ALGORITHMS FOR HIGHER ORDER AUTOMATIC DIFFERENTIATION IN MANY VARIABLES WITH APPLICATIONS TO BEAM PHYSICS

MARTIN BERZ

PROCEEDINGS WORKSHOP ON AUTOMATIC DIFFERENTIATION
BRECKENRIDGE, CO, JANUARY 1991
PUBLISHED BY SIAM 1991

Abstract. Efficient algorithms for automatic differentiation with several variables and high orders are presented. The algorithms are geared towards sparse vectors, which is particularly important in this case and allows significant savings in computer time. Besides the mere computation of derivatives, algorithms for the efficient composition and inversion of functions with sparse derivatives are discussed.

The algorithms are implemented in a FORTRAN library. The library can be utilized by a precompiler that transforms FORTRAN into code to perform the desired automatic differentiation task. The precompiler allows passing of variables into subroutines and functions and allows the user to provide functions. Besides the use with the precompiler, the routines can be accessed from a dedicated language environment. The language has the flavor of PASCAL, but provides object oriented features and nonlinear optimization at the language level.

The tools have been used in numerous cases for the computation and correction of aberrations of beam physics systems and the simulation and analysis of nonlinear dynamics problems, including the simulation of large particle accelerators.

1. Introduction. In recent years it has been shown that automatic differentiation methods, combined with methods of theoretical physics, can be used very fruitfully for the computation and analysis of beam physics systems [Ber88b, Ber89b, Ber90a, Ber87, Ber88a, Ber89a, FBI89, FB89]. These systems include particle accelerators, particle optical systems, and electron microscopy.

The high orders often needed for the computation of such maps and the complexity of the underlying systems require rather sophisticated computational algorithms. In this paper we will discuss in detail the core algorithms used in the implementation of the high order automatic differentiation methods.

Beam physics systems can be represented by a map relating final phase space coordinates $\vec{z_f}$ to initial coordinates $\vec{z_i}$ and system parameters $\vec{\delta}$ in the following way:

$$\vec{z}_f = \mathcal{M}(\vec{z}_i, \vec{\delta}) \tag{1}$$

Depending on the problem, the phase space variables can be sets of two or three positions and momenta, and can contain other quantities like the spin. The system parameters can include certain quantities that describe the system and the dependence on which is of interest. Note that the distinction between variables and parameters is somewhat arbitrary; we consider any quantity of interest a parameter if it stays constant throughout the system.

The transfer map is the (unique) flow of certain differential equations [Ber90b] describing the evolution of the variables:

$$\frac{d}{dt}\vec{z} = \vec{f}(\vec{z}, \vec{\delta}, t). \tag{2}$$

The partial derivatives of the transfer map (1) with respect to the phase space variables are called aberrations, and the ones involving system parameters are called sensitivities. In a very general sense, the task of beam physics is to find the aberration coefficients and sensitivities to a certain order, and to try to modify them in such a way that the map has certain desirable properties.

Beam Physics systems are characterized by the fact that the linear part of the flow is dominant, and that the Taylor series of the flow converges to the map rapidly. Typically, orders anywhere from two to ten are required to describe the system with the required accuracy.

In the past, it has been very difficult to obtain aberrations, and usually the orders were limited to at most three. It required considerable efforts to derive analytical formulas [Bro79, BW87] for the aberrations of all the important elements of which particle optical systems are made of [Wol87, Car87, WE65, BBB64, Wol65, Wol67, Wol68, MW70a, MW70b, MMW72]

In 1986, we outlined another way [Ber87] that allows a much more straightforward computation of such image aberrations. It is based on the replacement of all arithmetical steps in the numerical algorithm for the solution of the equations of motion by truncated power series. Being essentially an automatic differentiation method, it automatically allows the computation of arbitrary order aberrations of arbitrary systems in a very elegant way.

In order to compute the derivatives of $\vec{f}$, several evaluations of $\vec{f}$ at different positions are required; for example, the eighth order Runge Kutta algorithm used in COSY INFINITY [Ber91] requires thirteen evaluations of the function per time step. These evaluations of the right hand side of the differential equation are very costly, and they are indeed the limiting factor for the speed.

It turns out that in the important case in which $\vec{f}$ is time independent and origin preserving, the use of a numerical integrator can be avoided, and we can readily obtain all the required higher order behaviour of $\vec{f}$ with only one evaluation of $\vec{f}$.

To illustrate this technique, suppose we are interested in the behaviour of a function g of phase space, i.e., we want to know $g(\vec{x}(t))$, where $\vec{x}(t)$ is a solution of the equations of motion. Then we can infer

$$\begin{aligned}\frac{d}{dt}g &= \vec{\nabla}\, g \cdot \frac{d}{dt}\vec{x} + \frac{\partial g}{\partial t} \\ &= \vec{\nabla}\, g \cdot \vec{f} + \frac{\partial}{\partial t}g \\ &= L_f\, g.\end{aligned} \tag{3}$$

The operator L_f is usually called the Lie derivative of g. Using the operator L_f, also higher derivatives of g can be computed:

$$\begin{aligned}\frac{d^2}{dt^2}g &= L_f^2\, g, \\ \frac{d^3}{dt^3}g &= L_f^3\, g \quad \text{etc.}\end{aligned} \tag{4}$$

This approach is well known [CdB80] and in fact is even sometimes used in practice to derive analytical low order integration formulas for certain functions $\vec{f}$. The limitation is that unless $\vec{f}$ is very simple, it is usually impossible to compute the repeated action of L_f analytically, and this is why this approach has not been very useful in practice. However, once all the derivatives of $\vec{f}$ are known, the repetitive actions of the Lie derivative can readily be computed. To this end, it is necessary to compute a vector of derivatives for the partial derivative of a function from the vector of derivatives of the underlying function. This is merely a bookkeeping operation and is formally accomplished by the operator ∂_μ. So altogether, one just evaluates the derivatives of $\vec{f}$ and repeatedly uses the ∂_μ to compute the Lie derivatives

In this algorithm, the use of the derivations ∂_μ is apparently of prime importance. Since an algebra with a derivation is called a differential algebra [Rit50], the method is usually referred to as the differential algebraic (DA) approach.

2. Storage and Addition. Let ${}_nD_v$ the algebra of vectors of partial derivatives of order n in v variables used in automatic differentiation. According to [Ber92], any element of the structure ${}_nD_v$ can be written as a polynomial in the v nilpotent base elements d_1, ..., d_n, i.e., they form generators of the algebra. In the arrangement introduced in [Ber92], the element d_k, $k = 1, n$, has a one in slot $k+1$ and zeros everywhere else. The $(n+v)!/n!v!$ monomials generated by combinations of at most n of these generators form a basis of the real algebra ${}_nD_v$. Any product of more than n of these generators d_k vanishes.

In many high order calculations, and also in the computations involving beam physics systems, the derivative vectors are often very sparse. One reason for this effect is that a given intermediate variable does not depend at all on a certain initial variable. While in the case of ${}_1D_v$, this entails that one of the $v+1$ component vanishes, in higher orders the effects are much more dramatic. Of the $(n+v)!/n!/v!$ components, only $(n+v-1)!/n!/(v-1)!$ are nonzero, which is a fraction of $v/(n+v)$. So for problems around $n = 10$ and $v = 6$, which is somewhat typical for beam physics cases, less than half of the terms prevail. Of course, if the intermediate variable is independent of even more than one original variable, the results are even more noticeable.

The particular mathematics of beam physics entails that many quantities depend on certain initial conditions in a purely odd or even way, which further dramatically reduces the nonzero terms. Altogether, in a typical beam physics calculation, it is not unusual to have only about 10 % nonzero components on average.

So it is computationally advantageous to store and operate on the nonzero terms only. This requires some bookkeeping overhead; however, compared to the sophistication needed for efficient multiplication algorithms discussed in the next section, this effort is minor.

In order to facilitate addition and subtraction, it is essential that all nonzero entries are stored in an ordered way, for example following the natural lexicographical ordering [Ber92]. In case such an ordering is followed, addition and subtraction become simple merging operations.

3. The Multiplication Algorithm. As outlined in the previous section, vectors in the algebra ${}_nD_v$ used for automatic differentiation can be written and manipulated as polynomials such that products of terms whose order exceeds n always vanish. Suppose the $N = (n+v)!/n!/v!$ monomials are arranged in a certain order. Let M_i denote the monomial identified with the ith component, and let I_M denote the position of the monomial M.

In order to multiply two vectors and find the contribution to the ith component, it is necessary to find all factorizations of the monomial M_i:

$$c_i = \sum_{\substack{0 \leq \nu,\mu \leq N \\ M_\nu \cdot M_\mu = M_i}} a_\nu \cdot b_\mu \tag{5}$$

The computation of all these factorizations presents a difficult algorithm and could be quite time consuming. Thus in practice it is advantageous to rephrase the problem such that no factorizations in submonomials are searched, but rather each component of the first vector is multiplied by each component of the second vector and the product is stored at the place where the product monomial belongs.

For this process, the main algorithmic problem of an efficient multiplication algorithm becomes apparent: it is necessary to determine the address of the product monomial in a fast way.

Here we present a solution to this problem which produces minimal computational overhead and works for arbitrarily many variables and arbitrary order.

First, all $(n+v)!/n!/v!$ monomials M are coded with an integer $C(M)$ in the following way: Let $M = x_1^{i_1} \cdot ... \cdot x_v^{i_v}$, then $C(M)$ is defined as follows:

$$C(M) = C(x_1^{i_1} \cdot ... \cdot x_v^{i_v}) = i_1 \cdot (n+1)^0 + i_2 \cdot (n+1)^1 + ... + i_v \cdot (n+1)^{(v-1)} \tag{6}$$

So the exponents are just "decimals" in base $(n+1)$. Note that since $i_\nu \leq n$, the function $M \to C(M)$ is injective and hence the coding unique. Note also that no coding exceeds $(n+1)^v$, but not all such codings occur.

Now suppose two monomials M and N have to be multiplied and suppose their product has an order less than or equal to n. Since the multiplication corresponds to an addition of the exponents, it follows that

$$C(M \cdot N) = C(M) + C(N) \tag{7}$$

To exploit this for the finding of the desired coordinate position I_M of the product of two monomials, an array D is required that has the property

$$I_M = D(C(M)) \tag{8}$$

This array can be generated easily once the order n and number of variables v are fixed, and has to be computed only once. Since the codings are bounded by $(n+1)^v$, the array has to have at least this length. With 6 variables, this allows orders of 8 or 9 if one wants to stay inside the boundaries of computer storage; with 8 variables the order would decrease to about 4, and this is too strict a limitation. To circumvent this, a slight modification of the above coding and decoding will be presented.

Without loss of generality, we assume the number of variables v to be even; if it is not even, increase it by one and ignore the additional variable. We define two coding numbers C_1 and C_2 for any monomial in the following way:

$$\begin{aligned} C_1(x_1^{i_1} \cdot ... \cdot x_v^{i_v}) &= i_1 \cdot (n+1)^0 + i_2 \cdot (n+1)^1 + ... + i_{\frac{v}{2}} \cdot (n+1)^{(\frac{v}{2}-1)} \\ C_2(x_1^{i_1} \cdot ... \cdot x_v^{i_v}) &= i_{\frac{v}{2}+1} \cdot (n+1)^0 + i_{\frac{v}{2}+2} \cdot (n+1)^1 + ... + i_v \cdot (n+1)^{(\frac{v}{2}-1)} \end{aligned} \tag{9}$$

Next we store the $N(n,v)$ monomials in a special way. We note that this storage differs from the one used for the introduction of the lexicographical ordering described in [Ber92]; interestingly enough, however, the arrangement presented here defines another lexicographical total ordering.

We start with all monomials that have $C_2(M) = 0$ and group them by order; within one order, the monomials are stored according to ascending values of $C_1(M)$. Then we store all those with $C_2(M) = 1$, again by order, and so forth, going through all possible values of C_2. Because of the order-by-order arrangement within the monomials belonging to the same $C_1(M)$, it follows that again

$$\begin{aligned} C_1(M \cdot N) &= C_1(M) + C_1(N) \\ C_2(M \cdot N) &= C_2(M) + C_2(N) \end{aligned} \tag{10}$$

Finally we introduce some "inverse" arrays D_1 and D_2 in the following way:

$$
\begin{aligned}
&D_1(c_1) = (I_M \text{ of first monomial } M \text{ with } C_1(M) = c_1) \\
&D_2(c_2) = (I_M \text{ of first monomial } M \text{ with } C_2(M) = c_2) - 1
\end{aligned} \tag{11}
$$

Again the arrays D_1 and D_2 can be generated once during the setup process. Using the definitions of C_1, C_2, D_1 and D_2 and the storage scheme outlined above, it now follows that the address of the product of the monomials M and N can be found directly as

$$
I_{M \cdot N} = D_1[C_1(I_M) + C_1(I_N)] + D_2[C_2(I_M) + C_2(I_N)] \tag{12}
$$

For the sake of clarity, table 1 shows an example for the arrays C_1, C_2, D_1 and D_2 for $n = 3$ and $v = 4$. This example also illustrates equations (9) through (12).

(13)

I_M	i_1	i_2	i_3	i_4	C_1	C_2
1	0	0	0	0	0	0
2	1	0	0	0	1	0
3	0	1	0	0	4	0
4	2	0	0	0	2	0
5	1	1	0	0	5	0
6	0	2	0	0	8	0
7	3	0	0	0	3	0
8	2	1	0	0	6	0
9	1	2	0	0	9	0
10	0	3	0	0	12	0
11	0	0	1	0	0	1
12	1	0	1	0	1	1
13	0	1	1	0	4	1
14	2	0	1	0	2	1
15	1	1	1	0	5	1
16	0	2	1	0	8	1
17	0	0	0	1	0	4
18	1	0	0	1	1	4
19	0	1	0	1	4	4
20	2	0	0	1	2	4
21	1	1	0	1	5	4
22	0	2	0	1	8	4
23	0	0	2	0	0	2
24	1	0	2	0	1	2
25	0	1	2	0	4	2
26	0	0	1	1	0	5
27	1	0	1	1	1	5
28	0	1	1	1	4	5
29	0	0	0	2	0	8
30	1	0	0	2	1	8
31	0	1	0	2	4	8
32	0	0	3	0	0	3
33	0	0	2	1	0	6
34	0	0	1	2	0	9
35	0	0	0	3	0	12

j	$D_1(j)$	$D_2(j)$
0	1	0
1	2	10
2	4	22
3	7	31
4	3	16
5	5	25
6	8	32
7	0	0
8	6	28
9	9	33
10	0	0
11	0	0
12	10	34

Table 1: List of the ordering of the all monomials $M = x_1^{i_1} \cdot ... \cdot x_v^{i_v}$ for order $n = 3$ and number of variables $v = 4$. Also shown are the coding integers C_1 and C_2 and the arrays D_1 and D_2. For all M, one verifies $I_M = D_1(C_1(M)) + D_2(C_2(M))$

The coding defined in (9) entails that the required length of the arrays D_1 and D_2 is much smaller, namely only $(n+1)^{\frac{v}{2}}$. For a maximum length of 10^6, this limits the maximum order for a given number of variables to the values given in table 2.

number of variables	6	8	10	12
maximum order	99	30	14	10

Table 2: The maximum order for different numbers of variables due to the limitation of the length of the reverse addressing arrays D_1, D_2

Each multiplication of two monomials now requires three integer additions and six integer array look-ups besides the double precision multiplication of the coefficients. Since integer additions are usually executed much faster than double precision multiplications and array look-ups are faster yet, the extra amount of time for the bookkeeping is quite limited. To be specific, on a typical VAX computer, all the bookkeeping integer operations together take only about one third of the time required for the one double precision multiplication. Since the latter can of course never be avoided, the algorithm here is very nearly optimal and it should be very hard to improve significantly.

4. Additional Operations. Besides the operations outlined in the previous section, there are several more that are important for the practical use. First and foremost, this holds for intrinsic functions.

In [Ber92] we showed that indeed all real power series can be extended to the structure $\mathcal{L}$ and with it also to ${}_nD_v$ within their radius of convergence. In practice, it turns out that we often can simplify the computation considerably by exploiting certain addition theorems of the function of interest. In this case, it suffices to evaluate the series at nilpotent infinitesimals, where they converge in finitely many steps.

We illustrate this with the sine function. Suppose we are given a DA number which we write as $X+r$, X being its real part and r being the infinitely small rest. Then we obtain

$$\begin{aligned}\sin(X+r) &= \sin(X)\cdot\cos(r)+\cos(X)\cdot\sin(r) \Rightarrow\\ &= \sin(X)\cdot\sum_{i=0}^{\infty}(-1)^i\frac{r^{2i}}{(2i)!}+\cos(X)\cdot\sum_{i=0}^{\infty}(-1)^{i+1}\frac{r^{2i+1}}{(2i+1)!} \Rightarrow\\ &= \sin(X)\cdot\sum_{i=0}^{n}(-1)^i\frac{r^{2i}}{(2i)!}+\cos(X)\cdot\sum_{i=0}^{n}(-1)^{i+1}\frac{r^{2i+1}}{(2i+1)!}.\end{aligned} \tag{14}$$

So the addition theorem allows us to compute the sine of an element of the differential algebra in only finitely many steps. A very similar argument can be developed for the cosine, the exponential and the logarithm, as well as for inverses and roots. For inverse trigonometric functions, the situation becomes much more difficult, and it often requires a rather involved battery of equations.

The FORTRAN library developed using the algorithms described here contains routines for the computation of most intrinsic functions covered by the FORTRAN ANSI standard.

Besides the computation of intrinsic functions, the second most important operation is the derivation, which turns the algebras ${}_nD_v$ into differential algebras [Rit50]. Because of the significance of this operation for the computation of aberrations of beam physics systems (see section 1), the techniques are usually referred to as the differential algebraic methods [Ber90a, Ber89b, Ber88a].

The derivation operation computes the value and derivatives of the partial derivative of a function from the corresponding values for the function. This operation is readily achieved by subtracting the coding integers for the variable with respect to which to differentiate from the coding integers of the component under consideration. Note that the derivation operations always entail a loss of order by one, and algorithms have to be designed such that this does not matter. In a very similar way to the derivations, it is possible to perform integration.

Another important algorithm allows the composition of functions of which partial derivatives are known. So given $\vec{f_1}$ and $\vec{f_2}$ and their derivatives, we would like to compute the derivatives of $\vec{f_1} \cdot \vec{f_2}$. Of particular importance is the case where $\vec{f_2}$ vanishes, but its derivatives are nonzero.

As the need to compose functions occurs very frequently during beam physics computations, an efficient implementation of this method is crucial. Again, great care has to be exercised if the maps are sparse. In this case, the first step in composition algorithms is to find an optimally short tree that reaches all the nonzero monomials in $\vec{f_1}$, but requires the least number of monomials not occurring in $\vec{f_1}$. The next step is to traverse the tree in such a way that each new node requires only one polynomial multiplication.

We note that using iterative compositions and the inversion of a linear matrix, it is possible to compute the derivative of the inverse of a given map $\vec{f}$. For details, we refer to [Ber90a].

For practical use it is important to utilize the FORTRAN procedures described here in a convenient manner. At the present time, there are two different ways to use the differential algebra package. The first way is based on a precompiler which allows the transformation of FORTRAN code for the automatic computation of derivatives. This method is described in the next section.

The second approach is based on a full language system which provides a particularly elegant and powerful environment for the development of new code. This is discussed below.

5. The Precompiler. The precompiler DAFOR [Ber90f, Ber90e] represents an extension to standard FORTRAN 77 which allows the use of a differential algebraic data type. DAFOR is designed to allow rapid conversion of existing programs. It converts the extended FORTRAN code to regular FORTRAN code by expressing all DA operation by calls to subroutines from the DA library DAPRE. For a more detailed description of the precompiler, the reader is referred to the manual.

The first step in the conversion of a FORTRAN code to Differential Algebraic computation of derivatives is to identify the independent variables with respect to which to differentiate as well as the number of these variables and the maximum order to which derivatives are to be computed. These two numbers are then passed to an initial setup routine.

In the next step, in each program segment all variables that become DA have to be declared to the precompiler and their original FORTRAN declarations have to be removed. All declarations for the precompiler have to be located between the declaration and execution section of the FORTRAN program. Besides the DA variables, all REAL, INTEGER, or DOUBLE PRECISION variables occurring in arithmetic expressions in which DA variables occur have to be declared to the precompiler.

Besides the variables occurring in the program section, it is necessary to declare all external functions that occur in an expression that contains DA operations. Note that the precompiler supports all intrinsic functions required in the FORTRAN ANSI standard.

The main feature of the precompiler DAFOR is that it automatically converts assignment statements in which DA variables are computed in terms of others. Any assignment statement containing a DA variable has to be identified for precompilation with a *DA in columns 1 through 3. The statement can extend over several lines, which then also have to be marked with *DA in the first columns. The end of the command has to be denoted with a semicolon.

Besides assignment, DA variables can occur in calls to subroutines. No changes are necessary as long as the subroutine is converted accordingly and the DA variables are properly declared.

FORTRAN commands other than assignments and subroutine calls are not processed by DAFOR. In the case of outputting statements like WRITE or PRINT, it is necessary to rewrite the commands. There are routines to output the value of the quantity along with its derivatives with respect to the independent variables, i.e. the full DA variable.

6. The FOXY language. While the precompiler discussed in the previous section is particularly helpful for the conversion of existing FORTRAN code, we believe that the environment discussed in this section is the matter of choice for the development of new code. This language environment was used to write the new beam physics design and simulation code COSY INFINITY [Ber90d, Ber90b, Ber90c]. While being much more powerful than existing codes, its source is eminently readable and about a factor of 10 more compact than other codes. Besides the significant simplifications in the physics parts that are the consequence of the use of differential algebraic methods, this is largely due to the very powerful language environment.

The COSY language is similar to PASCAL, which provides power in a compact syntax that is easy to analyze. The language of COSY differs from PASCAL in its object oriented features. New data types and operations on them can easily be implemented. In particular, the language allows the direct use of differential algebraic objects. Among other objects being used, the picture object has proved very helpful for sophisticated manipulation of graphics.

Most commands of the COSY language consist of a keyword, followed by expressions and names of variables, and terminated by a semicolon. The individual entries and the semicolon are separated by blanks. The exceptions are the assignment statement, which does not have a keyword but is identified by the assignment identifier :=, and the call to a procedure, in which case the procedure name is used instead of the keyword.

The language consists of a tree-structured arrangement of nested program segments. There are three types of program segments. The first is the main program, of which there has to be exactly one and which has to begin at the top of the input file and which ends at the bottom. The others are procedures and functions.

Inside each program segment, there are three sections. The first section contains the declaration of local variables, the second section contains the local procedures and functions, and the third section contains the executable code.

All variables are visible inside the program segment in which they are declared as well as in all other program segments inside it. In case a variable has the same name as one that is visible from a higher level routine, its name and dimension override the name and properties of the higher level variable of the same name for the remainder of the procedure and all local procedures.

The next section of the program segment contains the declaration of local procedures and functions. Any such program segment is visible in the segment in which it was declared and in all program segments inside the segment in which it was declared, as long as the reference is physically located below the declaration of the local procedure. Recursive calls are permitted. Altogether, the local and global visibility of variables and procedures follows standard structured programming practice.

The third and final section of the program segment contains executable statements. Among the permissible executable statements are assignment statements, call to procedures and to FORTRAN subroutines.

There are also statements that control the program flow. These statements consist of matching pairs denoting the beginning and ending of a control structure and sometimes of a third statement that can occur between such beginning and ending statements. Control statements can be nested as long as the beginning and ending of the lower level control structure is completely contained inside the same section of the higher level control structure. The flow control statements supported by the COSY language are IF, WHILE, LOOP and FIT; the last statement goes beyond conventional languages and allows direct nonlinear optimization.

Besides the commands just presented, there are commands for input and output. They appear as commands and not as procedure calls because they have variable number of arguments. There are also commands to save code in compiled form. This allows later inclusion in another program without recompiling.

The COSY language is compiled by the program FOXY, which is written in standard FORTRAN 77 and has a length of about 3000 lines. The result of the compilation is metacode consisting of a sequence of integers. The metacode can be stored or executed using FOXY. Great care was taken to make FOXY as efficient as possible. In benchmark tests of compilation speed, it was only about 20 percent slower than an off the shelf PASCAL compiler.

As mentioned above, execution is controlled by machine independent metacode. In the case of operations with differential algebraic data types, the induced computational overhead is completely insignificant. For algorithms containing only double precision operations, an average slowdown by a factor of two has to be expected.

REFERENCES

[BBB64] K. L. Brown, R. Belbeoch, and P. Bounin. First- and second- order magnetic optics matrix equations for the midplane of uniform-field wedge magnets. *Review of Scientific Instruments*, 35:481, 1964.

[Ber87] M. Berz. The method of power series tracking for the mathematical description of beam dynamics. *Nuclear Instruments and Methods*, A258:431, 1987.

[Ber88a] M. Berz. Differential algebraic description and analysis of trajectories in vacuum electronic devices including spacecharge effects. *IEEE Transactions on Electron Devices*, 35-11:2002, 1988.

[Ber88b] M. Berz. Differential algebraic treatment of beam dynamics to very high orders including applications to spacecharge. *AIP Conference Proceedings*, 177:275, 1988.

[Ber89a] M. Berz. *The Description of Particle Accelerators using High Order Perturbation Theory on Maps, in: M. Month (Ed), Physics of Particle Accelerators*, volume 1, page 961. American Institute of Physics, 1989.

[Ber89b] M. Berz. Differential algebraic description of beam dynamics to very high orders. *Particle Accelerators*, 24:109, 1989.

[Ber90a] M. Berz. Arbitrary order description of arbitrary particle optical systems. *Nuclear Instruments and Methods*, A298:426, 1990.

[Ber90b] M. Berz. Computational aspects of design and simulation: COSY INFINITY. *Nuclear Instruments and Methods*, A298:473, 1990.

[Ber90c] M. Berz. COSY INFINITY, an arbitrary order general purpose optics code. *Computer Codes and the Linear Accelerator Community*, Los Alamos LA-11857-C:137, 1990.

[Ber90d] M. Berz. COSY INFINITY Version 3 reference manual. Technical Report MSUCL-751, National Superconducting Cyclotron Laboratory, Michigan State University, East Lansing, MI 48824, 1990.

[Ber90e] M. Berz. The DA precompiler DAFOR. Technical report, Lawrence Berkeley Laboratory, Berkeley, CA, 1990.

[Ber90f] M. Berz. Differential algebra precompiler version 3 reference manual. Technical Report MSUCL-755, Michigan State University, East Lansing, MI 48824, 1990.

[Ber91] M. Berz. COSY INFINITY Version 4 reference manual. Technical Report MSUCL-771, National Superconducting Cyclotron Laboratory, Michigan State University, East Lansing, MI 48824, 1991.

[Ber92] M. Berz. Automatic differentiation as nonarchimedean analysis. In *Computer Arithmetic and Enclosure Methods*, page 439, Amsterdam, 1992. Elsevier Science Publishers.

[Bro79] K. L. Brown. The ion optical program TRANSPORT. Technical Report 91, SLAC, 1979.

[BW87] M. Berz and H. Wollnik. The program HAMILTON for the analytic solution of the equations of motion in particle optical systems through fifth order. *Nuclear Instruments and Methods*, A258:364, 1987.

[Car87] D. C. Carey. *The Optics of Charged Particle Beams*. Harwood, 1987.

[CdB80] S. D. Conte and C. de Boor. *Elementary Numerical Analysis*. McGraw Hill, New York, 1980.

[FB89] E. Forest and M. Berz. *Canonical Integration and Analysis of Periodic Maps using Non-Standard Analysis and Lie Methods, in: Lie Methods in Optics II*, pages 47–66. Springer, Berlin, 1989.

[FBI89] E. Forest, M. Berz, and J. Irwin. Normal form methods for complicated periodic systems: A complete solution using Differential algebra and Lie operators. *Particle Accelerators*, 24:91, 1989.

[MMW72] T. Matsuo, H. Matsuda, and H. Wollnik. Particle trajectories in a toroidal condenser in a third order approximation. *Nuclear Instruments and Methods*, 103:515, 1972.

[MW70a] H. Matsuda and H. Wollnik. The influence of an inhomogeneous magnetic fringing field on the trajectories of charged particles in a third order approximation. *Nuclear Instruments and Methods*, 77:40, 1970.

[MW70b] H. Matsuda and H. Wollnik. Third order transfer matrices of the fringing field of an inhomogeneous magnet. *Nuclear Instruments and Methods*, 77:283, 1970.

[Rit50] J. F. Ritt. *Differential Algebra*. American Mathematical Society, Washington, D.C., 1950.

[WE65] H. Wollnik and H. Ewald. The influence of magnetic and electric fringing fields on the trajectories of charged particles. *Nuclear Instruments and Methods*, 36:93, 1965.

[Wol65] H. Wollnik. Second order approximation of the three-dimensional trajectories of charged particles in deflecting electrostatic and magnetic fields. *Nuclear Instruments and Methods*, 34:213, 1965.

[Wol67] H. Wollnik. Second order transfer matrices of real magnetic and electrostatic sector fields. *Nuclear Instruments and Methods*, 52:250, 1967.

[Wol68] H. Wollnik. Image aberrations of second order of electrostatic sector fields. *Nuclear Instruments and Methods*, 59:277, 1968.

[Wol87] H. Wollnik. *Charged Particle Optics*. Academic Press, Orlando, Florida, 1987.