters. Some source code is included in those papers[6,7]

G. W. Forbes has also published numerous papers in which he uses the techniques of power series manipulation to calculate multivariate optical aberration series[11-15] In addition, he has published two papers[16,17] that summarize the theory of multivariate power series manipulation and give pseudocode that illustrate his compact, optimized implementation. It will be shown in §2.4.3 that power series manipulation techniques are intimately related to automatic differentiation.

2.2.2 Unpublished Work

In 1979, Dr. Sidney Marshall of the Xerox Corporation implemented a concept called "differentials" in the Smalltalk programming language on the Xerox Alto microprogrammable computer. This turned out to be equivalent to automatic differentiation. Drawing on the object-oriented and LISP-like nature of the language, Marshall developed an algorithm and method of representation of differentials that is valid for an arbitrary partial derivative of a function of an arbitrary number of independent variables.

2.3 THESIS GOALS

The intent of this thesis is to develop an implementation of automatic differentiation in such a way that a casual user of the implementation could *easily* incorporate automatic differentiation into his or her programming without the need to be cognizant of all the underlying details. Given this, several distinct thesis goals can be developed:

(A) Implementation of automatic differentiation in the Ada programming language: Some of the desirable features in a successful implementation of automatic differentiation include (1) *data type abstraction*, and (2) *operator overloading*. *Data type abstraction* allows one to create and name a new data type that is appropriate to some application, and to define functions that operate on that data type. If it makes sense to consider such operations as +, −, *, and ÷ on this new data type, then *operator overloading* permits one to use the *symbols* +, −, *, and ÷ in a program to represent those operations. These two features were previously obtained by using a specially enhanced language (such as PASCAL-SC) or a language preprocessor (such as AUGMENT, CODEX, SUPER-CODEX). However, the Ada programming language provides as standard features data type abstraction and operator overloading. Ada also allows overloaded operators to be used in *infix* notation within a program. Thus, one could write code such as:

```
variable1:  MY_DATA_TYPE;
variable2:  MY_DATA_TYPE;
              .
              .
              .
```

```
variable1 := variable1 * variable2;
```

assuming that the data type "MY_DATA_TYPE" and the operator "*" were appropriately defined.

Another feature of the Ada programming language is the concept of *packages*, with which one can combine (or "package") the definition of a data type and various functions, procedures, and operators that operate on that data type, and hide the details of the implementation from a client, or user, program. Lastly, given that the Ada programming language is steadily growing in acceptance (as evidenced by the growing number of Ada compilers available), it makes a very appropriate implementation vehicle for automatic differentiation.

(B) Ease of use with higher order derivatives and multiple independent variables: Another goal for this thesis is to develop an implementation that is flexible enough to allow a user to define higher order derivatives and/or multiple independent variables just as easily as one would define a first order derivative of a function of one variable. The source code that has previously been published has put the onus of accommodating higher order derivatives or multiple variables on the user program.

Some of Marshall's work on "differentials" involved developing algorithms that were needed to handle higher order derivatives, especially for products and functions. Part of this second thesis goal is to bring over those algorithms into the Ada programming language and use them as the basis for extension of the concept of automatic differentiation to higher order partial derivatives.

(C) Applications: The final goal of this thesis is to write several client programs that use the features of automatic differentiation and illustrate its flexibility and utility. Since my background is in optical engineering, the client programs will perform some typical analyses on lens systems (similar to the work by Forbes and Anderson).

2.4 THEORY OF AUTOMATIC DIFFERENTIATION

2.4.1 Basic Theory

The following summary of the theory of automatic differentiation is taken largely from the discussion by L. B. Rall in [2]. A more rigorous derivation of the mathematical theory can be found in [2]–[4].

As mentioned in §2.1, automatic differentiation can be considered a type of "ordered pair" arithmetic. The simplest case is to consider pairs of real numbers

$$U = (u, u') \in R^2,$$

where the first element of the ordered pair is a number associated with a *value*, and the second element is a number associated with a *derivative*.

Consider two such pairs U and V. Given below are the rules of differential arithmetic for $U, V \in R^2$:

$$\text{Addition:} \qquad U+V=(u,u')+(v,v')=(u+v,u'+v') \tag{1}$$

$$\text{Subtraction:} \qquad U-V=(u,u')-(v,v')=(u-v,u'-v') \tag{2}$$

$$\text{Multiplication:} \qquad UV=(u,u')(v,v')=(uv,uv'+vu') \tag{3}$$

$$\text{Division:} \qquad \frac{U}{V}=\frac{(u,u')}{(v,v')}=\left(\frac{u}{v},\frac{vu'-uv'}{v^2}\right),\ v\neq 0 \tag{4}$$

Next, consider any rational function (function expressed as a ratio of polynomials) r: $R \rightarrow R$. This function may be extended to a function r^*: $R^2 \rightarrow R^2$

$$r^*(U) \equiv r^*((u,u')) = V \equiv (v,v'),$$

by substituting (u, u') for the independent variable in r, substituting $(c, 0)$ for any constant c in r, and applying (1)–(4) recursively. If the ordered pair $(u, 1)$ is substituted for the independent variable, then:

$$r^*((u,1)) = V \equiv (r(u), r'(u)), \tag{5}$$

i.e., both the value of the function *and* its derivative are immediately given.

Consider as an example the following rational function:

$$r(x)=\frac{2x^2-4x+7}{3x+1} \tag{6}$$

By substitution of $x=3$ into (6), we find that $r(3)=1.3$. If we extend this function to r^*: $R^2 \rightarrow R^2$:

$$r^*(U)=\frac{(2,0)UU-(4,0)U+(7,0)}{(3,0)U+(1,0)}. \tag{7}$$

Then, using (1)–(4), and substituting $U=(3, 1)$, we can easily find that

$$r^*((3,1))=\frac{(2,0)(3,1)(3,1)-(4,0)(3,1)+(7,0)}{(3,0)(3,1)+(1,0)}=\frac{(2,0)(9,6)-(12,4)+(7,0)}{(9,3)+(1,0)}$$

$$=\frac{(18,12)+(-5,-4)}{(10,3)}=\frac{(13,8)}{(10,3)}=(1.3,0.41). \tag{8}$$

From this calculation, we see that $r'(3) = 0.41$. To check this, we can symbolically differentiate (6):

$$r'(x)=\frac{(3x+1)(4x-4)-(2x^2-4x+7)(3)}{(3x+1)^2}=\frac{6x^2+4x-25}{(3x+1)^2}, \tag{9}$$

and, by substitution, we verify that $r'(3) = 0.41$.

Differentiation arithmetic can also be extended to arbitrary differentiable functions f: $R \rightarrow R$ (f^*: $R^2 \rightarrow R^2$) via:

$$f^*(U) \equiv f^*((u, u')) = (f(u), u'f'(u)). \tag{10}$$

For example:

$$e^U \equiv e^{(u, u')} = (e^u, u'e^u), \tag{11}$$

and

$$\sin(U) \equiv \sin(u, u') = (\sin(u), u'\cos(u)) . \tag{12}$$

Note that if $U=(u, 1)$, then Eq. (10) is true by definition [see, e.g., Eqs. (11) & (12)]. If $u' \neq 1$, then Eq. (10) is equivalent to the so-called *chain rule* for the derivatives of functions of functions. If f,g: $R \rightarrow R$, and f^*,g^*: $R^2 \rightarrow R^2$ are the extensions to f and g, then:

$$f^*(g^*(U)) \equiv f^*(g^*(u, 1)) = (f(g(u)), g'(u)f'(g(u))). \tag{13}$$

Here is an example that illustrates the use of Eq. (13). Let $f(x)=\sin(x)$, $g(x)=3x^2+4x^2-6$ and $h(x)=f(g(x))$. We desire to find $h(1)$ and $h'(1)$. This time, we do the symbolic differentiation first:

$$h(x)=\sin(3x^2+4x-6) \tag{14}$$

$$h'(x)=(6x+4)\cos(3x^2+4x-6), \tag{15}$$

so $h(1) \simeq 0.84147098$ and $h'(1) \simeq 5.4030231$.

Now, evaluate Eq. (14) using differentiation arithmetic:

$$h^*((1, 1))=\sin((3, 0)(1, 1)(1, 1)+(4, 0)(1, 1)-(6, 0))$$

$$=\sin((3, 0)(1, 2)+(4, 4)-(6, 0))$$

$$=\sin((3, 6)+(-2, 4)) = \sin((1, 10)),$$

$$= (\sin(1), 10\cos(1))$$

$$\simeq (0.84147098, 5.4030231).$$

Therefore, the rules (1)–(4) and (13) are sufficient to define an ordered pair arithmetic that can be used to calculate both the value and derivative of any differentiable function without the need to symbolically differentiate the formula for the function. The key to this technique is that many differentiable functions of interest are *composite* functions, i.e., they consist of a sequence of operators, each operator being chosen from a relatively small set or library of basic functions, e.g., $\{+, -, *, \div, \sin, \cos, \text{sqrt}, \ldots\}$. Thus only the formulas for the derivatives of the *basic* operators have to be derived (or programmed) and incorporated into the rules of differentiation arithmetic. From there, only *numbers* need to be manipulated in order to derive both the value *and* derivative of an arbitrary composite function.

2.4.2 Extension to higher order derivatives and multiple variables

The concept of an ordered pair arithmetic implementation of differentiation can be extended to higher order derivatives. As a next logical step, consider an ordered triplet $U=(u, u', u'')$, where u'' is associated with a second derivative. The rules (1)–(4) and (13) can easily be modified to handle an ordered triplet:

$$\text{Addition:} \qquad (u, u', u'')+(v, v', v'') = (u+v, u'+v', u''+v'') \tag{16}$$

$$\text{Subtraction:} \qquad (u, u', u'')-(v, v', v'') = (u-v, u'-v', u''-v'') \tag{17}$$

$$\text{Multiplication:} \qquad (u, u', u'')(v, v', v'') = (uv, uv'+vu', uv''+2u'v'+u''v) \tag{18}$$

$$\text{Division:} \qquad \frac{(u, u', u'')}{(v, v', v'')} = \left(\frac{u}{v}, \frac{vu'-uv'}{v^2}, \frac{v^2(vu''-uv'')-2vv'(vu'-uv')}{v^4} \right), v \neq 0. \tag{19}$$

$$\text{Function Composition:} \qquad f^*((u, u', u'')) = (f(u), u'f'(u), (u')^2 f''(u)+u''f'(u)). \tag{20}$$

For this case, $U=(u, 1, 0)$ represents the independent variable, and $C=(c, 0, 0)$ is a constant.

As an example of the application of Eqs. (16)–(19), consider again Eq. (6). It can easily be shown that:

$$r''(x) = \frac{462x+154}{(3x+1)^4}, \tag{21}$$

and, by substitution, $r''(3)=0.154$. By using Eqs. (16)–(19) and (6), one can alternatively find:

$$r^*((3, 1, 0)) = \frac{(2, 0, 0)(3, 1, 0)(3, 1, 0)-(4, 0, 0)(3, 1, 0)+(7, 0, 0)}{(3, 0, 0)(3, 1, 0)+(1, 0, 0)}$$

$$= \frac{(2, 0, 0)(9, 6, 2)-(12, 4, 0)+(7, 0, 0)}{(9, 3, 0)+(1, 0, 0)} = \frac{(18, 12, 4)+(-5, -4, 0)}{(10, 3, 0)}$$

$$= \frac{(13, 8, 4)}{(10, 3, 0)} = (1.3, 0.41, 0.154).$$

To show the use of Eq. (20), let us evaluate Eq. (14) via ordered triplets, with $U=(1, 1, 0)$:

$$h^*((1, 1, 0)) = \sin((3, 0, 0)(1, 1, 0)(1, 1, 0)+(4, 0, 0)(1, 1, 0)-(6, 0, 0))$$

$$= \sin((3, 0, 0)(1, 2, 2)+(4, 4, 0)-(6, 0, 0))$$

$$= \sin((3, 6, 6)+(-2, 4, 0))$$

$$= \sin((1, 10, 6))$$

$$= (\sin(1),\ 10\cos(1),\ -100\sin(1)+6\cos(1))$$

$$\simeq (0.84147098,\ 5.4030231,\ -80.905285).$$

Alternatively, from Eq. (15), it can be shown that

$$h''(x) = -(6x+4)^2 \sin(3x^2+4x-6) + 6\cos(3x^2+4x-6), \tag{22}$$

and, $h''(1) \simeq -80.905285$.

One can also extend automatic differentiation to multivariate functions. The simplest case is that of two independent variables, where the appropriate arithmetic is with ordered triplets $U = (u, u_1, u_2)$, in which u_i represents a first order partial derivative with respect to the i^{th} independent variable. The rules of the arithmetic [analogous to Eqs. (16) – (20)] are:

Addition: $$(u, u_1, u_2) + (v, v_1, v_2) = (u+v, u_1+v_1, u_2+v_2) \tag{23}$$

Subtraction: $$(u, u_1, u_2) - (v, v_1, v_2) = (u-v, u_1-v_1, u_2-v_2) \tag{24}$$

Multiplication: $$(u, u_1, u_2)(v, v_1, v_2) = (uv, uv_1+vu_1, uv_2+vu_2) \tag{25}$$

Division: $$\frac{(u, u_1, u_2)}{(v, v_1, v_2)} = \left(\frac{u}{v}, \frac{vu_1 - uv_1}{v^2}, \frac{vu_2 - uv_2}{v^2}\right), v \neq 0. \tag{26}$$

Function Composition: $$f^*((u, u_1, u_2)) = (f(u), u_1 f'(u), u_2 f'(u)) \tag{27}$$

In this case $U = (u, 1, 0)$ and $V = (v, 0, 1)$ represent the two independent variables, and $C = (c, 0, 0)$ is a constant.

As an example, consider:

$$r(x,y) = \frac{3x^2+17xy-8y^2+3}{7x-4}, \tag{28}$$

$$\frac{\partial}{\partial x} r(x,y) = \frac{21x^2+56y^2-24x-68y-21}{(7x-4)^2}, \tag{29}$$

$$\frac{\partial}{\partial y} r(x,y) = \frac{17x-16y}{7x-4}, \tag{30}$$

and, substituting $(x, y) = (2, 3)$:

$$r(2, 3) = 4.5$$

$$\frac{\partial}{\partial x} r(2, 3) = 3.15$$

$$\frac{\partial}{\partial y} r(2,3) = -1.4.$$

Using ordered triplet notation:

$$r^*((2,1,0),(3,0,1)) = \frac{\{(3,0,0)(2,1,0)(2,1,0)+(17,0,0)(2,1,0)(3,0,1) - (8,0,0)(3,0,1)(3,0,1)+(3,0,0)\}}{(7,0,0)(2,1,0)-(4,0,0)}$$

$$= \frac{(3,0,0)(4,4,0)+(17,0,0)(6,3,2)-(8,0,0)(9,0,6)+(3,0,0)}{(14,7,0)-(4,0,0)}$$

$$= \frac{(12,12,0)+(102,51,34)-(72,0,48)+(3,0,0)}{(10,7,0)}$$

$$= \frac{(45,63,-14)}{(10,7,0)} = (4.5, 3.15, -1.4).$$

From the above examples, one can see that, conceptually, automatic differentiation is extendible to arbitrary order partial derivatives of functions of an arbitrary number of independent variables. The only practical difficulty is developing a method to calculate the sum, difference, product, quotient, and function formulas for the higher order/multivariate terms.

We now introduce the differential operator $D_{\mathbf{s}}$. The index **s** is a vector that represents the indices of the independent variables that constitute the derivative. For example, let $\mathbf{s}=\{2,3,3,5\}$, and let x_2, x_3, and x_5 be independent variables. Then, the following notations are equivalent:

$$D_{\mathbf{s}}(f) \equiv D_{2335}(f) \equiv f_{2335} \equiv \frac{\partial^4}{\partial x_2 \partial x_3^2 \partial x_5}(f)$$

The formula for a generalized derivative of a sum (or difference) is:

$$D_{\mathbf{s}}(f \pm g) = f_{\mathbf{s}} \pm g_{\mathbf{s}}. \tag{31}$$

Since a quotient is merely an inversion operation (function) followed by a product, then the only operations left are product and function.

The following example illustrates the taking of successive derivatives of a product:

$$D_{334}(fg) \equiv \frac{\partial^3}{\partial x_3^2 \partial x_4}(fg)$$

$$= D_{33}(fg_4 + f_4 g) = D_3(fg_{34} + f_3 g_4 + f_4 g_3 + f_{34} g)$$

$$= fg_{334} + 2\,(f_3 g_{34}) + f_{33} g_4 + f_4 g_{33} + 2\,(f_{34} g_3) + f_{334} g. \tag{32}$$

In general, then:

$$D_{\mathbf{s}}(fg) = \sum_{\mathbf{p} \cup \mathbf{q} = \mathbf{s}} C_{\mathbf{p}}\,(f_{\mathbf{p}} g_{\mathbf{q}}), \tag{33}$$

where **s** is the disjoint union of **p** and **q**, and $C_{\mathbf{p}}$ is the number of distinct ways **p** is a subset of **s**.

The next example illustrates taking successive derivatives of a function $F: R \to R$:

$$D_{334}(F(g)) \equiv \frac{\partial^3}{\partial x_3^2 \partial x_4}(F(g))$$

$$= D_{33}(g_4 F'(g)) = D_3(g_{34} F'(g) + g_4 g_3 F''(g))$$

$$= F'(g)\, g_{334} + F''(g)\,(2\, g_3 g_{34} + g_4 g_{33}) + F'''(g)\, g_3 g_3 g_4, \tag{34}$$

where

$$F^{(i)}(g) \equiv \frac{\partial^i}{\partial \xi^i}[F(\xi)].$$

In general, the formula for the derivative of a composite function is:

$$D_{\mathbf{s}}[F(g)] = \sum_{i=1}^{m} F^{(i)}(g) \sum_{\mathbf{p}} C_{\mathbf{p}}{}' \prod_{k=1}^{i} g_{\mathbf{t}(\mathbf{p},k)} \tag{35}$$

where m is the order of the derivative (equal to the length of the vector **s**) and **t** is a function that returns an index vector according to the values of **p** and k.

Thus, Eqs. (31), (33), and (35) are sufficient to develop an "ordered tuplet" implementation of differentiation, given that the appropriate version of these equations are applied to each term of the tuplet.

2.4.3 Relationship to Power Series Manipulation

Assume that one is given two power series:

$$U(x) = u_0 + u_1 x + u_2 x^2 + \ldots \qquad V(x) = v_0 + v_1 x + v_2 x^2 + \ldots$$

Using the techniques of power series manipulation (see, e.g., Knuth [18]) one can find the sum, difference, product, and quotient of these two series, as well as find the function of a series ($f[U(x)]$). If U and V are sufficiently well behaved, then U and V can be expanded in a Taylor series, where the coefficients u_i and v_i are related to derivatives as shown in the following equations:

$$U(x) = \sum_{i=0}^{\infty} \frac{1}{i!} \frac{\partial^i U}{\partial x^i}\bigg|_{x=x_0} (x-x_0)^i; \qquad u_i = \frac{1}{i!} \frac{\partial^i U}{\partial x^i}\bigg|_{x=x_0},$$

$$V(x) = \sum_{j=0}^{\infty} \frac{1}{j!} \frac{\partial^j V}{\partial x^j}\bigg|_{x=x_0} (x-x_0)^j; \qquad v_j = \frac{1}{j!} \frac{\partial^j V}{\partial x^j}\bigg|_{x=x_0}.$$

Given these relationships, one can use the formalism of power series manipulation to derive relationships (or formulas) for the various derivatives. As examples, given below are the summation and product formulas for power series:

Summation: $$W(x) = U(x) \pm V(x) \Rightarrow w_i = u_i \pm v_i \tag{36}$$

Product: $$W(x) = U(x)V(x) \Rightarrow w_i = \sum_{k=0}^{i} u_k v_{i-k} \tag{37}$$

Eq. (36) implies that

$$\frac{\partial^i W}{\partial x^i}\bigg|_{x=x_0} = \frac{\partial^i U}{\partial x^i}\bigg|_{x=x_0} + \frac{\partial^i V}{\partial x^i}\bigg|_{x=x_0}, \tag{38}$$

which is equivalent to Eq. (31). Eq. (37) can be rewritten:

$$\frac{1}{i!}\frac{\partial^i W}{\partial x^i}\bigg|_{x=x_0} = \sum_{k=0}^{i} \frac{1}{(i-k)!k!} \frac{\partial^k U}{\partial x^k}\bigg|_{x=x_0} \frac{\partial^{i-k} V}{\partial x^{i-k}}\bigg|_{x=x_0},$$

or,

$$\frac{\partial^i W}{\partial x^i}\bigg|_{x=x_0} = \sum_{k=0}^{i} \binom{i}{k} \frac{\partial^k U}{\partial x^k}\bigg|_{x=x_0} \frac{\partial^{i-k} V}{\partial x^{i-k}}\bigg|_{x=x_0}, \tag{39}$$

which is equivalent to Eq. (33) for the univariate case of automatic differentiation.

Since there is a clear relationship between the theory of power series manipulation and the theory of automatic differentiation, it follows that some of the analytic techniques in power series manipulation may be applicable to automatic differentiation. For example, in the case of function composition, if the function f is the solution of a differential equation, such as sin, cos, $1/x$, etc., then there are recursive techniques that can be used to calculate the resulting coefficients w_i. For more detailed theory on power series manipulation, the reader is referred to [18] for Knuth's discussion of univariate series, and to [16]–[17] for Forbes' discussion on multivariate series. This is an area that has not been investigated in this thesis, but is suggested as a potential enhancement in §7.

3. DEVELOPMENT AND IMPLEMENTATION SYSTEM

As indicated in the Introduction, a good implementation of the concept of DIFFERENTIALS would allow one to define (1) an abstract data type called DIFFERENTIAL, and (2) functions that operate on this data type. The Ada programming language was chosen to implement DIFFERENTIALS because: (1) It supports data abstraction and function overloading, (2) it supports good software engineering practices such as separate compilation, library management, strong type checking, and package development, and (3) it is available on the RIT computer systems.

Initial development of the DIFFERENTIALS package took place on the RIT Computer Science VAX 11/780 computer CINEVAX (running under UNIX), and was written in Ada using Version 1.5 of the TeleSoft Ada compiler. Unfortunately, V1.5 was *not* a full implementation of Ada, and numerous problems were encountered in trying to use some of the software engineering oriented features of Ada, such as private data types, separate compilation, function overloading, etc. In addition, V1.5 only had a single predefined floating point type called FLOAT, with 6 digits of precision. These problems were sufficiently severe that development of this software was delayed until the release of TeleSoft's second generation Ada compiler.

The TeleGen2 VMS Ada compiler, TeleSoft's second generation Ada compiler, was received at RIT during July, 1986, and put up on the RIT ISC VAX/8600 computer VAXA, running under VMS. The current implementation of DIFFERENTIALS was developed using this compiler. No deviations from the Ada language standard[19] were noted during the software development.

Since the TeleGen2 Ada compiler did not come with a real functions package, and part of the DIFFERENTIALS package involved using sines/cosines/sqrts, etc., a real function package had to be developed. TeleGen2 Ada provided a method to interface to FORTRAN routines, so a small amount of code was written in FORTRAN 77 and compiled using the VMS FORTRAN compiler in order to provide the real functions needed.

4. FUNCTIONAL SPECIFICATION

When developing a package in Ada, the package specification is used to define the functions, variables, etc., that are visible to the user of the package. In this sense, the package specification is a necessary, and in most cases sufficient, part of the functional specification. Given below, therefore, is a detailed description of each element in the package specifications. First the DIFFERENTIALS package will be described, and then, since a real function package had to be developed, the REALFUNC package will be described.

4.1 DIFFERENTIALS PACKAGE

Listed in Fig. 4.1 is the complete specification to the DIFFERENTIALS package. It is composed of the following elements: (1) Required packages, (2) constants, (3) types, (4) exceptions, and (5) functions and procedures, which are described in detail below:

Required Packages

text_io; float_text_io
: These are supplied with the TeleGen2 Ada system, and are used to print out text and floating point numbers.

realfunc
: Some real function package is required. TeleGen2 Ada does not provide one, so this was developed using FORTRAN and is described in §4.2.

Constants

max_level
: The maximum order of a derivative that one can specify (e.g., $\partial^4/\partial x_1 \partial x_3{}^2 \partial x_6$ has order = 4).

max_length
: The maximum number of components of a DIFFERENTIAL (i.e., the maximum number of derivatives that may be calculated at any one time).

Types

DIFFERENTIAL
: A collection of the values of the various partial derivatives of the dependent variable with respect to the independent variable(s) (including the zeroth derivative). Each value is of type FLOAT.

LIST
: A data type that is used to refer to a specific derivative. A LIST is a vector of numbers, each number representing an independent variable. As an example, to specify the following derivative:

```
------------------------------------------------------------------------
------------------------------------------------------------------------

with text_io; use text_io;
with float_text_io; use float_text_io;
with realfunc; use realfunc;

package DIFFERENTIALS is

-- constants

   max_level: constant INTEGER := 11;        -- maximum order of a derivative
   max_length: constant INTEGER := 125;      -- maximum # of derivatives at once

-- type definitions

   type DIFFERENTIAL is private;
   type LIST is array(1..(max_level+1)) of INTEGER;

-- exceptions

   TOO_MANY_DIFFS: exception;                -- too many derivatives have been
                                             --   specified or are needed
   LIST_ERROR: exception;                    -- illegal LIST construction (last
                                             --   element /= 0)
   LIST_NOT_IN_NAMES: exception;             -- an uninitialized derivative has
                                             --   been requested
   DIFF_MATH_ERROR: exception;               -- math error such as divide by 0
                                             --   or sqrt of negative number

-- binary functions (in "diffbody.ada")

   function "*"(D,E: DIFFERENTIAL) return DIFFERENTIAL;
   function "*"(x: FLOAT; D: DIFFERENTIAL) return DIFFERENTIAL;
   function "*"(D: DIFFERENTIAL; y: FLOAT) return DIFFERENTIAL;
   function "+"(D,E: DIFFERENTIAL) return DIFFERENTIAL;
   function "+"(x: FLOAT; D: DIFFERENTIAL) return DIFFERENTIAL;
   function "+"(D: DIFFERENTIAL; y: FLOAT) return DIFFERENTIAL;
   function "-"(D,E: DIFFERENTIAL) return DIFFERENTIAL;
   function "-"(x: FLDAT; D: DIFFERENTIAL) return DIFFERENTIAL;
   function "-"(D: DIFFERENTIAL; y: FLOAT) return DIFFERENTIAL;
   function "/"(D,E: DIFFERENTIAL) return DIFFERENTIAL;
   function "/"(x: FLOAT; D: DIFFERENTIAL) return DIFFERENTIAL;
   function "/"(D: DIFFERENTIAL; y: FLOAT) return DIFFERENTIAL;

-- unary functions (in "diffbody.ada")

   function partial(D: DIFFERENTIAL; i: INTEGER) return DIFFERENTIAL;
   function sin(D: DIFFERENTIAL) return DIFFERENTIAL;
   function cos(D: DIFFERENTIAL) return DIFFERENTIAL;
   function exp(D: DIFFERENTIAL) return DIFFERENTIAL;
   function sqrt(D: DIFFERENTIAL) return DIFFERENTIAL;
   function recip(D: DIFFERENTIAL) return DIFFERENTIAL;
```

Fig 4.1 DIFFERENTIAL Package Specification

```
-- element access (in "diffbodya.ada")

   procedure dump_diff(R: in DIFFERENTIAL);
   procedure put(D: in DIFFERENTIAL; l: in LIST);
   function val0(D: DIFFERENTIAL) return FLOAT;
   function val(D: DIFFERENTIAL; l: LIST) return FLOAT;

-- Initialization (in "diffbodya.ada")

   procedure Set_Names(l: in LIST);
   procedure Set_Diff(D: in out DIFFERENTIAL; value,dval: in FLOAT;
                      i: in INTEGER);
   function Const_Diff(value: FLOAT) return DIFFERENTIAL;
   procedure Init_Diffs;

-- Debugging (in "diffbodya.ada")

   procedure set_debug;
   procedure clear_debug;
   procedure dump_order_table;
   procedure dump_times_table;
   procedure dump_funct_table;
   procedure dump_names;

-- private type

private
   type DIFFERENTIAL is array (1..max_length) of FLOAT;

end DIFFERENTIALS;
```

Fig 4.1, con't

$$\frac{\partial^4}{\partial x_1 \partial x_3^2 \partial x_6},$$

the associated LIST would be: {1,3,3,6,0,...,0}, where the total length of the LIST is max_level+1, and the final element *has* to be equal to 0 by definition of this data type. The zeroth derivative would be represented by {0,0,...,0}.

Exceptions

TOO_MANY_DIFFS

This exception will be raised if the user (either by calling Set_Names or Init_Diffs) has effectively specified more than "max_length" derivatives to be included with each DIFFERENTIAL.

LIST_ERROR

The last element of a LIST does not equal 0.

LIST_NOT_IN_NAMES

A derivative has been requested that hasn't been specified explicitly by Set_Names or implicitly by Init_Diffs.

DIFF_MATH_ERROR

Divide by 0 or square root of a negative number detected.

Functions and Procedures

Initialization

procedure Set_Names(l: in LIST)

This procedure is used to tell the system that the partial derivative represented by the LIST l should be included as part of a DIFFERENTIAL.

procedure Init_Diffs

This procedure must be called *after* all calls to Set_Names, and *before* any other procedure or function call. It constructs some internal tables that are necessary to manipulate DIFFERENTIALS. As part of that procedure, if some higher order derivatives have been specified via Set_Names, it automatically determines what lower order derivatives are necessary in order to manipulate DIFFERENTIALS, and includes them as part of a DIFFERENTIAL. It is during this procedure that it is most likely that the exception TOO_MANY_DIFFS will be raised.

procedure Set_Diff(D: in out DIFFERENTIAL; value,dval: in FLOAT; i: in INTEGER)

This procedure initializes a DIFFERENTIAL D by setting its initial value to "value" and setting the first partial derivative with respect to the i^{th} independent variable to "dval". This is used to associate a user's program variable with the implied i^{th} independent variable. All higher derivatives are set equal to 0. Typically dval is set to 1.0, though this is not necessary.

function Const_Diff(value: FLOAT) return DIFFERENTIAL

This function returns a constant DIFFERENTIAL (all derivatives = 0), whose zeroth derivative (or value) is set to value.

Arithmetic Functions

```
function "*"(D,E: DIFFERENTIAL) return DIFFERENTIAL
function "*"(x: FLOAT; D: DIFFERENTIAL) return DIFFERENTIAL
function "*"(D: DIFFERENTIAL; y: FLOAT) return DIFFERENTIAL
function "+"(D,E: DIFFERENTIAL) return DIFFERENTIAL
function "+"(x: FLOAT; D: DIFFERENTIAL) return DIFFERENTIAL
function "+"(D: DIFFERENTIAL; y: FLOAT) return DIFFERENTIAL
function "-"(D,E: DIFFERENTIAL) return DIFFERENTIAL
function "-"(x: FLOAT; D: DIFFERENTIAL) return DIFFERENTIAL
function "-"(D: DIFFERENTIAL; y: FLOAT) return DIFFERENTIAL
function "/"(D,E: DIFFERENTIAL) return DIFFERENTIAL
function "/"(x: FLOAT; D: DIFFERENTIAL) return DIFFERENTIAL
function "/"(D: DIFFERENTIAL; y: FLOAT) return DIFFERENTIAL
function sin(D: DIFFERENTIAL) return DIFFERENTIAL
function cos(D: DIFFERENTIAL) return DIFFERENTIAL
function exp(D: DIFFERENTIAL) return DIFFERENTIAL
function sqrt(D: DIFFERENTIAL) return DIFFERENTIAL
```

The dyadic functions (*, +, −, ÷) take 2 arguments, at least one of which is a DIFFERENTIAL, and return a DIFFERENTIAL. The monadic functions (sin, cos, exp, sqrt) take a DIFFERENTIAL

and return a DIFFERENTIAL. They use the rules of differentiation arithmetic as described in §2.4.

`function recip(D: DIFFERENTIAL) return DIFFERENTIAL`

This is the reciprocal, or $1/x$, function.

`function partial(D: DIFFERENTIAL; i: INTEGER) return DIFFERENTIAL`

This function allows one to take the partial derivative of a DIFFERENTIAL with respect to the i^{th} independent variable. The returned coefficients are merely the argument coefficients shifted by i, so the value of the resulting DIFFERENTIAL equals the first partial derivative with respect to the i^{th} independent variable of the argument D, etc.

Element Access

`procedure dump_diff(R: in DIFFERENTIAL)`

This procedure prints to STDOUT all coefficients of the DIFFERENTIAL R in the order that the coefficients of R are stored internally. This is not as useful as the next two functions.

`function val(D: DIFFERENTIAL; l: LIST) return FLOAT`

This function returns the value of the derivative specified by the LIST l in D.

`function val0(D: DIFFERENTIAL) return FLOAT`

This function returns the zeroth derivative (or value) of the DIFFERENTIAL D.

`procedure put(D: in DIFFERENTIAL; l: in LIST)`

This procedure prints to STDOUT the value of the derivative specified by l in D. This routine calls put(FLOAT), so the FORE, AFT, and EXP fields are defaulted to DEFAULT_FORE, DEFAULT_AFT, and DEFAULT_EXP, respectively.

Debugging

```
procedure set_debug
procedure clear_debug
procedure dump_order_table
procedure dump_times_table
procedure dump_funct_table
procedure dump_names
```

These functions are visible because they are required for debugging purposes. They are *not* intended to be called by the casual user.

4.2 REALFUNC PACKAGE

Fig. 4.2 is the complete specification for the REALFUNC package associated with the DIFFERENTIALS package.

Required Packages

`System`

Supplied by the TeleGen2 Ada system.

Functions and Procedures

```
function sin(arg: FLOAT) return FLOAT
function sin(arg: LONG_FLOAT) return LONG_FLOAT
function cos(arg: FLOAT) return FLOAT
```

```
------------------------------------------------------------------------
------------------------------------------------------------------------

with System; use System;

------------------------------------------------------------------------
-- Package Specification
------------------------------------------------------------------------
package realfunc is
   function sin(arg: FLOAT) return FLOAT;
   function sin(arg: LONG_FLOAT) return LONG_FLOAT;
   function cos(arg: FLOAT) return FLOAT;
   function cos(arg: LONG_FLOAT) return LONG_FLOAT;
   function exp(arg: FLOAT) return FLOAT;
   function exp(arg: LONG_FLOAT) return LONG_FLOAT;
   function sqrt(arg: FLOAT) return FLOAT;
   function sqrt(arg: LONG_FLOAT) return LONG_FLOAT;
end realfunc;
```

Fig 4.2 REALFUNC Package Specification.

```
function cos(arg: LONG_FLOAT) return LONG_FLOAT
function exp(arg: FLOAT) return FLOAT
function exp(arg: LONG_FLOAT) return LONG_FLOAT
function sqrt(arg: FLOAT) return FLOAT
function sqrt(arg: LONG_FLOAT) return LONG_FLOAT
```

These functions take an argument of either the type FLOAT or LONG_FLOAT, and return a result of the same type. For sin and cos, the argument is assumed to be in *radians*. Because the underlying implementation uses FORTRAN 77, the restrictions and characteristics of these functions are those of VMS FORTRAN 77.

5. SYSTEM DESIGN

5.1 INTERNAL DATA STRUCTURES

In the DIFFERENTIALS package there are a number of special data structures. The first is that of a DIFFERENTIAL itself. As indicated in the private section of the DIFFERENTIALS specification, a DIFFERENTIAL is an array of FLOATs. Each element of the array represents a particular partial derivative. As described in the functional specification, the user is not required to know which element corresponds to which derivative. In fact, in different instantiations of this package, a given partial derivative may be located at different element positions in the array. Since position is not an indicator of the level or composition of a derivative, there must be an auxiliary structure to keep track of that information. In this software, that is done using the "names" table.

The variable "names" is an array of LISTs; the length of the "names" array is the same as the length of a DIFFERENTIAL (max_length). As described previously, the LIST structure is used to denote a particular derivative; therefore, names is an array of LISTs that gives the identification of each derivative in a DIFFERENTIAL. Names is filled using the user-callable routines Set_Names and Init_Diffs. Since calls to Set_Names can occur in any order (as long as they precede a call to Init_Diffs), the individual order of partial derivatives may vary from program to program.

The most complicated structure in the program is that of the two tables funct_table and times_table. Looking at the formula for the function of a DIFFERENTIAL [Eq. (35) of §2.4.2] and the corresponding example [Eq. (34) of §2.4.2], one can see that one has to form a sum of products of two terms. The first term is the ith derivative of the function itself $F^{(i)}(g)$. The second term is the sum of the products of various terms of the input DIFFERENTIAL g. The index **t** indicates a particular partial derivative; which **t**'s are used in the evaluation of a derivative depend upon the order of the derivative (m) and which derivative D_s is being requested. The formula for the product of two DIFFERENTIALS [Eq. (33)] is slightly less complicated in that it is only a "sum of products"

Funct_table and times_table are complex linked list structures that are used to implement Eqs. (33) and (35) in the evaluation of DIFFERENTIALS. They are constructed *dynamically* by the Init_Diffs routine. Funct_table and times_table are both max_length long arrays; each element is constructed by Init_Diffs according to what derivative is positionally implied in the associated names table. For example, if element #10 of a DIFFERENTIAL represents the partial derivative {1,3,4,7,0, . .}, Init_Diffs constructs an entry in location #10 of funct_table and times_table that tells the "function" and "product" routines (such as "sin" and "*") how to calculate the {1,3,4,7,0, . . .} term of the resulting DIFFERENTIAL. Each entry is a linked list or tree, whose leaves contain various index numbers into the DIFFERENTIAL array that are to be used to calculate the function or product.

I have defined various auxiliary data types to help denote the particular elements of the linked list. First consider the function of a DIFFERENTIAL. Given below is Eq. (35) rewritten:

$$D_s[F(g)] = \sum_{i=1}^{m} F^{(i)}(g) \sum_{p} C_p' \prod_{k=1}^{i} g_{t(p,k)}$$

Each element of the linked list represents a different term of this equation. The reader should refer to Fig. 5.1 during the following discussion.

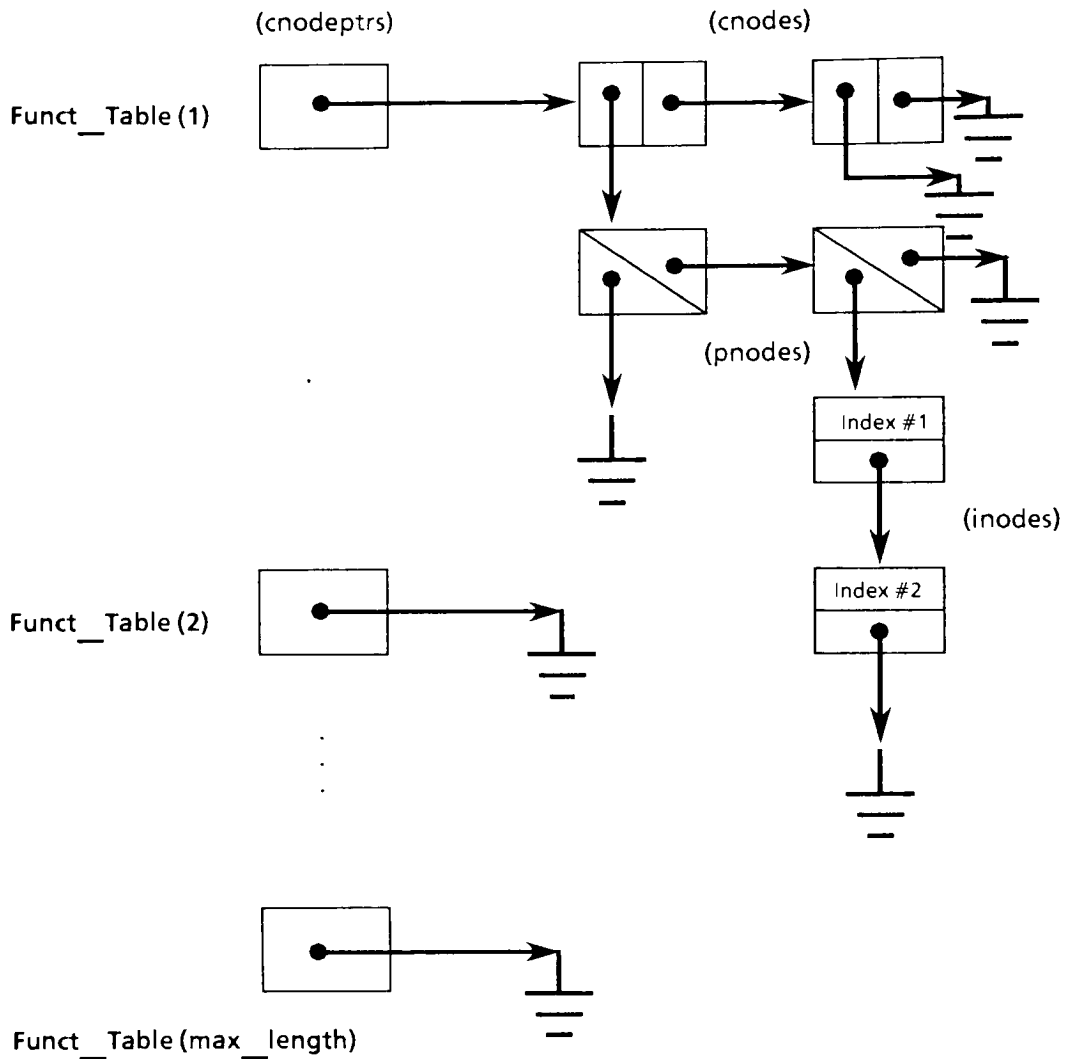

Fig 5.1 Structure diagram for the Funct structure

Linked List Components

inode

An inode consists of 2 fields: a pointer to another inode, and a val field of type INTEGER. The sublist consisting of a list of inodes represents the inner product

$$\prod_{k=1}^{i} g_{t(\mathbf{p},k)}$$

of Eq. (35); i.e., the procedure that finds the function of a DIFFERENTIAL knows to use the val field of each inode as an index into the input DIFFERENTIAL, and to multiply the appropriate terms together. In the example of taking a function of a DIFFERENTIAL [Eq. (34)], we see that sometimes a constant $C_{\mathbf{p}}$ multiplies this inner product. This represents more that one occurrence of a particular product. In the current implementation of DIFFERENTIALS, each occurrence of that product is calculated separately and represented by its own inode list, so the value $C_{\mathbf{p}}$ is not needed.

pnode

A pnode consists of two fields: a pointer to another pnode, and a pointer to an inode. The "value" of a pnode can be considered the product of the terms of the inode it points to. A list of pnodes represents a sum of terms; each term being represented by the value of a pnode.

cnode

A cnode consists of 2 fields: a pointer to another cnode, and a pointer to a pnode. The value of a list of cnodes gives the value of the appropriate derivative; it is a summation of the values of each individual cnode. The value of a cnode equals the value of the list of pnodes it points to times the value of the corresponding function derivative $F^{(i)}$; i.e., the first cnode value is multiplied by F', the second by F'', etc.

The relationships given above are best illustrated by an example. Fig 5.2 is the actual structure that would be constructed by Init_Diffs if the derivative {3,3,4,0, . . .} was encountered in names, assuming that the following assignments hold for a DIFFERENTIAL:

```
DIFF(0)  → {0,0,0,...}
DIFF(6)  → {3,0,0,...}
DIFF(7)  → {4,0,0,...}
DIFF(14) → {3,3,0,...}
DIFF(15) → {3,4,0,...}
DIFF(24) → {3,3,4,0,...}
```

This structure should be compared to Eq. (34), which is rewritten below:

$$D_{334}(F(g)) = F'(g)\, g_{334} + F''(g)\ (2\, g_3 g_{34} + g_4 g_{33}) + F'''(g)\, g_3 g_3 g_4$$

Now, consider Eq. (33):

$$D_{\mathbf{s}}(fg) = \sum_{\mathbf{p}\cup\mathbf{q}=\mathbf{s}} C_{\mathbf{p}}\, (f_{\mathbf{p}} g_{\mathbf{q}})$$

This can be represented by a single cnode pointing to a list of pnodes. Given in Fig. 5.3 is the entry in times_table that would be constructed by Init_Diffs (note that "duplicate" entries here are also calculated separately) for the example above. Note that all inode lists are only two elements long. The first index represents a term of the left-hand argument DIFFERENTIAL, and the second index represents a term of the right-hand argument DIFFERENTIAL. This example should be compared to Eq. (32), which is rewritten below

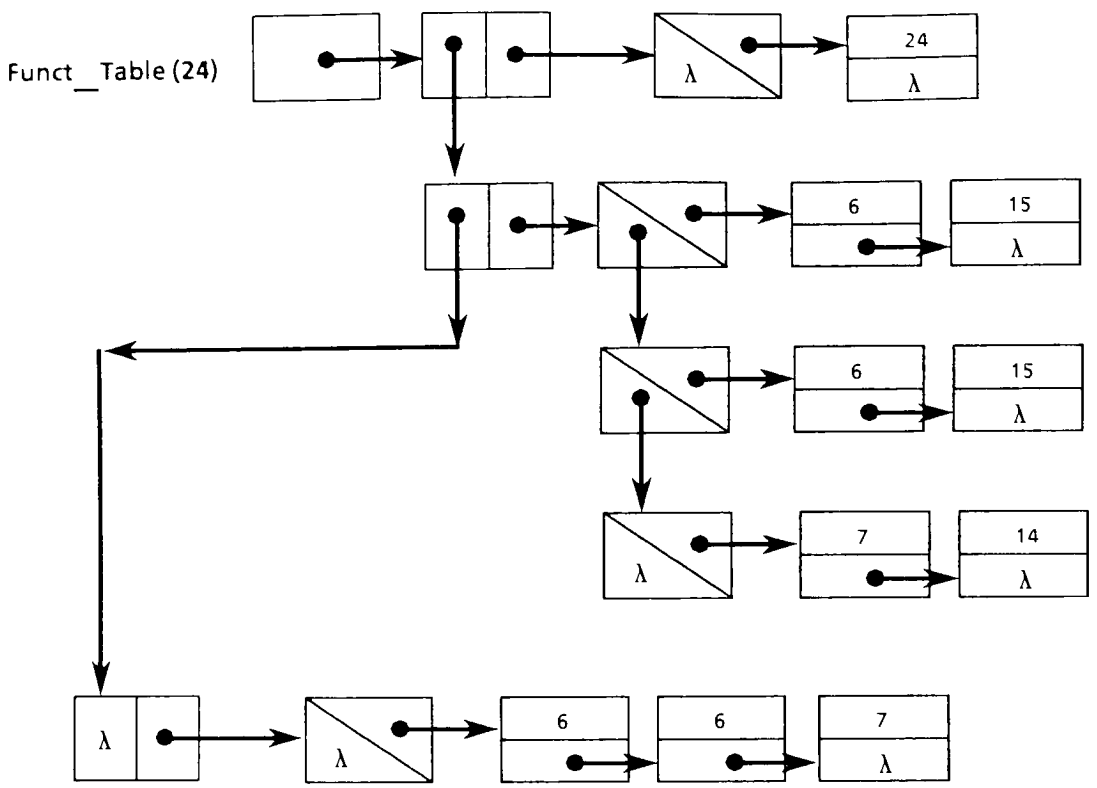

Fig 5.2 $D_{334}[F(g)]$

$$D_{334}(fg) = fg_{334} + f_3 g_{34} + f_3 g_{34} + f_{33} g_4 + f_4 g_{33} + f_{34} g_3 + f_{34} g_3 + f_{334} g.$$

As mentioned previously, these tables are constructed at run-time by `Init_Diffs`. These structures allow the user to have maximum flexibility in defining DIFFERENTIALS, but at the penalty of increased program space required. One can see that this data space will grow exponentially as the level of the desired derivatives increases.

The last internal data structure to be discussed is the `order_table`. In some implementations of DIFFERENTIALS (not this one), it may be necessary to calculate lower order derivatives before higher order ones. The `order_table` is provided if a future implementation wants to know a proper order for derivative evaluation. It is merely an array that is `max_length` long, consisting of a list of indices into a DIFFERENTIAL, sorted by increasing order of derivative.

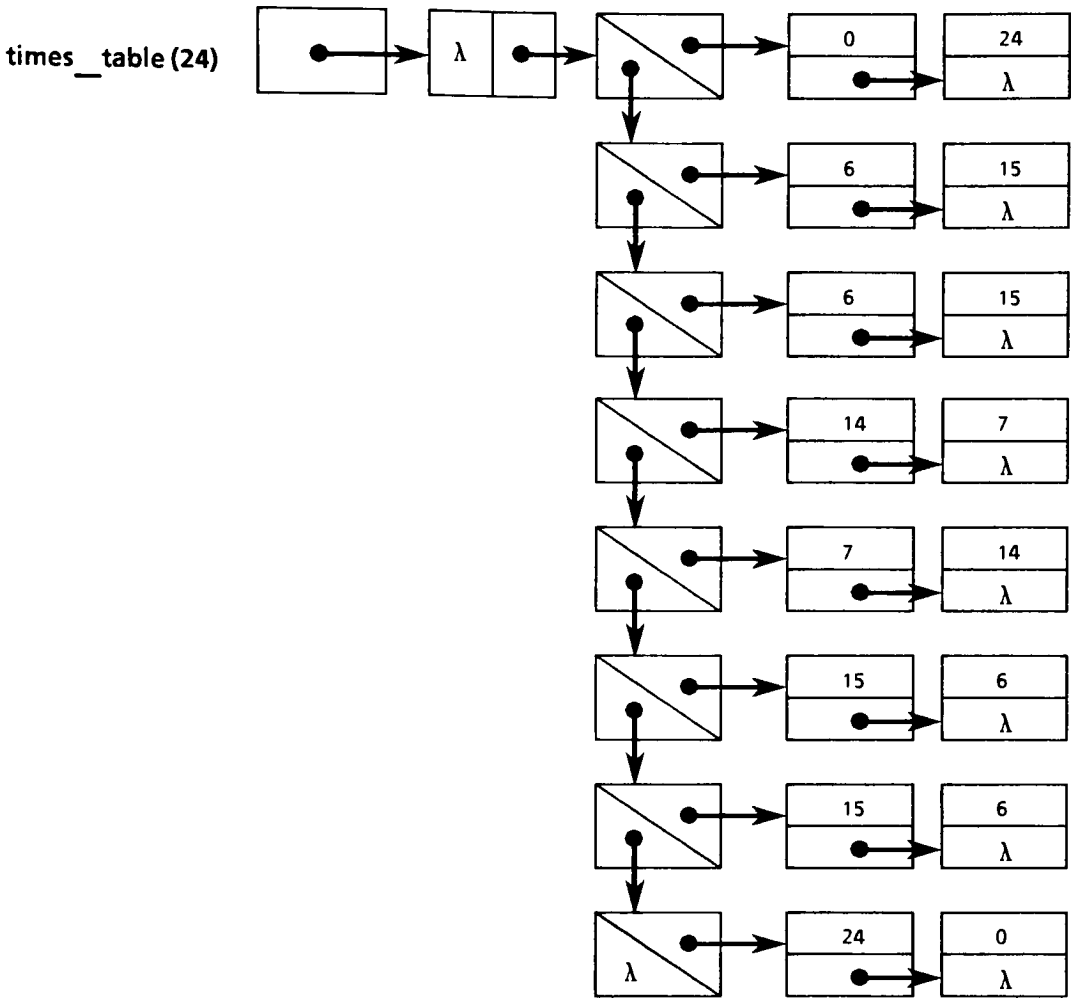

Fig 5.3 $D_{334}[fg]$

5.2 INTERNAL ROUTINE DESCRIPTIONS: DIFFERENTIALS

The internal routines (hidden from the user) are divided into three sections: (1) vector manipulation, (2) table manipulation, and (3) debugging. The reader should refer to the source code listings in §9.5 for the appropriate procedure listings.

5.2.1 Vector Manipulation

Since any particular derivative is specified by a `LIST` structure such as {3,3,4,0, . . .}, and the `Init_Diffs` routine needs to analyze and manipulate `LIST` structures in order to construct the `funct_table` and `times_table` entries, vector manipulation procedures need to be provided.

`function "+"(l,ri: LIST) return LIST`

This function concatenates two LISTS `l` and `ri`. The exception `LIST_INDEX_ERROR` is raised if the resulting LIST would be longer than `max_level` (the source code is in §9.4).

`function list_prepend(l: LIST; i: INTEGER) return LIST`

Returns a new LIST, composed of the INTEGER `i` as the first element, and the LIST `l` as the rest of the LIST.

`procedure dump_list(l: in LIST)`

Prints the elements of `l` to STOOUT.

`function as_list(i: INTEGER) return LIST`

Returns the LIST $\{i,0,\ldots,0\}$.

`function len(l: LIST) return INTEGER`

Returns the length of the LIST `l`, defined as the number of elements before the first occurrence of a 0.

`function equals(l,r: LIST) return BOOLEAN`

Returns TRUE if every element of `l` equals the corresponding one of `r`, otherwise, FALSE is returned.

`function car(l: LIST) return LIST`

Returns the LIST $\{l1,0,\ldots\}$, where `l1` is the first element of `l`.

`function cdr(l: LIST) return LIST`

Returns the LIST $\{l2,l3,\ldots,l(len(l)),0,\ldots\}$.

`function empty(l: LIST) return BOOLEAN`

Returns TRUE if the first element of `l` = 0, otherwise, returns FALSE.

`function max(i,j: INTEGER) return INTEGER`

Returns the larger of `i` and `j`.

5.2.2 Table Manipulation

`function names_lookup(l: LIST; insert_flag: BOOLEAN) return INTEGER`

This routine sees if the LIST `l` is located anywhere in the `names` table. If it is, it returns the array index value where the LIST can be found. If not, then the exception `LIST_NOT_IN_NAMES` is raised, unless the `insert_flag` is TRUE, in which case it tries to append the LIST `l` into the `names` table. If successful, it returns the new location; if not, the exception `TOO_MANY_DIFFS` is raised.

`procedure Init_times_table`

This procedure is called by `Init_Diffs`. It constructs the `times_table` as described in §5.1 using the recursive procedure `gen_times` that simulates the process of finding the derivative of a product.

`procedure Init_funct_table`

This routine is called by `Init_Diffs`. It constructs the `funct_table` as described in §5.1 using the complex recursive procedure `gen_funct` that simulates the process of finding the derivative of a function.

procedure order_tables

This procedure generates the entries in the order_table.

5.2.3 Debugging

procedure set_debug

There is a local BOOLEAN variable in the DIFFERENTIALS package called DEBUG. If set to TRUE, then after each DIFFERENTIAL operation (+,−,*,÷,sin,cos,etc.), the values of *all* elements of the argument and answer DIFFERENTIALS are output to STDOUT. This flag is initialized to FALSE. Set_Debug set the value of DEBUG to TRUE.

procedure clear_debug

This procedure sets DEBUG to FALSE.

procedure dump_order_table
procedure dump_times_table
procedure dump_funct_table
procedure dump_names

These procedures dump to STDOUT the elements of each of these tables.

5.3 INTERNAL ROUTINE DESCRIPTIONS: REALFUNC

The REALFUNC package provides the functions sine, cosine, exponential, and square root (of both FLOAT and LONG_FLOAT arguments) to any Ada program. They are actually implemented using the Ada Interface Pragma, specifying a FORTRAN routine that should be called. The routines sin, cos, exp, and sqrt first convert their arguments to LONG_FLOAT if required. Then, the *addresses* of these arguments (obtained via the Ada 'ADDRESS attribute) are passed to the FORTRAN routines FSIN, FCOS, FEXP, and FSQRT, respectively. The results are converted back to FLOAT (if appropriate) and then returned to the Ada client procedure. An example of the TeleSoft Ada definition required to do this is:

```
procedure FSQRT (farg,fresIt: in System'Address);
pragma Interface (VMS,FSQRT);
```

In addition, the proper linking of the object code generated by the FORTRAN compiler is required. TeleGen2 Ada provides the appropriate linking instructions to do this. The Ada source code can be found in §9.1, and the FORTRAN source code can be found in §9.2.

6. APPLICATION TO OPTICS

To truly exercise the automatic differentiation software, one should try to come up with an application that involves higher order derivatives of multivariate functions. In addition, one should have an alternate method of calculating the derivatives so that comparisons can be made between the results of automatic differentiation and the other calculational method.

It turns out that the tracing of light rays through a series of lenses is an ideal area of application for automatic differentiation. The equations that govern the path of the rays are algebraically straightforward, involving only the arithmetic operators, sin, cos, and sqrt. They are sufficiently complex so that for all but the simplest cases one cannot calculate symbolically the derivatives of the ray tracing "function". The aberrations, or imperfections of the lens system (such as astigmatism, distortion, etc.), are related to third and fifth order partial derivatives of the ray tracing "function" However, optical engineers have been designing lenses for over 100 years, and, given certain assumptions about rotational symmetry in the lens, a methodology has been developed that can be used to calculate these aberrations without the need to actually do a symbolic differentiation. Thus, ray tracing of lens systems and aberration theory provide a sufficiently complex series of partial derivatives that can be calculated via automatic differentiation, and it provides a means to check those results.

In §6.1 a very brief tutorial on the theory of geometrical optics (ray tracing) is given. For a more detailed discussion, the reader is referred to [20] and [21]. Section 6.2 gives a detailed description of an OPTICS package that implements some standard optics ray tracing software, and §6.3 gives several examples of programs that use the OPTICS and DIFFERENTIALS packages to calculate various lens system characteristics that are related to partial derivatives. A comparison of the automatic differentiation results is made with the results of classical lens aberration analysis.

6.1 TUTORIAL

6.1.1 Light Rays

When a pebble is dropped into a pond, a series of expanding circular waves are produced. The wavelets are evenly spaced, and travel outwards at some velocity. In many respects, the behavior of light is very similar to this. The light that is emitted from a hypothetical point source has a spherical shape, with a specific wavelength and velocity. For visible light, the wavelength varies from 0.4μ (violet light) to 0.7μ (red light). [One micron, or μ, is one-one millionth of a meter, or about forty-millionths of an inch.] In a vacuum, the speed of light is about 3×10^8 m/sec, or about 186,400 miles per second. In other materials (such as glass), the speed of light is always less than that in a vacuum. The ratio of the speed of light in a vacuum to that in a given material is called the index of refraction N of the material. For common glass, N is about 1.5 (i.e., light travels about two-thirds as fast in glass as in a vacuum).

If one connects corresponding points on successive wavefronts, as shown in Fig 6.1, one sees that a straight line is formed. This path is a so-called "light ray". It is a *very* useful, though fictitious, construct for understanding how light travels through materials.

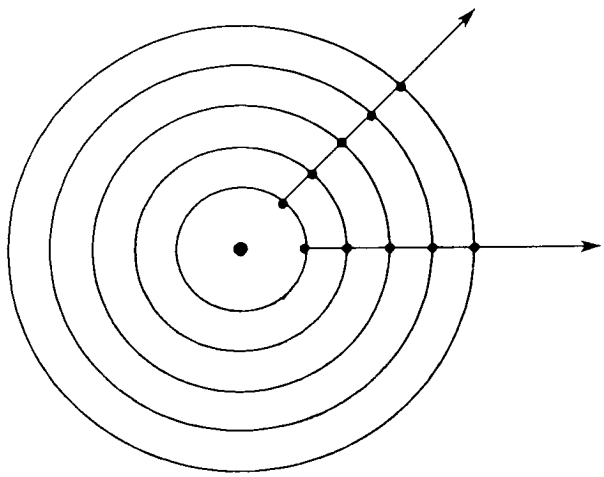

Fig 6.1 Light Ray Construct

In any given homogeneous material, the light rays travel in straight lines. At a boundary between two different materials, the light rays bend. If the angle of incidence of the ray measured with respect to the normal to the boundary is I_1, then the exiting angle I_2 of the ray measured with respect to the boundary normal is given by (see Fig. 6.2):

$$\sin I_2 = \frac{N_1}{N_2} \sin I_1. \tag{1}$$

This is "Snell's Law". Note that the boundary does *not* have to be flat, but can be curved. As long as the ray incidence angle is measured with respect to the perpendicular to the surface, Snell's Law can be used to calculate how the ray bends. In fact, Eq. (1) is the only optics-specific formula needed for ray tracing; all other formulas involve 3D geometry and the calculation of the intersection of lines in space with spheres. These will be given in §6.1.3.

6.1.2 Lens Fundamentals

A piece of glass on which one or two curved surfaces have been formed is called a *lens element*, or more simply, a lens. However, a collection of lens elements can also be called a lens (or lens system), so the use of the term *lens*, unless otherwise noted, will refer to *both* types of lenses.

The purpose of a lens is to take a particular distribution of light in its so-called *object plane*, and form another distribution of light in its so-called *image plane*. The simplest case is that of a point source, as illustrated in Fig 6.3. The light rays that emanate from the point

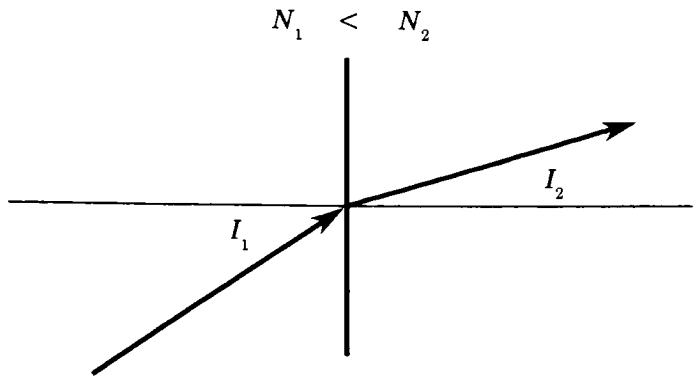

Fig 6.2 Illustration of Snell's Law

source A (in the object plane) are focused to another point source A′ in the image plane. The so-called *optical axis* is the axis of rotational symmetry of the lens. If the point source A is moved off the axis to a new point B, the image A′ moves to a location B′ A generalized object in the object plane acts as a collection of point sources; it is this collection of point sources that is imaged in the image plane. [For example, when a person's face is photographed, each part of the face reflects light that is incident on it, and scatters light towards the lens. Each part thus acts as a point source.] Note that the length A′B′ does not necessarily equal the length AB; the ratio:

$$\frac{\overline{A'B'}}{\overline{AB}}$$

is called the lens *magnification.*

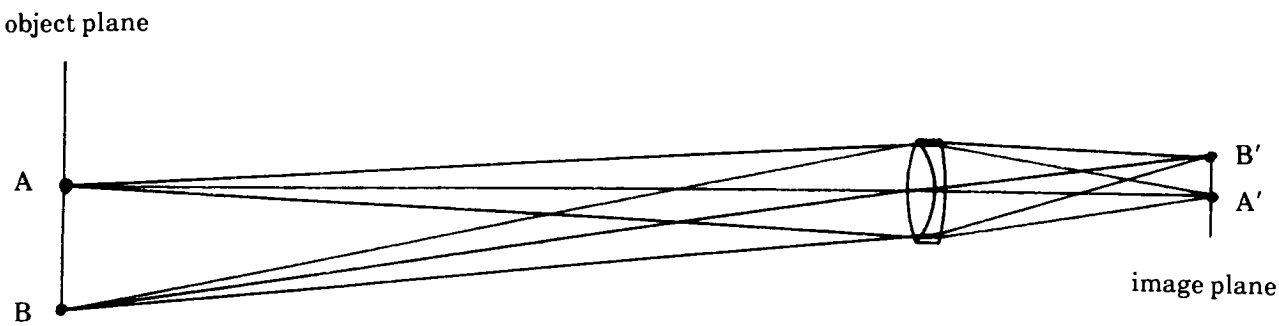

Fig 6.3 Illustration of lens imaging

In reality, the rays that leave a point source in the object plane do *not* intersect as a perfect point image; there is some small spread of the light rays. This is illustrated in Fig. 6.4; the measure of the spreading of the light rays is called the lens *aberration*. Typically, in a single lens element, the rays that make a large angle with respect to the optical axis (the axis of symmetry; e.g., the line AA′ in Fig. 6.3) are focused *closer* to the lens than those rays close to the axis. These aberrations tend to make an image look "fuzzy" rather than sharp.

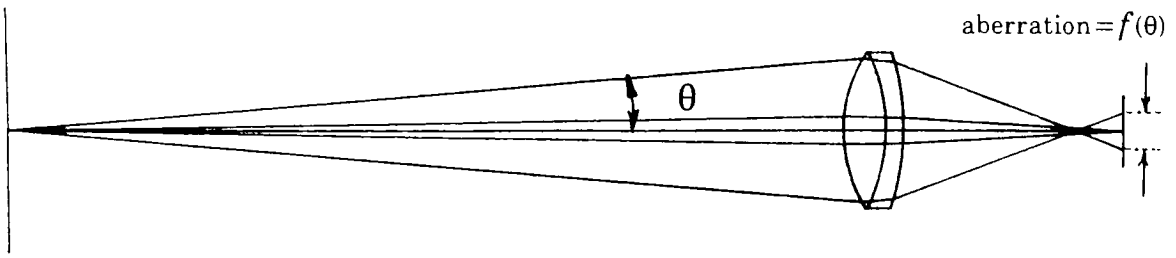

Fig 6.4 Lens exhibiting aberrations

In Fig. 6.4, one can see that the intersection of a ray with the image plane is a function of the angle θ the ray makes initially in object space. One can therefore consider various order *derivatives* of this function with respect to angle; the various aberrations of a lens are related to the values of the various derivatives of this so-called ray function. Here is where automatic differentiation becomes applicable in lens analysis and design.

The final topic that will be covered as part of this discussion on lens fundamentals is the important concept of the *lens stop*, or *aperture*. In Fig. 6.4, it is the size of the lens itself that limits how much of the fan of rays leaving the point source is collected and imaged. Fig 6.5 shows a more typical situation, in which an auxiliary aperture (commonly known as an iris diaphragm) is place somewhere within a lens system. It determines how much light gets through the lens, and, as one can see, it determines through which part of the lens elements the various light rays from the different object points go. There is a particular ray that goes through the center of the aperture stop. This is called the *chief ray*; note that it is coincident with the optical axis when the object point is on the axis. It is actually *this* ray from which all aberrations (at any given object point) are measured. In fact, the amount and type of aberration varies in a lens as the object point is moved from on-axis to off-axis, because at different object heights the rays travel through different parts of the lenses, and at different angles.

From this one can see that there are two fundamental parameters for any given ray: (1) the height of the object from which it came, and (2) the angle of the ray with respect to the chief ray, or, equivalently, the point on the aperture stop through which the ray passes. The ray tracing problem is easily seen to be a multiple variable problem, whose various derivatives include higher order mixed partial derivatives.

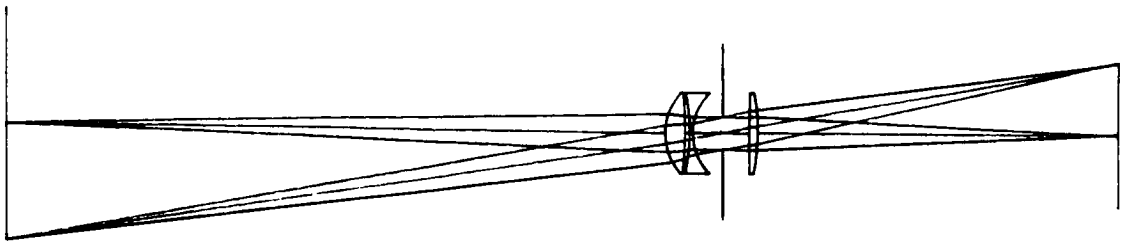

Fig 6.5 Lens System with Interior Aperture

6.1.3 Ray Tracing Formulas

The following discussion on ray tracing formulas is adapted from [22]. A complex lens system, such as in Fig. 6.5, can be considered a collection of surfaces, each surface separating two materials of different refractive indices. The process of ray tracing can then be broken down into two steps: (1) transferring from the back of one surface to the front of the next one, and (2) bending (or refracting) through the surface. Fig 6.6 shows such a situation for a general ray in 3D space (note that the lens surfaces are spheres). Associated with each surface is a tangent plane, tangent to the surface at the optical axis. Distances at a surface are measured assuming the tangent plane and optic axis are the origin of a *local* coordinate system.

A general ray in space is uniquely specified by 6 parameters: the x, y, and z locations of the intersection of the ray with the local surface, and the k, l, and m *optical direction cosines* of the ray, which are merely the product of the actual direction cosines and the index of refraction of the material following the surface. Thus, the ray tracing procedure can be reduced to: (1) Given x, y, z, k, l, and m on a given surface, calculate x', y', z', k', l', and m' on the next surface, and (2) repeat this calculation (iterate) for all surfaces, from the object plane to the image plane of the lens.

Given below are the equations to be used to go from one surface to the next. For a derivation of these formulas, refer to [22]. If one is given x, y, z, k, l, and m, then, referring to Fig. 6.6:

$$\frac{d}{n} = \frac{t-z}{m} \tag{2}$$

$$y_t = y + \frac{d}{n} l \tag{3}$$

$$x_t = x + \frac{d}{n} k \tag{4}$$

$$\xi_1 = c(x_t^2 + y_t^2) \tag{5}$$

$$\xi_2 = m - c(y_t l + x_t k) \tag{6}$$

$$\xi_3 = n\left[\left(\frac{\xi_2}{n}\right)^2 - c\xi_1\right]^{\frac{1}{2}} \tag{7}$$

$$\frac{A}{n} = \frac{\xi_1}{\xi_2 + \xi_3} \tag{8}$$

$$x' = x_t + \frac{A}{n} k \tag{9}$$

$$y' = y_t + \frac{A}{n} l \tag{10}$$

$$z' = \frac{A}{n} m \tag{11}$$

$$\xi_4 = n'\left[\left(\frac{\xi_3}{n'}\right)^2 - \left(\frac{n}{n'}\right)^2 + 1\right]^{\frac{1}{2}} \tag{12}$$

$$\xi_5 = \xi_4 - \xi_3 \tag{13}$$

$$k' = k - x' c \xi_5 \tag{14}$$

$$l' = l - y' c \xi_5 \tag{15}$$

$$m' = m - (z' c - 1)\xi_5, \tag{16}$$

where: d = distance *along the ray* from the previous surface to the tangent plane
n = index of refraction before the second surface
t = separation (along the axis) of the two surfaces
x_t, y_t = coordinates of the ray on the tangent plane of the second surface
$\xi_1, \xi_2, \xi_3, \xi_4, \xi_5$ = temporary variables
c = curvature of the second surface (1/radius of curvature)

A = distance *along the ray* from the tangent plane to the second surface
n' = index of refraction after the second surface.

Eq. (12) is actually Snell's Law [Eq. (1)] rewritten for a 3D coordinate system centered at the tangent plane of the surface.

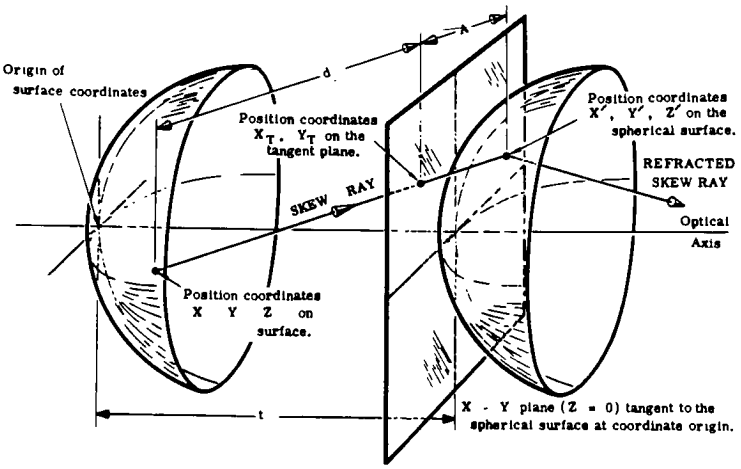

Fig 6.6 Illustration of a Ray between two surfaces [From Ref. 22]

These formulas are simple in that they only involve $+, -, *, \div$, and sqrt. (If the surfaces are tilted, then Eqs. (2) – (16) have to be modified slightly; the modifications involve the use of sines and cosines.) However, one can easily see that they are complicated enough that it would be impractical to calculate x', y', z', k', l', and m' (at the image plane of a complex lens) *symbolically* as a function of x, y, z, k, l, and m (at the object plane), and even more impractical to *symbolically* calculate derivatives; a perfect situation to illustrate the utility of automatic differentiation.

6.1.4 Aberration Theory Overview

The point of ray tracing is to calculate the ray coordinates (x', y', z', k', l', m') at the image plane as a function of the ray coordinates (x, y, z, k, l, m) at the object plane. However, not all of these coordinates are independent. Given that the coordinates are measured with respect to a local coordinate system centered on the tangent plane of a surface, the z and z' coordinates of the ray at the object and image surfaces (which are planes) are obviously 0. The optical direction cosines are related via the formula:

$$k^2 + l^2 + m^2 = N^2,$$

where N is the current index of refraction, so only two of the three parameters are independent. One can alternatively consider using direction tangents:

$$u = \frac{k}{m}, \qquad v = \frac{l}{m}$$

as the independent parameters. Given this, the problem reduces to finding (x', y', u', v') at the

image plane as a function of (x, y, u, v) at the object plane. This can be rewritten:

$$x' = X(x, y, u, v) \tag{17}$$

$$y' = Y(x, y, u, v) \tag{18}$$

$$u' = U(x, y, u, v) \tag{19}$$

$$v' = V(x, y, u, v). \tag{20}$$

So, for any general optical system, there exists functions X, Y, U, and V that relate the coordinates x', y', u', and v' and the image plane to x, y, u, and v at the object plane. These functions can be expanded in a Taylor series:

$$x' = x_0' + \frac{\partial x'}{\partial x}(x - x_0) + \frac{\partial x'}{\partial y}(y - y_0) + \frac{\partial x'}{\partial u}(u - u_0) + \frac{\partial x'}{\partial v}(v - v_0) + \ldots \tag{21}$$

$$y' = y_0' + \frac{\partial y'}{\partial x}(x - x_0) + \frac{\partial y'}{\partial y}(y - y_0) + \frac{\partial y'}{\partial u}(u - u_0) + \frac{\partial y'}{\partial v}(v - v_0) + \ldots \tag{22}$$

$$u' = u_0' + \frac{\partial u'}{\partial x}(x - x_0) + \frac{\partial u'}{\partial y}(y - y_0) + \frac{\partial u'}{\partial u}(u - u_0) + \frac{\partial u'}{\partial v}(v - v_0) + \ldots \tag{23}$$

$$v' = v_0' + \frac{\partial v'}{\partial x}(x - x_0) + \frac{\partial v'}{\partial y}(y - y_0) + \frac{\partial v'}{\partial u}(u - u_0) + \frac{\partial v'}{\partial v}(v - v_0) + \ldots \tag{24}$$

The first order partial derivatives are the so-called *paraxial coefficients*. The higher order derivatives are related to the lens aberrations.

Note that the functions X, Y, U, and V are also functions of the location of the object plane and location of the image plane. So there exist a whole family of functions X, Y, U, and V that completely characterize the path of a ray between any plane in object space and any plane in image space. There is actually only one location for a plane in image space to form a relatively sharp "image" of the object, the traditional "image" plane. Referring to Fig. 6.3 we see that the common concept of an image plane is where all the rays that come from the object intersect at the same point (to *first* order). This implies that the final height (x', y') of a ray is *not* a function of u' and v', to first order. Thus, we will know that the output plane is actually located at the image plane *if*, when calculating the first order derivatives in Eqs. (21) – (24):

$$\frac{\partial x'}{\partial u} = \frac{\partial x'}{\partial v} = \frac{\partial y'}{\partial u} = \frac{\partial y'}{\partial v} = 0.$$

In Eq. (21), the derivative

$$\frac{\partial x'}{\partial x}$$

indicates how much the image point moves off axis as the object point is moved off axis. This

is the *magnification* of the lens. Other fundamental lens paraxial parameters, such as *focal length*, can also be derived from the derivatives in Eqs. (21) – (24).

It turns out that not all the first order derivatives in Eqs. (21) – (24) are independent for real optical systems. This is a consequence of Snell's Law. Assume that the initial coordinate system is aligned with its z axis along a ray about which the optical ray trace function is to be expanded. In most cases this ray will be coincident with the optical axis. For this case, $x_0 = y_0 = u_0 = v_0 = 0$. If we rewrite Eqs. (21) – (24):

$$x' = A_{11}x + B_{11}u + A_{12}y + B_{12}v + \ldots \tag{25}$$

$$u' = C_{11}x + D_{11}u + C_{12}y + D_{12}v + \ldots \tag{26}$$

$$y' = A_{21}x + B_{21}u + A_{22}y + B_{22}v + \ldots \tag{27}$$

$$v' = C_{21}x + D_{21}u + C_{22}y + D_{22}v + \ldots, \tag{28}$$

then the following 6 fundamental relationships hold[23] (if N in object and image space equals 1.0):

$$A_{11}C_{12} - A_{12}C_{11} + A_{21}C_{22} - A_{22}C_{21} = 0 \tag{29}$$

$$A_{11}D_{12} - B_{12}C_{11} + A_{21}D_{22} - B_{22}C_{21} = 0 \tag{30}$$

$$B_{11}C_{12} - A_{12}D_{11} + B_{21}C_{22} - A_{22}D_{21} = 0 \tag{31}$$

$$B_{11}D_{12} - B_{12}D_{11} + B_{21}D_{22} - B_{22}D_{21} = 0 \tag{32}$$

$$A_{11}D_{11} - B_{11}C_{11} + A_{21}D_{21} - B_{21}C_{21} = 1 \tag{33}$$

$$A_{12}D_{12} - B_{12}C_{12} + A_{22}D_{22} - B_{22}C_{22} = 1 \tag{34}$$

It should be emphasized here that the implication of Eqs. (21) – (24) is that the paraxial coefficients of an optical system are measured *with respect to* a given ray. In other words, a single ray is traced through the optical system, and the paraxial coefficients are measured with respect to that traced ray. (For a rotationally symmetric system, that ray is normally the one that goes down the axis of the system.) In addition, for Eqs. (29) – (34) to be valid, the initial and final *coordinate systems* have to be aligned along that ray. This is illustrated in Fig. 6.7.

Rotationally Symmetric Formulation

Classical aberration analysis has assumed that all lens surfaces are centrally located along an optical axis. The expansion of the ray tracing function also takes place about this optical axis. In this case, the terms x_0', y_0', u_0', and v_0' in Eqs. (21) – (24) all equal 0. In addition, it can be shown[24] that all even order derivatives are identically equal to 0. Furthermore, if one

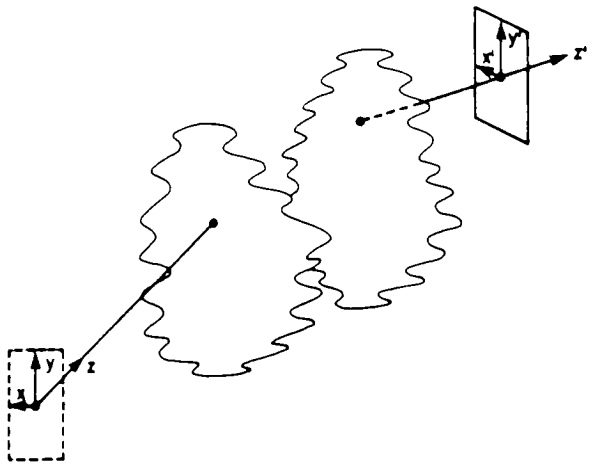

Fig 6.7 Illustration of Alignment of Coordinate Systems to Chief Ray

considers a plane that contains the optical axis and launches a ray in that plane, the ray *stays* in that plane throughout the lens. This says that if a ray has an initial y component (y_0 or v_0) and no initial x component, it will have no x component anywhere in the lens (this ray is a so-called *meridional* ray). This implies that the "cross-term" derivatives in Eqs. (21)–(24), i.e., $A_{12} \ldots D_{12}$ and $A_{21} \ldots D_{21}$ in Eqs. (25)–(28), are also identically 0. Given these conditions, Eqs. (25) and (27) can be rewritten:

$$x' = A_{11}x + B_{11}u + O(3) + O(5)\ldots \tag{35}$$

$$y' = A_{22}y + B_{22}v + O(3) + O(5)\ldots \tag{36}$$

where $O(n)$ indicates terms of order n. If the image plane is actually the paraxial image plane, then the coefficients B_{11} and B_{22} are equal to 0. Given this, then Eqs. (35) and (36) can be rewritten:

$$x' - Mx = O(3) + O(5)\ldots \tag{37}$$

$$y' - My = O(3) + O(5)\ldots \tag{38}$$

where $A_{11} = A_{22} = M$ is the magnification of the lens system. The values Mx and My are the heights of the so-called *paraxial chief ray* at the image plane. Eqs. (37) and (38) show that the lens aberrations are measured with respect to this ray.

Fig. 6.8 illustrates the classical coordinates used for aberration analysis. The quantities ρ and θ are polar coordinate representations for the intersection point of the ray with the aperture stop. (More correctly, the intersection point is at the *image* of the aperture stop in object space, which is called the *entrance pupil*. This is what you see when you look into a lens and see an "image" of the iris diaphragm.) If the coordinates in the entrance pupil are (x, y), then $y = \rho\cos\theta$ and $x = \rho\sin\theta$. H is the height of the object (assumed to be along the y axis); t_0 is the distance from the object to the entrance pupil, and $\overline{H} = H/t_0$ is essentially the direction tangent v of the ray from the object to the center of the entrance pupil. If we define $\varepsilon_y' = y' - My$ and $\varepsilon_x' = x' - Mx$, then it is given in [24], p. 298, that:

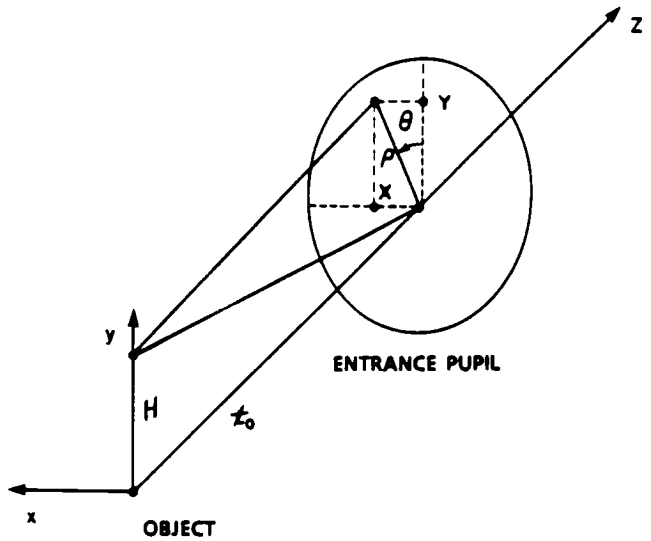

Fig 6.8 Illustration of Classical Expansion Variables

$$\varepsilon_y' = \sigma_1\rho^3\cos\theta + \sigma_2\rho^2\overline{H}(2+2\cos2\theta) + (3\sigma_3+\sigma_4)\rho\overline{H}^2\cos\theta + \sigma_5\overline{H}^3 + O(5)\ldots \tag{38}$$

$$\varepsilon_x' = \sigma_1\rho^3\sin\theta + \sigma_2\rho^2\overline{H}\sin2\theta + (\sigma_3+\sigma_4)\rho\overline{H}^2\sin\theta + O(5)\ldots \tag{39}$$

The quantities $\sigma_1 \ldots \sigma_5$ are the *aberration coefficients* of the lens. Given below are their more common names:

σ_1 = spherical aberration
σ_2 = coma
σ_3 = astigmatism
σ_4 = Petzval
σ_5 = distortion.

In [24], Buchdahl also gives a method of calculating $\sigma_1 \ldots \sigma_5$ from the lens characteristics. It is these calculations that will be used as a check of the calculations of these coefficients by automatic differentiation.

To tie classical aberration analysis with automatic differentiation, it is necessary to derive a relationship between the higher order mixed partial derivatives of Eqs. (21)–(24) and $\sigma_1 \ldots \sigma_5$. If we redefine Eqs. (21)–(24) slightly so that x and y correspond to x and y in the *entrance pupil*, and v corresponds to $\overline{H}$, then the following relations hold between the higher order mixed partial derivatives and $\sigma_1 \ldots \sigma_5$:

TABLE 6.1

derivative	y'	x'
$\frac{\partial^3}{\partial x^3}$	0	$6\sigma_1$
$\frac{\partial^3}{\partial x^2 \partial y}$	$2\sigma_1$	0
$\frac{\partial^3}{\partial x^2 \partial v}$	$2\sigma_2$	0
$\frac{\partial^3}{\partial x \partial y^2}$	0	$2\sigma_1$
$\frac{\partial^3}{\partial x \partial y \partial v}$	0	$2\sigma_2$
$\frac{\partial^3}{\partial x \partial v^2}$	0	$2(\sigma_3+\sigma_4)$
$\frac{\partial^3}{\partial y^3}$	$6\sigma_1$	0
$\frac{\partial^3}{\partial y^2 \partial v}$	$6\sigma_2$	0
$\frac{\partial^3}{\partial y \partial v^2}$	$2(3\sigma_3+\sigma_4)$	0
$\frac{\partial^3}{\partial v^3}$	$6\sigma_5$	0

6.2 OPTICS PACKAGE

6.2.1 Functional Spec

The purpose of the OPTICS package is to provide to the user the ability to do ray tracing through a wide variety of optical systems. In particular, this package provides the ability to calculate *derivatives* of ray trace parameters with respect to other system variables using the automatic DIFFERENTIALS package. As with the DIFFERENTIALS package, the OPTICS package specification provides the user with an appropriate functional specification. The following discussion should therefore be read with reference to Fig. 6.9, which is the OPTICS package specification.

The OPTICS package provides the user with the ability to read in lens data from a file and to trace a ray through the specified lens. Various output options are also provided.

Constants

MAXLINE
: The maximum length (in characters) of an input line in the lens data file.

MAXSURF
: The maximum allowable number of lens surfaces in the lens input deck.

MAXNAME
: The maximum length of the file name of the file that contains the lens information.

Data types

surface
: This defines a data type used to represent a single lens surface (any lens is a collection of lens surfaces). The various fields of this record are explained in Fig. 6.9; some additional comments are below:

  ty: currently the software handles 3 types of surfaces:
  0 = sphere, 1 = y-direction cylinder, 2 = x-direction cylinder

  ind: index of refraction *after* the surface

  th: distance to the *next* surface

  dx,dy,alpha,beta,gamma: these indicate that the current surface should have the these transformations applied to it (with respect to the *previous* surface)

  control parameters: These INTEGERS are used to indicate to the DIFFERENTIALS package which independent variable the corresponding parameter is; e.g., if thc = 6, then the system recognizes this surface's thickness to be independent variable #6, and when this thickness is used in a calculation, it will be treated as a DIFFERENTIAL with the initialization {th, 0, 0, . . . , 1, 0, . . .}, where the 1 will be in the $\partial/\partial x_6$ position.

  ray: The 6 parameters of a ray (x, y, z, k, l, m) are the fields of this record; each field is of type DIFFERENTIAL.

```
-------------------------------------------------------------------------
-------------------------------------------------------------------------

with text_io; use text_io;
with integer_text_io; use integer_text_io;
with float_text_io; use float_text_io;
with DIFFERENTIALS; use DIFFERENTIALS;

package optics is

-- constants

   MAXLINE: constant INTEGER := 50;              -- maximum length of input line
   MAXSURF: constant INTEGER := 30;              -- max number of surfaces
   MAXNAME: constant INTEGER := 10;              -- max length of file name

-- types

   type surface is record
        ty: INTEGER;            -- type
        cv: FLOAT;              -- curvature
        th: FLOAT;              -- thickness
        ind: FLOAT;             -- index of refraction
        dx: FLOAT;              -- x-decenter
        dy: FLOAT;              -- y-decenter
        alpha: FLOAT;           -- alpha rotation (+z -> +y)
        beta: FLOAT;            -- beta rotation (+x -> +z)
        gamma: FLOAT;           -- gamma rotation (+x -> +y)
        cvc: INTEGER;           -- curvature control
        thc: INTEGER;           -- thickness control
        indc: INTEGER;          -- index control
        dxc: INTEGER;           -- dx control
        dyc: INTEGER;           -- dy control
        alphac: INTEGER;        -- alpha control
        betac: INTEGER;         -- beta control
        gammac: INTEGER;        -- gamma control
    end record;
   type RAY is record
                x: DIFFERENTIAL;                -- current x coordinate
                y: DIFFERENTIAL;                -- current y coordinate
                z: DIFFERENTIAL;                -- current z coordinate
                k: DIFFERENTIAL;                -- n * current x dir. cosine
                l: DIFFERENTIAL;                -- n * current y dir. cosine
                m: DIFFERENTIAL;                -- n * current z dir. cosine
      end record;

-- exceptions

   LENS_DECK_ERROR: exception;

-- Global variables

   system: array(0..MAXSURF) of surface;        -- the collection of surfaces
                                                --   constituting a lens system
   NSURF: integer;                              -- # of surfaces read in from
                                                --   lens deck
```

Fig. 6.9 OPTICS Package Specification

```
-- Data input procedures

   procedure input_lens(surf: out INTEGER);
   procedure set_ray(r: in out RAY;
                     x,dx: in FLOAT; ix: in INTEGER;
                     y,dy: in FLOAT; iy: in INTEGER;
                     z,dz: in FLOAT; iz: in INTEGER;
                     k,dk: in FLOAT; ik: in INTEGER;
                     l,dl: in FLOAT; il: in INTEGER;
                     m,dm: in FLOAT; im: in INTEGER);

-- Data output procedures

   procedure dump_lens(numsurf: in INTEGER);
   procedure print_ray(r1: in RAY);

-- Ray Tracing procedures

   procedure traverse(r1: in out RAY; n1,n2: in INTEGER; p,q: in BOOLEAN);
   procedure decenter(r1: in out RAY; dx,dy: in DIFFERENTIAL);
   procedure rotZY(r1: in out RAY; sa,ca: in DIFFERENTIAL);
   procedure rotXZ(r1: in out RAY; sb,cb: in DIFFERENTIAL);
   procedure rotXY(r1: in out RAY; sg,cg: in DIFFERENTIAL);
   procedure align_axes(r1: in out RAY);

end optics;
```

Fig 6.9, con't

Exceptions

LENS_DECK_ERROR

Raised when an error is encountered when reading from the lens input file.

Global variables

system()

This array consists of the collection of surfaces that constitutes a lens.

NSURF

The number of surfaces successfully read in from the lens deck. (*Has* to be set via `input_lens` and its `OUT` parameter.)

Data Input Procedures

procedure input_lens(surf: out INTEGER)

Lens data is input using this routine. It prompts the user for the name of a file which contains the lens information to be input. It then reads in the information from the file, and returns in `surf` the number of surfaces successfully read. A lens deck consists of a sequence of surface definitions; each surface is defined by at least one (and optionally three) line of input in the following format:

```
(Column #:)          123....
(optional            DECE   dx   dy   alpha   beta   gamma
```

```
        lines)      OECC   dxc  dyc  alphac  betac  gammac
(required line)     incv   incvc   th   thc   ind   inc
```

The optional lines give the orientation of the surface with respect to the previous surface, where the decenters are applied first, then the rotations alpha, beta, and gamma are applied successively. If these two lines (which *must* come together) are not included, those parameters of the surface default to 0.0 (or 0).

All surface parameters are *required* to be FLOATs. All control parameters are *required* to be INTEGERS. Given below is an example of a valid deck:

```
: Surface #0
0.0   0     5.9   4   1.0 1                <--  Surface #0
: Surface #1
DECE  0.0   0.0   3.5   2.6   99.7
OECC  1     2     3     0     0
-0.445 0    3.4 0   1.5 0                  <-- Surface #1
: Surface #2
45.6  200   0.0   0   1.0   0              <-- Surface #2
```

Note that there can be an arbitrary number of spaces between entries (as long as the total length of each line is less that MAXLINE. The incv and incvc lens deck parameters are then interpreted to generate the ty, cv, and cvc surface parameters via the following kludge:

```
        incvc                  how to generate cv, ty, cvc
        -----                  ---------------------------
    a. 0..9          =>        ty=0; cvc=incvc; cv=incv (Sphere)
    b. 10..19        =>        ty=1; cvc=incvc-10; cv=incv (Y-cylinder)
    c. 20..29        =>        ty=2; cvc=incvc-20; cv=incv (X-cylinder)
Radius of curvature input
    a. 200..209      =>        ty=0; cvc=incvc-200; cv=1.0/incv (Sphere)
    a. 210..219      =>        ty=1; cvc=incvc-210; cv=1.0/incv (Y-cylinder)
    a. 220..229      =>        ty=2; cvc=incvc-220; cv=1.0/incv (X-cylinder)
```

Note that if any line in the lens deck starts with a ":", it is treated as a comment and skipped.

```
procedure set_ray(r: in out RAY;
                  x,dx: in FLOAT; ix: in INTEGER;
                  y,dy: in FLOAT; iy: in INTEGER;
                  z,dz: in FLOAT; iz: in INTEGER;
                  k,dk: in FLOAT; ik: in INTEGER;
                  l,dl: in FLOAT; il: in INTEGER;
                  m,dm: in FLOAT; im: in INTEGER)
```

This procedure is used to set the initial parameters of a ray (x, y, z, k, l, m). If, in addition, the user is interested in derivatives with respect to some of these ray input parameters, then the appropriate derivative values dx . . . dm can be set (most often to 1.0) and they are associated with independent variables via ix . . . im. For example, if we want to trace a ray down the lens axis, and we will later want to know about derivatives with respect to the initial y-height (which we want to be the 5^{th} independent variable), then the appropriate call to set_ray is:

```
set_ray(r,0.0,0.0,0,
        0.0,1.0,5,
        0.0,0.0,0
        0.0,0.0,0
        0.0,0.0,0
        1.0,0,0,0);
```

Data Output Procedures

`procedure dump_lens(numsurf: in INTEGER)`

This procedure outputs to STDOUT the stored lens surface information, from surface #0 to surface #numsurf.

`procedure print_ray(r1: in RAY)`

Outputs to STDOUT *only* the current values (no derivatives) of x, y, z, and the direction *tangents* (k/m and l/m) of the ray r1.

Ray Tracing Procedures

`procedure traverse(r1: in out RAY; n1,n2: in INTEGER; p,q: in BOOLEAN)`

This procedure traces the r1 (assumed to be initialized via set_ray) from surface #n1 to surface #n2. If p is TRUE, then print out ray value at each surface. If q is FALSE, then do *not* do any surface transformations.

`procedure decenter(r1: in out RAY; dx,dy: in DIFFERENTIAL)`

Perform a decenter of the coordinate system by shifting it dx in the x direction and dy in the y direction. This is equivalent to *subtracting* dx from the ray's x intercept and dy from the ray's y intercept.

`procedure rotZY(r1: in out RAY; sa,ca: in DIFFERENTIAL)`

Perform a rotation of the coordinate system about the x axis by an angle whose sine is sa and whose cosine is ca. A positive angle rotates the $+z$ axis towards the $+y$ axis.

`procedure rotXZ(r1: in out RAY; sb,cb: in DIFFERENTIAL)`

Perform a rotation of the coordinate system about the y axis by an angle whose sine is sb and whose cosine is cb. A positive angle rotates the $+x$ axis towards the $+z$ axis.

`procedure rotxy(r1: in out RAY; sg,cg: in DIFFERENTIAL)`

Perform a rotation of the coordinate system about the z axis by an angle whose sine is sg and whose cosine is cg. A positive angle rotates the $+x$ axis towards the $+y$ axis.

`procedure align_axes(r1: in out RAY)`

Transform (decenter and rotate) the coordinate system such that the ray is pointed along the z axis of the new coordinate system. When tracing a skew ray through a centered optical system, the final coordinates of the ray are invariably non-zero. In order to expand the system about that ray and to have the paraxial relationships (29) – (34) hold, the final coordinate system *must* be aligned along the ray.

6.2.2 Internal Procedures

(The reader should refer to the "opticsbody" section, §9.7, for the appropriate code listings.

Constants

`rad_per_deg: constant FLOAT := 0.017453293`

Number of radians per degree.

`zero: constant DIFFERENTIAL := Const_Diff(0.0)`

This is the DIFFERENTIAL constant 0. It's primary use is for type casting; i.e., adding zero to something converts the something into a DIFFERENTIAL (in an expression).

Procedures

`procedure clear_system`

Set all surface parameters of all surfaces to 0 or 0.0, except set ind = 1.0.

`procedure transform(r1: in out RAY; i: in INTEGER; q: BOOLEAN)`

This procedure transforms the current coordinate system of the ray r1 (assumed that of the $(i-1)^{st}$ surface) into that of the i^{th} surface. If q is FALSE, then no transform is done. Transform gets the surface transform information from system, then calls decenter and rotate (which calls rotzy, then rotxz, and finally, rotxy).

Parts of "Input_Lens"

`procedure clear(line: out STRING)`

Returns a cleared input line buffer (used to read in information from the lens file).

`procedure parse_surf(line: in STRING; surf: in INTEGER)`

Uses line to read in surface information from the lens file. Assumes that information should be associated with surface #surf in system.

`procedure parse_dec(line: in STRING; surf: in INTEGER)`

Parses the two decenter lines associated with a surface in the input file, if the two lines are there. Puts information into system(surf). Note that for this implementation one *cannot* specify transformation quantities to be a DIFFERENTIAL.

Parts of "traverse"

`procedure sphere(r1: in out RAY; i: in INTEGER)`

Refracts the ray r1 through a sphere at surface #i.

`procedure cylX(r1: in out RAY; i: in INTEGER)`

Refracts the ray r1 through an x-cylinder at surface #i.

`procedure cylY(r1: in out RAY; i: in INTEGER)`

Refracts the ray r1 through a y-cylinder at surface #i.

6.3 CLIENT PROGRAMS AND TEST RESULTS

This section describes in detail two client programs that use the DIFFERENTIALS and OPTICS packages. The first to be described will be a paraxial optics program that calculates all 16 first order partial derivatives of an optics system [Eqs. (21) – (24)]. The other program is one that calculates all 10 third order partial derivatives for a rotationally symmetric optics system (from which the 5 third order aberration coefficients can be derived). Each program will be described line-by-line to illustrate the proper use of the procedures in the DIFFERENTIALS and OPTICS packages.

6.3.1 Paraxial Optics

The listing of this program is given in Fig. 6.10. This program calculates all 16 paraxial coefficients of an optics system. These paraxial coefficients are the first order partial derivatives in the expansion of the 4 final ray coordinates (x', y', u', v') with respect to the 4 input ray coordinates (x, y, u, v). Given below is a line-by-line description of the paraxial optics program.

Lines 9-14:

This is a list of the required packages.

17:

R1 is the ray that will be traced through the system.

18-21:

I1 . I4 are LISTs that specify which derivatives are to be taken. Since it was known that DIFFERENTIALS was compiled with max_level=11, a LIST is an array of 12 INTEGERS, as shown here. I1 specifies a first partial derivative with respect to independent variable #1, I2 specifies a first partial derivative with respect to independent variable #2, etc.

22-34:

These define the local variables:

x0,y0,z0:	initial ray heights (z0 is actually not ever used)
k0,l0,m0:	initial ray direction cosines
u0,v0:	initial ray direction tangents
u,v:	final ray direction tangents
n:	used as index of refraction of initial surface
surf:	temporary
f:	temporary

36-43:

This is merely a subroutine that will be used to output the 16 paraxial coefficients in a certain format. For example, line 41 says to take the FLOATing point value of the derivative of r specified by the LIST I, and print it out with FORE=3, AFT=6, and EXP=0 (i.e., in ±xxx.xxxxxx format).

47-50:

These calls to Set_Names tell the DIFFERENTIALS package which derivatives will be required. In this case, we have specified 4 first partial derivatives, with respect to independent variables #1, 2, 3, and 4.

51-54:

The call to init_diffs initializes the DIFFERENTIALS package so that only the derivatives specified in Set_Names are calculated. The put_line's are merely used to indicate to the user how long it took to do init_diffs.

55:

Inputs information from a lens file. NSURF (which is a global variable in OPTICS) is used to store the number of lens surfaces read in.

56:

Requests a printout of the lens surface information.

```
1   ------------------------------------------------------------------------
2   --
3   -- Test of optics package:  paraxial coefficients
4   -- Robert Herloski
5   -- 11:18      09-Jan-87          vaxa
6   --
7   ------------------------------------------------------------------------
8
9   with text_io; use text_io;
10  with integer_text_io; use integer_text_io;
11  with float_text_io; use float_text_io;
12  with realfunc; use realfunc;
13  with DIFFERENTIALS; use DIFFERENTIALS;
14  with optics; use optics;
15
16  procedure parax is
17     r1: RAY;
18     l1: LIST  := (1,0,0,0,0,0,0,0,0,0,0,0);
19     l2: LIST  := (2,0,0,0,0,0,0,0,0,0,0,0);
20     l3: LIST  := (3,0,0,0,0,0,0,0,0,0,0,0);
21     l4: LIST  := (4,0,0,0,0,0,0,0,0,0,0,0);
22     x0: DIFFERENTIAL;
23     y0: DIFFERENTIAL;
24     z0: DIFFERENTIAL;
25     k0: DIFFERENTIAL;
26     l0: DIFFERENTIAL;
27     m0: DIFFERENTIAL;
28     u0: DIFFERENTIAL;
29     v0: DIFFERENTIAL;
30      u: DIFFERENTIAL;
31      v: DIFFERENTIAL;
32      n: DIFFERENTIAL;
33   surf: INTEGER;
34      f: FLOAT;
35
36     procedure put_coeff(s: in STRING; r: in DIFFERENTIAL; l: in LIST) is
37
38        begin
39           put(s);
40           put(":  ");
41           put(val(r,l),3,6,0);
42           put_line("");
43     end put_coeff;
44
45
46     begin
47        Set_Names(l1);
48        Set_Names(l2);
49        Set_Names(l3);
50        Set_Names(l4);
51        put_line("Starting with Init_Diffs.");
52        Init_Diffs;
53        put_line("All done with Init_Diffs.");
54        put_line("");
55        input_lens(NSURF);
56        dump_lens(NSURF);
57        f := 0.0;
58  --      put("Input x0:  ");
59  --      get(f);
60        Set_Diff(x0,f,1.0,1);
```

Fig 6.10 Paraxial Optics Program

```
61       --      put("Input u0:  ");
62       --      get(f);
63               Set_Diff(u0,f,1.0,2);
64       --      put("Input y0:  ");
65       --      get(f);
66               Set_Diff(y0,f,1.0,3);
67       --      put("Input v0:  ");
68       --      get(f);
69               Set_Diff(v0,f,1.0,4);
70               r1.x := x0;                              -- set initial ray position
71               r1.y := y0;
72               Set_Diff(r1.z,0.0,0.0,1);
73               surf := 1;
74               if (system(surf).indc /= 0) then
75                  Set_Diff(n,system(surf).ind,1.0,system(surf).indc);
76               else
77                  Set_Diff(n,system(surf).ind,0.0,1);
78               end if;
79               r1.m := n / sqrt(1.0 + u0 * u0 + v0 * v0);  -- set in. ray optical
80               r1.k := u0 * r1.m;                          --   direction cosines
81               r1.l := v0 * r1.m;
82               traverse(r1,surf,NSURF,TRUE,TRUE);
83               align_axes(r1);
84               put(" Im:  ");
85               print_ray(r1);
86               u := r1.k / r1.m;
87               v := r1.l / r1.m;
88               put_line("");
89               put_line("Paraxial Coefficients");
90               put_line("---------------------");
91               put_line("");
92               put_coeff("A11",r1.x,1);
93               put_coeff("B11",r1.x,2);
94               put_coeff("C11",u,1);
95               put_coeff("D11",u,2);
96               put_coeff("A12",r1.x,3);
97               put_coeff("B12",r1.x,4);
98               put_coeff("C12",u,3);
99               put_coeff("D12",u,4);
100              put_coeff("A21",r1.y,1);
101              put_coeff("B21",r1.y,2);
102              put_coeff("C21",v,1);
103              put_coeff("D21",v,2);
104              put_coeff("A22",r1.y,3);
105              put_coeff("B22",r1.y,4);
106              put_coeff("C22",v,3);
107              put_coeff("D22",v,4);
108              put("A11C12   A12C11 + A21C22   A22C21   ");
109              f := val(r1.x,1) * val(u,3)   val(r1.x,3) * val(u,1);
110              f := f + val(r1.y,1) * val(v,3)   val(r1.y,3) * val(v,1);
111              put(f,3,6,0);
112              put_line("");
113              put("A11D12   B12C11 + A21D22   B22C21   ");
114              f := val(r1.x,1) * val(u,4)   val(r1.x,4) * val(u,1);
115              f := f + val(r1.y,1) * val(v,4)   val(r1.y,4) * val(v,1);
116              put(f,3,6,0);
117              put_line("");
118              put("B11C12   A12D11 + B21C22   A22D21   ");
119              f := val(r1.x,2) * val(u,3)   val(r1.x,3) * val(u,2);
120              f := f + val(r1.y,2) * val(v,3)   val(r1.y,3) * val(v,2);
```

Fig. 6.10, con't

```
121             put(f,3,6,0);
122             put_line("");
123             put("B11D12 - B12D11 + B21D22 - B22D21 = ");
124             f := val(r1.x,12) * val(u,14) - val(r1.x,14) * val(u,12);
125             f := f + val(r1.y,12) * val(v,14) - val(r1.y,14) * val(v,12);
126             put(f,3,6,0);
127             put_line("");
128             put("A11D11 - B11C11 + A21D21 - B21C21 = ");
129             f := val(r1.x,11) * val(u,12) - val(r1.x,12) * val(u,11);
130             f := f + val(r1.y,11) * val(v,12) - val(r1.y,12) * val(v,11);
131             put(f,3,6,0);
132             put_line("");
133             put("A12D12 - B12C12 + A22D22 - B22C22 = ");
134             f := val(r1.x,13) * val(u,14) - val(r1.x,14) * val(u,13);
135             f := f + val(r1.y,13) * val(v,14) - val(r1.y,14) * val(v,13);
136             put(f,3,6,0);
137             put_line("");
138
139     end parax;
```

Fig 6.10 , con't

57-69:

These lines initialize the DIFFERENTIALS x0,y0,u0, and v0. Note the temporary variable f is set to 0.0, so, for example, line #63 says to (1) set the initial value of u0 to 0.0, and (2) let u0 be proportional to independent variable #2, with an initial derivative of 1.0, which means that u0 actually *is* independent variable #2. The comment lines can be included if one wants to set the initial values to other than 0.0.

70-71:

Set the initial x and y positions of ray r1 to x0 and y0.

72:

Sets the initial z position of ray r1 to 0.0; since the third parameter to Set_Diff is 0.0, it doesn't matter what the value of the fourth parameter is.

73:

surf is set to 1 (we will trace starting from the first surface of the lens, which we will always assume is the entrance pupil).

74-81:

These lines initialize the initial optical direction cosines. As mentioned before, optical direction cosines are just the ray direction cosines multiplied by the current index of refraction. The if statement is required because the initialization of the index of refraction DIFFERENTIAL n depends upon whether it is an independent variable or not (i.e., is indc=0?). In all test cases, it will be equal to 0. Lines 79–81 convert from direction tangents to optical direction cosines.

82:

This routine traces the DIFFERENTIAL ray r1 from surface #1 to surface #NSURF. TRUE,TRUE means (1) to print out ray information at each surface during the trace, and (2) do *not* skip any tilts or decenters specified at a surface.

83:

This is *required* in order for Eqs. (29) – (34) to be valid.

84-85:

Prints out the final ray data; since it is after an `align_axes`, x', y', z', u', and v' should all equal 0.

86-87:

Calculate final direction tangents.

88-107:

These lines output the 16 paraxial coefficients. For example, line #92 outputs `A11`, which in this case is the derivative with respect to independent variable #1 (because `I1` was specified) of the x value of the ray `r1`. Since independent variable #1 was associated with `x0` in line #60, the effective result is the derivative of the final x height with respect to the initial ray x height ($\partial x'/\partial x$), which, from Eq. (25), is A_{11}.

108-137:

These lines output the values of the six relationships (28) – (33) for the lens system. This is the *check* of the calculations to see if the differentiation was done properly; if so, the Eqs. (29) – (34) should hold. Note the use of the routine `val`, which returns the specified derivative value (using a `LIST`) of the specified `DIFFERENTIAL`.

Fig. 6.11 gives the first example of the use of this program. A diagram of the lens is shown in Fig. 6.12. The first part of Fig. 6.11 is the listing of the input file. Following that is the execution of the `PARAX` program. First the lens data is printed, then the ray trace information is output. All ray parameters are equal to zero, which means that the ray traveled down the lens axis. The final table output is that of the 16 paraxial coefficients and the values of the 6 relationships. Since this system is rotationally symmetric, the cross term coefficients $A_{12} \ldots D_{21}$ should be equal to zero, and one can see that they are. The image plane (surface #9) is located at the back focus of the lens. This means that all incoming rays parallel to the axis focus at the same point (to *first* order), hence $\partial x'/\partial x = \partial y'/\partial y = 0$. From the table we see that $A_{11} = A_{22} = 0$. In addition, the 6 relationships do hold.

Fig 6.13 is the second example of the use of `PARAX`; in it, the location of the image plane has been moved toward the lens by 0.5 (refer to Fig. 6.12). In this case, $\partial x'/\partial x$ and $\partial y'/\partial y$ should not equal 0, and, as can be seen in Fig. 6.13, $A_{11} = A_{22} = 0.5$. (The reason that $A_{11} = A_{22} = 0.5 =$ the shift away from focus is that the *focal length* of the lens equals 1.0.) Note also that D_{11} and D_{22} ($\partial u'/\partial u$ and $\partial v'/\partial v$) haven't changed from Fig 6.11, which is also to be expected.

The final example is tabulated in Fig. 6.14, and illustrated in Fig. 6.15. The last element has been shifted and tilted somewhat. Fig. 6.15 shows the path of the ray that started down the axis of the lens; referring to Fig. 6.14, one can see that after surface #6 the ray parameter values are non-zero. Because of this tilt, rotational symmetry has been lost, and hence the cross-term coefficients $A_{21} \ldots D_{21}$ should be non-zero. In fact, looking at Fig. 6.14, all 16 coefficients are non-zero. However, the 6 fundamental relationships (described in §6.1.4) still hold, which further validates the correctness of the calculated results.

```
$ type thes01.len

 0.0       0   0.0         0   1.0     0
 0.0       0  -0.1132      0   1.0     0
 4.82439   0   0.040278    0   1.6162  0
-0.753929  0   0.016851    0   1.0     0
-1.64505   0   0.0096145   0   1.5725  0
 5.11794   0   0.041353    0   1.0     0
:DECE    -0.14   0.25    5.3   -3.4    0.0
:DECC    0 0 0 0 0
 0.0       0   0.097385    0   1.0     0
 0.310726  0   0.0313246   0   1.6162  0
 -1.46116  0   0.836       0   1.0     0
 0.0       0   0.0         0   1.0     0

$ run parax
Starting with Init_Diffs.
All done with Init_Diffs.

Input lens file name:  thes01.len

Surf TY  Curvature  CVC      Thickness    THC      Index    INC
---- --  ---------  ---      ---------    ---      -----    ---

  0  0   0.00000000   0         0.000000  0      1.000000  0
  1  0   0.00000000   0        -0.113200  0      1.000000  0
  2  0   4.82438993   0         0.040278  0      1.616200  0
  3  0  -0.75392902   0         0.016851  0      1.000000  0
  4  0  -1.64505005   0         0.009615  0      1.572500  0
  5  0   5.11793995   0         0.041353  0      1.000000  0
  6  0   0.00000000   0         0.097385  0      1.000000  0
  7  0   0.31072599   0         0.031325  0      1.616200  0
  8  0  -1.46115994   0         0.836000  0      1.000000  0
  9  0   0.00000000   0         0.000000  0      1.000000  0

Surf        X           Y           Z          TAN X       TAN Y
----    --------    --------    --------    --------    --------

  1:    0.000000    0.000000    0.000000    0.000000    0.000000
  2:    0.000000    0.000000    0.000000    0.000000    0.000000
  3:    0.000000    0.000000    0.000000    0.000000    0.000000
  4:    0.000000    0.000000    0.000000    0.000000    0.000000
  5:    0.000000    0.000000    0.000000    0.000000    0.000000
  6:    0.000000    0.000000    0.000000    0.000000    0.000000
  7:    0.000000    0.000000    0.000000    0.000000    0.000000
  8:    0.000000    0.000000    0.000000    0.000000    0.000000
  9:    0.000000    0.000000    0.000000    0.000000    0.000000
 Im:    0.000000    0.000000    0.000000    0.000000    0.000000

Paraxial Coefficients
---------------------

A11:    0.000000
B11:    1.000001
C11:   -0.999999
D11:    1.032178
A12:    0.000000
B12:    0.000000
C12:    0.000000
D12:    0.000000
```

Fig. 6.11 First Example of PARAX Program

```
A21:     0.000000
B21:     0.000000
C21:     0.000000
D21:     0.000000
A22:     0.000000
B22:     1.000001
C22:    -0.999999
D22:     1.032178
A11C12    A12C11 + A21C22    A22C21      0.000000
A11D12    B12C11 + A21D22    B22C21      0.000000
B11C12    A12D11 + B21C22    A22D21      0.000000
B11D12    B12D11 + B21D22    B22D21      0.000000
A11D11    B11C11 + A21D21    B21C21      1.000000
A12D12    B12C12 + A22D22    B22C22      1.000000

$
```

Fig 6.11 , con't

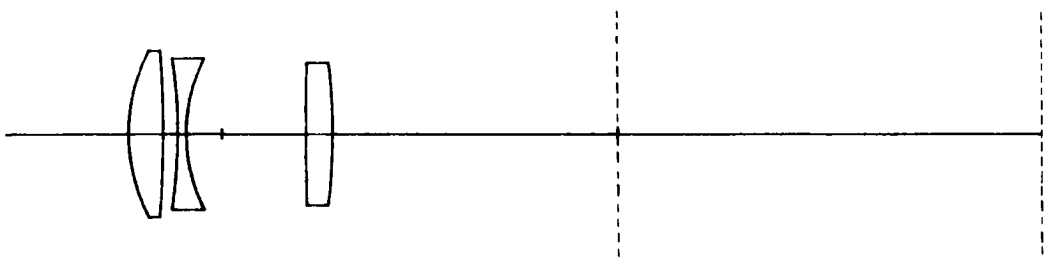

Fig 6.12 Lens Diagram for listing in Fig. 6.11

6.3.2 Third Order Optics

In §6.1.4, it was pointed out that rotationally symmetric lens systems have no second order terms (in fact, no *even* order terms) in their aberration expansion, but only third, fifth, . , order terms. In such a system, there are 5 fundamental third order aberration coefficients which are related to the 10 third order partial derivatives of the system, as given in Table 6.1. It can also be shown that there are 12 fundamental 5th order aberration coefficients, related to the 21 fifth order partial derivatives.

Fig. 6.16 gives a listing of a program that uses the OPTICS and DIFFERENTIALS packages to calculate third and fifth order coefficients (only the 3rd order results will be discussed in detail). Given below are selected line-by-line descriptions of the program.

Lines 18-48:

These lines define the various LISTs that will be used during the execution of the program.

```
$ type thes02.len
 0.0       0   0.0         0   1.0     0
 0.0       0  -0.1132      0   1.0     0
 4.82439   0   0.040278    0   1.6162  0
-0.753929  0   0.016851    0   1.0     0
-1.64505   0   0.0096145   0   1.5725  0
 5.11794   0   0.041353    0   1.0     0
:DECE    -0.14    0.25    5.3    -3.4    0.0
:DECC    0 0 0 0 0
 0.0       0   0.097385    0   1.0     0
 0.310726  0   0.0313246   0   1.6162  0
 -1.46116  0   0.336       0   1.0     0
 0.0       0   0.0         0   1.0     0

$ run parax
Starting with Init_Diffs.
All done with Init_Diffs.

Input lens file name:  thes02.len

Surf  TY  Curvature   CVC      Thickness    THC      Index     INC
----  --  ---------   ---      ---------    ---      -----     ---

  0   0   0.00000000   0         0.000000   0      1.000000   0
  1   0   0.00000000   0        -0.113200   0      1.000000   0
  2   0   4.82438993   0         0.040278   0      1.616200   0
  3   0  -0.75392902   0         0.016851   0      1.000000   0
  4   0  -1.64505005   0         0.009615   0      1.572500   0
  5   0   5.11793995   0         0.041353   0      1.000000   0
  6   0   0.00000000   0         0.097385   0      1.000000   0
  7   0   0.31072599   0         0.031325   0      1.616200   0
  8   0  -1.46115994   0         0.336000   0      1.000000   0
  9   0   0.00000000   0         0.000000   0      1.000000   0

Surf        X           Y           Z          TAN X       TAN Y
----     --------    --------    --------    --------    --------

  1:     0.000000    0.000000    0.000000    0.000000    0.000000
  2:     0.000000    0.000000    0.000000    0.000000    0.000000
  3:     0.000000    0.000000    0.000000    0.000000    0.000000
  4:     0.000000    0.000000    0.000000    0.000000    0.000000
  5:     0.000000    0.000000    0.000000    0.000000    0.000000
  6:     0.000000    0.000000    0.000000    0.000000    0.000000
  7:     0.000000    0.000000    0.000000    0.000000    0.000000
  8:     0.000000    0.000000    0.000000    0.000000    0.000000
  9:     0.000000    0.000000    0.000000    0.000000    0.000000
 Im:     0.000000    0.000000    0.000000    0.000000    0.000000

Paraxial Coefficients
---------------------

A11:     0.500000
B11:     0.483912
C11:    -0.999999
D11:     1.032178
A12:     0.000000
B12:     0.000000
C12:     0.000000
D12:     0.000000
```

Fig. 6.13 Second Example of PARAX Program

```
A21:     0.000000
B21:     0.000000
C21:     0.000000
D21:     0.000000
A22:     0.500000
B22:     0.4B3912
C22:    -0.999999
D22:     1.03217B
A11C12    A12C11 + A21C22    A22C21      0.000000
A11D12    B12C11 + A21D22    B22C21      0.000000
B11C12    A12D11 + B21C22    A22D21      0.000000
B11D12    B12D11 + B21D22    B22D21      0.000000
A11D11    B11C11 + A21D21    B21C21      1.000000
A12D12    B12C12 + A22D22    B22C22      1.000000

$
```

Fig 6.13, con't

For example, line #40 indicates that `113` represents a fifth order partial derivative $(\partial^5/\partial v_1 \partial v_2^2 \partial v_3^2)$.

51-60:

This is another output formatting subroutine; note that the ray `r` is passed in, and the various derivatives (indicated in the `LIST 1`) of the fields `r.x` and `r.y` are printed.

64-88:

Again, this is the initialization of the `DIFFERENTIALS` package. Note that we only have to do a `Set_Names()` on the higher order derivatives; `DIFFERENTIALS` automatically figures out which lower order derivatives are needed. For this case, there are 21 5^{th} order derivatives. This will required 15 4^{th} order derivatives, 10 3^{rd} order derivatives, 6 2^{nd} order derivatives, 3 1^{st} order derivatives, and 1 0^{th} order derivative (i.e., the value), for a total of 56 derivatives.

89-90:

Read in lens data and output it.

91-97:

Initialize the `DIFFERENTIAL` variables and associate program variables with the system independent variables.

98:

Trace the ray (and, at the same time, calculate *all* derivatives).

99-113:

Output the third order aberration coefficients.

114-137:

Output the fifth order aberration coefficients.

Given in Fig. 6.17 are the results of executing this program with an input lens called the "Cruickshank triplet" It is diagrammed in Fig. 6.18. Comparing the printed values of the third order derivatives to Table 6.1, we see that all derivatives that ought to be zero actually

```
$ type thes03.len

 0.0       0   0.0        0   1.0     0
 0.0       0  -0.1132     0   1.0     0
 4.82439   0   0.040278   0   1.6162  0
-0.753929  0   0.016851   0   1.0     0
-1.64505   0   0.0096145  0   1.5725  0
 5.11794   0   0.041353   0   1.0     0
OECE    0.0    0.03    22.0    5.0    0.0
OECC    0 0 0 0 0
 0.0       0   0.097385   0   1.0     0
 0.310726  0   0.0313246  0   1.6162  0
 -1.46116  0   0.336      0   1.0     0
 0.0       0   0.5        0   1.0     0
 0.0       0   0.0        0   1.0     0

$ run parax
Starting with Init_Diffs.
All done with Init_Diffs.

Input lens file name:  thes03.len

Surf TY  Curvature  CVC      Thickness   THC      Index    INC
---- --  ---------  ---      ---------   ---      -----    ---

  0  0   0.00000000   0         0.000000  0      1.000000  0
  1  0   0.00000000   0        -0.113200  0      1.000000  0
  2  0   4.82438993   0         0.040278  0      1.616200  0
  3  0  -0.75392902   0         0.016851  0      1.000000  0
  4  0  -1.64505005   0         0.009615  0      1.572500  0
  5  0   5.11793995   0         0.041353  0      1.000000  0
  6     OECE      0.000000      0.030000     22.000000      5.000000      0.000000
        OECC         0             0             0              0             0
     0   0.00000000   0         0.097385  0      1.000000  0
  7  0   0.31072599   0         0.031325  0      1.616200  0
  8  0  -1.46115994   0         0.336000  0      1.000000  0
  9  0   0.00000000   0         0.500000  0      1.000000  0
 10  0   0.00000000   0         0.000000  0      1.000000  0

Surf        X           Y           Z          TAN X       TAN Y
----     --------    --------    --------    --------    --------

  1:     0.000000    0.000000    0.000000    0.000000    0.000000
  2:     0.000000    0.000000    0.000000    0.000000    0.000000
  3:     0.000000    0.000000    0.000000    0.000000    0.000000
  4:     0.000000    0.000000    0.000000    0.000000    0.000000
  5:     0.000000    0.000000    0.000000    0.000000    0.000000
  6:    -0.000000   -0.032356    0.000000    0.087489   -0.405569
  7:     0.008592   -0.072185    0.000821    0.050250   -0.228790
  8:     0.009896   -0.078124   -0.004546    0.073384   -0.302594
  9:     0.034887   -0.181171    0.000000    0.073384   -0.302594
 10:     0.071579   -0.332468    0.000000    0.073384   -0.302594
 Im:     0.000000    0.000000    0.000000    0.000000    0.000000

Paraxial Coefficients
---------------------

A11:     0.058037
B11:    -1.010400
C11:     0.983115
```

Fig. 6.14 Last example of PARAX program

```
D11:    -0.973601
A12:     0.013113
B12:    -0.260079
C12:     0.255464
D12:    -0.249836
A21:    -0.025385
B21:     0.250834
C21:    -0.275964
D21:     0.238571
A22:     0.090918
B22:    -0.957466
C22:     1.064630
D22:    -0.907552
A11C12    A12C11 + A21C22    A22C21      0.000000
A11D12    B12C11 + A21D22    B22C21     -0.000000
B11C12    A12D11 + B21C22    A22D21      0.000000
B11D12    B12D11 + B21D22    B22D21     -0.000000
A11D11    B11C11 + A21D21    B21C21      1.000000
A12D12    B12C12 + A22D22    B22C22      1.000000
```

Fig 6.14, con't

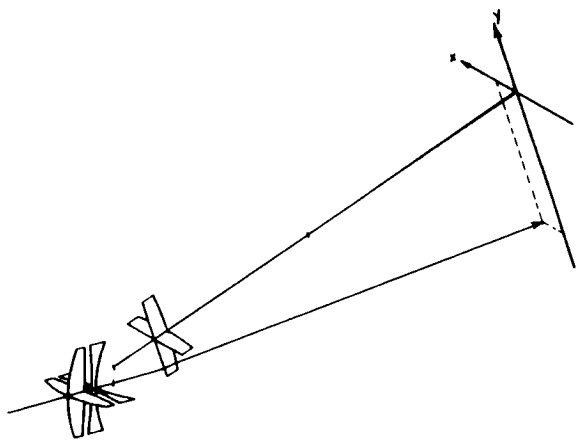

Fig 6.15 Lens Diagram for listing in Fig. 6.14

are. From the printed table, and using Table 6.1, we can derive:

Table 6.2

$$\sigma_1 = -0.366303$$
$$\sigma_2 = -0.037697$$
$$\sigma_3 = +0.049805$$
$$\sigma_4 = -0.202323$$
$$\sigma_5 = -0.016516.$$

To use as a check on these calculation, Fig. 6.19 is included here; it is a copy of Table IV from [24] and gives Buchdahl's calculated coefficients for this system. In terms of his notation in his Table IV:

```
1   ------------------------------------------------------------------------
2   --
3   -- Test of optics package -- Fifth order aberration coefficients
4   -- Robert Herloski
5   -- 12:34       09-Jan-86      vaxa
6   --
7   ------------------------------------------------------------------------
8
9   with text_io; use text_io;
10  with integer_text_io; use integer_text_io;
11  with float_text_io; use float_text_io;
12  with realfunc; use realfunc;
13  with DIFFERENTIALS; use DIFFERENTIALS;
14  with optics; use optics;
15
16  procedure fifth is
17     r1: RAY;
18     l01: LIST  := (1,1,1,0,0,0,0,0,0,0,0,0);
19     l02: LIST  := (1,1,2,0,0,0,0,0,0,0,0,0);
20     l03: LIST  := (1,1,3,0,0,0,0,0,0,0,0,0);
21     l04: LIST  := (1,2,2,0,0,0,0,0,0,0,0,0);
22     l05: LIST  := (1,2,3,0,0,0,0,0,0,0,0,0);
23     l06: LIST  := (1,3,3,0,0,0,0,0,0,0,0,0);
24     l07: LIST  := (2,2,2,0,0,0,0,0,0,0,0,0);
25     l08: LIST  := (2,2,3,0,0,0,0,0,0,0,0,0);
26     l09: LIST  := (2,3,3,0,0,0,0,0,0,0,0,0);
27     l010: LIST  := (3,3,3,0,0,0,0,0,0,0,0,0);
28     l1: LIST  := (1,1,1,1,1,0,0,0,0,0,0,0);
29     l2: LIST  := (1,1,1,1,2,0,0,0,0,0,0,0);
30     l3: LIST  := (1,1,1,1,3,0,0,0,0,0,0,0);
31     l4: LIST  := (1,1,1,2,2,0,0,0,0,0,0,0);
32     l5: LIST  := (1,1,1,2,3,0,0,0,0,0,0,0);
33     l6: LIST  := (1,1,1,3,3,0,0,0,0,0,0,0);
34     l7: LIST  := (1,1,2,2,2,0,0,0,0,0,0,0);
35     l8: LIST  := (1,1,2,2,3,0,0,0,0,0,0,0);
36     l9: LIST  := (1,1,2,3,3,0,0,0,0,0,0,0);
37     l10: LIST := (1,1,3,3,3,0,0,0,0,0,0,0);
38     l11: LIST := (1,2,2,2,2,0,0,0,0,0,0,0);
39     l12: LIST := (1,2,2,2,3,0,0,0,0,0,0,0);
40     l13: LIST := (1,2,2,3,3,0,0,0,0,0,0,0);
41     l14: LIST := (1,2,3,3,3,0,0,0,0,0,0,0);
42     l15: LIST := (1,3,3,3,3,0,0,0,0,0,0,0);
43     l16: LIST := (2,2,2,2,2,0,0,0,0,0,0,0);
44     l17: LIST := (2,2,2,2,3,0,0,0,0,0,0,0);
45     l18: LIST := (2,2,2,3,3,0,0,0,0,0,0,0);
46     l19: LIST := (2,2,3,3,3,0,0,0,0,0,0,0);
47     l20: LIST := (2,3,3,3,3,0,0,0,0,0,0,0);
48     l21: LIST := (3,3,3,3,3,0,0,0,0,0,0,0);
49      v: DIFFERENTIAL;
50
51     procedure put_coeff(s: in STRING; r: in RAY; l: in LIST) is
52
53        begin
54           put(s);
55           put(":  ");
56           put(val(r.y,l),3,6,0);
57           put("      ");
58           put(val(r.x,l),3,6,0);
59           put_line("");
60     end put_coeff;
```

Fig. 6.16 Third and Fifth Order Aberration Program

```
61
62
63          begin
64             Set_Names(11);
65             Set_Names(12);
66             Set_Names(13);
67             Set_Names(14);
68             Set_Names(15);
69             Set_Names(16);
70             Set_Names(17);
71             Set_Names(18);
72             Set_Names(19);
73             Set_Names(110);
74             Set_Names(111);
75             Set_Names(112);
76             Set_Names(113);
77             Set_Names(114);
78             Set_Names(115);
79             Set_Names(116);
80             Set_Names(117);
81             Set_Names(118);
82             Set_Names(119);
83             Set_Names(120);
84             Set_Names(121);
85             put_line("Starting with Init_Diffs.");
86             Init_Diffs;
87             put_line("All done with Init_Diffs.");
88             put_line("");
89             input_lens(NSURF);
90             dump_lens(NSURF);
91             Set_Diff(v,0.0,1.0,3);
92             Set_Diff(r1.x,0.0,1.0,1);
93             Set_Diff(r1.y,0.0,1.0,2);
94             Set_Diff(r1.z,0.0,0.0,1);
95             Set_Diff(r1.k,0.0,0.0,1);
96             r1.m := system(1).ind / sqrt(1.0 + v * v);
97             r1.l := v * r1.m;
98             traverse(r1,1,NSURF,TRUE,FALSE);
99             put_line("");
100            put_line("Third Order Derivatives");
101            put_line("-----------------------");
102            put_line("");
103            put_coeff("Dxxx",r1,101);
104            put_coeff("Dxxy",r1,102);
105            put_coeff("Dxxh",r1,103);
106            put_coeff("Dxyy",r1,104);
107            put_coeff("Dxyh",r1,105);
108            put_coeff("Dxhh",r1,106);
109            put_coeff("Dyyy",r1,107);
110            put_coeff("Dyyh",r1,108);
111            put_coeff("Dyhh",r1,109);
112            put_coeff("Dhhh",r1,1010);
113            put_line("");
114            put_line("Fifth Order Derivatives");
115            put_line("-----------------------");
116            put_line("");
117            put_coeff("Dxxxxx",r1,11);
118            put_coeff("Dxxxxy",r1,12);
119            put_coeff("Dxxxxh",r1,13);
120            put_coeff("Dxxxyy",r1,14);
```

Fig. 6.16, con't

```
121             put_coeff("Dxxxyh",r1,15);
122             put_coeff("Dxxxhh",r1,16);
123             put_coeff("Dxxyyy",r1,17);
124             put_coeff("Dxxyyh",r1,18);
125             put_coeff("Dxxyhh",r1,19);
126             put_coeff("Dxxhhh",r1,110);
127             put_coeff("Dxyyyy",r1,111);
128             put_coeff("Dxyyyh",r1,112);
129             put_coeff("Dxyyhh",r1,113);
130             put_coeff("Dxyhhh",r1,114);
131             put_coeff("Dxhhhh",r1,115);
132             put_coeff("Dyyyyy",r1,116);
133             put_coeff("Dyyyyh",r1,117);
134             put_coeff("Dyyyhh",r1,118);
135             put_coeff("Dyyhhh",r1,119);
136             put_coeff("Dyhhhh",r1,120);
137             put_coeff("Dhhhhh",r1,121);
138
139     end fifth;
```

Fig 6.16, con't

$$\begin{aligned}\sigma_1 &= -A\\ \sigma_2 &= +A\\ \sigma_3 &= -\tfrac{1}{2}B\\ \sigma_4 &= -C+\tfrac{1}{2}B\\ \sigma_5 &= +C.\end{aligned}$$

Applying these formulas to the numbers in Fig. 6.19 gives:

Table 6.3

$$\begin{aligned}\sigma_1 &= -0.36632\\ \sigma_2 &= -0.03768\\ \sigma_3 &= +0.049805\\ \sigma_4 &= -0.202325\\ \sigma_5 &= -0.016516,\end{aligned}$$

which are in excellent agreement with the results of automatic differentiation in Table 6.2.

One can also compare selected results of the fifth order derivatives in Fig. 6.17 and the S_i's in Fig. 6.19. For example, the following relationship holds:

$$-120S_1 = D_{xxxxx} = D_{yyyyy}$$

and, using the results of Fig. 6.19 and 6.17:

$$517.84 \simeq 517.829$$

Again, this is excellent agreement, given the finite precision (6 places) of the computer calculations and of Buchdahl's hand calculations (in some places, only 4 significant figures).

```
$ type cru.len
 0.0          0    1.0E7       0      1.0      0
 0.0          0   -0.273004    0      1.0      0
 2.05096      0    0.135       0      1.6578   0
-2.02632      0    0.03        0      1.6538   0
 0.336286     0    0.11846     0      1.0      0
-1.51639      0    0.03        0      1.5095   0
 2.60844      0    0.075318    0      1.0      0
 1.14458      0    0.03        0      1.6538   0
 3.37367      0    0.14        0      1.6578   0
-1.65244      0    0.719312    0      1.0      0
 0.0          0    0.0         0      1.0      0

$ run fifth
Starting with Init_Diffs.
All done with Init_Diffs.

Input lens file name:  cru.len

Surf TY  Curvature  CVC      Thickness    THC      Index    INC
---- --  ---------  ---      ---------    ---      -----    ---

  0  0   0.00000000   0 10000000.000000  0      1.000000  0
  1  0   0.00000000   0       -0.273004  0      1.000000  0
  2  0   2.05096006   0        0.135000  0      1.657800  0
  3  0  -2.02631998   0        0.030000  0      1.653800  0
  4  0   0.33628601   0        0.118460  0      1.000000  0
  5  0  -1.51638997   0        0.030000  0      1.509500  0
  6  0   2.60843992   0        0.075318  0      1.000000  0
  7  0   1.14458001   0        0.030000  0      1.653800  0
  8  0   3.37367010   0        0.140000  0      1.657800  0
  9  0  -1.65243995   0        0.719312  0      1.000000  0
 10  0   0.00000000   0        0.000000  0      1.000000  0

Surf        X           Y           Z          TAN X       TAN Y
----    --------    --------    --------    --------    --------

  1:    0.000000    0.000000    0.000000    0.000000    0.000000
  2:    0.000000    0.000000    0.000000    0.000000    0.000000
  3:    0.000000    0.000000    0.000000    0.000000    0.000000
  4:    0.000000    0.000000    0.000000    0.000000    0.000000
  5:    0.000000    0.000000    0.000000    0.000000    0.000000
  6:    0.000000    0.000000    0.000000    0.000000    0.000000
  7:    0.000000    0.000000    0.000000    0.000000    0.000000
  8:    0.000000    0.000000    0.000000    0.000000    0.000000
  9:    0.000000    0.000000    0.000000    0.000000    0.000000
 10:    0.000000    0.000000    0.000000    0.000000    0.000000

Third Order Derivatives
-----------------------

Dxxx:    0.000000       -2.197819
Dxxy:   -0.732606        0.000000
Dxxh:   -0.075384        0.000000
Dxyy:    0.000000       -0.732606
Dxyh:    0.000000       -0.075383
```

Fig. 6.17 Example of use of FIFTH program

```
Dxhh:     0.000000        -0.305028
Dyyy:    -2.197819         0.000000
Dyyh:    -0.226151         0.000000
Dyhh:    -0.105791         0.000000
Dhhh:    -0.099098         0.000000

Fifth Order Derivatives
-----------------------

Dxxxxx:     0.000000       517.829468
Dxxxxy:   103.565926         0.000000
Dxxxxh:     6.148937         0.000000
Dxxxyy:     0.000000       103.566017
Dxxxyh:     0.000000         7.510225
Dxxxhh:     0.000000        22.934950
Dxxyyy:   103.566010         0.000000
Dxxyyh:     7.056433         0.000000
Dxxyhh:    17.661415         0.000000
Dxxhhh:     1.134005         0.000000
Dxyyyy:     0.000000       103.565926
Dxyyyh:     0.000000         7.510229
Dxyyhh:     0.000000        19.185225
Dxyhhh:     0.000000         1.759343
Dxhhhh:     0.000000        11.303902
Dyyyyy:   517.829468         0.000000
Dyyyyh:    36.189880         0.000000
Dyyyhh:    87.605011         0.000000
Dyyhhh:     6.821688         0.000000
Dyhhhh:     7.084752         0.000000
Dhhhhh:   -19.319765         0.000000

$
```

Fig 6.17, con't

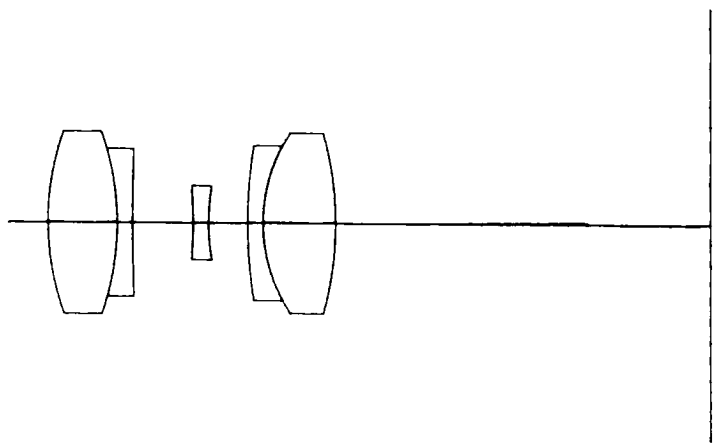

Fig 6.18 Lens Diagram for listing in Fig. 6.17

TABLE IV. Σ_3: The final aberration coefficients of orders three, five, and seven, and the coefficient of ninth-order spherical aberration.

A	0.36632	T_1	−52.394
$\bar{A}$	−0.03768	$\bar{T}_1$	2.626
B	−0.099610	T_2	13.483
C	0.15252	$\bar{T}_2$	−8.6838
$\bar{C}$	−0.016516	T_3	−18.525
		$\bar{T}_3$	1.8456
S_1	−4.3153	T_4	−8.0508
$\bar{S}_1$	0.2563	$\bar{T}_4$	−1.2620
S_2	1.2518	T_5	3.6432
$\bar{S}_2$	−2.5041	$\bar{T}_5$	−7.2301
S_3	−1.9112	T_6	−2.3658
$\bar{S}_3$	0.09448	$\bar{T}_6$	0.78632
S_4	−2.8850	T_7	−2.6256
$\bar{S}_4$	0.18069	$\bar{T}_7$	0.5095
S_5	0.29320	T_8	−2.1688
$\bar{S}_5$	0.17578	$\bar{T}_8$	0.4560
S_6	−0.47100	T_9	0.9914
$\bar{S}_6$	−0.16103	$\bar{T}_9$	2.6581
		T_{10}	−0.09120
		$\bar{T}_{10}$	−0.35311
		Q_1	−466.39

¶ The diagrams are drawn with $\mathfrak{e}_y'$ positive upwards. If U_1 is taken as positive then they are really upside down as viewed from the system, since positive $\mathfrak{e}_y'$ corresponds to a displacement away from the axis of the system.

Fig. 6.19 Buchdahl's Calculations on the "Cruickshank Triplet" (From Ref. [24])

7. FINAL STATUS AND CONCLUSIONS

The results of this thesis have confirmed that automatic differentiation is a very powerful technique for analysis of problems where symbolic or numeric differentiation is just not adequate. The ideas behind automatic differentiation are very simple (that of an ordered pair arithmetic); this technique can be programmed in any computer language, though the features of such modern programming languages as Ada and Modula-2 make it especially easy to implement with them.

As in most any program, there are some shortcomings and room for improvement. Listed below, in no particular order, are the major shortcomings of the current implementation of automatic differentiation and opportunities for improvement.

(1) The only problem encountered with this version of TeleSoft Ada (July, 1986) was a subtle bug in the printout of floating point numbers. The interpretation of AFT was such that rounding occurred at the AFT digit behind the *fixed* decimal point, not behind the *floating* decimal point, as it should. For example, the result of executing:

```
put(0.02/3.0,3,6,3);
```

is

```
6.667666E-03
```

instead of

```
6.666667E-03.
```

This required printing out all critical results in a fixed point format (EXP = 0) rather than in the preferred floating point format. TeleSoft claims to have fixed the bug in the next release of the software (February, 1987).

(2) The method by which multiplication and function of DIFFERENTIALS is implemented is admittedly inefficient. The funct and times table entries grow exponentially in size as the order of the derivative increases. This is due to the fact that the entries are calculated iteratively, i.e., as one would actually do by hand. The advantage of this implementation was that once the tables are constructed, the actual calculation of a derivative takes as short a time as possible, because of the use of pointer arithmetic (i.e., following a linked list) as opposed to a complex address calculation each time a derivative was encountered. This also allows the derivatives to be stored in an *arbitrary* order within a DIFFERENTIAL, transparent to the user.

In both cases, if the specified derivative contains the occurrence of an independent variable more than once (e.g., $\partial^5/\partial x_1 \partial x_2^2 \partial x_3^2$), then, referring back to Eq. (33) and (35) of §2.4.2, the constants C_p and C_p' are not equal to 1. However, rather than calculating C_p

and C_p', this implementation calculates the corresponding term C_p (C_p') times and adds the results together, which is an additional inefficiency.

(3) A major enhancement to the program would be to remove the inefficiencies listed above. Given in [17] is a method by which this may be achieved. There are several key ideas to Forbes' optimization of power series manipulation techniques:

(a) The individual coefficients are ordered appropriately within the (DIFFERENTIAL) array. With this ordering, Forbes derives a so-called *address function* A that, given the indices $\mathbf{r}$ of the coefficient (or derivative), one can calculate the address in the array of that term.

(b) Forbes defines an auxiliary addressing table K, which takes 2 index addresses $A(\mathbf{r})$ and $A(\mathbf{s})$ and has as an entry the address of the derivative (or coefficient) to which these two terms contribute. This implies that, instead of calculating one *resulting* coefficient at a time, Forbes takes succeeding *input* coefficients, figures out in which terms these contribute, and adds each partial sum to the corresponding *result* coefficient.

(c) My implementation of "functions" is similar to Forbes' implementation of "function composition". However, Forbes shows that for simple functions such as quotient, sine, cosine, square root, etc., there are simple *recursive* formulas that can be used to calculate the resulting series terms (or derivatives). For example,Forbes shows that if $h(\mathbf{x})=f(\mathbf{x})/g(\mathbf{x})$, then one can derive the resulting power series coefficients via:

$$h_{\mathbf{r}} = \frac{(f_{\mathbf{r}} - \sum_{\mathbf{s}\neq\mathbf{0}}^{\mathbf{s}\leq\mathbf{r}} h_{\mathbf{r}-\mathbf{s}} g_{\mathbf{s}})}{g_{\mathbf{0}}}.$$

This recursive equation could be adapted to the calculation of derivatives of quotients, and similar equations for the sine, cosine, square root, etc., of a series could be adapted to calculating derivatives of a sine, cosine, square root, etc. of a DIFFERENTIAL.

8. REFERENCES

1. M. Webster and S. Steinberg, "A Survey of Symbolic Differentiation Implementations," Proceedings of the 1984 MACSYMA User's Conference, General Electric, Schenectady, New York, pp. 330 – 355 (1984).

2. L. B. Rall, "The Arithmetic of Differentiation," *Mathematics Magazine* 59:5, pp. 275 – 282 (1986).

3. L. B. Rall, Automatic Differentation: Techniques and Applications. Lecture Notes in Computer Science, vol. 120, Springer-Verlag, New York, 1981.

4. G. Kedem, "Automatic Differentiation of Computer Programs," *ACM Trans. on Math. Soft.* 6:2, pp. 150 – 165 (1980).

5. L. B. Rall, "Differentiation in Pascal-SC: Type GRADIENT," *ACM Trans. on Math. Soft.* 10:2, pp. 161 – 184 (1984).

6. T. B. Anderson, "Automatic computation of optical aberration coefficients," *Appl. Opt.* 19:22, pp. 3800 – 3816 (1980).

7. T. B. Anderson, "Optical aberration coefficients: FORTRAN subroutines for symmetrical systems," *Appl. Opt.* 20:18, pp. 3263 – 3268 (1981).

8. T. B. Anderson, "Optical aberration functions: derivatives with respect to axial distances for symmetrical systems," *Appl. Opt.* 21:10, pp. 1817 – 1823 (1982).

9. T. B. Anderson, "Optical aberration functions: chromatic aberrations and derivatives with respect to refractive indices for symmetrical systems," *Appl. Opt.* 21:22, pp. 4040 – 4044 (1982).

10. T. B. Anderson, "Optical aberration functions: derivatives with respect to surface parameters for symmetrical systems," *Appl. Opt.* 24:8, pp. 1122 – 1129 (1985).

11. G. W. Forbes and M. Andrews, "Concatenation of symmetric systems in Hamiltonian Optics," *J. Opt. Soc. Am.* 73:6, pp. 776 – 781 (1983).

12. G. W. Forbes, "Order doubling in the computation of aberration coefficients," *J. Opt. Soc. Am.* 73:6, pp. 782 – 788 (1983).

13. G. W. Forbes, "Chromatic coordinates in aberration theory," *J. Opt. Soc. Am. A* 1:4, pp. 344 – 349 (1984).

14. G. W. Forbes, "Weighted truncation of power series and the computation of chromatic aberration coefficients," *J. Opt. Soc. Am. A* 1:4, pp. 350 – 355 (1984).

15. G. W. Forbes, "Extension of the convergence of multivariate aberration series," *J. Opt. Soc. Am. A* 3:9, pp. 1376 – 1383 (1986).

16. G. W. Forbes, "Truncation and manipulation of multivariate power series," *J. Comp. and Appl. Math.* 15, pp. 27 – 36 (1986).

17. G. W. Forbes, "Automation of the manipulation of multivariate power series," *J. Comp. and Appl. Math.* 15, pp. 37 – 58 (1986).

18. D. Knuth, Seminumerical Algorithms: The Art of Computer Programming, Addison-Wesley, Massachusetts, 1981.

19. "ANSI/MIL – STD – 1815A: Ada Programming Language," Department of Defense, Washington, 22 January 1983.

20. M. Born and E. Wolf, Principles of Optics, Pergamon, New York, 1975.

21. W. J. Smith, Modern Optical Engineering, McGraw-Hill, New York, 1966.

22. MIL – HDBK – 141, Military Standardization Handbook on Optical Design, Department of Defense, Washington, 1962.

23. R. K. Luneburg, Mathematical Theory of Optics, University of California Press, California, p. 219 (1964).

24. H. A. Buchdahl, Optical Aberration Coefficients, Dover, New York, 1968.

9. PROGRAM LISTINGS

9.1 REALFUNC.ADA

```
------------------------------------------------------------------------
------------------------------------------------------------------------
-- Package to Implement Real Functions
-- Copyright (c) 1986,1987 Robert P. Herloski
-- All rights reserved.
--
-- 14:31       23-Oec-86      VAXA
-- Compiler:  TeleGen2 tsada/vms (by TeleSoft)
--
-- Since there is no real function package provided by TeleSoft, I have
-- provided my own package, which interfaces to VMS FORTRAN real function
-- calls.
------------------------------------------------------------------------
------------------------------------------------------------------------

with System; use System;

------------------------------------------------------------------------
-- Package Specification
------------------------------------------------------------------------
package realfunc is
   function sin(arg: FLOAT) return FLOAT;
   function sin(arg: LONG_FLOAT) return LONG_FLOAT;
   function cos(arg: FLOAT) return FLOAT;
   function cos(arg: LONG_FLOAT) return LONG_FLOAT;
   function exp(arg: FLOAT) return FLOAT;
   function exp(arg: LONG_FLOAT) return LONG_FLOAT;
   function sqrt(arg: FLOAT) return FLOAT;
   function sqrt(arg: LONG_FLOAT) return LONG_FLOAT;
end realfunc;

------------------------------------------------------------------------
-- Package Body
------------------------------------------------------------------------
package body realfunc is

   ---------------------------------------------------------------------
   -- These are the definitions for the FORTRAN Real function calls.  Note
   -- that values are passed by address.
   ---------------------------------------------------------------------
   procedure FSQRT(farg,freslt: in System.Address);
   pragma Interface(VMS,FSQRT);
   procedure FSIN(farg,freslt: in System.Address);
   pragma Interface(VMS,FSIN);
   procedure FCOS(farg,freslt: in System.Address);
   pragma Interface(VMS,FCOS);
   procedure FEXP(farg,freslt: in System.Address);
   pragma Interface(VMS,FEXP);

   ---------------------------------------------------------------------
   -- function sqrt
   ---------------------------------------------------------------------

   function sqrt(arg: FLOAT) return FLOAT is
      temp1,temp2: LONG_FLOAT;
```

```
begin
   temp1 := LONG_FLOAT(arg);
   FSQRT(temp1'Address,temp2'Address);
   return FLOAT(temp2);
end sqrt;

-------------------------------------------------------------------------
-- function sqrt
-------------------------------------------------------------------------

function sqrt(arg: LONG_FLOAT) return LONG_FLOAT is
   temp1,temp2: LONG_FLOAT;

begin
   temp1 := arg;
   FSQRT(temp1'Address,temp2'Address);
   return temp2;
end sqrt;

-------------------------------------------------------------------------
-- function sin
-------------------------------------------------------------------------

function sin(arg: FLOAT) return FLOAT is
   temp1,temp2: LONG_FLOAT;

begin
   temp1 := LONG_FLOAT(arg);
   FSIN(temp1'Address,temp2'Address);
   return FLOAT(temp2);
end sin;

-------------------------------------------------------------------------
-- function sin
-------------------------------------------------------------------------

function sin(arg: LONG_FLOAT) return LONG_FLOAT is
   temp1,temp2: LONG_FLOAT;

begin
   temp1 := arg;
   FSIN(temp1'Address,temp2'Address);
   return temp2;
end sin;

-------------------------------------------------------------------------
-- function cos
-------------------------------------------------------------------------

function cos(arg: FLOAT) return FLOAT is
   temp1,temp2: LONG_FLOAT;

begin
   temp1 := LONG_FLOAT(arg);
   FCOS(temp1'Address,temp2'Address);
   return FLOAT(temp2);
end cos;

-------------------------------------------------------------------------
-- function cos
-------------------------------------------------------------------------
```

```
   function cos(arg: LONG_FLOAT) return LONG_FLOAT is
      temp1,temp2: LONG_FLOAT;

   begin
      temp1 := arg;
      FCOS(temp1'Address,temp2'Address);
      return temp2;
   end cos;

   ------------------------------------------------------------------------
   -- function exp
   ------------------------------------------------------------------------

   function exp(arg: FLOAT) return FLOAT is
      temp1,temp2: LONG_FLOAT;

   begin
      temp1 := LONG_FLOAT(arg);
      FEXP(temp1'Address,temp2'Address);
      return FLOAT(temp2);
   end exp;

   ------------------------------------------------------------------------
   -- function exp
   ------------------------------------------------------------------------

   function exp(arg: LONG_FLOAT) return LONG_FLOAT is
      temp1,temp2: LONG_FLOAT;

   begin
      temp1 := arg;
      FEXP(temp1'Address,temp2'Address);
      return temp2;
   end exp;

end realfunc;
```

9.2 REALFUNC.FOR

```
C ***************************************************************************
C ** Fortran interface to Ada subroutines
C ** Copyright (c) 1986,1987 Robert P. Herloski
C ** All rights reserved.
C **
C ** 14:36      23-Dec-86      VAXA
C ** Compiler: VMS FORTRAN
C **
C ** These are the FORTRAN real function routines called by the Ada Inter-
C ** face procedures.
C ***************************************************************************
C
      subroutine FSQRT(arg,reslt)
      real*8 arg,reslt
      reslt=dsqrt(arg)
      end
C
      subroutine FSIN(arg,reslt)
      real*8 arg,reslt
      reslt=dsin(arg)
      end
C
      subroutine FCOS(arg,reslt)
      real*8 arg,reslt
      reslt=dcos(arg)
      end
C
      subroutine FEXP(arg,reslt)
      real*8 arg,reslt
      reslt=dexp(arg)
      end
```

9.3 DIFFSPEC.ADA

```
------------------------------------------------------------------------------
------------------------------------------------------------------------------
-- Package to implement differentials
-- Copyright (c) 1986,1987 Robert P. Herloski
-- All rights reserved.
--
-- 09:15    23-Dec-86     VAXA
-- Compiler:  TeleGen2 tsada/vms (by TeleSoft)
--
------------------------------------------------------------------------------
--
-- File: diffspec.ada
--
-- This file contains the entire package definition for the package
-- DIFFERENTIALS.  The purpose of this package is to define a new data type
-- called DIFFERENTIAL and define some functions that operate on this data
-- type, such as +,-.*,/,sin,exp, etc.  The full user interface to this package
-- is given here.
--
--
-- Description of DIFFERENTIALS
--
-- A full description of the intent of this package is given in the thesis
-- proposal: "Automatic Differentiation:  Implementation in Ada", dated
-- April 14, 1986.  In brief: The process of differentiation is relatively
-- mechanical; that is, differentiation follows a set of well defined rules.
-- There exist many packages that SYMBOLICALLY differentiate given functions.
-- However, there also exists a method to differentiate a given function and
-- evaluate that new function at a given point (at the same time).  This pro-
-- cedure is called AUTOMATIC DIFFERENTIATION.  It is based on several facts:
--
-- (1) Many functions programmed into computers are composed from a basic set
--     of library functions, such as:  {+,-.*,/,sin,cos,sqrt}, etc.
--
-- (2) Many functions can be decomposed into a corresponding Taylor Series,
--     which contains as coefficients of the expansion the value of the func-
--     tion and values of all derivatives of the function at the expansion
--     point.
--
-- (3) Given one or two Taylor series, there exist simple formulas that allow
--     one to find the +,-,*, and / of two series, and the sin, cos, sqrt,
--     and exp of a single series.
--
-- (4) Starting with an initial Taylor series, one can apply the formulas in
--     (3) according to the function definition (1); the result will be a
--     Taylor series whose coefficients are the value and derivatives of the
--     function of interest.
--
-- A logical abstraction of this procedure leads one to the concept of a
-- data type called "DIFFERENTIAL".  This type is, at its simplest, an
-- array of values (say, of type FLOAT) that correspond to the value and par-
-- tial derivatives of the appropriate function, evaluated at the specified
-- point (i.e., a Taylor series).  One can then define functions that
-- operate on this data type, such as +, ,*,/,sin, etc.  Embedded in the def-
-- initions of the functions are the chain rules of differentiation that
-- apply to that function.  A very simple example is:
--
--                    m             m       m
--                   d             d D     d E
```

```
--                    ----- (D + E) = ----- + -----
--                      m               m       m
--                     dx              dx      dx
--
-- In other words, if I am given a DIFFERENTIAL D and a DIFFERENTIAL E, to find
-- their sum, I just add together each of the terms in the array that consti-
-- stutes the DIFFERENTIAL.  Note that this is done NUMERICALLY (ie, with num-
-- bers plugged in everywhere) rather than SYMBOLICALLY.
--
-- USE OF DIFFERENTIALS
--
-- The data type LIST is used to refer to specific derivatives of interest.
-- In the user's program the user has associated his independent variables with
-- integers using Set_Diff [Set_Diff initializes a differential with a value
-- and a 1st partial derivative w.r.t. the ith independent variable].  These
-- integers can be used in a LIST array to access the various derivatives.
-- For example, if one desires to know the value of:
--
--                                4
--                               d
--                          ------------  (D),
--                                2
--                           dv dv  dv
--                             1  3   5
--
-- one can set a LIST variable, say L1, to (1,3,3,5,0,0,...) and then call
-- val(D,L1) to access that value.
--
-- Set_Names is used to tell the system what the maximum-order derivatives
-- will be.  Init_Diffs is then called to initialize the internal structures
-- and figure out what lower order derivatives are needed.  If it turns out
-- too many derivatives are needed, the exception TOO_MANY_DIFFS is raised.
--
-- ACKNOWLEDGEMENTS
--
-- The original implementation of these algorithms was in Smalltalk, on the
-- Xerox Alto microprogrammable workstation.  Dr. Sidney Marshall, a scientist
-- at the Xerox Webster Research Center, developed these algorithms and methods
-- of representing differentiation around 1979.  This Ada implementation is
-- thus an adaptation of the original Smalltalk code.
--
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
--
-- Package SPECIFICATION
--
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

with text_io; use text_io;
with float_text_io; use float_text_io;
with realfunc; use realfunc;

package DIFFERENTIALS is

-- constants

   max_level: constant INTEGER := 11;        -- maximum order of a derivative
   max_length: constant INTEGER := 125;      -- maximum # of derivatives at once

-- type definitions

   type DIFFERENTIAL is private;
```

```
   type LIST is array(1..(max_level+1)) of INTEGER;

-- exceptions

   TOO_MANY_DIFFS: exception;              -- too many derivatives have been
                                           --   specified or are needed
   LIST_ERROR: exception;                  -- illegal LIST construction (last
                                           --   element /= 0)
   LIST_NOT_IN_NAMES: exception;           -- an uninitialized derivative has
                                           --   been requested
   DIFF_MATH_ERROR: exception;             -- math error such as divide by 0
                                           --   or sqrt of negative number

-- binary functions (in "diffbody.ada")

   function "*"(D,E: DIFFERENTIAL) return DIFFERENTIAL;
   function "*"(x: FLOAT; D: DIFFERENTIAL) return DIFFERENTIAL;
   function "*"(D: DIFFERENTIAL; y: FLOAT) return DIFFERENTIAL;
   function "+"(D,E: DIFFERENTIAL) return DIFFERENTIAL;
   function "+"(x: FLOAT; D: DIFFERENTIAL) return DIFFERENTIAL;
   function "+"(D: DIFFERENTIAL; y: FLOAT) return DIFFERENTIAL;
   function "-"(D,E: DIFFERENTIAL) return DIFFERENTIAL;
   function "-"(x: FLOAT; D: DIFFERENTIAL) return DIFFERENTIAL;
   function "-"(D: DIFFERENTIAL; y: FLOAT) return DIFFERENTIAL;
   function "/"(D,E: DIFFERENTIAL) return DIFFERENTIAL;
   function "/"(x: FLOAT; D: DIFFERENTIAL) return DIFFERENTIAL;
   function "/"(D: DIFFERENTIAL; y: FLOAT) return DIFFERENTIAL;

-- unary functions (in "diffbody.ada")

   function partial(D: DIFFERENTIAL; i: INTEGER) return DIFFERENTIAL;
   function sin(D: DIFFERENTIAL) return DIFFERENTIAL;
   function cos(D: DIFFERENTIAL) return DIFFERENTIAL;
   function exp(D: DIFFERENTIAL) return DIFFERENTIAL;
   function sqrt(D: DIFFERENTIAL) return DIFFERENTIAL;
   function recip(D: DIFFERENTIAL) return DIFFERENTIAL;

-- element access (in "diffbodya.ada")

   procedure dump_diff(R: in DIFFERENTIAL);
   procedure put(D: in DIFFERENTIAL; l: in LIST);
   function val0(D: DIFFERENTIAL) return FLOAT;
   function val(D: DIFFERENTIAL; l: LIST) return FLOAT;

-- Initialization (in "diffbodya.ada")

   procedure Set_Names(l: in LIST);
   procedure Set_Diff(D: in out DIFFERENTIAL; value,dval: in FLOAT;
                      i: in INTEGER);
   function Const_Diff(value: FLOAT) return DIFFERENTIAL;
   procedure Init_Diffs;

-- Debugging (in "diffbodya.ada")

   procedure set_debug;
   procedure clear_debug;
   procedure dump_order_table;
   procedure dump_times_table;
   procedure dump_funct_table;
   procedure dump_names;

-- private type
```

```
private
   type DIFFERENTIAL is array (1..max_length) of FLOAT;

end DIFFERENTIALS;
```

9.4 DIFFBODY.ADA

```
------------------------------------------------------------------------------
------------------------------------------------------------------------------
-- Package to implement differentials
-- Copyright (c) 1986,1987 Robert P. Herloski
-- All rights reserved
--
-- 12:39    05-Jan-87      vaxa
-- Compiler: TeleGen2 tsada/vms (by TeleSoft)
--
------------------------------------------------------------------------------
--
-- File: diffbody.ada
--
-- This file contains the body of the package DIFFERENTIALS.  Some of the
-- procedure definitions are merely stubs; those bodies are in a separate
-- file called "diffbodya.ada".
--
------------------------------------------------------------------------------
------------------------------------------------------------------------------
--
-- Package BODY
--
------------------------------------------------------------------------------
------------------------------------------------------------------------------

with text_io; use text_io;
with integer_text_io; use integer_text_io;
with float_text_io; use float_text_io;
with realfunc; use realfunc;

package body DIFFERENTIALS is

-- types

   --*************************************************************************
   --**
   --** These types define several types of nodes that are used in the funct
   --** and times tables, which are complex linked-list structures.
   --**
   --*************************************************************************

   type INODE;                                        -- Index Node
   type INODEPTR is access INODE;
   type INODE is record
                   val: INTEGER;      -- an index into the DIFFEREN-
                                      --   TIAL array

                   nexti: INODEPTR;
                 end record;
   type PNODE;                                        -- Product Node
   type PNODEPTR is access PNODE;
   type PNODE is record
                   nextp: PNODEPTR;
                   nexti: INODEPTR;
                 end record;
   type CNODE;                                        -- Coefficient Node
   type CNODEPTR is access CNODE;
   type CNODE is record
                   nextc: CNODEPTR;
                   nextp: PNDDEPTR;
```

```
            end record;

-- Scalars

   tc: CNODEPTR;                                  -- temporary
   tp: PNODEPTR;                                  -- temporary
   ti: INODEPTR;                                  -- temporary
   length: INTEGER := 1;                          -- number of coefficients used
   user_length: INTEGER := 1;                     -- # of coeff. used by user
   level: INTEGER := 0;                           -- temporary
   DEBUG: BOOLEAN := FALSE;                       -- flag to indicate debug mode

-- vectors

   --*************************************************************************
   --**
   --** "names" is an array of LISTs that define which derivative each element
   --** of a DIFFERENTIAL is.  No special order is implied in names.  The pro-
   --** cedure "Init_Diffs" constructs the linked-list structures funct_table
   --** and times_table, which are used by unary functions and "*", respec-
   --** tively, to calculate the appropriate derivative value for each element
   --** of a DIFFERENTIAL.  Order_table is generated to tell which order one
   --** has to calculate the derivatives (i.e., low order first).  "dvect" is
   --** an auxiliary vector used by unary functions in the calculation of
   --** derivatives; in use it contains the value of the derivative of the
   --** unary function itself; e.g.:
   --**
   --**                    d                          d
   --**                 ------- {f[g(x)]} = f' * ------- [g(x)]
   --**                   dx                         dx
   --**
   --** (dvect holds the values f').
   --**
   --*************************************************************************

   funct_table, times_table: array (1..max_length) of CNODEPTR;
   order_table: array (1..max_length) of INTEGER := (1..max_length => 0);
   dvect: array (1..(max_level+1)) of FLOAT;
   names: array(1..max_length) of LIST := (1..max_length =>
                                           (1..(max_level+1) => 0));

-- stubs

   -- VECTOR MANIPULATION
      ---------------------------------------
      -- function "+"
      --
      -- (This function should be a stub, but can't because of Ada.)
      -- This function concatenates two LISTs (the end of a list is des-
      -- ignated by the first occurrence of a "0").
      ---------------------------------------
      function "+"(l,ri: LIST) return LIST is
         R: LIST := (others => 0);
         i,j: INTEGER := 1;
         LIST_index_error: exception;

         begin
            while (l(i) /= 0) loop
               if ((i > l'LAST) or (j > R'LAST)) then
                  raise LIST_index_error;
               end if;
               R(j) := l(i);                    -- put elements of l in R
               i := i + 1;
```

```
            j := j + 1;
         end loop;
         i := 1;
         while (ri(i) /= 0) loop
            if ((i > ri'LAST) or (j > R'LAST)) then
               raise LIST_index_error;
            end if;
            R(j) := ri(i);                    -- then put elements of ri in R
            i := i + 1;
            j := j + 1;
         end loop;
         return R;
      end "+";
   ------------------------------------------------------------------
   function list_prepend(l: LIST; i: INTEGER) return LIST is separate;
   procedure dump_list(l: in LIST) is separate;
   function as_list(i: INTEGER) return LIST is separate;
   function len(l: LIST) return INTEGER is separate;
   function equals(l,r: LIST) return BOOLEAN is separate;
   function car(l: LIST) return LIST is separate;
   function cdr(l: LIST) return LIST is separate;
   function empty(l: LIST) return BOOLEAN is separate;
   function max(i,j: INTEGER) return INTEGER is separate;

-- TABLE MANIPULATION
   function names_lookup(l: LIST; insert_flag: BOOLEAN) return INTEGER is
      separate;
   procedure Init_Times_Table is separate;
   procedure Init_Funct_Table is separate;
   procedure Order_tables is separate;

-- ELEMENT ACCESS
   procedure dump_diff(R: in DIFFERENTIAL) is separate;
   function val0(D: DIFFERENTIAL) return FLOAT is separate;
   function val(D: DIFFERENTIAL; l: LIST) return FLOAT is separate;
   procedure put(D: in DIFFERENTIAL; l: in LIST) is separate;

-- DEBUGGING
   procedure set_debug is separate;
   procedure clear_debug is separate;
   procedure dump_names is separate;
   procedure dump_order_table is separate;
   procedure dump_times_table is separate;
   procedure dump_funct_table is separate;

-- INITIALIZATION
   procedure Set_Names(l: in LIST) is separate;
   procedure Init_Diffs is separate;
   procedure Set_Diff(D: in out DIFFERENTIAL; value, dval: in FLOAT;
                      i: in INTEGER) is separate;
   function Const_Diff(value: FLOAT) return DIFFERENTIAL is separate;

--*************************************************************************
--**
--** Binary Functions
--**
--*************************************************************************

------------------------------------------------------------------------
--
-- Function: funct
--
-- This function reads through the funct_table and applies the rules of dif-
```

```
-- ferentiation programmed therein to each element of the DIFERENTIAL array.
-- dvect holds the values of the total derivatives of the functions, evaluated
-- appropriately.
--
-------------------------------------------------------------------------------

   function funct(D: DIFFERENTIAL) return DIFFERENTIAL is

      sum,coeff,prod: FLOAT;
      j,k,l: INTEGER;
      R: DIFFERENTIAL;

      begin
         for l in 1..length loop
            k := order_table(l);
            sum := 0.0;
            case k is
               when 1       => j := 1;       -- kludge to handle diff. btwn.
               when others  => j := 2;       --  value and derivative eval.
            end case;
            tc := funct_table(k);
            while (tc /= null) loop      -- for all coeff. (* by dvect)
               coeff := 0.0;
               tp := tc.nextp;
               while (tp /= null) loop       -- for all products (add them)
                  prod := 1.0;
                  ti := tp.nexti;
                  while (ti /= null) loop-- for all terms (* them)
                     prod := prod * D(ti.val);
                     ti := ti.nexti;     -- next term (index of array)
                  end loop;
                  coeff := coeff + prod;
                  tp := tp.nextp;     -- next product
               end loop;
               sum := sum + coeff * dvect(j);
               j := j + 1;
               tc := tc.nextc;           -- next coefficient
            end loop;
            R(k) := sum;
         end loop;
         return R;

   end funct;

-------------------------------------------------------------------------------
--
-- Function: DIFF * DIFF
--
-- This (overloaded) function takes the product of two differentials, and uses
-- the times_table to get the rules (like the function funct above),
--
-------------------------------------------------------------------------------

   function "*"(D,E: DIFFERENTIAL) return DIFFERENTIAL is
      l: INTEGER;
      R: DIFFERENTIAL;
      sum: FLOAT;
      k: INTEGER;

      begin
         for l in 1..length loop
            k := order_table(l);
            sum := 0.0;
```

```
            tp := times_table(k).nextp;
            while (tp /= null) loop      -- sum up the products
               sum := sum + D(tp.nexti.val) * E(tp.nexti.nexti.val);
               tp := tp.nextp;           -- next product
            end loop;
            R(k) := sum;
         end loop;
         if (DEBUG) then
            put_line("-----DIFF * DIFF-----");
            put("     ");
            dump_diff(D);
            put("     ");
            dump_diff(E);
            put("     ");
            dump_diff(R);
            put_line("");
         end if;
         return R;
      end "*";

-------------------------------------------------------------------------
--
-- Function: DIFF * FLOAT
--
-- This function finds the product of a DIFFERENTIAL and a FLOAT.  The differ-
-- entiation rule in this case is obvious.
--
-------------------------------------------------------------------------

   function "*"(D: DIFFERENTIAL; y: FLOAT) return DIFFERENTIAL is
      R: DIFFERENTIAL;
      k: INTEGER;
      begin
         for k in 1..length loop
            R(k) := D(k) * y;         -- this is all there is!!
         end loop;
         if (DEBUG) then
            put_line("-----DIFF * FLOAT-----");
            put("     ");
            dump_diff(D);
            put("     ");
            put(y,3,6,3);
            put_line("");
            put("     ");
            dump_diff(R);
            put_line("");
         end if;
         return R;
      end "*";

-------------------------------------------------------------------------
--
-- Function: FLOAT * DIFF
--
-- You have to explicitly define the product for the terms in reverse order.
--
-------------------------------------------------------------------------

   function "*"(x: FLOAT; D: DIFFERENTIAL) return DIFFERENTIAL is
      R: DIFFERENTIAL;
      k: INTEGER;
      begin
         for k in 1..length loop
```

```
            R(k) := x * D(k);
         end loop;
         if (DEBUG) then
            put_line("-----FLOAT * DIFF-----");
            put("      ");
            put(x,3,6,3);
            put_line("");
            put("      ");
            dump_diff(D);
            put("      ");
            dump_diff(R);
            put_line("");
         end if;
         return R;
      end "*";


-------------------------------------------------------------------------------
--
-- Function: DIFF + DIFF
--
-------------------------------------------------------------------------------

   function "+"(D,E: DIFFERENTIAL) return DIFFERENTIAL is
      R: DIFFERENTIAL;
      k: INTEGER;
      begin
         for k in 1..length loop
            R(k) := D(k) + E(k);      -- here's the rule for addition
         end loop;
         if (DEBUG) then
            put_line("-----DIFF + DIFF-----");
            put("      ");
            dump_diff(D);
            put("      ");
            dump_diff(E);
            put("      ");
            dump_diff(R);
            put_line("");
         end if;
         return R;
   end "+";


-------------------------------------------------------------------------------
--
-- Function: FLOAT + DIFF
--
-------------------------------------------------------------------------------

   function "+"(x: FLOAT; D: DIFFERENTIAL) return DIFFERENTIAL is
      R: DIFFERENTIAL;
      k: INTEGER;
      begin
         R(1) := x + D(1);
         for k in 2..length loop
            R(k) := D(k);      -- here's the rule for addition
         end loop;
         if (DEBUG) then
            put_line("-----FLOAT + DIFF-----");
            put("      ");
            put(x,3,6,3);
            put_line("");
```

```
            put("     ");
            dump_diff(D);
            put("     ");
            dump_diff(R);
            put_line("");
         end if;
         return R;
   end "+";


-------------------------------------------------------------------------------
--
-- Function: DIFF + FLDAT
--
-------------------------------------------------------------------------------

   function "+"(D: DIFFERENTIAL; y: FLDAT) return DIFFERENTIAL is
      R: DIFFERENTIAL;
      k: INTEGER;
      begin
         R(1) := D(1) + y;
         for k in 2..length loop
            R(k) := D(k);      -- here's the rule for addition
         end loop;
         if (DEBUG) then
            put_line("-----DIFF + FLDAT-----");
            put("     ");
            dump_diff(D);
            put("     ");
            put(y,3,6,3);
            put_line("");
            put("     ");
            dump_diff(R);
            put_line("");
         end if;
         return R;
   end "+";


-------------------------------------------------------------------------------
--
-- Function: DIFF   DIFF
--
-------------------------------------------------------------------------------

   function "-"(D,E: DIFFERENTIAL) return DIFFERENTIAL is
      R: DIFFERENTIAL;
      k: INTEGER;
      begin
         for k in 1..length loop
            R(k) := D(k)   E(k);      -- here's the rule for sub.
         end loop;
         if (DEBUG) then
            put_line("-----DIFF   DIFF-----");
            put("     ");
            dump_diff(D);
            put("     ");
            dump_diff(E);
            put("     ");
            dump_diff(R);
            put_line("");
         end if;
         return R;
```

```
   end "-";


-------------------------------------------------------------------------------
--
-- Function: FLOAT - DIFF
--
-------------------------------------------------------------------------------

   function "-"(x: FLOAT; D: DIFFERENTIAL) return DIFFERENTIAL is
      R: DIFFERENTIAL;
      k: INTEGER;
      begin
         R(1) := x - D(1);
         for k in 2..length loop
            R(k) := 0.0 - D(k);       -- here's the rule for subtraction
         end loop;
         if (DEBUG) then
            put_line("-----FLOAT - DIFF-----");
            put("     ");
            put(x,3,6,3);
            put_line("");
            put("     ");
            dump_diff(D);
            put("     ");
            dump_diff(R);
            put_line("");
         end if;
         return R;
   end "-";


-------------------------------------------------------------------------------
--
-- Function: DIFF - FLOAT
--
-------------------------------------------------------------------------------

   function "-"(D: DIFFERENTIAL; y: FLOAT) return DIFFERENTIAL is
      R: DIFFERENTIAL;
      k: INTEGER;
      begin
         R(1) := D(1) - y;
         for k in 2..length loop
            R(k) := D(k);      -- here's the rule for subtraction
         end loop;
         if (DEBUG) then
            put_line("-----DIFF - FLOAT-----");
            put("     ");
            dump_diff(D);
            put("     ");
            put(y,3,6,3);
            put_line("");
            put("     ");
            dump_diff(R);
            put_line("");
         end if;
         return R;
   end "-";

-------------------------------------------------------------------------------
--
-- Function: recip
```

```
--
-- This function takes the reciprocal of a DIFFERENTIAL.  The format here is
-- the same as for all other (unary) functions:  First calculate the terms of
-- dvect (the total derivatives of the function itself with respect to its
-- (symbolic) argument), and then call funct.
--
-------------------------------------------------------------------------------

   function recip(D: DIFFERENTIAL) return DIFFERENTIAL is
      r,value: FLOAT;
      k: INTEGER;

      begin
         if (D(1)   0.0) then
            raise DIFF_MATH_ERROR;
         end if;
         value := 1.0/D(1);
         dvect(1) := value;
         r := 0.0   value;
         for k in 1..level loop
            value := value * r * FLOAT(k);
            dvect(k+1) := value;
         end loop;
         return funct(D);
   end recip;

-------------------------------------------------------------------------------
--
-- Function: DIFF / DIFF
--
-- This function is the division function.  It first finds recip(D), then calls
-- the * function.
--
-------------------------------------------------------------------------------

   function "/"(D,E: DIFFERENTIAL) return DIFFERENTIAL is
      R: DIFFERENTIAL;
      l: INTEGER;

      begin
         R := recip(E);
         R := D * R;
         if (DEBUG) then
            put_line("-----DIFF / DIFF-----");
            put("     ");
            dump_diff(D);
            put("     ");
            dump_diff(E);
            put("     ");
            dump_diff(R);
            put_line("");
         end if;
         return R;
   end "/";


-------------------------------------------------------------------------------
--
-- Function: FLOAT / DIFF
--
-- This function is the division function.  It first finds recip(D), then calls
-- the * function.
--
```

```
------------------------------------------------------------------------------

   function "/"(x: FLOAT; D: DIFFERENTIAL) return DIFFERENTIAL is
      R: DIFFERENTIAL;
      l: INTEGER;

      begin
         R := recip(D);
         R := x * R;
         if (DEBUG) then
            put_line("-----FLOAT / DIFF-----");
            put("     ");
            put(x,3,6,3);
            put_line("");
            put("     ");
            dump_diff(D);
            put("     ");
            dump_diff(R);
            put_line("");
         end if;
         return R;
   end "/";


------------------------------------------------------------------------------
--
-- Function: DIFF / FLOAT
--
-- This function is the division function.  It first finds recip(D), then calls
-- the * function.
--
------------------------------------------------------------------------------

   function "/"(D: DIFFERENTIAL; y: FLOAT) return DIFFERENTIAL is
      R: DIFFERENTIAL;
      l: INTEGER;

      begin
         if (y = 0.0) then
            raise DIFF_MATH_ERROR;
         end if;
         R := D * (1.0 / y);
         if (DEBUG) then
            put_line("-----DIFF / FLOAT-----");
            put("     ");
            dump_diff(D);
            put("     ");
            put(y,3,6,3);
            put_line("");
            put("     ");
            dump_diff(R);
            put_line("");
         end if;
         return R;
   end "/";

--*****************************************************************************
--**
--** Unary Functions
--**
--*****************************************************************************

------------------------------------------------------------------------------
```

```
--
-- Function: sin
--
-- This is the sin function for DIFFERENTIALS.  Note how simply dvect is cal-
-- culated.
--
-------------------------------------------------------------------------------

   function sin(D: DIFFERENTIAL) return DIFFERENTIAL is
      value: FLOAT;
      i: INTEGER;
      R: DIFFERENTIAL;

      begin
         value := D(1);
         for i in 1..(level+1) loop
            case i is
               when 1      => dvect(i) := sin(value);
               when 2      => dvect(i) := cos(value);
               when 3|4    => dvect(i) := -dvect(i-2);
               when others => dvect(i) := dvect(i-4);
            end case;
         end loop;
         R := funct(D);
         if (DEBUG) then
            put_line("-----sin(DIFF)-----");
            put("     ");
            dump_diff(D);
            put("     ");
            put_line("");
            put("     ");
            dump_diff(R);
            put_line("");
         end if;
         return R;
   end sin;


-------------------------------------------------------------------------------
--
-- Function: cos
--
-- This is the cos function for DIFFERENTIALS.  Note how simply dvect is cal-
-- culated.
--
-------------------------------------------------------------------------------

   function cos(D: DIFFERENTIAL) return DIFFERENTIAL is
      value: FLOAT;
      i: INTEGER;
      R: DIFFERENTIAL;

      begin
         value := D(1);
         for i in 1..(level+1) loop
            case i is
               when 1      => dvect(i) := cos(value);
               when 2      => dvect(i) := -sin(value);
               when 3|4    => dvect(i) := -dvect(i-2);
               when others => dvect(i) := dvect(i-4);
            end case;
         end loop;
         R := funct(D);
```

```
         if (DEBUG) then
            put_line("-----cos(DIFF)-----");
            put("     ");
            dump_diff(D);
            put("     ");
            put_line("");
            put("     ");
            dump_diff(R);
            put_line("");
         end if;
         return R;
   end cos;

-------------------------------------------------------------------------------
--
-- Function: exp
--
-------------------------------------------------------------------------------

   function exp(D: DIFFERENTIAL) return DIFFERENTIAL is
      value: FLOAT;
      l: INTEGER;
      R: DIFFERENTIAL;

      begin
         value := exp(D(1));
         for l in 1..(level+1) loop
            dvect(l) := value;            -- this is a REAL easy rule
         end loop;
         R := funct(D);
         if (DEBUG) then
            put_line("-----exp(DIFF)-----");
            put("     ");
            dump_diff(D);
            put("     ");
            dump_diff(R);
            put_line("");
         end if;
         return R;
   end exp;

-------------------------------------------------------------------------------
--
-- Function: sqrt
--
-------------------------------------------------------------------------------

   function sqrt(D: DIFFERENTIAL) return DIFFERENTIAL is
      r,value: FLOAT;
      l: INTEGER;
      RR: DIFFERENTIAL;

      begin
         if (D(1) < 0.0) then
            raise DIFF_MATH_ERROR;
         end if;
         value := sqrt(D(1));
         dvect(1) := value;
         r := 1.0/D(1);
         for l in 1..level loop
            value := value * r * (1.5 - FLOAT(l));
            dvect(l+1) := value;
         end loop;
```

```
        RR := funct(D);
        if (DEBUG) then
           put_line("-----sqrt(DIFF)-----");
           put("     ");
           dump_diff(D);
           put("     ");
           dump_diff(RR);
           put_line("");
        end if;
        return RR;
   end sqrt;

-------------------------------------------------------------------------
--
-- Function: partial
--
-- This function allows one to take the partial derivative of a DIFFERENTIAL
-- with respect to the ith variable.  This function merely shifts the Taylor
-- coefficients to correspond to a derivative operation.
--
-------------------------------------------------------------------------

   function partial(D: DIFFERENTIAL; i: INTEGER) return DIFFERENTIAL is
      loc: INTEGER;
      j: INTEGER;
      l1: LIST;
      R: DIFFERENTIAL;

   begin
      for j in 1..length loop
         l1 := list_prepend(names(j),i);
         lookup:
            begin
               loc := names_lookup(l1,FALSE);
               R(j) := D(loc);
            exception
               when LIST_NOT_IN_NAMES => R(j) := 0.0;
         end lookup;
      end loop;
      return R;
   end partial;

end DIFFERENTIALS;
```

9.5 DIFFBODYA.ADA

```
------------------------------------------------------------------------------
------------------------------------------------------------------------------
-- File of separately compiled routines for package DIFFERENTIALS
-- Copyright (c) 1986,1987 Robert P. Herloski
-- All rights reserved.
--
-- 12:24      05-Jan-87      VAXA
-- Compiler:  TeleGen2 tsada/vms (by TeleSoft)
------------------------------------------------------------------------------
------------------------------------------------------------------------------

-- ****************************************************************************
-- ****************************************************************************
-- **********   Vector Manipulation   ********************************************
-- ****************************************************************************
-- ****************************************************************************

   ---------------------------------------------------------------------------
   -- function list_prepend
   --
   -- This effectively does a "+" or concatenation of the LIST {i,0,....}
   -- with the LIST l.
   ---------------------------------------------------------------------------

separate (DIFFERENTIALS)
   function list_prepend(l: LIST; i: INTEGER) return LIST is
      j: INTEGER;
      l1: LIST;

   begin
      for j in reverse 2..l'LAST loop
         l1(j) := l(j-1);
      end loop;
      l1(1) := i;
      return l1;
   end list_prepend;

   ---------------------------------------------------------------------------
   -- procedure dump_list
   --
   -- Outputs the elements of the LIST l to STDOUT
   ---------------------------------------------------------------------------

separate (DIFFERENTIALS)
   procedure dump_list(l: in LIST) is
      i: INTEGER;

      begin
         for i in 1..l'LAST loop
            put(l(i),2);
            if (i < l'LAST) then
               put(",");
            end if;
         end loop;
         put_line("");
   end dump_list;

   ---------------------------------------------------------------------------
   -- function as_list
```

```
   --
   -- Returns the list {i,0,...}
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   function as_list(i: INTEGER) return LIST is
      R: LIST := (others => 0);

      begin
         R(1) := i;
         return R;
   end as_list;

   ------------------------------------------------------------------------
   -- function len
   --
   -- Returns the length of the LIST l, which is defined as the number of
   -- elements before the first occurrence of the integer 0.
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   function len(l: LIST) return INTEGER is
      i: INTEGER := 0;

      begin
         while (l(i+1) /= 0) loop
            i := i + 1;
         end loop;
         return i;
   end len;

   ------------------------------------------------------------------------
   -- function equals
   --
   -- Sees if two LISTs l and r are identical
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   function equals(l,r: LIST) return BOOLEAN is
      i: INTEGER;

      begin
         for i in 1..l'LAST loop
            if (l(i) /= r(i)) then
               return FALSE;
            end if;
         end loop;
         return TRUE;
   end equals;

   ------------------------------------------------------------------------
   -- function car
   --
   -- Returns the first element of l as a LIST (not exactly the same as the
   -- LISP notion of car).
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   function car(l: LIST) return LIST is
      R: LIST;
      i: INTEGER;

      begin
```

```
         R(1) := l(1);
         for i in 2..R'LAST loop
            R(i) := 0;
         end loop;
         return R;
   end car;

   ------------------------------------------------------------------------
   -- function cdr
   --
   -- Returns the tail of the LIST l (minus the first element) as a LIST.
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   function cdr(l: LIST) return LIST is
      R: LIST;

      begin
         R(1..(R'LAST-1)) := l(2..(l'LAST));
         R(R'LAST) := 0;
         return R;
   end cdr;

   ------------------------------------------------------------------------
   -- function empty
   --
   -- Tests to see if the LIST is empty (1st element = 0).
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   function empty(l: LIST) return BOOLEAN is

      begin
         if (l(1) = 0) then
            return TRUE;
         else
            return FALSE;
         end if;
   end empty;

   ------------------------------------------------------------------------
   -- function max
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   function max(i,j: INTEGER) return INTEGER is

      begin
         if (i > j) then
            return i;
         else
            return j;
         end if;
   end max;

-- ************************************************************************
-- ************************************************************************
-- **********   Table Manipulation   ***************************************
-- ************************************************************************
-- ************************************************************************

   ------------------------------------------------------------------------
   -- function names_lookup
```

```
   --
   -- names_lookup takes a LIST l and sees if it is in the "names" table any-
   -- where.  If it is, it returns the index value into "names" of its
   -- location.  insert_flag tells the routine whether or not to try to in-
   -- sert the LIST in the "names" if the LIST is not found.
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   function names_lookup(l: LIST; insert_flag: BOOLEAN) return INTEGER is
      i: INTEGER;

      begin
         for i in 1..length loop
            if (equals(l,names(i))) then
               return i;
            end if;
         end loop;
         if insert_flag then
            if (length   names'LAST) then
               raise TOO_MANY_DIFFS;
            end if;
            length := length + 1;
            names(length) := l;
            return length;
         else
            raise LIST_NOT_IN_NAMES;
         end if;
   end names_lookup;

   ------------------------------------------------------------------------
   -- procedure Init_Times_Table
   --
   -- This is the procedure that constructs the times_table from the infor-
   -- mation in "names".  Each element of "names" is passed to the recursive
   -- procedure "gen_times", which constructs the linked list structure
   -- based upon the LIST passed it.
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   procedure Init_Times_Table is
      i: INTEGER;
      null_list: LIST := (others => 0);

      ---------------------------------------------------------------
      -- procedure gen_times
      --
      -- This recursive procedure manipulates the elements of the LIST
      -- owt passed it initially to generate a linked list structure
      -- that represents how to find the appropriate derivative of the
      -- product of two DIFFERENTIALS (Taylor Series).  This algorithm
      -- was developed by Sidney Marshall of Xerox Corporation in 1979.
      ---------------------------------------------------------------

      procedure gen_times(left,right,owt: in LIST) is
         head, tail: LIST := (others => 0);

         begin
            if (empty(owt)) then
               if (tc.nextp   null) then
                  tp := new PNODE;
                  tc.nextp := tp;
               else
                  tp.nextp := new PNODE;
```

```
                tp := tp.nextp;
             end if;
             ti := new INODE;
             tp.nexti := ti;
                ti.val := names_lookup(left,TRUE);
             ti.nexti := new INODE;
             ti := ti.nexti;
                ti.val := names_lookup(right,TRUE);
          else
             head := car(owt);
             tail := cdr(owt);
             gen_times(left+head,right,tail);
             gen_times(left,right+head,tail);
          end if;
    end gen_times;

    begin      ---------------- of Init_Times_Table --------------------
       i := 1;
       while (i <= length) loop
          tc := new CNODE;
          times_table(i) := tc;
          gen_times(null_list,null_list,names(i));
          i := i + 1;
       end loop;
 end Init_Times_Table;

 ------------------------------------------------------------------------
 -- procedure Init_Funct_Table
 --
 -- This is the procedure that constructs the funct_table from the infor-
 -- mation in "names".  Each element of "names" is passed to the recursive
 -- procedure "gen_funct", which constructs the linked list structure
 -- based upon the LIST passed it.
 ------------------------------------------------------------------------

separate (DIFFERENTIALS)
 procedure Init_Funct_Table is
    i,j: INTEGER;
    null_list: LIST := (others => 0);
    c: array (1..null_list'LAST) of CNODEPTR;      -- so it's same length as
                                                   --  a LIST

    ---------------------------------------------------------------
    -- procedure gen_funct
    --
    -- This recursive procedure manipulates the elements of the LIST
    -- owt passed it initially to generate a linked list structure
    -- that represents how to find the appropriate derivative of the
    -- function of a DIFFERENTIAL (Taylor Series).  This algorithm
    -- was developed by Sidney Marshall of Xerox Corporation in 1979.
    ---------------------------------------------------------------

    procedure gen_funct(sets,inn,owt,left: in LIST; k: in INTEGER) is
       head,tail: LIST := (others => 0);
       null_list: LIST := (others => 0);
       result: LIST;
       i,l: INTEGER;

       begin
          if (not empty(left)) then
             head := car(left);
             tail := cdr(left);
             gen_funct(sets,inn+head,owt,tail,k);
```

```
            gen_funct(sets,inn,owt+head,tail,k);
         elsif (not empty(owt)) then
            gen_funct(sets+as_list(names_lookup(inn,TRUE)),car(owt),
                      null_list,cdr(owt),k);
         else
            result := sets + as_list(names_lookup(inn,TRUE));
            l := len(result);
            if (c(l)   null) then
               i := l;
               while (c(i)   null) loop
                  c(i) := new CNODE;
                  exit when i   1;
                  i := i   1;
               end loop;
               while (i < l) loop
                  c(i).nextc := c(i+1);
                  i := i + 1;
               end loop;
                if (funct_table(k)   null) then
                  funct_table(k) := c(1);
               end if;
            end if;
            if (c(l).nextp   null) then
               tp := new PNODE;
               c(l).nextp := tp;
            else
               tp := c(l).nextp;
               while (tp.nextp /= null) loop
                  tp := tp.nextp;
               end loop;
               tp.nextp := new PNODE;
               tp := tp.nextp;
            end if;
            ti := new INODE;
            tp.nexti := ti;
               ti.val := result(1);
            i := 2;
            while (result(i) /= 0) loop
               ti.nexti := new INODE;
               ti := ti.nexti;
                  ti.val := result(i);
               i := i + 1;
            end loop;
         end if;
      end gen_funct;

      begin      ----------------- of Init_Funct_Table ------------------
         i := 1;
         while (i <= length) loop
            for j in 1..c'LAST loop
               c(j) := null;
            end loop;
            if (i   1) then
               tc := new CNODE;
               funct_table(i) := tc;
                  tp := new PNODE;
                  tc.nextp := tp;
            else
               gen_funct(null_list,car(names(i)),null_list,cdr(names(i)),i);
            end if;
            i := i + 1;
         end loop;
   end Init_Funct_Table;
```

```
   ------------------------------------------------------------------------
   -- procedure Order_tables
   --
   -- Calculates the values in order_table, which tells the system in which
   -- order it should calculate the various order derivatives for times and
   -- functions.
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   procedure Order_tables is
      temp_level,i,j: INTEGER;

      begin
         order_table(1) := 1;
         j := 2;
         for temp_level in 1..level loop
            for i in 2.. length loop
               if (len(names(i)) = temp_level) then
                  order_table(j) := i;
                  j := j + 1;
               end if;
            end loop;
         end loop;
   end Order_tables;

-- ************************************************************************
-- ************************************************************************
-- **********   Element Access   ******************************************
-- ************************************************************************
-- ************************************************************************

   ------------------------------------------------------------------------
   -- procedure dump_diff
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   procedure dump_diff(R: in DIFFERENTIAL) is
      j: INTEGER;

   begin
      for j in 1..length loop
         put(R(j),3,6,3);
         put("  ");
      end loop;
      put_line("");
   end dump_diff;

   ------------------------------------------------------------------------
   -- procedure val0
   --
   -- Returns the value of the first element of a DIFFERENTIAL; i.e., the
   -- zeroeth order derivative, or value, of the Taylor Series.
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   function val0(D: DIFFERENTIAL) return FLOAT is
      begin
         return D(1);
   end val0;

   ------------------------------------------------------------------------
   -- function val
```

```
   --
   -- Returns the value of the derivative specified by the LIST l of the
   -- DIFFERENTIAL D.
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   function val(D: DIFFERENTIAL; l: LIST) return FLOAT is
      loc: INTEGER;

      begin
         if (len(l)  l'LAST) then
            raise LIST_ERROR;
         end if;
         loc := names_lookup(l,FALSE);
         return D(loc);
   end val;

   ------------------------------------------------------------------------
   -- procedure put
   --
   -- Puts the value of the derivative specified by the LIST l of the DIFF-
   -- ERENTIAL D to STDOUT.
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   procedure put(D: in DIFFERENTIAL; l: in LIST) is
      loc: INTEGER;

      begin
         if (len(l)   l'LAST) then
            raise LIST_ERROR;
         end if;
         loc := names_lookup(l,FALSE);
         put(D(loc));
   end put;

-- ************************************************************************
-- ************************************************************************
-- **********   Debugging   ***********************************************
-- ************************************************************************
-- ************************************************************************

   ------------------------------------------------------------------------
   -- procedure set_debug
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   procedure set_debug is
      begin
         DEBUG := TRUE;
   end set_debug;

   ------------------------------------------------------------------------
   -- procedure clear_debug
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   procedure clear_debug is
      begin
         DEBUG := FALSE;
   end clear_debug;

   ------------------------------------------------------------------------
```

```
   -- procedure dump_names
   --
   -- Outputs "names" to STDOUT
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   procedure dump_names is
      i: INTEGER;

      begin
         for i in 1..length loop
            put("Names(");
            put(i,2);
            put(") = ");
            dump_list(names(i));
         end loop;
   end dump_names;

   ------------------------------------------------------------------------
   -- procedure dump_order_table
   --
   -- procedure that outputs the contents of order_table to STDOUT.
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   procedure dump_order_table is
      i: INTEGER;

      begin
         for i in 1..length loop
            put("order_Table(");
            put(i,2);
            put(") = ");
            put(order_table(i),2);
            put_line("");
         end loop;
   end dump_order_table;

   ------------------------------------------------------------------------
   -- procedure dump_times_table
   --
   -- This procedure outputs the contents of the linked list structure
   -- "times_table" to STDOUT
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   procedure dump_times_table is
      i: INTEGER;

      begin
         put_line("Times_Table = ");
         for i in 1..length loop
            tp := times_table(i).nextp;
            while (tp /= null) loop
               put(tp.nexti.val,1);
               put(",");
               put(tp.nexti.nexti.val,1);
               if (tp.nextp /= null) then
                  put("  ;  ");
               end if;
               tp := tp.nextp;
            end loop;
            put_line("");
```

```
         end loop;
         put_line("End of times table.");
   end dump_times_table;

   ------------------------------------------------------------------------
   -- procedure dump_funct_table
   --
   -- This procedure outputs the contents of the linked list structure
   -- "funct_table" to STDOUT.
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   procedure dump_funct_table is
      i: INTEGER;

      begin
         put_line("Funct_Table   ");
         for i in 1..length loop
            tc := funct_table(i);
            while (tc /= null) loop
               tp := tc.nextp;
               while (tp /= null) loop
                  ti := tp.nexti;
                  while (ti /= null) loop
                     put(ti.val,1);
                     if (ti.nexti /= null) then
                        put(",");
                     end if;
                     ti := ti.nexti;
                  end loop;
                  if (tp.nextp /= null) then
                     put("  ;  ");
                  end if;
                  tp := tp.nextp;
               end loop;
               put_line("");
               if (tc.nextc /= null) then
                  put("   ");
               end if;
               tc := tc.nextc;
            end loop;
         end loop;
         put_line("End of funct_table.");
   end dump_funct_table;

-- ************************************************************************
-- ************************************************************************
-- **********   Initialization   ******************************************
-- ************************************************************************
-- ************************************************************************

   ------------------------------------------------------------------------
   -- procedure Set_Names
   --
   -- This procedure tries to put the LIST l into "names", and calculates
   -- the system variables length, user_length, and level.
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   procedure Set_Names(l: in LIST) is

      begin
         if (len(l)   l'LAST) then
```

```
            raise LIST_ERROR;
         end if;
         if (length=names'LAST) then
            raise TOO_MANY_DIFFS;
         end if;
         user_length := user_length + 1;
         length := length + 1;
         names(user_length) := l;
         level := max(level,len(l));
   end Set_Names;

   ------------------------------------------------------------------------
   -- procedure Init_Diffs
   --
   -- User-called procedure to initialize the system tables based on the
   -- entries in "names".
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   procedure Init_Diffs is

      begin
         Init_Times_Table;
         Init_Funct_Table;
         Order_tables;
   end Init_Diffs;

   ------------------------------------------------------------------------
   -- procedure Set_Diff
   --
   -- This procedure initializes a DIFFERENTIAL for the user.  It is typically
   -- used to associate a user variable with the ith independent variable;
   -- in such case the initial value is "value", and the initial derivative
   -- value "dval" is obviously 1 (dx/dx = 1).  However, one can set dval /=
   -- 1; this is useful in calculating what a constant times the derivative
   -- of the ith independent variable might be.
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   procedure Set_Diff(D: in out DIFFERENTIAL; value,dval: in FLOAT;
                      i: in INTEGER) is
      l: LIST := (others => 0);
      loc,k: INTEGER;

      begin
         D(1) := value;
         if (dval /= 0.0) then
            l(1) := i;
            loc := names_lookup(l,FALSE);
         else
            loc := 3;                          -- doesn't care what i is;
         end if;                               -- (Effectively declares a const.)
         for k in 2..(loc-1) loop
            D(k) := 0.0;
         end loop;
         D(loc) := dval;
         for k in (loc+1)..length loop
            D(k) := 0.0;
         end loop;
   end Set_Diff;

   ------------------------------------------------------------------------
   -- function Const_Diff
```

```
   --
   -- This function returns a constant DIFFERENTIAL with a val0 of value.
   ------------------------------------------------------------------------

separate (DIFFERENTIALS)
   function Const_Diff(value: FLOAT) return DIFFERENTIAL is
      R: DIFFERENTIAL;

      begin
         R(1) := value;
         R(2..R'LAST) := (2..R'LAST => 0.0);
         return R;
   end Const_Diff;
```

9.6 OPTICSSPEC.ADA

```
--------------------------------------------------------------------------
--------------------------------------------------------------------------
-- Package of Optics System Analysis
-- Copyright (c) 1986,1987 Robert P. Herloski
-- All rights reserved.
--
-- 08:42      06-Jan-87      vaxa
--
-- File: opticsspec.ada
--
-- This package is a collection of routines that allows one to do ray-based
-- analyses of optical systems.
--
-- An optical system is composed of a collection of surfaces of arbitrary
-- orientation.  The data type "surface" is defined below; the parameters of
-- a surface are as given there.  Some additional explanation:
--
--    ty:  the surface type:  0 = sphere; 1 = y-cylinder; 2 = x-cylinder
--
--    control parameter:  this is an integer that indicates that the correspon-
--                        ding surface parameter is the "ith" independent var-
--                        iable for DIFFERENTIALs.  If the value of the control
--                        parameter is 0, then the surface parameter is a
--                        constant.
--
--
-- Lens data is input using the "input_lens" routine.  It prompts the user for
-- the name of a file which contains the lens deck information to be input.
-- A lens deck consists of a sequence of surface definitions; each surface
-- is defined by at least one (and optionally three) lines of input in the
-- following format:
--
-- (Column #:)        123....
-- (optional          DECE  dx   dy   alpha   beta   gamma
--          lines)    DECC  dxc  dyc  alphac  betac  gammac
-- (required line)    incv   incvc   th   thc   ind   inc
--
-- The optional lines give the orientation of the surface with respect to the
-- previous surface, where the decenters are applied first, then the rotations
-- alpha, beta, and gamma are applied (following the CODE-V convention).  If
-- these two lines (which MUST come together) are not included, those par-
-- ameters of the surface default to 0.0 (or 0).
--
-- All surface parameters are REQUIRED to be FLOATs.  All control parameters
-- are REQUIRED to be INTEGERs.  Given below is an example of a valid deck:
--
--   0.0   0       5.9   4   1.0 0              <-   Surface #0
--   DECE   0.0    0.0   3.5  2.6   99.7
--   DECC   1      2     3    0     0
--   -0.445 0   3.4  0  1.5 0                   <-   Surface #1
--   45.6   200  0.0  0  1.0  0                 <-   Surface #2
--
-- Note that there can be an arbitrary number of spaces between entries (as
-- long as the total length of each line is less than "maxline".  The incv and
-- incvc lens deck entries are then interpreted to generate the ty, cv, and cvc
-- surface parameters via the following kludge:
--
--         incvc            how to generate cv, ty, cvc
--         -----            ---------------------------
```

```
--      a. 0..9        =>  ty=0; cvc=incvc; cv=incv (Sphere)
--      b. 10..19  =>  ty=1; cvc=incvc-10; cv=incv (Y-cylinder)
--      c. 20..29  =>  ty=2; cvc=incvc-20; cv=incv (X-cylinder)
--  Radius of curvature input
--      d. 200..209    =>  ty=0; cvc=incvc-200; cv=1.0/incv (Sphere)
--      e. 210..219    =>  ty=1; cvc=incvc-210; cv=1.0/incv (Y-Cylinder)
--      f. 220..229    =>  ty=2; cvc=incvc-220; cv=1.0/incv (X-Cylinder)
--
--
-- The rest of the routines are standard ray tracing procedures.
-------------------------------------------------------------------------
-------------------------------------------------------------------------

with text_io; use text_io;
with integer_text_io; use integer_text_io;
with float_text_io; use float_text_io;
with DIFFERENTIALS; use DIFFERENTIALS;

package optics is

-- constants

   MAXLINE: constant INTEGER := 50;              -- maximum length of input line
   MAXSURF: constant INTEGER := 30;              -- max number of surfaces
   MAXNAME: constant INTEGER := 10;              -- max length of file name

-- types

   type surface is record
        ty: INTEGER;           -- type
        cv: FLOAT;          -- curvature
        th: FLOAT;          -- thickness
        ind: FLOAT;         -- index of refraction
        dx: FLOAT;          -- x-decenter
        dy: FLOAT;          -- y-decenter
        alpha: FLOAT;           -- alpha rotation (+z -> +y)
        beta: FLOAT;            -- beta rotation (+x -> +z)
        gamma: FLOAT;           -- gamma rotation (+x -> +y)
        cvc: INTEGER;           -- curvature control
        thc: INTEGER;           -- thickness control
        indc: INTEGER;          -- index control
        dxc: INTEGER;           -- dx control
        dyc: INTEGER;           -- dy control
        alphac: INTEGER;        -- alpha control
        betac: INTEGER;             -- beta control
        gammac: INTEGER;        -- gamma control
    end record;
    type RAY is record
                x: DIFFERENTIAL;                    -- current x coordinate
                y: DIFFERENTIAL;                    -- current y coordinate
                z: DIFFERENTIAL;                    -- current z coordinate
                k: DIFFERENTIAL;                    -- n * current x dir. cosine
                l: DIFFERENTIAL;                    -- n * current y dir. cosine
                m: DIFFERENTIAL;                    -- n * current z dir. cosine
      end record;

 -- exceptions

    LENS_DECK_ERRDR: exception;

 -- Global variables

    system: array(0..MAXSURF) of surface;          -- the collection of surfaces
```

```
                                                --   constituting a lens system
   NSURF: integer;                              -- # of surfaces read in from
                                                --   lens deck

-- Data input procedures

   procedure input_lens(surf: out INTEGER);
   procedure set_ray(r: in out RAY;
                     x,dx: in FLOAT; ix: in INTEGER;
                     y,dy: in FLOAT; iy: in INTEGER;
                     z,dz: in FLOAT; iz: in INTEGER;
                     k,dk: in FLOAT; ik: in INTEGER;
                     l,dl: in FLOAT; il: in INTEGER;
                     m,dm: in FLOAT; im: in INTEGER);

-- Data output procedures

   procedure dump_lens(numsurf: in INTEGER);
   procedure print_ray(r1: in RAY);

-- Ray Tracing procedures

   procedure traverse(r1: in out RAY; n1,n2: in INTEGER; p,q: in BOOLEAN);
   procedure decenter(r1: in out RAY; dx,dy; in DIFFERENTIAL);
   procedure rotZY(r1: in out RAY; sa,ca: in DIFFERENTIAL);
   procedure rotXZ(r1: in out RAY; sb,cb: in DIFFERENTIAL);
   procedure rotXY(r1: in out RAY; sg,cg: in DIFFERENTIAL);
   procedure align_axes(r1: in out RAY);

end optics;
```

9.7 OPTICSBODY.ADA

```
------------------------------------------------------------------------------
------------------------------------------------------------------------------
-- This is the body of the package "optics"
-- Copyright (c) 1986,1987 Robert P. Herloski
-- All rights reserved.
--
-- 13:28      06-Jan-87     vaxa
------------------------------------------------------------------------------
------------------------------------------------------------------------------

with text_io; use text_io;
with integer_text_io; use integer_text_io;
with float_text_io; use float_text_io;
with realfunc; use realfunc;
with DIFFERENTIALS; use DIFFERENTIALS;

package body optics is

-- <<<<<<constants>>>>>>

   rad_per_deg: constant FLOAT := 0.017453293;
   zero: constant DIFFERENTIAL := Const_Diff(0.0);

-- <<<<<<Internal procedures>>>>>>

   ---------------------------------------------------------------------
   -- procedure clear_system
   --
   -- This procedure initializes all surface parameters to 0.0 (or 0),
   -- except for the refractive index ind which is set to 1.0
   ---------------------------------------------------------------------

   procedure clear_system is
      i: INTEGER;

      begin
         for i in 0..MAXSURF loop
            system(i).ty := 0;
            system(i).cv := 0.0;
            system(i).th := 0.0;
            system(i).ind := 1.0;
            system(i).dx := 0.0;
            system(i).dy := 0.0;
            system(i).alpha := 0.0;
            system(i).beta := 0.0;
            system(i).gamma := 0.0;
            system(i).cvc := 0;
            system(i).thc := 0;
            system(i).indc := 0;
            system(i).dxc := 0;
            system(i).dyc := 0;
            system(i).alphac := 0;
            system(i).betac := 0;
            system(i).gammac := 0;
         end loop;
   end clear_system;

   ---------------------------------------------------------------------
   -- procedure transform
```

```
--
-- This procedure transforms the coordinates of ray r1 from the current
-- coordinate system to the new coordinate system defined by the tilt
-- and decenter parameters of the ith surface.  The BOOLEAN q is pro-
-- vided to allow the calling procedure to skip the meat of the routine
-- (which takes a long time for DIFFERENTIALS).
-------------------------------------------------------------------------

procedure transform(r1: in out RAY; i: in INTEGER; q: in BOOLEAN) is
   dx,dy,a,b,g: DIFFERENTIAL;

   ---------------------------------------------------------
   -- procedure rotate
   --
   -- This procedure rotates the ray r1 through the angles a, b, and g,
   -- in that order, which are an alpha rotation about the x axis, a beta
   -- rotation about the y axis, and a gamma rotation about the z axis,
   -- respectively.
   ---------------------------------------------------------

   procedure rotate(r1: in out RAY; a,b,g: in DIFFERENTIAL) is
      t: DIFFERENTIAL;
      s,c: DIFFERENTIAL;

      begin
         t := a * rad_per_deg;
         s := sin(t);
         c := cos(t);
         rotZY(r1,s,c);
         t := b * rad_per_deg;
         s := sin(t);
         c := cos(t);
         rotXZ(r1,s,c);
         t := g * rad_per_deg;
         s := sin(t);
         c := cos(t);
         rotXY(r1,s,c);
   end rotate;

   ---------------------------------------------------------
   -- procedure go_back
   --
   -- This procedure resets the ray parameters such that its z-value
   -- with respect to the current coordinate system is zero (i.e., it
   -- "backs-up" along the ray to the xy plane.
   ---------------------------------------------------------

   procedure go_back(r1: in out RAY) is
      don: DIFFERENTIAL;

      begin
         don := r1.z / r1.m;
         don := 0.0 - don;
         r1.x := r1.x + don * r1.k;
         r1.y := r1.y + don * r1.l;
         Set_Diff(r1.z,0.0,0.0,1);
   end go_back;

   begin  ------------------------------BEGIN of TRANSFORM-------------
      if (q) then
         dx := system(i).dx + zero;
         dy := system(i).dy + zero;
         a := system(i).alpha + zero;
```

```
            b := system(i).beta + zero;
            g := system(i).gamma + zero;
            decenter(r1,dx,dy);
            rotate(r1,a,b,g);
            go_back(r1);
         end if;
      end transform;

-- -- <<<<<<Data Input procedures>>>>>>

      ------------------------------------------------------------------------
      -- procedure input_lens
      --
      -- This procedure reads in the lens surface information from a user-
      -- specified file.
      ------------------------------------------------------------------------

      procedure input_lens(surf: out INTEGER) is
         file1: FILE_TYPE;
         file_name: STRING(1..MAXNAME);
         last: NATURAL;
         tsurf: INTEGER;
         input_line: STRING(1..MAXLINE);

         ------------------------------------------------------------
         -- procedure clear
         --
         -- This procedure clears the input line buffer.
         ------------------------------------------------------------

         procedure clear(line: out STRING) is
            i: INTEGER;

            begin
               for i in 1.. MAXLINE loop
                  line(i..i) := " ";
               end loop;
         end clear;

         ------------------------------------------------------------
         -- procedure parse_surf
         --
         -- This procedure parses the input line.
         ------------------------------------------------------------

         procedure parse_surf(line: in STRING; surf: in INTEGER) is
            cv,th,ind: FLOAT;
            cvc,thc,indc: INTEGER;
            last: NATURAL;

            begin
               get(line,cv,last);
               get(line(last+1..MAXLINE),cvc,last);
               get(line(last+1..MAXLINE),th,last);
               get(line(last+1..MAXLINE),thc,last);
               get(line(last+1..MAXLINE),ind,last);
               get(line(last+1..MAXLINE),indc,last);
               if (cvc >= 200) then
                  cv := 1.0 / cv;
                  cvc := cvc   200;
               end if;
               case cvc is
                  when 0..9   =>  system(surf).ty := 0;
```

```
            when 10..19=>  system(surf).ty := 1;
                cvc := cvc - 10;
            when 20..29=>  system(surf).ty := 2;
                cvc := cvc - 20;
            when others=>  raise LENS_DECK_ERROR;
         end case;
         system(surf).cv := cv;
         system(surf).cvc := cvc;
         system(surf).th := th;
         system(surf).thc := thc;
         if (indc /= 0) then
            put_line("IND Diffs not implemented yet");
            raise LENS_DECK_ERROR;
         else
            system(surf).ind := ind;
            system(surf).indc := indc;
         end if;
   end parse_surf;

   ------------------------------------------------------------
   -- procedure parse_dec
   --
   -- This procedure parses the DECE and DECC lines.
   ------------------------------------------------------------

   procedure parse_dec(line: in STRING; surf: in INTEGER) is
      dx,dy,alpha,beta,gamma: FLOAT;
      dxc,dyc,alphac,betac,gammac: INTEGER;
      last: NATURAL;
      nline: STRING(1..MAXLINE);

      begin
         get(line(5..MAXLINE),dx,last);
         get(line(last+1..MAXLINE),dy,last);
         get(line(last+1..MAXLINE),alpha,last);
         get(line(last+1..MAXLINE),beta,last);
         get(line(last+1..MAXLINE),gamma,last);
         clear(nline);
         get_line(file1,nline,last);
         if (nline(1..4) /= "DECC") then
            put_line("Error!  No decenter control card!");
            raise LENS_DECK_ERROR;
         else
            get(nline(5..MAXLINE),dxc,last);
            get(nline(last+1..MAXLINE),dyc,last);
            get(nline(last+1..MAXLINE),alphac,last);
            get(nline(last+1..MAXLINE),betac,last);
            get(nline(last+1..MAXLINE),gammac,last);
            if (dxc /= 0) then
               put_line("DX Diffs not yet implemented!");
               raise LENS_DECK_ERROR;
            else
               system(surf).dx := dx;
               system(surf).dxc := dxc;
            end if;
            if (dyc /= 0) then
               put_line("DY Diffs not yet implemented!");
               raise LENS_DECK_ERROR;
            else
               system(surf).dy := dy;
               system(surf).dyc := dyc;
            end if;
            if (alphac /= 0) then
```

```
            put_line("ALPHA Diffs not yet implemented!");
            raise LENS_DECK_ERROR;
         else
            system(surf).alpha := alpha;
            system(surf).alphac := alphac;
         end if;
         if (betac /= 0) then
            put_line("BETA Diffs not yet implemented!");
            raise LENS_DECK_ERROR;
         else
            system(surf).beta := beta;
            system(surf).betac := betac;
         end if;
         if (gammac /= 0) then
            put_line("GAMMA Diffs not yet implemented!");
            raise LENS_DECK_ERROR;
         else
            system(surf).gamma := gamma;
            system(surf).gammac := gammac;
         end if;
      end if;
   end parse_dec;

   ------------------------------START OF INPUT_LENS--------------------

   begin
      file_name(1..MAXNAME) := (1..MAXNAME => ' ');
      put("Input lens file name:  ");
      get_line(file_name,last);
      if (last < file_name'LAST) then
        file_name(last+1..file_name'LAST) := (last+1..file_name'LAST =>' ');
      end if;
      open(file1,IN_FILE,file_name);
      tsurf := 0;
      while not END_OF_FILE(file1) loop
         clear(input_line);
         get_line(file1,input_line,last);
         if (input_line(1..1) /= ":") then
            if (input_line(1..4)   "DECE") then
               parse_dec(input_line,tsurf);
            else
               parse_surf(input_line,tsurf);
               tsurf := tsurf + 1;
            end if;
         end if;
      end loop;
      surf := tsurf   1;
end input_lens;

-------------------------------------------------------------------------
-- procedure set_ray
--
-- This procedure sets the components of the ray, which are DIFFERENTIALS,
-- to the appropriate values.
-------------------------------------------------------------------------

procedure set_ray(r: in out RAY;
          x,dx: in FLOAT; ix: in INTEGER;
          y,dy: in FLOAT; iy: in INTEGER;
          z,dz: in FLOAT; iz: in INTEGER;
          k,dk: in FLOAT; ik: in INTEGER;
          l,dl: in FLOAT; il: in INTEGER;
          m,dm: in FLOAT; im: in INTEGER) is
```

```
      begin
         Set_Diff(r.x,x,dx,ix);
         Set_Diff(r.y,y,dy,iy);
         Set_Diff(r.z,z,dz,iz);
         Set_Diff(r.k,k,dk,ik);
         Set_Diff(r.l,l,dl,il);
         Set_Diff(r.m,m,dm,im);
   end set_ray;

-- <<<<<<Data output procedures>>>>>>

   ------------------------------------------------------------------------
   -- procedure dump_lens
   --
   -- This procedure dumps the lens surface data to STDOUT
   ------------------------------------------------------------------------

   procedure dump_lens(numsurf: in INTEGER) is
      i: INTEGER;

      begin
         put_line("");
         put_line("");
   put_line("Surf TY    Curvature  CVC       Thickness  THC      Index    INC");
   put_line("---- --   ----------  ---      ---------  ---    --------  ---");
         put_line("");
         for i in 0..numsurf loop
            put(i,3);
            if ((system(i).dx /= 0.0) or
                (system(i).dy /= 0.0) or
                (system(i).alpha /= 0.0) or
                (system(i).beta /= 0.0) or
                (system(i).gamma /= 0.0)) then
               put("    DECE");
               put(system(i).dx,6,6,0);
               put(system(i).dy,6,6,0);
               put(system(i).alpha,6,6,0);
               put(system(i).beta,6,6,0);
               put(system(i).gamma,6,6,0);
               put_line("");
               put("       DECC");
               put(system(i).dxc,8);
               put(system(i).dyc,13);
               put(system(i).alphac,13);
               put(system(i).betac,13);
               put(system(i).gammac,13);
               put_line("");
               put("   ");
            end if;
            put(system(i).ty,4);
            put(system(i).cv,4,8,0);
            put(system(i).cvc,4);
            put(system(i).th,9,6,0);
            put(system(i).thc,4);
            put(system(i).ind,6,6,0);
            put(system(i).indc,4);
            put_line("");
         end loop;
   end dump_lens;

   ------------------------------------------------------------------------
   -- procedure print_ray
```

```
   --
   -- This procedure dumps the values (NOT the derivatives) of components
   -- of the ray r1.
   ------------------------------------------------------------------------

   procedure print_ray(r1: in RAY) is

      begin
         put(val0(r1.x),3,6,0);
         put("  ");
         put(val0(r1.y),3,6,0);
         put("  ");
         put(val0(r1.z),3,6,0);
         put("  ");
         put(val0(r1.k/r1.m),3,6,0);
         put("  ");
         put(val0(r1.l/r1.m),3,6,0);
         put_line("");
   end print_ray;

-- <<<<<<Ray Tracing Procedures>>>>>

   ------------------------------------------------------------------------
   -- procedure decenter
   --
   -- This procedure decenters a ray by the x and y values dx and dy.
   ------------------------------------------------------------------------

   procedure decenter(r1: in out RAY; dx,dy: in DIFFERENTIAL) is

      begin
         r1.x := r1.x   dx;
         r1.y := r1.y - dy;
   end decenter;

   ------------------------------------------------------------------------
   -- procedure rotZY
   --
   -- This procedure performs a rotation of alpha degrees about the x axis.
   -- Sa and ca are the sine and cosine of alpha, respectively.  A positive
   -- rotation rotates the +z axis towards the +y axis.
   ------------------------------------------------------------------------

   procedure rotZY(r1: in out RAY; sa,ca: in DIFFERENTIAL) is
      t: DIFFERENTIAL;

      begin
         t := r1.y * ca   r1.z * sa;
         r1.z := r1.y * sa + r1.z * ca;
         r1.y := t;
         t := r1.l * ca   r1.m * sa;
         r1.m := r1.l * sa + r1.m * ca;
         r1.l := t;
   end rotZY;

   ------------------------------------------------------------------------
   -- procedure rotXZ
   --
   -- This procedure performs a rotation of beta degrees about the y axis.
   -- Sb and cb are the sine and cosine of beta, respectively.  A positive
   -- rotation rotates the +x axis towards the +z axis.
   ------------------------------------------------------------------------
```

```
procedure rotXZ(r1: in out RAY; sb,cb: in DIFFERENTIAL) is
   t: DIFFERENTIAL;

   begin
      t := r1.z * cb - r1.x * sb;
      r1.x := r1.z * sb + r1.x * cb;
      r1.z := t;
      t := r1.m * cb - r1.k * sb;
      r1.k := r1.m * sb + r1.k * cb;
      r1.m := t;
end rotXZ;

-------------------------------------------------------------------------
-- procedure rotXY
--
-- This procedure performs a rotation of gamma degrees about the z axis.
-- Sg and cg are the sine and cosine of gamma, respectively.  A positive
-- rotation rotates the +x axis towards the +y axis.
-------------------------------------------------------------------------

procedure rotXY(r1: in out RAY; sg,cg: in DIFFERENTIAL) is
   t: DIFFERENTIAL;

   begin
      t := r1.y * cg - r1.x * sg;
      r1.x := r1.y * sg + r1.x * cg;
      r1.y := t;
      t := r1.l * cg - r1.k * sg;
      r1.k := r1.l * sg + r1.k * cg;
      r1.l := t;
end rotXY;

-------------------------------------------------------------------------
-- procedure align_axes
--
-- This procedure allows the user to rotate the ray into a coordinate
-- system such that it points along the z axis.
-------------------------------------------------------------------------

procedure align_axes(r1: in out RAY) is
   st,ct: DIFFERENTIAL;
   t: FLOAT;
   dx,dy: DIFFERENTIAL;

   begin
      dx := val0(r1.x) + zero;
      dy := val0(r1.y) + zero;
      decenter(r1,dx,dy);
      t := sqrt(val0(r1.k) * val0(r1.k) + val0(r1.l) * val0(r1.l));
      if (t > 1.0E-6) then
         st := zero - val0(r1.k) / t;
         ct := zero + val0(r1.l) / t;
         rotXY(r1,st,ct);
         rotZY(r1,t + zero,val0(r1.m) + zero);
      end if;
end align_axes;

-------------------------------------------------------------------------
-- procedure traverse
--
-- This procedure traces a ray r1 through the optical system from surface
-- number n1 to surface number n2. p is a BOOLEAN to indicate whether or
-- not to print out ray surface information, and q is a BOOLEAN that in-
```

```
-- dicates whether or not to execute the transform function (lots of time
-- is saved if one traces a ray through a centered optical system).
-------------------------------------------------------------------------

procedure traverse(r1: in out RAY; n1,n2: in INTEGER; p,q: in BOOLEAN) is
   don: DIFFERENTIAL;
   i: INTEGER;
   dth: DIFFERENTIAL;

   ---------------------------------------------------------
   -- procedure sphere
   --
   -- This procedure refracts the ray r1 through the sphere at surface
   -- number (i+1).
   ---------------------------------------------------------

   procedure sphere(r1: in out RAY; i: in INTEGER) is
      nm1,n: FLOAT;
      h,b,nci,aon,ncip,g: DIFFERENTIAL;
      c: DIFFERENTIAL;

      begin
         nm1 := system(i).ind;
         if (system(i+1).cvc /= 0) then
            Set_Diff(c,system(i+1).cv,1.0,system(i+1).cvc);
         else
            Set_Diff(c,system(i+1).cv,0.0,1);
         end if;
           n := system(i+1).ind;
           h := c * ((r1.x * r1.x) + (r1.y * r1.y));
           b := r1.m   (c * ((r1.y * r1.l) + (r1.x * r1.k)));
         nci := sqrt(b * b   c * nm1 * nm1 * h);
         aon := h / (b + nci);
        r1.x := r1.x + (aon * r1.k);
        r1.y := r1.y + (aon * r1.l);
        r1.z := aon * r1.m;
        ncip := 1.0 + ((nci * nci) / (n * n))   ((nm1 / n) * (nm1 / n));
        ncip := n * sqrt(ncip);
           g := ncip   nci;
        r1.k := r1.k   (r1.x * c * g);
        r1.l := r1.l   (r1.y * c * g);
        r1.m := r1.m   (((r1.z * c)   1.0) * g);
    end sphere;

    ---------------------------------------------------------
    -- procedure cylX
    --
    -- This procedure refracts the ray r1 through the x-cylinder at sur-
    -- face number (i+1).
    ---------------------------------------------------------

    procedure cylX(r1: in out RAY; i: in INTEGER) is
       nm1,n: FLOAT;
       h,b,nci,aon,ncip,g: DIFFERENTIAL;
       c: DIFFERENTIAL;

       begin
          nm1 := system(i).ind;
          if (system(i+1).cvc /= 0) then
             Set_Diff(c,system(i+1).cv,1.0,system(i+1).cvc);
          else
             Set_Diff(c,system(i+1).cv,0.0,1);
          end if;
```

```
            n := system(i+1).ind;
            h := c * r1.x * r1.x;
            b := r1.m - (c * r1.x * r1.k);
          nci := sqrt(b * b - (c * (r1.k * r1.k + r1.m * r1.m) * h));
          aon := h / (b + nci);
         r1.x := r1.x + (aon * r1.k);
         r1.y := r1.y + (aon * r1.l);
         r1.z := aon * r1.m;
         ncip := 1.0 + ((nci * nci) / (n * n)) - ((nm1 / n) * (nm1 / n));
         ncip := n * sqrt(ncip);
            g := ncip - nci;
         r1.k := r1.k - (r1.x * c * g);
         r1.m := r1.m - (((r1.z * c) - 1.0) * g);
   end cylX;

   ------------------------------------------------------------
   -- procedure cylY
   --
   -- This procedure refracts the ray r1 through the y-cylinder at sur-
   -- face number (i+1).
   ------------------------------------------------------------

   procedure cylY(r1: in out RAY; i: in INTEGER) is
      nm1,n: FLOAT;
      h,b,nci,aon,ncip,g: DIFFERENTIAL;
      c: DIFFERENTIAL;

      begin
         nm1 := system(i).ind;
         if (system(i+1).cvc /= 0) then
            Set_Diff(c,system(i+1).cv,1.0,system(i+1).cvc);
         else
            Set_Diff(c,system(i+1).cv,0.0,1);
         end if;
           n := system(i+1).ind;
           h := c * r1.y * r1.y;
           b := r1.m - (c * r1.y * r1.l);
         nci := sqrt(b * b - (c * (r1.l * r1.l + r1.m * r1.m) * h));
         aon := h / (b + nci);
        r1.x := r1.x + (aon * r1.k);
        r1.y := r1.y + (aon * r1.l);
        r1.z := aon * r1.m;
        ncip := 1.0 + ((nci * nci) / (n * n)) - ((nm1 / n) * (nm1 / n));
        ncip := n * sqrt(ncip);
           g := ncip - nci;
        r1.l := r1.l - (r1.y * c * g);
        r1.m := r1.m - (((r1.z * c) - 1.0) * g);
   end cylY;

   begin    -----------------------BEGIN of TRAVERSE-------------------
      if (p) then
put_line("");
put_line("Surf        X           Y           Z          TAN X       TAN Y");
put_line("----    --------    --------    --------    --------    --------");
put_line("");
         put(n1,3);
         put(":  ");
         print_ray(r1);
      end if;
      for i in n1..(n2-1) loop
         if (system(i).thc /= 0) then
            Set_Diff(dth,system(i).th,1.0,system(i).thc);
         else
```

```
            Set_Diff(dth,system(i).th,0.0,1);
          end if;
          don := dth   r1.z;
          don := don / r1.m;
         r1.y := r1.y + (don * r1.l);            -- transfer to tangent plane
         r1.x := r1.x + (don * r1.k);
          Set_Diff(r1.z,0.0,0.0,1);
          transform(r1,i+1,q);                   -- rotate into local coord.
                                                 --   system.
          case system(i+1).ty is                 -- refract appropriately
             when 0     =>  sphere(r1,i);
             when 1     =>  cylY(r1,i);
             when 2     =>  cylX(r1,i);
             when others=>  raise LENS_DECK_ERROR;
          end case;
          if (p) then
             put(i+1,3);
             put(":  ");
             print_ray(r1);
          end if;
       end loop;
   end traverse;

end optics;
```