Higher Order Multivariate Automatic Differentiation and Validated Computation of Remainder Bounds

MARTIN BERZ and KYOKO MAKINO
Department of Physics and Astronomy
Michigan State University
East Lansing, MI 48824
USA
berz@msu.edu, makino@msu.edu http://bt.pa.msu.edu/

Abstract: High order automatic differentiation has been used in recent years for simultaneous computation of Taylor expansions of functional dependencies as well as validated enclosures for the remainder over a certain domain. The resulting Taylor model methods offer certain advantages compared to other validated methods, including an approximation by a functional form with an accuracy scaling with a high order, as well as the ability to suppress the dependency problem.

In this paper we describe the implementation of the method in a validated setting. Several steps are taken to increase computational efficiency. Computations of coefficients are not performed in interval arithmetic, but rather in floating point arithmetic with simultaneous accurate accounting of the possible computational errors. Furthermore, the storage and elementary operations of the objects supports sparsity of the coefficients.

Key-Words: Multivariate, automatic differentiation, validation, interval, Taylor model, COSY Infinity.

1 Introduction

Taylor model methods[1, 2, 3] represent a combination of high order automatic differentiation[4, 5, 6, 7, 8] and interval methods[6, 9] to simultaneously determine a Taylor expansion as well as an enclosure for the remainder bound for a given functional dependency. The method allows a far reaching suppression of the dependency problem[10] that often hinders sharp enclosures by interval methods. Different from Interval Automatic Differentiation methods, the contributions to the remainder bounds are always computed from the currently accumulated floating point Taylor coefficients, and not via interval evaluation of AD code for derivatives; for all but simple functional dependencies, this significantly increases the sharpness of the resulting bounds.

To satisfy the requirements of rigor demanded by validated computations, it is necessary to properly account for errors in the coefficient arithmetic. This could be achieved by representing each coefficient by an interval; however, this approach has several disadvantages, including the need to represent each coefficient by two numbers instead of one, additional computational cost, and the undesirable growth of the size of the coefficients in extended calculations.

We set the stage by determining the requirements imposed on the floating point unit of the system as well as a set of interval tools. Next we address the tools for storage and addition of Taylor models, followed by the discussion of algorithms for multiplication and advanced operations.

2 Requirements for Floating Point and Interval Arithmetic

In the following, we describe in detail the current implementation of Taylor model arithmetic in version 8.1 of the code COSY INFINITY. Since in the Taylor model approach, the coefficients are floating point (FP) numbers, care must be taken that the inaccuracies of conventional FP arithmetic are properly accounted for. We begin with the requirements we impose on the underlying floating point arithmetic.

Definition 1 ***(Admissible FP Arithmetic)*** *We assume computation is performed in a floating point environment supporting the four elementary operations* $\oplus$, $\otimes$, $\ominus$, $\oslash$. *We call the arithmetic admissible if there are two positive constants denoted*

$$\varepsilon_u : \textit{underflow threshold}$$
$$\varepsilon_m : \textit{relative accuracy of elementary operations}$$

such that

1. *If the FP numbers* a, b *are such that* $a * b$ *exceeds* ε_u *in magnitude, then the product* $a * b$ *differs from the floating point multiplication result* $a \otimes b$ *by not more than* $|a \otimes b| \otimes \varepsilon_m$.
2. *The sum* $a + b$ *of FP numbers* a *and* b *differs from the floating point addition result* $a \oplus b$ *by not more than* $\max(|a|, |b|) \otimes \varepsilon_m$.

Definition 2 ***(Admissible Interval Arithmetic)*** *We assume that besides an admissible FP environment, there is an interval arithmetic environment of four elementary operations* $\oplus$, $\otimes$, $\ominus$, $\oslash$, *as well as a set* S *of intrinsic functions. We call the interval arithmetic admissible if for any two intervals* $[a_1, b_1]$ *and* $[a_2, b_2]$ *of floating point numbers and any* $\bigcirc \in \{\oplus, \otimes, \ominus, \oslash\}$ *and corresponding real operation* $\circ \in \{+, \times, -, /\}$, *we have*

$$[a_1, b_1] \bigcirc [a_2, b_2] \supset \{x \circ y | x \in [a_1, b_1],\ y \in [a_2, b_2]\},$$

and furthermore, for any interval intrinsic Ⓢ $\in S$ *representing the real function* s, *we have*

$$Ⓢ([a, b]) \supset \{\mathrm{s}(x) | x \in [a, b]\}.$$

Remark 1 ***(Testing of the COSY Interval Arithmetic)*** *Besides the tests performed in the development of the program, various other tests have been performed. Corliss and Yu performed extensive tests of the COSY interval tools by porting of COSY interval results to Maple in binary format and comparison with Maple computations with nearly* 1000 *digits of accuracy. Several thousand cases that are to be considered particularly difficult as well as around* 10^6 *random tests spanning all orders of magnitude of allowed domains of the intrinsics were performed[11]. Independently, Revol performed around* 10^8 *random tests of the interval arithmetic by comparison with a guaranteed precision library for elementary operations and intrinsic functions[12].*

For the specific purposes of Taylor model arithmetic, some additional considerations are necessary. First we note that combinatorial arguments show [13] that the number of nonzero coefficients in a polynomial of order n in v variables cannot exceed $(n+v)!/(n! \cdot v!)$. Furthermore, as also shown in [13], the number of multiplications necessary to determine all coefficients up to order n of the product polynomial of two such polynomials cannot exceed $(n + 2v)!/\,(n! \cdot (2v)!)$.

Definition 3 ***(Taylor Model Arithmetic Constants)*** *Let* n *and* v *be the order and dimension of the Taylor model computation. Then we fix constants denoted*

$$\varepsilon_c : \textit{cutoff threshold}$$
$$e : \textit{contribution bound}$$

such that

1. $\varepsilon_c^2 > \varepsilon_u$
2. $2 \geq e > 1 + 2 \cdot \varepsilon_m \cdot (n + 2v)!/\,(n! \cdot (2v)!)$

We remark that in a conventional double precision floating point environment, typical values for the constants of the admissible FP arithmetic may be $\varepsilon_u = 10^{-307}$ and $\varepsilon_m = 10^{-15}$. The Taylor arithmetic cutoff threshold ε_c can be chosen over a wide possible range, but since it is later used to control the number of coefficients actively retained in the Taylor model arithmetic, a value not too far below ε_m, such as $\varepsilon_c = 10^{-20}$, is a good choice for many cases. Furthermore, for essentially all practically conceivable cases of n and v, the choice $e = 2$ is satisfactory, and this is the number used in our implementation.

Under the assumption of the above properties of the floating point arithmetic, interval arithmetic, and the Taylor model arithmetic constants, we now describe the algorithms for Taylor model arithmetic, which will lead to the definition of admissible FP Taylor model arithmetic.

3 Storage, Error Tallying, and Sweeping

In the COSY implementation, a Taylor model T of order n and dimension v is represented by a collection of nonzero floating point coefficients a_i, as well as two coding integers $n_{i,1}$ and $n_{i,2}$ that contain unique information allowing to identify the term to which the coefficient a_i belongs. The coefficients are stored in an ordered list, sorted in increasing order first by size of $n_{i,1}$, and second, for each value of $n_{i,1}$, by size of $n_{i,2}$. For the purposes of the present discussion, the details about the meaning of the coding integers $n_{i,1}$ and $n_{i,2}$ is immaterial; their significance will become apparent in the discussion of algorithms for multiplication discussed below. There is also other information stored in the Taylor model, in particular the information of the expansion point and the domain, as well as various intermediate bounds that are useful for the necessary computation of range bounds; however this information is not critical for the further discussion. For simplicity of the subsequent arguments, all coefficients are always stored normalized to the interval $[-1, 1]$ with expansion point 0.

Only coefficients a_i exceeding the cutoff threshold ε_c in magnitude, i.e. satisfying $|a_i| > \varepsilon_c$, are retained, which in many practical cases can save significant computational expense. This is done for two reasons; first, in many high order calculations, the derivative vectors are often rather 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 first order case, this entails that one of the $v + 1$ component vanishes, in higher orders the effects are more significant. 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 $v = 6$ 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.

Another important aspect is that over the normalized interval $[-1, 1]$, Taylor models with a remainder bound that is meaningfully small are characterized by coefficients that fall of quickly in higher orders; thus it is likely that many of those of high order are actually of reduced significance and can be safely eliminated by ramping them into the remainder bound.

Since by requirement, $\varepsilon_c^2 > \varepsilon_u$, the multiplication of two retained coefficients can never lead to underflow. Besides the coefficients and coding integers, each TM also contains an interval I composed of two floating point numbers representing rigorous enclosures of the remainder bound.

In the elementary operations of Taylor models, the errors due to floating point arithmetic are accumulated in a floating point "tallying variable" t which in the end is used to increase the remainder bound interval I by an interval of the form $e \otimes \varepsilon_m \otimes [-t, t]$. The factor e assures a safe upper bound of all floating point errors of adding up the (positive) contributions to t. Accounting for the error through a single floating point variable t with the factor $e \cdot \varepsilon_m$ "factored out" notably increases computational efficiency. In addition, there is a "sweeping variable" s that will be used to absorb terms that fall below the cutoff threshold ε_c and are thus not explicitly retained.

4 Scalar Multiplication and Addition

The multiplication of a Taylor model T with coefficients a_i, coding integers $(n_{i,1}, n_{i,2})$ and remainder bound interval I with a floating point real number c is performed in the following manner. The tallying variable t and the sweeping variable s are initialized to zero. Going through the list of terms in the Taylor polynomial, each floating point coefficient a_i is multiplied by the floating point number c to yield the floating point result $b_k = a_i \otimes c$. The tallying variable t is incremented by $|b_k|$, accounting for the roundoff error in the calculation of b_k. If

$|b_k| \geq \varepsilon_c$, the term will be included in the resulting polynomial, and k will be incremented. If $|b_k| < \varepsilon_c$, the sweeping variable s is incremented by $|b_k|$. After all terms have been treated, the total remainder bound of the result of the scalar multiplication is set to be $[c, c] \otimes I \oplus e \otimes \varepsilon_m \otimes [-t, t] \oplus e \otimes [-s, s]$, which is performed in outward rounded interval arithmetic.

Addition of two Taylor models $T^{(1)}$ and $T^{(2)}$ with coefficients $a_i^{(1)}$ and $a_j^{(2)}$, coding integers ($n_{i,1}^{(1)}$, $n_{i,2}^{(1)}$) and ($n_{j,1}^{(2)}$, $n_{j,2}^{(2)}$), and remainder bounds I_1, I_2, respectively, is performed similar to the merging of two ordered lists. The pointers i, j of the two lists and pointer of the merged list k are initialized to 1. Then iteratively, the terms $(n_{i,1}^{(1)}, n_{i,2}^{(1)})$ and $(n_{j,1}^{(2)}, n_{j,2}^{(2)})$ are compared. In case $(n_{i,1}^{(1)}, n_{i,2}^{(1)}) \neq (n_{j,1}^{(2)}, n_{j,2}^{(2)})$, the term that should come first according to the ordering is merely copied, and its pointer as well as k are incremented. In case $(n_{i,1}^{(1)}, n_{i,2}^{(1)}) = (n_{j,1}^{(2)}, n_{j,2}^{(2)})$, we proceed as follows. We determine the floating point coefficient $b_k = a_i^{(1)} \oplus a_j^{(2)}$. To account for the error, we increment t by $\max(|a_i^{(1)}|, |a_j^{(2)}|)$. If $|b_k| \geq \varepsilon_c$, the term will be included in the resulting polynomial, and k will be incremented. If $|b_k| < \varepsilon_c$, the sweeping variable s is incremented by $|b_k|$. Finally i, j are incremented by one. After both the lists of $T^{(1)}$ and $T^{(2)}$ are completely transversed, the remainder bound is determined via interval arithmetic as $I_1 \oplus I_2 \oplus e \otimes \varepsilon_m \otimes [-t, t] \oplus e \otimes [-s, s]$, which is performed in outward rounded interval arithmetic.

5 Multiplication

The multiplication of two Taylor models $T^{(1)}$ and $T^{(2)}$ of order n with coefficients $a_i^{(1)}$ and $a_j^{(2)}$ and coding integers ($n_{i,1}^{(1)}$, $n_{i,2}^{(1)}$) and ($n_{j,1}^{(2)}$, $n_{j,2}^{(2)}$), respectively, is performed as follows.

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:

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

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.

Next, the terms of the polynomial $T^{(2)}$ are sorted into pieces $T_m^{(2)}$ of exact order m respectively. Then, each term in $T^{(1)}$ with order k is multiplied with all those terms of $T^{(2)}$ of order $(n-k)$ or less.

As a first step, we describe the determination of the location l of the product based on the coding integers $(n_1^{(1)}, n_2^{(1)})$ and $(n_1^{(2)}, n_2^{(2)})$, which is related to perhaps the main algorithmic difficulty of the method. For the purpose of being sufficiently transparent, we begin with a simplified version of the method, and in the next step present the full algorithm in detail.

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

$$\begin{aligned} n_c(M) &= n_c(x_1^{i_1} \cdot ... \cdot x_v^{i_v}) \\ &= i_1 \cdot (n+1)^0 + i_2 \cdot (n+1)^1 + ... + i_v \cdot (n+1)^{v-1}. \end{aligned}$$

So the exponents are just "digits" in a base $(n+1)$ representation. Note that since $i_\nu \leq n$, the function $M \to n(M)$ 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

$$n_c(M \cdot N) = n_c(M) + n_c(N).$$

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

$$I_M = p(n_c(M)).$$

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 below say 10^6 entries; with 8 variables the order would decrease to about 4, and this is too strict a limitation. To circumvent this, the two-stage generalization of the above coding and decoding based on two coding integers mentioned above is useful. It will become apparent how the method can be generalized to more stages at the expense of increased computational effort and reduced storage requirements.

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 n_1 and n_2 for any monomial in the following way:

$$\begin{aligned} n_1(x_1^{i_1}\cdot\dots\cdot x_v^{i_v}) =& i_1\cdot(n+1)^0 + i_2\cdot(n+1)^1 \\ &+\dots+ i_{\frac{v}{2}}\cdot(n+1)^{(\frac{v}{2}-1)} \\ n_2(x_1^{i_1}\cdot\dots\cdot x_v^{i_v}) =& i_{\frac{v}{2}+1}\cdot(n+1)^0 + i_{\frac{v}{2}+2}\cdot(n+1)^1 \\ &+\dots+ i_v\cdot(n+1)^{(\frac{v}{2}-1)}. \end{aligned} \tag{1}$$

Next we store the $N(n,v)$ monomials in a particular way. We note that this storage differs from the one used for the introduction of the lexicographical ordering described in [14]; interestingly enough, however, the arrangement presented here defines another lexicographical total ordering.

We start with all monomials that have $n_2(M) = 0$ and group them by order; within one order, the monomials are stored according to ascending values of $n_1(M)$. Then we store all those with $n_2(M) = 1$, again by order, and so forth, going through all possible values of n_2. Because of the order-by-order arrangement within the monomials belonging to the same $n_1(M)$, it follows that again

$$\begin{aligned} n_1(M\cdot N) &= n_1(M) + n_1(N) \\ n_2(M\cdot N) &= n_2(M) + n_2(N). \end{aligned}$$

Finally we introduce some "inverse" arrays p_1 and p_2 in the following way: For all n_1 and n_2 that appear as valid coding integers, we set $p_1(n_1) = (I_M$ of first monomial M with first coding integer $n_1)$ and $p_2(n_2) = (I_M$ of first monomial M with second coding integer $n_2) - 1$.

Again the arrays p_1 and p_2 can be generated once during the setup process. Using the definitions of n_1, n_2, p_1 and p_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

$$I_{M\cdot N} = p_1[n_1(I_M)+n_1(I_N)]+p_2[n_2(I_M)+n_2(I_N)]. \tag{2}$$

For the sake of clarity, table 1 shows an example for the arrays n_1, n_2, p_1 and p_2 for $n = 3$ and $v = 4$. This example also illustrates equations (1) through (2).

In a system with $s > 2$ stages, the exponents will be grouped into s blocks, and arranged in such a way that the ordering of block s takes precedence over block $s-1$, which takes precedence over that of block $s-2$, etc. Each monomial is assigned s coding integers $n_1...n_s$, and there are s "inverse" arrays $p_1...p_s$. The address computation computation generalizing eq. (2) then comprises s terms, each consisting of the addition of the n_i of the factors and the lookup of the sum in p_i.

The coding defined in (1) entails that the required length of the arrays p_1 and p_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.

In the two-stage scheme, 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

I_M	i_1	i_2	i_3	i_4	n_1	n_2	I_M	i_1	i_2	i_3	i_4	n_1	n_2	j	p_1	p_2
1	0	0	0	0	0	0	19	0	1	0	1	4	4	0	1	0
2	1	0	0	0	1	0	20	2	0	0	1	2	4	1	2	10
3	0	1	0	0	4	0	21	1	1	0	1	5	4	2	4	22
4	2	0	0	0	2	0	22	0	2	0	1	8	4	3	7	31
5	1	1	0	0	5	0	23	0	0	2	0	0	2	4	3	16
6	0	2	0	0	8	0	24	1	0	2	0	1	2	5	5	25
7	3	0	0	0	3	0	25	0	1	2	0	4	2	6	8	32
8	2	1	0	0	6	0	26	0	0	1	1	0	5	7	0	0
9	1	2	0	0	9	0	27	1	0	1	1	1	5	8	6	28
10	0	3	0	0	12	0	28	0	1	1	1	4	5	9	9	33
11	0	0	1	0	0	1	29	0	0	0	2	0	8	10	0	0
12	1	0	1	0	1	1	30	1	0	0	2	1	8	11	0	0
13	0	1	1	0	4	1	31	0	1	0	2	4	8	12	10	34
14	2	0	1	0	2	1	32	0	0	3	0	0	3			
15	1	1	1	0	5	1	33	0	0	2	1	0	6			
16	0	2	1	0	8	1	34	0	0	1	2	0	9			
17	0	0	0	1	0	4	35	0	0	0	3	0	12			
18	1	0	0	1	1	4										

Table 1: The arrangement of the 35 monomials $M = x_1^{i1} \cdot ... \cdot x_v^{i_v}$ for order $n = 3$ and number of variables $v = 4$ for the two-stage addressing scheme. Also shown are the coding integers $C1$ and $C2$ and the arrays $D1$ and $D2$. For all M, one verifies $I_M = D1(C1(M)) + D2(C2(M))$

v		4	6	8	10	12	16	20
n	$l = 10^6$	999	99	30	14	9	4	2
n	$l = 10^8$	9999	463	99	38	20	9	5

Table 2: The maximum order for different numbers of variables due to the limitation of the length l of the reverse addressing arrays $D1$, $D2$

time for the bookkeeping is quite limited. To be specific, on a typical UNIX 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.

We now address the error counting and the proper determination of the contributions to the remainder bound. As the first step of the multiplication, the contributions I to the remainder bound due to orders greater than n are computed using interval arithmetic as outlined in [1]. Then, for each of the monomial multiplications, after the address calculation is completed, we determine the floating point product $a = a_i^{(1)} \otimes a_j^{(2)}$ of the coefficients. To account for the error, we increment t by $|a|$. We add the term a to the coefficient b_l. To account for the error, we increment t by $\max(|a|, |b_l|)$.

After all monomial multiplications have been executed, all resulting total coefficients b_l of the product polynomial will be studied for sweeping. If $|b_l| \geq \varepsilon_c$, the term will be included in the resulting polynomial, and l will be incremented. If $|b_l| < \varepsilon_c$, the sweeping variable s is incremented by $|b_l|$, but l will not be incremented, i.e. the term is not retained. In the end, the remainder bound I is incremented by $e \otimes \varepsilon_m \otimes [-t, t] \oplus e \otimes [-s, s]$ which is executed in outward rounded interval arithmetic.

6 Summary

We have discussed the basic operations of the validated methods for the Taylor model coefficient arithmetic, and we summarize the results in a series of remarks. First we note that intrinsic functions, while their actual mathematical description is somewhat involved, can now be readily treated from an algorithmic point of view.

Remark 2 ***(Intrinsic Functions)*** *All intrinsic functions can be expressed as linear combinations of monomials of Taylor models, plus an interval remainder bound I_i[1]. The coefficients are obtained via interval arithmetic, including elementary interval operations and interval intrinsic functions. The necessary scalar multiplications, additions, and multiplications are executed based on the previous algorithms, and in the end the interval remainder bound I_i is added to the thus far accumulated remainder bound.*

The treatment of validated Taylor models via interval coefficients has certain performance advantages:

Remark 3 ***(Floating Point Versus Interval Coefficients)*** *Apparently the storage required is only approximately half of what would be required with intervals, and so for the same amount of storage, the accuracy of the representation can be increased; in the one dimensional case, this amounts to twice the order as would be possible with interval coefficients. Also, the amount of floating point arithmetic necessary to perform validated computations is reduced by about a factor of two compared to an interval implementation.*

The various algorithms just discussed form the basis of a computer implementation of Taylor model arithmetic; we state the COSY contains such an arithmetic, and comment on a variety of tests of it.

Definition 4 ***(Admissible FP Taylor Model Arithmetic)*** *We call a Taylor model arithmetic admissible if it is based on an admissible FP and interval arithmetic and it adheres to the algorithms for storage, scalar multiplication, addition, multiplication, and intrinsic functions described above.*

Remark 4 ***(FP Taylor Model Arithmetic in COSY INFINITY)*** *The code COSY INFINITY[15, 16, 17] contains an admissible Taylor model arithmetic in arbitrary order and in arbitrarily many variables. The code consists of around* 50,000 *lines of FORTRAN77 source that also cross-compiles to standard C. It can be used in the environment of the COSY language, as well as in F77 and C. It is also available as classes in F90 and C++. The code is highly optimized for performance in that any overhead for addressing of polynomial coefficients amounts to less than 30 percent of the floating point arithmetic necessary for the coefficient arithmetic. It also has full sparsity support in that coefficients below the cutoff threshold do not contribute to execution time and storage.*

Remark 5 ***(Verification and Validation of the COSY FP Taylor Model Arithmetic)*** *The FP TM arithmetic implemented in COSY is currently being verified and validated by two outside groups[11, 12] with a suite of challenging test problems. Independently, the validity of the algorithms forming the core of the COSY Taylor model FP algorithm have been verified by Revol[18].*

Acknowledgements

We gratefully acknowledge financial support through the US Department of Energy, Grant Number DE-FG02-95ER4093, the National Science Foundation, the German National Merit Foundation, the Illinois Consortium for Accelerator Research, and an Alfred P. Sloan Fellowship.

References:

[1] K. Makino. *Rigorous Analysis of Nonlinear Motion in Particle Accelerators.* PhD thesis, Michigan State University, East Lansing, Michigan, USA, 1998. Also MSUCL-1093.

[2] K. Makino and M. Berz. Remainder differential algebras and their applications. In M. Berz, C. Bischof, G. Corliss, and A. Griewank, editors, *Computational Differentiation: Techniques, Applications, and Tools*, pages 63–74, Philadelphia, 1996. SIAM.

[3] K. Makino and M. Berz. New applications of Taylor model methods. In G. Corliss, C. Faure, A. Griewank, L. Hascoët, and U. Naumann, editors, *Automatic Differentiation of Algorithms from Simulation to Optimization*, pages 359–364. Springer, 2002.

[4] A. Griewank. *Evaluating Derivatives - Principles and Techniques of Algorithmic Differentiation.* SIAM, Philadelphia, 2000.

[5] L. B. Rall. *Automatic Differentiation: Techniques and Applications.* Springer-Verlag, Berlin; Heidelberg; New York, 1981.

[6] R. E. Moore. *Interval Analysis.* Prentice-Hall, Englewood Cliffs, NJ, 1966.

[7] A. Griewank and G. F. Corliss, editors. *Automatic Differentiation of Algorithms: Theory, Implementation and Application.* SIAM, Philadelphia, 1991.

[8] M. Berz, C. Bischof, A. Griewank, G. Corliss, and Eds. *Computational Differentiation: Techniques, Applications, and Tools.* SIAM, Philadelphia, 1996.

[9] R. E. Moore. *Methods and Applications of Interval Analysis.* SIAM, 1979.

[10] K. Makino and M. Berz. Efficient control of the dependency problem based on Taylor model methods. *Reliable Computing*, 5(1):3–12, 1999.

[11] George F. Corliss. Private communication.

[12] Nathalie Revol. Private communication.

[13] M. Berz. *Modern Map Methods in Particle Beam Physics.* Academic Press, San Diego, 1999.

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

[15] M. Berz and K. Makino. COSY INFINITY Version 8.1 - user's guide and reference manual. Technical Report MSUHEP-20704, Department of Physics and Astronomy, Michigan State University, East Lansing, MI 48824, 2001. see also http://cosy.pa.msu.edu.

[16] M. Berz, J. Hoefkens, and K. Makino. COSY INFINITY Version 8.1 - programming manual. Technical Report MSUHEP-20703, Department of Physics and Astronomy, Michigan State University, East Lansing, MI 48824, 2001. see also http://cosy.pa.msu.edu.

[17] M. Berz et al. The COSY INFINITY web page. http://cosy.pa.msu.edu.

[18] N. Revol, K. Makino, and M. Berz. Taylor models and floating-point arithmetic: Proof that arithmetic operations are validated in COSY. *submitted*, 2003. University of Lyon LIP Report RR 2003-11, http://www.ens-lyon.fr/pub/LIP/Rapports/RR/RR2003/RR2003-11.ps.Z, INRIA Report RR-4737, http://www.inria.fr/rrrt/rr-4737.html, MSU Department of Physics Report MSUHEP-30212, http://bt.pa.msu.edu/pub.