Taylor models and floating-point arithmetic: proof that arithmetic operations are validated in COSY ⋆

N. Revol [a], K. Makino [b], M. Berz [c]

[a] *INRIA, LIP (UMR CNRS, ENS Lyon, INRIA, Univ. Claude Bernard Lyon 1), École Normale Supérieure de Lyon, 46 allée d'Italie, 69364 Lyon Cedex 07, France*

[b] *Department of Physics, University of Illinois at Urbana-Champaign, 1110 Green Street, Urbana, IL 61801-3080, USA*

[c] *Department of Physics and Astronomy, Michigan State University, East Lansing, MI 48824, USA*

Abstract

The goal of this paper is to prove that the implementation of Taylor models in COSY, based on floating-point arithmetic, computes results satisfying the "containment property", *i.e.* guaranteed results.

First, Taylor models are defined and their implementation in the COSY software by Makino and Berz is detailed. Afterwards IEEE-754 floating-point arithmetic is introduced. Then the core of this paper is given: the algorithms implemented in COSY for multiplying a Taylor model by a scalar, for adding or multiplying two Taylor models are given and are proven to return Taylor models satisfying the containment property.

Key words: Taylor model, COSY software, floating-point operation, rounding error, containment property, validated result.

1 Introduction

Computing with floating-point arithmetic and rounding errors and still being able to provide guaranteed results can be achieved in various ways. In this

⋆ Supported by the US Department of Energy, the Alfred P. Sloan Foundation, the National Science Foundation and Illinois Consortium for Accelerator Research.

Email addresses: `Nathalie.Revol@ens-lyon.fr` (N. Revol), `makino@uiuc.edu` (K. Makino), `berz@msu.edu` (M. Berz).

paper, techniques are studied for Taylor model computations. Taylor models constitute a way to rigorously manipulate and evaluate functions using floating-point arithmetic. They are composed of a polynomial part, which can be seen as an expansion of the function at a given point, and of an interval part which brings in the certification of the result, *i.e.* an enclosure of all errors which have occurred (truncation, roundings). Thus the Taylor models are a hybrid between conventional floating-point arithmetic and computer algebra. Their data size is limited even after a long sequence of operations, many operations can be defined, and yet the results of computations are rigorous like with interval methods (which correspond to Taylor models of order 0). Various algorithms exist for solutions of ODEs [7], quadrature [8] and range bounding [16] [15] [17], implicit equations [13] [6], etc.

The focus in this paper is to prove that the implementation in the COSY software [3] provides validated results, *i.e.* enclosures of the results, even if operations are performed using floating-point operations. The considered arithmetic operations are the multiplication of a Taylor model by a scalar in Section 4, the addition in Section 5 and the product in Section 6 of two Taylor models. Section 2 defines Taylor models and Section 3 recalls useful facts about IEEE-754 floating-point arithmetic. The algorithms are detailed before being proven correct: they are taken from COSY sources. They can also be found in Makino's thesis [15], along with the details of the data structure which are not recalled here.

2 Taylor models

A Taylor model is a convenient way to represent and manipulate a function on a computer. In the following, we first introduce Taylor models from the mathematical point of view, *i.e.* an exact arithmetic is assumed. Then the use of floating-point arithmetic and the modifications it implies are detailed. Finally, another, computationally more convenient, way of storing Taylor models on a computer using floating-point arithmetic and a sparse representation is given. This last subsection corresponds to the way Taylor models are represented in the COSY software [3].

2.1 Taylor models with exact arithmetic

Let f be a function on v variables: $f : [-1,1]^v \to \mathbb{R}$, a Taylor model of order ω for f is a pair (T_ω, I_R) where T_ω is the Taylor expansion of order ω for f at the point $(0,\ldots,0)$ and I_R is an interval enclosing the truncation error, I_R will also be called the *interval remainder* of the Taylor model.

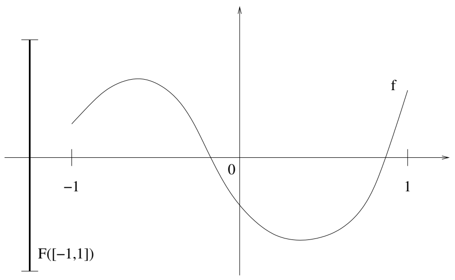

Fig. 1. Graphical representation of the function f and an enclosure of its range.

The interval remainder is required to satisfy the following so-called high order *scaling property*: if we consider the function f_h defined for $-1 \leq h \leq 1$, by [1], $f_h(x) = f(h \times x)$ and determine its remainder bound $I_{R,h}$, then as $h \to 0$, the width of $I_{R,h}$ behaves as $\mathcal{O}(h^{\omega+1})$. For instance, I_R could be computed as a Lagrange remainder as:

$$I_R = [-\alpha, \alpha] \text{ with } \alpha = \frac{1}{(\omega+1)!} \|f^{(\omega+1)}\|_\infty$$

where the $\| \ \|_\infty$ norm is taken over $[-1,1]^v$. However, determining I_R from a Lagrange remainder is in practice very difficult, certainly more so than bounding the original function itself, and so it is not very practical in most cases. In particular, in the COSY approach, remainder bounds are calculated in parallel to the computation of the floating-point representation of the coefficients from previous remainder bounds and the coefficients [15]. It suffices that the scaling property and the following *containment property* hold: $\forall x \in [-1,1]^v$, $f(x) \in [T_\omega(x), T_\omega(x)] + I_R$.

This property may be better illustrated in figures. Figure 1 shows a graphical representation of the function f. On the left the vertical bar represents an interval enclosure of the range of f over the whole domain. In Figure 2 a solid line corresponds to f whereas the dashed line corresponds to T_ω; for several arguments x, the vertical interval represents $[T_\omega(x), T_\omega(x)] + I_R$, and it contains $f(x)$. If this is repeated for every argument x, one obtains an enclosure of the graph of the f function in the dotted tube, shown on the right of Figure 2.

To simplify notations and algorithms, without loss of generality all considered Taylor models will be considered as having the same order ω, which must be in practice less or equal to the minimum of their actual orders. Indeed, it is meaningless to consider an order higher than the smallest of the orders of the summands when adding two Taylor models for instance, and the order of the result cannot exceed this value either.

[1] Throughout this paper, $\times$ wil be used as symbol for the multiplication in order to be visible when needed. In particular, it will not be needed inside a monomial, since monomials will be "transparent", cf. end of Subsection 2.3.

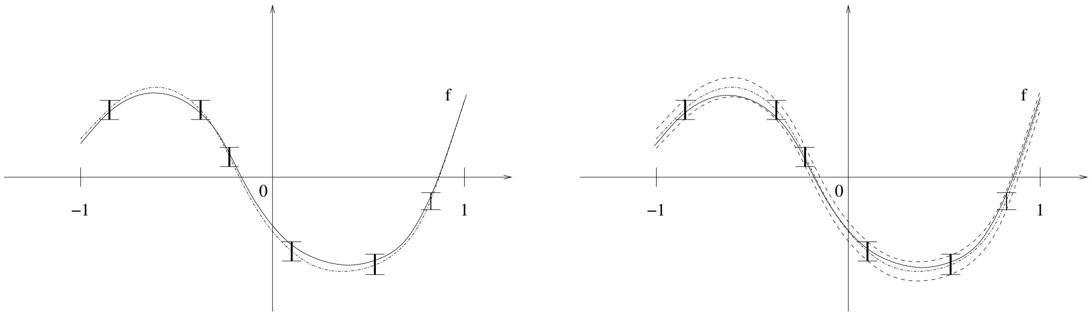

Fig. 2. Enclosures of $f(x)$ for various x (left) and enclosure of the graph of f (right).

Various operations can be performed on Taylor models, such as arithmetic operations ($+$, $\times$, $/$), computing their exponential or other algebraic or elementary functions ($\sqrt{\ }$, log, sin, arctan, cosh...), composing Taylor models, integrating or differentiating them and so on. In the following, we will focus on the multiplication of a Taylor model by a scalar (cf. Section 4), the addition (cf. Section 5) and multiplication (cf. Section 6) of two Taylor models.

2.2 Taylor models using floating-point arithmetic

In the previous definition, exact arithmetic is assumed: for instance the coefficients of the Taylor expansion are exactly represented. If floating-point arithmetic is assumed, then the coefficients of the polynomial must be floating-point numbers (typically double precision floating-point numbers of IEEE-754 arithmetic). So must be the representation of the remainder interval (its lower and upper bounds if intervals are represented by their endpoints).

Furthermore, rounding errors will inevitably occur during various computations involving Taylor models. To get validated results, the rounding errors due to approximate representation and to computations must be accounted for.

When floating-point arithmetic is used, a Taylor model is defined in the following way: let f be a function on v variables: $f : [-1, 1]^v \to \mathbb{R}$. In floating-point arithmetic, a Taylor model of order ω for f is a pair (T_ω, I_R). In this pair, T_ω is a polynomial in v variables of order ω with floating-point coefficients, these coefficients being floating-point representations of the coefficients of the exact Taylor expansion of order ω for f at the point $(0, \ldots, 0)$. The second member of this pair, I_R, is an interval; I_R encloses on the one hand the truncation error and on the other hand the rounding errors made in the construction of this Taylor model, both in the approximation of exact coefficients by floating-point arithmetic and during the various floating-point operations. It can be thought of as the sum of the interval remainder and of an enclosure of rounding errors.

Again, with floating-point arithmetic, the containment property still holds: $\forall x \in [-1,1]^v$, $f(x) \in [T_\omega(x), T_\omega(x)] + I_R$ if $T_\omega(x)$ is assumed to be exact, or if the rounding errors implied by its evaluation are accounted for in I_R.

2.3 Taylor models using floating-point arithmetic and sparsity

Since the algorithms analysed in this paper are the ones implemented in COSY, let us consider Taylor models as they are represented in COSY. COSY uses a sparse representation of Taylor models, *i.e.* it stores only the monomials that have a non-zero coefficient. In addition to this, COSY only stores coefficients with a "relevant" magnitude, *i.e.* whose absolute value is greater than a prescribed threshold. To preserve the property of validated results, monomials with a coefficient below this threshold are "swept" into the interval part, according to the following inclusion property:

$$\forall (x_1, \ldots, x_v) \in [-1,1]^v,\ \forall c \in \mathbb{R},\ \text{and natural } \omega_i,\ c \times x_1^{\omega_1} \ldots x_v^{\omega_v} \in [-|c|, |c|].$$

Sweeping a monomial $c \times x_1^{\omega_1} \ldots x_v^{\omega_v}$ corresponds to adding $[-|c|, |c|]$ to the interval remainder.

To sum up, in COSY, a Taylor model of order ω for a function f in v variables on $[-1,1]^v$ is a pair (T_ω, I). In this pair, T_ω is a polynomial in v variables of order ω with floating-point coefficients; these coefficients are floating-point representations of the coefficients of the exact Taylor expansion of order ω for f at the point $(0, \ldots, 0)$ whose absolute value is greater than a prescribed threshold. The second part of the pair, I, is an interval enclosing the sum of the following contributions:

- the truncation error,
- the rounding errors made in the construction of this Taylor model,
- the swept terms.

Conventions.

- Every Taylor model is assumed to be initialized to 0, *i.e.* every coefficient is initialized to 0 and the interval to $[0,0]$. This is used in the algorithms of Sections 4 to 6, given without initializations. For instance, in Section 6, the coefficients b_k are not set to 0 prior to their use as accumulators.
- To avoid tedious notations, the polynomial part T_ω will be represented as a tuple of coefficients $(a_i)_{1 \le i \le n}$ and the exact correspondance between the index i and the degree $(i_1, \ldots, i_v)$ of the corresponding monomial $x^{i_1} \ldots x^{i_v}$ will never be detailed.

3 IEEE-754 floating-point arithmetic and Taylor models in COSY

In order to bound rounding errors from above and to incorporate these estimates into the interval part of Taylor models, it is necessary to detail rounding errors for arithmetic operations with floating-point operands. This Section introduces floating-point arithmetic, as it is defined by the IEEE-754 standard, as well as some properties satisfied by this floating-point arithmetic and useful later on. To avoid burdening the reader, for the results presented in this Section, the proofs are relegated to the Appendix.

3.1 IEEE-754 floating-point arithmetic

3.1.1 IEEE-754 Floating-point numbers

The IEEE-754 standard [1] defines a binary floating-point system and an arithmetic that behaves in the same manner on every architecture (see also [2,9,14]). The goals of this standardization are the portability of numerical codes and the reproducibility of numerical computations. Furthermore it provides sound specifications that make possible proofs of the correct behaviour of programs, as in the remainder of this paper. The standard also specifies the handling of arithmetic exceptions.

Definition 1. IEEE-754 floating-point number system
A floating-point number system $\mathbb{F}$ *with base* β*, precision* p *and exponent bounds* $e_{\min}$ *and* $e_{\max}$ *is composed of a subset of* $\mathbb{R}$ *and some extra values; as far as real values are concerned, it contains* floating-point numbers *which have the form* $\pm mantissa \times \beta^e$*, where* β *is the base - in the following* β *will be equal to 2 - and mantissa is a real number whose representation in base* β *is* $m_0.m_1 \dots m_{p-1}$ *with digits* m_i *satisfying* $0 \leq m_i \leq \beta - 1$ *for* $0 \leq i \leq p-1$*; finally* e *is an integer such that* $e_{\min} \leq e \leq e_{\max}$*. In particular, 0 is represented twice, as* $+0$ *and* -0*. The other elements of* $\mathbb{F}$ *are* $+\infty$*,* $-\infty$ *and NaN (Not a Number, used for invalid operations).*
$\mathbb{F}$ *contains normalized and subnormal numbers. A* normalized number *is a number with* $m_0 \neq 0$*; when the base* β *equals 2, this implies that* $m_0 = 1$ *and* m_0 *does not have to be represented. A* subnormal number *is a number with* $e = e_{\min} - 1$ *and* $m_0 = 0$*. The threshold between normalized and subnormal numbers, also called* underflow threshold*, is* $\varepsilon_u = \beta^{e_{\min}}$.

With subnormal numbers, 0 can be represented and results between $-\varepsilon_u$ and ε_u have more accuracy.
The IEEE-754 standard defines two floating-point formats: for both of them, the base is $\beta = 2$. The single precision format has mantissas of length 24 bits ($p = 24$) and $e_{\min} = -126$, $e_{\max} = 127$ (a floating-point number fits into

a single word: 32 bits). The double precision format is defined by $p = 53$, $e_{\min} = -1022$ and $e_{\max} = 1023$ (a floating-point number is stored in 64 bits).

3.1.2 Ulp, rounding modes and rounding errors

Definition 2. u**: ulp (unit in the last place)**
Let 1^+ *denote the smallest floating-point number strictly larger than 1, then* $u = 1^+ - 1$*:* u *is called* ulp *for* unit in the last place *of the number 1.*

With the notations of Definition 1, $u = \beta^{-p+1}$. For formats defined by the IEEE-754 standard, in single precision $u = 2^{-23} \simeq 1.2 \times 10^{-7}$ and in double precision $u = 2^{-52} \simeq 2.2 \times 10^{-16}$.

A floating-point number system contains only a finite number of elements and it is thus not possible to represent every real number. A floating-point approximation $\mathrm{fl}(x)$ to a real number x is one of the two floating-point numbers surrounding x (except if x is exactly representable as a floating-point number, then $\mathrm{fl}(x) = x$, or for exceptional cases where $|x|$ is too large: *overflow*). The choice of one of these two floating-point numbers is determined by the active rounding mode. The IEEE-754 standard defines four rounding modes: rounding to nearest (even), rounding to $+\infty$, rounding to $-\infty$ and rounding to 0. With directed rounding modes, $\mathrm{fl}(x)$ is chosen as the floating-point number in the indicated direction. With rounding to nearest (even), $\mathrm{fl}(x)$ is chosen as the floating-point wich is the nearest of x; in case of a tie, *i.e.* when x is the middle of these two surrounding floating-point numbers, the one with the last bit m_{p-1} equal to 0 is chosen. The IEEE-754 standard also defines the behaviour of the four arithmetic operations $+$, $-$, $\times$, $/$ and of $\sqrt{\ }$. The result of these operations must be the same as if the exact result (in $\mathbb{R}$) were computed and then rounded.

Notation. Symbols without a circle denote exact operations and symbols with a circle denote either floating-point operations or, if some operands are intervals, outward rounded interval operations.
In the following, ε_M will denote an upper bound of the rounding error; it equals $u/2$ for rounding to nearest and $\varepsilon_M = u$ for the other rounding modes.

A consequence of the specifications for the arithmetic operations given by the IEEE-754 standard is the following: let $*$ be an arithmetic operation and $\circledast$ be its rounded counterpart, if $a \circledast b$ is neither a subnormal number nor an infinity nor a NaN, then $|(a \circledast b) - (a * b)| \leq \varepsilon_M |a * b|$, *i.e.*

$$
\begin{aligned}
|(a \circledast b) - (a * b)| &\leq \tfrac{1}{2} u |a * b| && \text{with rounding to nearest (even),} \\
|(a \circledast b) - (a * b)| &\leq u |a * b| && \text{with the other rounding modes.}
\end{aligned}
$$

Furthermore, it is possible to prove that the relative rounding error performed by each floating-point operation can be bounded from above using floating-point operation, as it is detailed in the following lemma.

Lemma 1. Estimating the rounding error using floating-point arithmetic
In what follows, a and b are assumed to be normalized floating-point numbers.

(1) If the floating-point numbers a, b are such that $a \times b$ neither overflows nor falls below ε_u (the underflow threshold) in magnitude, then the product $a \times b$ differs from the floating-point multiplication result $a \otimes b$ by no more than $|a \otimes b| \otimes (2\varepsilon_M)$. Since the floating-point multiplication by 2 in "$(2\varepsilon_M)$" is exact, there is no need to explicit it with $\times$ or $\otimes$.
(2) The sum $a + b$ of floating-point numbers a and b differs from the floating-point addition result $a \oplus b$ by no more than $|a \oplus b| \otimes (2\varepsilon_M)$, if $a \oplus b$ neither overflows nor falls below ε_u.
(3) With the same assumption, the sum $a + b$ of floating-point numbers a and b differs from the floating-point addition result $a \oplus b$ by no more than $\max(|a|, |b|) \otimes (2\varepsilon_M)$.

The proof of this lemma can be found in the appendix.

3.1.3 Rounding errors in sums

Let us denote by $S_n = \sum_{j=1}^n s_j$ and $\hat{S}_n = \bigoplus_{j=1}^n s_j$ this sum computed using floating-point arithmetic and any order on the s_j.

In the following, only nonnegative terms are added. The following lemma gives a formula using the computed sum that bounds the error from above.

Lemma 2.
If $\forall j \in \{1, \cdots, n\}$, $s_j \geq 0$ and if $(n-1) \times \varepsilon_M < 1$ then the error $E_n = S_n - \hat{S}_n$ is bounded as follows:

$$|E_n| \leq (n-1) \times \varepsilon_M \times \left(\bigoplus_{j=1}^n s_j \right).$$

This implies that $S_n = \sum_{j=1}^n s_j \leq (1 + (n-1)\varepsilon_M)\, \hat{S}_n = (1 + (n-1)\varepsilon_M) \left(\bigoplus_{j=1}^n s_j \right).$

The Lemmas 1 and 2 will be used in the following to prove that the algorithms studied in this paper provide guaranteed bounds even if they compute using floating-point operations only.

Some notations and assumptions used in COSY are now introduced. One of these assumptions is classical in rounding error analysis [12]: it stipulates that the number of floating-point operations multiplied by the rounding error bound ε_M is less than a given quantity $\eta < 1$, and quite often η is chosen as $1/2$. It has been proven in [5] (Chapter 2, p.96, Eq. (2.60)) that for Taylor models of order ω in v variables, the maximal number of floating-point operations involved in an operation between two Taylor models is less or equal to $(\omega + 2v)!/(\omega!(2v)!)$. A last lemma, using these assumptions, is then given: it relates an exact sum with its computed counterpart.

Notations and assumptions: constants in Taylor model arithmetic
Let ω and v be the order and dimension of the Taylor models. We fix constants denoted by

$$\begin{aligned}
\varepsilon_m &: \textit{an error factor which only has to satisfy } \varepsilon_m \geq 2\varepsilon_M \textit{ (cf. [15])} \\
\varepsilon_c &: \textit{cutoff threshold} \\
\eta &: \textit{accumulated rounding errors} \\
e &: \textit{contribution bound (a floating-point number)}
\end{aligned}$$

such that the following inequalities hold:

(1) $\varepsilon_c^2 > \varepsilon_u$,
(2) $1 > \eta > \varepsilon_m(\omega + 2v)!/(\omega!(2v)!)$,
(3) $e \geq (1 + \varepsilon_m/2)^3 \times (1 + \eta)$.

In a conventional double precision floating-point environment, typical values for these constants may be $\varepsilon_u \sim 10^{-307}$ and $\varepsilon_m \sim 10^{-15}$. The Taylor arithmetic cutoff threshold ε_c can be chosen over a wide possible range, but since it is used to control the number of coefficients actively retained in the Taylor model arithmetic, a value not too far below ε_m, like $\varepsilon_c = 10^{-20}$, is a good choice.
A classical value for η is $1/2$ and it then implies that assumption (3) is satisfied with $e = 2$ for usual floating-point precisions.

The following lemma derives from Lemma 2 and will be intensively used to prove that rounding errors in Taylor models operations are properly accounted for in the computation of the interval remainder.

Lemma 3. Link between a floating-point sum and an exact sum
If the previous assumptions are satisfied and if $\forall j,\ s_j \geq 0$, then:

$$\sum_{j=1}^{n} \Big(\varepsilon_M \otimes s_j\Big) \leq e \otimes \varepsilon_M \otimes \bigoplus_{j=1}^{n} s_j.$$

The proof is to be found in the appendix.

Our “floating-point arithmetic toolbox” is now complete. We can turn to the core of this paper, which is the proof that arithmetic operations on Taylor models, as they are implemented in COSY using floating-point operations, are correct.

4 Multiplication of a Taylor model by a scalar

The first operation considered here is the simplest one, in terms of its proof. Furthermore, the structure of the proof appears clearly and this scheme will be reproduced and adapted for the other operations.

4.1 Algorithm using exact arithmetic

Let us multiply the Taylor model $T = ((a_i)_{1 \leq i \leq n}, I)$ by a floating-point scalar c and let us denote by $T' = ((b_k)_{1 \leq k \leq n}, J)$ the result of this multiplication.

The algorithm is the following:
  for $k = 1$ to n do
    $b_k = c \times a_k$
  $J = c \times I$

4.2 Identification of rounding errors

The goal is now to identify the source of rounding errors and to give an upper bound of these errors using only floating-point operations. The previous algorithm is recalled on the left and rounding errors are mentioned in the right column.

previous algorithm	rounding error bounded by
for $k = 1$ to n do	
$b_k = c \times a_k$	$\varepsilon_m \otimes \lvert c \otimes a_k \rvert$
$J = c \times I$	no error since interval arithmetic is used

Furthermore, in COSY implementation of Taylor models, only coefficients above the given threshold ε_c are kept, the others are temporarily swept into a *sweeping variable* and then into the interval part. The corresponding algorithm is given below, with s denoting the sweeping variable, and again rounding errors are identified in the right column.

algorithm	rounding error bounded by
$s = 0$	
for $k = 1$ to n do	
$\quad b_k = c \times a_k$	$\varepsilon_m \otimes \lvert c \otimes a_k \rvert$
$\quad$ if $\lvert b_k \rvert < \varepsilon_c$ then	
$\quad\quad s = s + \lvert b_k \rvert$	$\varepsilon_m \otimes \max(s, \lvert b_k \rvert)$, with s taken before assignement
$\quad\quad b_k = 0$	
$J = c \times I + [-s, s]$	no error since interval arithmetic is used

4.3 Algorithm using floating-point arithmetic

One more variable t, called the *tallying variable*, is introduced: $\varepsilon_m \otimes t$ collects every upper bound of the rounding errors shown in the right column above. More precisely, t collects every rounding factor and is multiplied by ε_m and by e as a safety factor before being incorporated into the interval part, as it is shown in the following algorithm, which corresponds to the COSY implementation:

$t = 0$

$s = 0$

for $k = 1$ to n do

$\quad b_k = c \otimes a_k$

$\quad t = t \oplus |b_k|$

$\quad$ if $|b_k| < \varepsilon_c$ then

$\quad\quad s = s \oplus |b_k|$

$\quad\quad b_k = 0$

$J = c \otimes I \oplus e \otimes (\varepsilon_m \otimes [-t, t]) \oplus e \otimes [-s, s]$

Algorithm for the multiplication of a Taylor model by a scalar in COSY.

In the last line, circled interval operations denote outward rounded interval operations, *i.e.* guaranteed floating-point interval operations.

4.4 Proof that this algorithm is correct

To prove that this algorithm returns a Taylor model satisfying the property

$$\forall y_x \in [T(x), T(x)] + I,\ c \times y_x \in [T'(x), T'(x)] + J,$$

we have to prove that J encloses the interval $c \times I$ plus all rounding errors and swept terms. This means that we have to prove that the "extra" term $e \otimes (\varepsilon_m \otimes [-t, t]) \oplus e \otimes [-s, s]$ encloses the exact sum of all rounding error bounds and of all swept terms. The proof is decomposed into the following sub-tasks:

(1) prove that the rounding errors are correctly bounded by $e \otimes \varepsilon_m \otimes t$: the rounding errors made in each multiplication plus the rounding errors made in the accumulation in t;
(2) prove that the swept terms and the rounding errors made in the computation of s are correctly bounded from above by $e \times s$;
(3) the last computation is an interval computation and thus there is no need to take care of rounding errors. Actually, only the multiplication $c \otimes I$, the multiplication by e and the two additions need to be performed using interval arithmetic, the multiplication $\varepsilon_m \otimes t$ can be done using floating-point arithmetic. If $e = 2$ and IEEE-754 arithmetic is employed, then the multiplication by e is exact and again no interval arithmetic is required.

Proof of (1)
Let us first prove that the tallying term t takes correctly into account the accumulation of rounding errors made on the multiplications "$c \otimes a_k$".
For each k, the error on b_k is bounded by $\varepsilon_m \otimes |b_k|$ (cf. Lemma 1) thus the sum of every such error is bounded by $\sum_{k=1}^{n} \varepsilon_m \otimes |b_k|$. That $\sum_{k=1}^{n} \varepsilon_m \otimes |b_k|$ is less or equal to the term added to J, $e \otimes \varepsilon_m \otimes \bigoplus_{k=1}^{n} |b_k|$ is given by Lemma 3 and assumption (3) of the definition of Taylor model arithmetic constants, since $n\frac{\varepsilon_m}{2}$ is bounded from above by η.

Proof of (2)
Let us now prove that the term $e \otimes [-s, s]$ takes correctly into account the swept terms along with the rounding errors induced by the floating-point computation of s. Since $\otimes$ is here an interval operation, $e \otimes [-s, s]$ encloses $e \times [-s, s]$.
Let K denote the set $\{k : |b_k| < \varepsilon_c\}$ and $\sharp K$ its number of elements, we have to prove the inequality $e \times s = e \times \bigoplus_{k \in K} |b_k| \geq \sum_{k \in K} |b_k| +$ error on this sum.

We already know that (1st part of Lemma 2) the error on this sum is smaller than $\sharp K \times \varepsilon_m / 2 \times \bigoplus_{k\in K} |b_k|$, thus, using also the 2nd part of Lemma 2 to bound $\sum_{k\in K} |b_k|$,

$$\sum_{k\in K} |b_k| + \text{ error on this sum} \leq (1 + \sharp K \times \varepsilon_m) \bigoplus_{k\in K} |b_k|$$

and again, using assumption (2): $\sharp K \times \varepsilon_m \leq \eta$ and assumption (3): $1 + \eta \leq e$ in the definition of Taylor model arithmetic constants, we obtain that

$$\sum_{k\in K} |b_k| + \text{ error on this sum} \leq e \times \bigoplus_{k\in K} |b_k| = e \times s.$$

The tallying variable and the sweeping variable, as computed in the previous algorithm using floating-point arithmetic, thus fulfill their role.

5 Addition of two Taylor models

In this Section, the algorithm for adding two Taylor models using floating-point arithmetic and the proof that the computed Taylor model satisfies the containment property are given.

5.1 Algorithm using exact arithmetic

Let us add the Taylor model $T^{(1)} = \left((a_i^{(1)})_{1\leq i\leq n}, I^{(1)}\right)$ to the Taylor model $T^{(2)} = \left((a_j^{(2)})_{1\leq j\leq n}, I^{(2)}\right)$ and let us denote by $T = ((b_k)_{1\leq k\leq n}, J)$ the result of this addition.

The algorithm is the following:

for $k = 1$ to n do
    $b_k = a_k^{(1)} + a_k^{(2)}$
$J = I^{(1)} + I^{(2)}$

5.2 Identification of rounding errors

Let us proceed as in Subsection 4.3. The sweeping variable s is incorporated in the algorithm (left column) and the right column gives bounds on the rounding errors, every time such an error occurs.

algorithm	rounding error bounded by
$s = 0$	
for $k = 1$ to n do	
$\quad b_k = a_k^{(1)} + a_k^{(2)}$	$\varepsilon_m \otimes \max(\lvert a_k^{(1)}\rvert, \lvert a_k^{(2)}\rvert)$
$\quad$ if $b_k < \varepsilon_c$ then	
$\qquad s = s + \lvert b_k\rvert$	$\varepsilon_m \otimes \max(s, \lvert b_k\rvert)$ (with s taken before assignment)
$\qquad b_k = 0$	
$J = I^{(1)} + I^{(2)} + [-s, s]$	no error since interval arithmetic is used

5.3 Algorithm using floating-point arithmetic

The final algorithm is the following: the tallying variable t is invoked to collect all rounding errors.

$t = 0$

$s = 0$

for $k = 1$ to n do

$\quad b_k = a_k^{(1)} \oplus a_k^{(2)}$

$\quad t = t \oplus \max(|a_k^{(1)}|, |a_k^{(2)}|)$

$\quad$ if $|b_k| < \varepsilon_c$ then

$\qquad s = s \oplus |b_k|$

$\qquad b_k = 0$

$J = I^{(1)} \oplus I^{(2)} \oplus e \otimes (\varepsilon_m \otimes [-t, t]) \oplus e \otimes [-s, s]$

Algorithm for the addition of two Taylor models in COSY.

We note that in the actual implementation, because of the sparsity, addition of elements in the loop happens only if both of the matching entries are nonzero; if one of them vanishes, a mere copying is executed, and if both of them vanish, a zero is generated.

5.4 *Proof that this algorithm is correct*

Again, the goal is to prove that J correctly encloses the interval remainder plus all rounding errors and swept terms. As in Subsection 4.4, the proof is split into three sub-proofs.

(1) Proof that the rounding errors are correctly bounded from above by $e \otimes \varepsilon_m \otimes t$, *i.e.* the accumulation of rounding errors made in each addition. To achieve this, the corresponding sub-proof of Subsection 4.4 applies.
(2) Proof that the swept terms and the rounding errors made in the computation of s are correctly bounded from above by $e \times s$. Again, the corresponding sub-proof of Subsection 4.4 can be copied without a single modification.
(3) Again, the last computation is an interval computation and thus there is no need to take care of rounding errors. Actually, only the three additions and possibly the multiplications by e, if $e \neq 2$ or if an arithmetic not having 2 as radix is used, need to be performed using interval arithmetic, the multiplication $\varepsilon_m \otimes t$ can be done using floating-point arithmetic.

6 Multiplication of two Taylor models

In this Section, the algorithm multiplying two Taylor models using floating-point arithmetic is given: for multiplication, operations can be performed in various orders and here we stick to the one implemented in COSY. Then the proof that the computed Taylor model satisfies the containment property is presented.

6.1 *Algorithm using exact arithmetic*

Let us multiply the Taylor model $T^{(1)} = \left((a_i^{(1)})_{1 \le i \le n}, I^{(1)}\right)$ by the Taylor model $T^{(2)} = \left((a_j^{(2)})_{1 \le j \le n}, I^{(2)}\right)$ and let us denote by $T = ((b_k)_{1 \le k \le n}, J)$ the result of this multiplication. The polynomial part of T is the truncated product of the polynomial parts of $T^{(1)}$ and $T^{(2)}$, with a truncation at order ω. The interval part of T contains an enclosure of the truncated terms plus the product $I^{(1)} \times I^{(2)}$ and also plus the product of $I^{(1)}$ by an enclosure of the range of the $T_\omega^{(2)}$ over $[-1,1]^v$ and the product of $I^{(2)}$ by an enclosure of the range of the $T_\omega^{(1)}$ over $[-1,1]^v$. If necessary, more details can be found in [15].

Let us just recall that an enclosure of the range of a monomial $a \times x_1^{\omega_1} \dots x_v^{\omega_v}$ over $[-1,1]^v$ is simply $[-|a|, |a|]$. Let us finally denote by J_{tmp} a temporary interval variable.

The algorithm is the following:

for $i = 1$ to n do
  $J_{tmp} = [0, 0]$
  for $j = 1$ to n do
    if the corresponding monomial in the product is of order $\leq \omega$ then
      $p = a_i^{(1)} \times a_j^{(2)}$ $\quad (*)$
      $b_k = b_k + p$ $\quad (**)$
    else
      $J_{tmp} = J_{tmp} + [-|a_j^{(2)}|, |a_j^{(2)}|]$
  $J = J + [-|a_i^{(1)}|, |a_i^{(1)}|] \times (J_{tmp} + I^{(2)})$
$J = J + I^{(1)} \times \left(I^{(2)} + \sum_{j=1}^{n}[-|a_j^{(2)}|, |a_j^{(2)}|]\right)$

For the sake of readability, the determination of the index k from the i^{th} monomial of $T^{(1)}$ and the j^{th} monomial of $T^{(2)}$ is not detailed in the given algorithms because it is immaterial for the purpose of validation; details can be found in [4].

6.2 Identification of rounding errors

The only rounding errors that occur happen for $(*)$, the product $p = a_i^{(1)} \times a_j^{(2)}$, and for $(**)$, the accumulation in b_k: $b_k = b_k + p$.
For $(*)$, the rounding error is bounded from above by $\varepsilon_m \otimes |a_i^{(1)} \times a_j^{(2)}|$ and for $(**)$, it is bounded by $\varepsilon_m \otimes \max(|b_k|, |p|)$ (with the value of b_k before the assignment). Every other arithmetic operation being an interval operation, no other rounding error occurs.

Finally, coefficients b_k below the threshold ε_c are swept. This is achieved by the following lines which are appended at the end of the previous algorithm.

$s = 0$
for $k = 1$ to n do
  if $|b_k| < \varepsilon_c$ then
    $s = s + |b_k|$
    $b_k = 0$
$J = J + e \times [-s, s]$

In the final version of the algorithm, rounding errors (up to a factor $e \times \varepsilon_m$) are accumulated in the tallying variable t.

$t = 0$

for $i = 1$ to n do

  $J_{tmp} = [0, 0]$

  for $j = 1$ to n do

    if the corresponding monomial in the result is of order $\leq \omega$ then

      $p = a_i^{(1)} \otimes a_j^{(2)}$

      $t = t \oplus |a_i^{(1)} \otimes a_j^{(2)}|$

      $t = t \oplus \max(|b_k|, |p|)$

      $b_k = b_k \oplus p$

    else

      $J_{tmp} = J_{tmp} \oplus [-|a_j^{(2)}|, |a_j^{(2)}|]$

  $J = J \oplus [-|a_i^{(1)}|, |a_i^{(1)}|] \otimes (J_{tmp} \oplus I^{(2)})$

$J = J \oplus I^{(1)} \otimes \left(I^{(2)} \oplus \bigoplus_{j=1}^{n} [-|a_j^{(2)}|, |a_j^{(2)}|]\right)$

$s = 0$

for $k = 1$ to n do

  if $|b_k| < \varepsilon_c$ then

    $s = s \oplus |b_k|$

    $b_k = 0$

$J = J \oplus e \otimes \varepsilon_m \otimes [-t, t] \oplus e \otimes [-s, s]$

Algorithm for the product of two Taylor models in COSY.

For the sake of completeness, let us mention that this algorithm is performed twice in COSY, with the loops on i and j exchanged the second time. This leads to the computation of two different intervals for J and the resulting J is the intersection of these two intervals; it is expected that frequently a tighter J is returned. Anyway, the following proof also applies to the algorithm with the two loops exchanged and thus the intersection of the two computed intervals encloses the truncation and rounding error terms.

Again, the goal is to prove that J correctly encloses the interval remainder plus all rounding errors and swept terms. As in Subsection 4.4, the proof is split into three sub-proofs.

(1) Proof that the rounding errors, *i.e.* the accumulation of rounding errors made in each addition or multiplication, are correctly bounded from above by $e \times \varepsilon_m \times t$.
(2) Proof that the swept terms and the rounding errors made in the computation of s are correctly bounded from above by $e \times s$. Again, the corresponding sub-proof of Subsection 4.4 can be copied without a single modification.
(3) Again, for every interval computation, there is no need to take care of rounding errors.

Proof of (1)
Let us prove that $e \times \varepsilon_m \times t$ is greater than the sum of all rounding errors.
As previously, in the following formulae k is implicitely a function of i and j.
It is known from Lemma 1 that

$$\begin{aligned}\textstyle\sum |\text{rounding error}| &\leq \textstyle\sum_{i,j} \varepsilon_m \left(|a_i^{(1)} \otimes a_j^{(2)}| + \max(|b_k|, |a_i^{(1)} \otimes a_j^{(2)}|)\right) \\ &\leq \varepsilon_m \textstyle\sum_{i,j} \left(|a_i^{(1)} \otimes a_j^{(2)}| + \max(|b_k|, |a_i^{(1)} \otimes a_j^{(2)}|)\right).\end{aligned}$$

Let us denote by N the total number of operations. From the second part of Lemma 2, the right hand side satisfies

$$\begin{aligned}&\varepsilon_m \textstyle\sum_{i,j} \left(|a_i^{(1)} \otimes a_j^{(2)}| + \max(|b_k|, |a_i^{(1)} \otimes a_j^{(2)}|)\right) \\ &\qquad \leq \varepsilon_m \times \left(1 + N\frac{\varepsilon_m}{2}\right) \textstyle\bigoplus_{i,j} \left(|a_i^{(1)} \otimes a_j^{(2)}| \oplus \max(|b_k|, |a_i^{(1)} \otimes a_j^{(2)}|)\right)\end{aligned}$$

where the floating-point sum is performed in an arbitrary order: in particular this sum can be t. Thus

$$\varepsilon_m \sum_{i,j} \left(|a_i^{(1)} \otimes a_j^{(2)}| + \max(|b_k|, |a_i^{(1)} \otimes a_j^{(2)}|)\right) \leq \varepsilon_m \times \left(1 + N\frac{\varepsilon_m}{2}\right) \times t \leq \varepsilon_m \times e \times t$$

since $N \leq (\omega + 2v)!/(\omega!(2v)!)$ holds [5].
Finally, it has been proven that

$$\sum |\text{rounding error}| \leq e \times \varepsilon_m \times t$$

and, since the interval added to J is $e \otimes \varepsilon_m \otimes [-t, t]$ where $\otimes$ are (outward rounded) interval multiplications, this proves that J encloses the rounding errors made during the computation.

7 Conclusion

In this paper, the multiplication of a Taylor model by a scalar and the sum or product of two Taylor models are proven to return an interval enclosing every possible rounding error in addition to the truncation error. This means that the evaluation of a Taylor model at a point, using Horner's scheme, will also return an enclosure of the result.

So-called "intrinsics", such as division or square root and elementary functions (exp, log, sin, arctan, cosh...) are also available in COSY. They are computed using their Taylor expansions and an explicit knowledge of a bounding term for the truncated part. It is thus possible to compose an intrinsic (in 1 variable) with a Taylor model, using this explicit bounding term to compute its interval remainder and using Horner's scheme for its polynomial part. However, let us compose a function $f(x) = f_0 + f_1 x + \ldots$ by exp for instance: $\exp(f(x))$ is performed as $\exp(f_0) \times \exp(f_1 x + \ldots) = \exp(f_0) \times (1 + g(x) + g(x)^2/2 + \ldots)$ and evaluating $\exp(f_0)$ must be possible. Thus, for the implementation of intrinsics, what is needed is a bound on the rounding errors made during the floating-point evaluation of these functions. Unfortunately, such bounds do not exist in the IEEE-754 standard for elementary functions...

More sophisticated mechanisms are implemented in COSY. In particular the *linear dominated bounds* algorithm, or in short LDB [17], computes an enclosure of the range of a function, given by a Taylor model, over an interval; the result is usually tighter than the one obtained by simply replacing variables by the corresponding intervals in the polynomial part. The LDB algorithm is also based on floating-point arithmetic and it would be worth proving that it returns an enclosure of the sought range. Integration of ODEs with initial conditions is also performed in COSY [7]; it is based on Picard's iterations. Again, this algorithm should be proven to return validated results, using the same approach as in this paper.

In the algorithms presented in this paper, rounding errors are bounded from above using formulae of Lemma 1. It is a question to determine for which kind of algorithms such estimate of rounding errors could reveal useful, *i.e.* return tight upper bound of the rounding errors. Indeed, tightness was not an issue of this paper. In fact, in actual calculations for reasonable orders [5,15], the contributions to the remainder bounds due to the truncation of the series usually dominate the contributions due to floating-point errors, and so the computed intervals are usually satisfactorily narrow. It is still a question, anyway, to study and possibly improve the tightness of these bounds: more elaborate results of floating-point arithmetic [11,19,20], such as the *fast-two-sum* algorithm [10] or Sterbenz theorem [21], could yield tighter results than the systematic application of Lemma 1, probably at the price of a loss of speed.

Appendix: proofs of the lemmas of section 3

A.1 Proof of Lemma 1

Lemma 1.
We use here $\varepsilon_m = 2\varepsilon_M$. The original version of the lemma holds since floating-point multiplications and divisions by 2 are exact.

(1) If the floating-point numbers a, b are such that $a \times b$ neither overflows nor falls below ε_u (the underflow threshold) in magnitude, then the product $a \times b$ differs from the floating-point multiplication result $a \otimes b$ by no more than $|a \otimes b| \otimes \varepsilon_m$.
(2) The sum $a + b$ of floating-point numbers a and b differs from the floating-point addition result $a \oplus b$ by no more than $|a \oplus b| \otimes \varepsilon_m$, if $a \oplus b$ neither underflows nor overflows.
(3) With the same assumption, the sum $a + b$ of floating-point numbers a and b differs from the floating-point addition result $a \oplus b$ by no more than $\max(|a|, |b|) \otimes \varepsilon_m$.

Proof of (1)
A consequence of the correct rounding assumption in IEEE-754 arithmetic is that

$$|(a \otimes b) - (a \times b)| \leq \frac{1}{2}\varepsilon_m |a \times b|$$

(from [12], equation (2.4): $(a \circledast b) = (a * b)(1 + \delta)$ with $|\delta| \leq \varepsilon_m/2$), and

$$|(a \otimes b) - (a \times b)| \leq \frac{1}{2}\varepsilon_m |a \otimes b|$$

(from [12], equation (2.5): $(a \circledast b) = (a * b)/(1 + \delta)$ with $|\delta| \leq \varepsilon_m/2$). It follows that, if $\varepsilon_m \otimes |a \otimes b|/2$ does not fall below ε_u,

$$|(a \otimes b) - (a \times b)| \leq \frac{1}{2}\varepsilon_m |a \otimes b| \leq (1 + \varepsilon_m/2)\left(\frac{1}{2}\varepsilon_m \otimes |a \otimes b|\right).$$

Since $1 + \varepsilon_m/2 \leq 2$ and since floating-point multiplications by 2 are exact, eventually it holds

$$|(a \otimes b) - (a \times b)| \leq \varepsilon_m \otimes |a \otimes b|.$$

In case $\varepsilon_m \otimes |a \otimes b|/2 \leq \varepsilon_u$, it is still greater or equal to μ the smallest positive (subnormal) floating-point number, and from [18],

$$\varepsilon_m \otimes |a \otimes b|/2 \leq \varepsilon_m \otimes |a \otimes b|/2 + \mu \leq \varepsilon_m \otimes |a \otimes b|/2$$

i.e. assumption (1) is satisfied.

Proof of (2)
A proof similar to the previous one establishes that $|(a \oplus b) - (a + b)| \leq \varepsilon_m \otimes |a \oplus b|$.

Proof of (3)
If a and b have opposite signs, then $|a \oplus b| \leq \max(|a|, |b|)$ and thus $|(a \oplus b) - (a + b)| \leq \varepsilon_m \otimes \max(|a|, |b|)$, since $|(a \oplus b) - (a + b)| \leq \varepsilon_m \otimes |a \oplus b|$.

If a and b are of the same sign, without loss of generality they can be assumed to be both nonnegative with $0 \leq b \leq a$. The proof distinguishes several cases.

- If $a = b$, then $a + b = a \oplus b = 2a$ since floating-point multiplications by 2 are exact in IEEE-754 arithmetic (as long as no overflow occurs) and the error is zero. It holds that error $= 0 \leq \varepsilon_m \otimes \max(|a|, |b|)$.

- If $b < a$ and if b can be written as $a \times (1 - \beta)$ with $\varepsilon_m \leq \beta \leq 1$ then $a+b = (2-\beta)\times a = (2-\beta)\times\max(|a|, |b|)$ whereas $a \oplus b = (1+\delta)\times(2-\beta)\times a = (1 + \delta) \times (2 - \beta) \times \max(|a|, |b|)$ with $|\delta| \leq \varepsilon_m/2$.

$$(a \oplus b) - (a + b) \; = (2 - \beta) \times \delta \times \max(|a|, |b|)$$

$$|(a \oplus b) - (a + b)| \leq (2 - \beta) \times (1 + \delta') \times \tfrac{1}{2} \times (\varepsilon_m \otimes \max(|a|, |b|))$$
$$\text{with } |\delta'| \leq \varepsilon_m/2.$$

  Since $\varepsilon_m \leq \beta \leq 1$ and $-\varepsilon_m/2 \leq \delta' \leq \varepsilon_m/2$, we have

$$0 \leq 1/2 \times (2 - \beta) \times (1 + \delta') \leq (1 - \varepsilon_m/2) \times (1 + \varepsilon_m/2) = 1 - \varepsilon_m^2/4 \leq 1,$$

  and thus $|(a \oplus b) - (a + b)| \leq \varepsilon_m \otimes \max(|a|, |b|)$.
- If $b < a$ and $b = (1 - \beta) \times a$ with $0 < \beta < \varepsilon_m$, all possibilities for b are enumerated and checked. The study must distinguish between whether the rounding mode is to the nearest (and thus $\varepsilon_m = u$) or not (and then $\varepsilon_m = 2u$); a distinction is made between a being a power of 2 or not; in the case of another rounding mode, one must further distinguish between a^- being a power of 2 or not (in what follows, the exponent "$-$" on any x denotes the largest floating-point number strictly smaller than x).

  In each subcase, b can take only a small number of values (1, 2 or 3) and for each value of b the error $|(a + b) - (a \oplus b)|$ can be exactly expressed and bounded from above, as shown in the Table below.

	rounding to nearest (even)		other rounding modes		
	$\varepsilon_m = u$		$\varepsilon_m = 2u$		
	b	error		b	error
$a = 2^t$	a^-	$ua/2 \le \varepsilon_m \otimes a$		a^-	$ua/2 \le \varepsilon_m \otimes a$
	a^{--}	$0 \le \varepsilon_m \otimes a$		a^{--}	$0 \le \varepsilon_m \otimes a$
				a^{---}	$ua/2 \le \varepsilon_m \otimes a$
$2^t < a < 2^{t+1}$	a^-	$ua \le \varepsilon_m \otimes a$	$a^- = 2^t$	a^-	$ua^- \le \varepsilon_m \otimes a$
				a^{--}	$3ua^-/2 \le \varepsilon_m \otimes a$
				a^{---}	$0 \le \varepsilon_m \otimes a$
			$2^t < a^-$	a^-	$u2^t \le \varepsilon_m \otimes a$
				a^{--}	$0 \le \varepsilon_m \otimes a$

A.2 Proof of Lemma 2

Lemma 2.
If $\forall j \in \{1, \cdots, n\}$, $s_j \ge 0$ and if $(n-1) \times \varepsilon_M < 1$ then the error $E_n = S_n - \hat{S}_n$ is bounded as follows:

$$|E_n| \le (n-1) \times \varepsilon_M \times \left(\bigoplus_{j=1}^{n} s_j \right).$$

This implies that

$$S_n = \sum_{j=1}^{n} s_j \le (1 + (n-1)\varepsilon_M)\, \hat{S}_n = (1 + (n-1)\varepsilon_M) \left(\bigoplus_{j=1}^{n} s_j \right).$$

Proof of Lemma 2
Inequality (4.2) in [12] states that:

$$E_n = S_n - \hat{S}_n = \sum_{i=2}^{n} \delta_i \hat{T}_i$$

where $\hat{T}_i$ is the computed sum of i terms among $\{s_1, \ldots, s_n\}$ (depending on which order is used to sum up the s_j) and δ_i is the rounding error performed when summing one of the s_j to $\hat{T}_{i-1}$ to obtain $\hat{T}_i$. The δ_is satisfy $|\delta_i| \le \varepsilon_M$ and here we use the fact that, since the s_j are nonnegative, $\hat{T}_i \le \bigoplus_{j=1}^{n} s_j$.

Using these two inequalities to bound the right hand side, we get

$$|E_n| = |S_n - \hat{S}_n| \leq \varepsilon_M \times \sum_{i=2}^{n} \bigoplus_{j=1}^{n} s_j \leq (n-1)\varepsilon_M \bigoplus_{j=1}^{n} s_j.$$

Finally, this leads to

$$-(n-1)\varepsilon_M \hat{S}_n \leq S_n - \hat{S}_n \leq (n-1)\varepsilon_M \hat{S}_n$$

and using only the right inequality yields the desired bound for S_n.

A.3 Proof of Lemma 3

Let us multiply both sides of the inequality of Lemma 3 by 2 and use the fact that floating-point multiplications and divisions by 2 are exact.

Lemma 3.
We use here $\varepsilon_m = 2\varepsilon_M$. If the assumptions of Section 3.2 on Taylor models are satisfied and if $\forall j,\ s_j \geq 0$, then:

$$\sum_{j=1}^{n} \varepsilon_m \otimes s_j \leq e \otimes \varepsilon_m \otimes \bigoplus_{j=1}^{n} s_j.$$

It is assumed that no overflows occurs. Considering the case of an overflow would not be useful for our purpose: here the sum of the s_j is also computed, and the rounding error on this sum is of interest if the sum has a finite floating-point representation.

Proof of Lemma 3
Let us first prove that $\sum_{j=1}^{n} \varepsilon_m \otimes s_j \leq \varepsilon_m(1+\varepsilon_m/2)\sum_{j=1}^{n} s_j$: to get rid of the floating-point operations, which are neither associative nor distributive, let us go back to exact arithmetic; we have to multiply everything by $(1+\varepsilon_m/2)$ to be able to do it:

$$\varepsilon_m \otimes s_j \leq (1+\varepsilon_m/2)\,\varepsilon_m s_j.$$

The right hand side of the inequality to be proven is

$$e \otimes \varepsilon_m \otimes \bigoplus_{j=1}^{n} s_j$$

and, getting rid of the floating-point multiplication in the same manner: $a \otimes b \geq 1/(1+\frac{\varepsilon_m}{2})a \times b$,

$$e \otimes \varepsilon_m \otimes \bigoplus_{j=1}^{n} s_j \geq \frac{e\varepsilon_m}{\left(1+\frac{\varepsilon_m}{2}\right)^2} \bigoplus_{j=1}^{n} s_j.$$

The question is now whether the following inequality holds:

$$\varepsilon_m \left(1 + \frac{\varepsilon_m}{2}\right) \sum_{j=1}^{n} s_j \leq \frac{e\varepsilon_m}{\left(1 + \frac{\varepsilon_m}{2}\right)^2} \bigoplus_{j=1}^{n} s_j? \tag{1}$$

Using Lemma 2:

$$\sum_{j=1}^{n} s_j \leq \left(1 + n\frac{\varepsilon_m}{2}\right) \bigoplus_{j=1}^{n} s_j,$$

the left hand side part of inequality (1) can be upper bounded as follows:

$$\varepsilon_m \left(1 + \frac{\varepsilon_m}{2}\right) \sum_{j=1}^{n} s_j \leq \varepsilon_m \left(1 + \frac{\varepsilon_m}{2}\right) \left(1 + n\frac{\varepsilon_m}{2}\right) \bigoplus_{j=1}^{n} s_j.$$

Is the right part, in turn, upper bounded by the greatest part of inequality (1)? In other words, does the following inequality hold?

$$\left(1 + \frac{\varepsilon_m}{2}\right)^3 \left(1 + n\frac{\varepsilon_m}{2}\right) \leq e?$$

The answer is yes, it is given by assumption (3) of the definition of Taylor model arithmetic constants, since $n\frac{\varepsilon_m}{2}$ is bounded above by η.

References

[1] American National Standards Institute and Institute of Electrical and Electronic Engineers. IEEE standard for binary floating-point arithmetic. *ANSI/IEEE Standard, Std 754-1985,* New York, 1985.

[2] American National Standards Institute and Institute of Electrical and Electronic Engineers. IEEE standard for radix independent floating-point arithmetic. *ANSI/IEEE Standard, Std 854-1987,* New York, 1987.

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

[4] M. Berz. Forward algorithms for high orders and many variables. *Automatic Differentiation of Algorithms: Theory, Implementation and Application*, SIAM, 1991.

[5] M. Berz. *Modern Map Methods in Particle Beam Physics.* Academic Press, San Diego, 1999. Also available at http://bt.pa.msu.edu/pub.

[6] M. Berz and J. Hoefkens. Verified high-order inversion of functional dependencies and superconvergent interval Newton methods. *Reliable Computing*, 7(5):379–398, 2001.

[7] M. Berz and K. Makino. Verified integration of ODEs and flows using differential algebraic methods on high-order Taylor models. *Reliable Computing*, 4(4):361–369, 1998.

[8] M. Berz and K. Makino. New methods for high-dimensional verified quadrature. *Reliable Computing*, 5(1):13–22, 1999.

[9] W. J. Cody, J. T. Coonen, D. M. Gay, K. Hanson, D. Hough, W. Kahan, R. Karpinski, J. Palmer, F. N. Ris, and D. Stevenson. A proposed radix-and-word-length-independent standard for floating-point arithmetic. *IEEE MICRO*, 4(4):86–100, August 1984.

[10] T. J. Dekker. A floating-point technique for extending the available precision. *Numerische Mathematik*, 18:224-242, 1971.

[11] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–47, 1991.

[12] N. J. Higham, Accuracy and Stability of Numerical Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.

[13] J. Hoefkens and M. Berz. Verification of invertibility of complicated functions over large domains. *Reliable Computing*, 8,1:1–16, 2002.

[14] W. Kahan. Lecture notes on the status of IEEE-754. `http://http.cs.berkeley.edu/~wkahan/ieee754status/ieee754.ps`, 1996.

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

[16] K. Makino and M. Berz. Higher order verified inclusions of multidimensional systems by Taylor models. *Nonlinear Analysis*, 47:3503–3514, 2001.

[17] K. Makino and M. Berz. Methods for range bounding by Taylor models: LDB, QDB and related algorithms. Submitted.

[18] A. Neumaier. *Interval methods for systems of equations*. Cambridge University Press, 1990.

[19] D. M. Priest. *Algorithms for arbitrary precision floating-point arithmetic.* Proceedings of the 10th Symposium on Computer Arithmetic (P. Kornerup et D. Matula, eds.), (Grenoble, France), pp. 132-144, 1991.

[20] J. R. Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete and Computational Geometry*, vol. 18, pp. 305-363, 1997.

[21] P. H. Sterbenz. *Floating-point computation.* Prentice Hall, 1974.