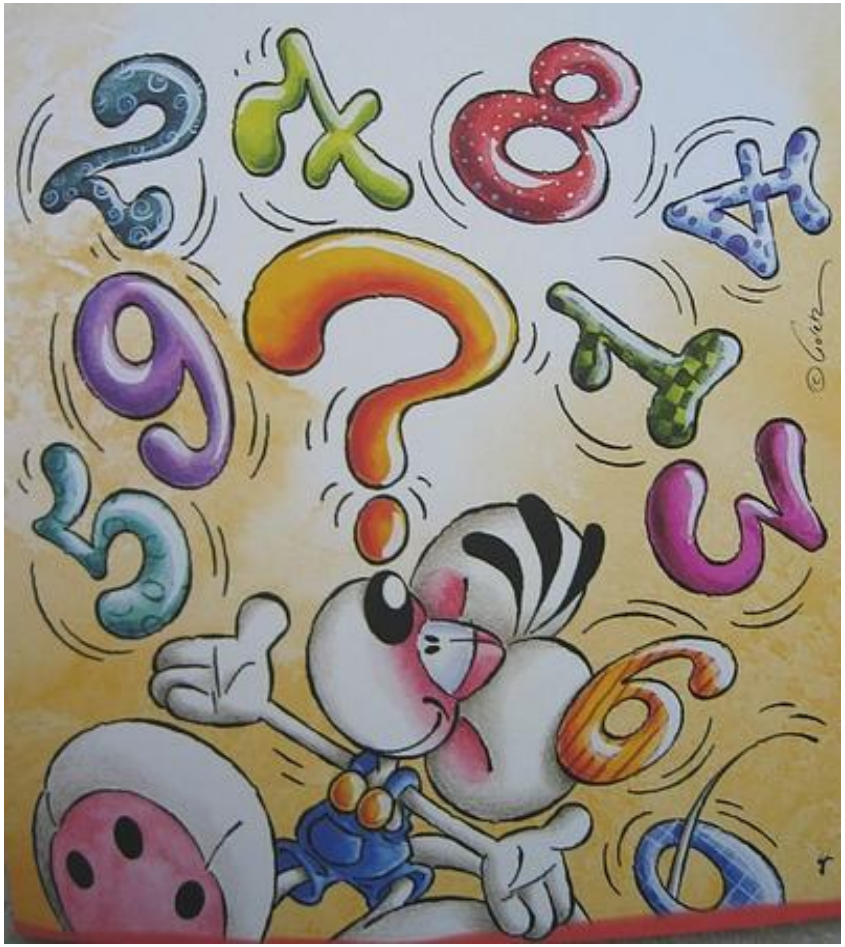

Diddl - Goletz

(cf. http://www.col-camus-soufflenheim.ac-strasbourg.fr/Page.php?IDP=137&IDD=0)

Implementing Taylor models with floating-point arithmetic

Nathalie Revol
INRIA, LIP-ENS Lyon, France
Nathalie.Revol@ens-lyon.fr

ICAM 2007, Zurich

Definition of a Taylor model for a function f

A function f can be represented by a Taylor model (p, J) on I where p is a polynomial and I, J are intervals if

$$\forall x \in I \subset D_f,\ f(x) \in p(x) + J.$$

(p, J) is a **Taylor model** for f.

Typically, p is the Taylor expansion of f and J encloses the truncation error of I, hence the name of **Taylor model**.

Assumption: interval $[-1, 1]$ as domain.

Implementation of Taylor models arithmetic

Cf. COSY.

Implementation of Taylor models
using floating-point arithmetic:

- coefficients of the polynomial and endpoints of the interval = floating-point numbers
- operations on Taylor models performed using floating-point arithmetic.

Advantage: benefit from the speed of floating-point arithmetic (implemented in hardware, thus very fast).

Implementation of Taylor models arithmetic

Roundoff errors must be taken into account.

Idea: for each computed coefficient,
bound the error on the computed coefficient by E
and add $[-E, E]$ to the interval remainder J.

J thus becomes a big "bin", enclosing every possible source of error (truncation error, roundoff error...).

Implementation of Taylor models arithmetic

Roundoff errors must be taken into account.

Example: addition of $(\sum_{i=0}^{n} a_i x^i, I)$ and $(\sum_{j=0}^{n} b_j x^j, J)$.

Using exact arithmetic:
$(\sum_{i=0}^{n} a_i x^i, I) + (\sum_{j=0}^{n} b_j x^j, J) = (c, K)$

where
$c = \sum_{k=0}^{n} c_k x^k$
with $c_k = a_k + b_k$
and
$K = I + J$.

Implementation of Taylor models arithmetic

Roundoff errors must be taken into account.

Example: addition of $(\sum_{i=0}^{n} a_i x^i, I)$ and $(\sum_{j=0}^{n} b_j x^j, J)$.

Using exact arithmetic:

$(\sum_{i=0}^{n} a_i x^i, I) + (\sum_{j=0}^{n} b_j x^j, J) = (c, K)$

where

$c = \sum_{k=0}^{n} c_k x^k$

with $c_k = a_k + b_k$

and

$K = I + J.$

Using floating-point arithmetic:

$(\sum_{i=0}^{n} a_i x^i, I) \oplus (\sum_{j=0}^{n} b_j x^j, J) = (\hat{c}, \hat{K})$

where

$\hat{c} = \sum_{k=0}^{n} \hat{c_k} x^k$

with $\hat{c_k} = a_k \oplus b_k$

and

$K = I + J.$

Implementation of Taylor models arithmetic

Elementary roundoff errors:

$$e_k = c_k - \hat{c}_k.$$

Let $E \geq \sum_{k=0}^{n} |e_k|$,
then when x varies in $[-1, 1]$,
the difference between $c(x)$ and $\hat{c}(x)$ lies in $[-E, E]$.

Roundoff errors are properly accounted for if

$$\hat{K} = K + [-E, E] = I + J + [-E, E].$$

Getting more accuracy

Techniques for reducing the overestimation on E.

Techniques for gaining even more accuracy, not only on E?

Outline

- introduction
  - definition of TM
  - TM and FP arithmetic
  - TM: getting more accuracy
- **Taylor models: implementation using floating-point arithmetic**
  - **addition of two Taylor models**
  - multiplication of two Taylor models
  - better multiplication of two Taylor models
- TM: getting more accuracy
  - increase the order of the TM
  - decrease the width of the input interval
  - increase the computing precision
  - change the polynomial basis
- conclusion

Addition of two Taylor models using FP arithmetic

Addition of $(\sum_{i=0}^{n} a_i x^i, I)$ **and** $(\sum_{j=0}^{n} b_j x^j, J)$ using FP arithmetic:

$$(\sum_{i=0}^{n} a_i x^i, I) \oplus (\sum_{j=0}^{n} b_j x^j, J) = (\hat{c}, \hat{K})$$

where

$$\hat{c} = \sum_{k=0}^{n} \hat{c_k} x^k$$

with $\hat{c_k} = a_k \oplus b_k$

$$e_k = (a_k + b_k) - (a_k \oplus b_k)$$
$$E = (1 \oplus n\varepsilon) \odot \bigoplus_{k=0}^{n} |e_k|$$
$$\hat{K} = I + J + [-E, E]$$

Addition of two Taylor models using FP arithmetic

$e_k = (a_k + b_k) - (a_k \oplus b_k)$

for $k = 0$ to n, e_k is computed using the TwoSum algorithm

more precisely, $(\hat{c_k}, e_k) = \text{TwoSum}\ (a_k, b_k)$

$E = (1 \oplus n\varepsilon) \odot \bigoplus_{k=0}^{n} |e_k|$

where ε is 1 ulp, $(1 + n\varepsilon)$ is computed exactly with FP arith.

and the factor $(1 + n\varepsilon)$ accounts for roundoff

when computing E

$\hat{K} = I + J + [-E, E]$

$\hat{K}$ is computed using interval arithmetic.

TwoSum algorithm (works with IEEE-754 FP arithmetic)

Algo TwoSum:

$$\begin{aligned} s &= x \oplus y \\ y' &= s \ominus x \\ x' &= s \ominus y' \\ \delta_y &= y \ominus y' \\ \delta_x &= x \ominus x' \\ r &= \delta_x \oplus \delta_y \end{aligned}$$

The mathematical equality holds: $s + r = x + y$
i.e. r is the roundoff error on the addition of x and y.

This algo avoids comparing x and y (a comparison can be costly).

Outline

- introduction
  - definition of TM
  - TM and FP arithmetic
  - TM: getting more accuracy
- **Taylor models: implementation using floating-point arithmetic**
  - addition of two Taylor models
  - **multiplication of two Taylor models**
  - better multiplication of two Taylor models
- TM: getting more accuracy
  - increase the order of the TM
  - decrease the width of the input interval
  - increase the computing precision
  - change the polynomial basis
- conclusion

Multiplication of two Taylor models using FP arith.

Multiplication of $(\sum_{i=0}^{n} a_i x^i, I)$ by $(\sum_{j=0}^{n} b_j x^j, J)$ using FP arithmetic:

$$(\sum_{i=0}^{n} a_i x^i, I) \cdot (\sum_{j=0}^{n} b_j x^j, J) = (\hat{c}, \hat{K})$$

where
$\hat{c} = \sum_{k=0}^{n} \hat{c}_k x^k$
with $\hat{c}_k = \bigoplus_{i+j=k} a_i \odot b_j$

$e_k = c_k - \hat{c}_k$
$E = (1 \oplus n\varepsilon) \odot \bigoplus_{k=0}^{n} |e_k|$
$\hat{K} = \ldots + I \times J + [-E, E]$

Multiplication of two Taylor models using FP arith.

$e_k = \sum_{i=0}^{k} a_i \cdot b_{k-1} - \bigoplus_{i=0}^{k} a_i \odot b_{k-i}$

for each operation ($\oplus$ or $\odot$),

the roundoff error is computed

using either a TwoSum or a TwoMult

finally, e_k is computed by summing (using $\oplus$) all these terms

and by multiplying by a security factor $((1 + 2k\varepsilon))$.

$E = (1 \oplus n\varepsilon) \odot \bigoplus_{k=0}^{n} |e_k|$

where the factor $(1 + n\varepsilon)$ accounts for roundoff

when computing E

$\hat{K} = \ldots + I \times J + [-E, E]$

$\hat{K}$ is computed using interval arithmetic.

TwoMult algorithm works with IEEE-754 arithmetic)

x, y two normal FP nbs, rounding to nearest, no overflow for $x \otimes y$.

Theorem: $r = (x \times y) - (x \otimes y)$ is representable.

Denote by $s = \lceil \frac{p}{2} \rceil$.

Algo TwoMult: (aka Dekker)

$$
\begin{aligned}
x' &= x \otimes (2^s \oplus 1) \\
x_h &= (x \ominus x') \oplus x' \\
x_l &= x \ominus x_h \\
&\text{ibid. for } y \\
r_h &= x \otimes y \\
r_l &= (((x_h \otimes y_h \ominus r_h) \\
&\qquad \oplus x_h \otimes y_l) \oplus x_l \otimes y_h) \\
&\qquad \oplus x_l \otimes y_l
\end{aligned}
$$

The mathematical equality holds: $r_h + r_l = x \times y$

i.e. r_l is the roundoff error on the multiplication of x and y.

17 operations: 7 $\otimes$ and 10 $\oplus, \ominus$.

Outline

- introduction
  - definition of TM
  - TM and FP arithmetic
  - TM: getting more accuracy
- **Taylor models: implementation using floating-point arithmetic**
  - addition of two Taylor models
  - multiplication of two Taylor models
  - **better multiplication of two Taylor models**
- TM: getting more accuracy
  - increase the order of the TM
  - decrease the width of the input interval
  - increase the computing precision
  - change the polynomial basis
- conclusion

Better multiplication of two Taylor models using FP arithmetic

Multiplication of $(\sum_{i=0}^{n} a_i x^i, I)$ by $(\sum_{j=0}^{n} b_j x^j, J)$
using FP arith.:

$$(\sum_{i=0}^{n} a_i x^i, I) \cdot (\sum_{j=0}^{n} b_j x^j, J) = (\hat{c}, \hat{K})$$

where
$\hat{c} = \sum_{k=0}^{n} \hat{c_k} x^k$
with $\hat{c_k} = \bigoplus_{i+j=k} a_i \odot b_j$
or equivalently $c_k = \{(a_i)^t \odot (b_{k-i})\}$ is a FP dot product

$e_k = c_k - \hat{c_k}$
$E = (1 \oplus n\varepsilon) \odot \bigoplus_{k=0}^{n} |e_k|$
$\hat{K} = \ldots + I \times J + [-E, E]$

Accurate dot product (Ogita, Rump and Oishi 2004)

$$
\begin{array}{l}
\texttt{function [res, err]} = \texttt{DotErr1}(x, y) \\
\quad [p, s] = \texttt{TwoMult}(x_1, y_1) \\
\quad \texttt{err} = |\texttt{s}| \\
\quad \texttt{for } i = 2 : n \\
\qquad [\texttt{h},\texttt{r}] = \texttt{TwoMult}(x_i, y_i) \\
\qquad [\texttt{p},\texttt{q}] = \texttt{TwoSum}(\texttt{p},\texttt{h}) \\
\qquad \texttt{s} = \texttt{s} \oplus (\ \texttt{q} \oplus \texttt{r}) \\
\qquad \texttt{err} = \texttt{err} \oplus (|\texttt{q}| \oplus |\texttt{r}|) \\
\quad \texttt{res} = \texttt{p} \oplus \texttt{s} \\
\quad \texttt{err} = \texttt{err} \oslash (1 - (n+2)\varepsilon)
\end{array}
$$

Better multiplication of $(\sum_{i=0}^{n} a_i x^i, I)$ by $(\sum_{j=0}^{n} b_j x^j, J)$

$$(\textstyle\sum_{i=0}^{n} a_i x^i, I) \cdot (\sum_{j=0}^{n} b_j x^j, J) = (\hat{c}, \hat{K})$$

where
$\hat{c} = \sum_{k=0}^{n} \hat{c_k} x^k$
with $(\hat{c_k}, e_k) = \text{DotErr1}\ ((a_i), (b_{k-i}))$

$E = (1 \oplus n\varepsilon) \odot \bigoplus_{k=0}^{n} |e_k|$
where the factor $(1 + n\varepsilon)$ accounts for roundoff
when computing E

$\hat{K} = \ldots + I \times J + [-E, E]$
$\hat{K}$ is computed using interval arithmetic.

Outline

- introduction
  - definition of TM
  - TM and FP arithmetic
  - TM: getting more accuracy
- Taylor models: implementation using floating-point arithmetic
  - addition of two Taylor models
  - multiplication of two Taylor models
  - better multiplication of two Taylor models
- **TM: getting more accuracy**
  - **increase the order of the TM**
  - **decrease the width of the input interval**
  - **increase the computing precision**
  - **change the polynomial basis**
- conclusion

Increase the order of the Taylor model

$T = (P, J)$ a Taylor model for $f : I \longrightarrow J$, P of degree d.
Theorem: (Taylor)

$$\forall x \in I,\ |f(x) - P(x)| \leq c.w(I)^d \text{ **local error**}.$$

c is unknown, but $c.w(I)^d \simeq w(J)$.

When to increase d?
When $w(J) >$ threshold.
How? Until $w(J) \leq$ threshold.

Problem: extra computational effort to work with a high order.
Number of coefficients (v variables and degree d) is $\begin{pmatrix} v+d \\ d \end{pmatrix}$.

Decrease the width of the domain I

Theorem: (Neumaier 1990)

$$w(f(I)) \leq \begin{cases} c_1.w(I) + c_2.\varepsilon_m \\ c_1.(w(I))^2 + c_2.\varepsilon_m \end{cases} \quad \textbf{global error.}$$

c_1 and c_2 are unknown, but $w(f(I)) \simeq w(P(I) + J)$
(if $c_2.\varepsilon_m$ is small enough).

When to decrease $w(I)$?

Trial and error: when $w(P(I) + J) >$ threshold, bisect I into $I_1 \cup I_2$,

OK if $\begin{cases} w(P_1(I_1) + J_1) \leq w(P(I) + J) \\ w(P_2(I_2) + J_2) \leq w(P(I) + J) \end{cases}$,

try something else otherwise.

Decrease the width of the domain I

Bisect until $\begin{cases} w(P_1(I_1) + J_1) \leq \text{ threshold} \\ w(P_2(I_2) + J_2) \leq \text{ threshold.} \end{cases}$

Problem: choice of the direction for bisection in multivariable case?

Problem: this yields the exponential complexity of interval algorithms (exponential in the number of variables)...

Increase the computing precision

Theorem: (Neumaier 1990)

$$w(f(I)) \leq \begin{cases} c_1.w(I) + c_2.\varepsilon_m \\ c_1.(w(I))^2 + c_2.\varepsilon_m \end{cases} \quad \textbf{global error.}$$

When to increase the computing precision?

When $w(P(I) + J) >$ threshold and bisection of I failed.
Then $c_2.\varepsilon_m$ must be large.
Double the computing precision (or multiply it by $\sqrt{2}$ or...).

Problem: multiple precision or arbitrary precision operations must be implemented.

Remark: implementing a high precision arithmetic

- implement double-double as coefficients (already done when we used TwoSum and TwoMult)
- implement triple-double as coefficients (work to get TripleSum and TripleMult)

Remark: implementing a high precision arithmetic

- implement double-double as coefficients (already done when we used TwoSum and TwoMult)
- implement triple-double as coefficients (work to get TripleSum and TripleMult)
- implement "multiple double", better known as expansions (cf. Shewchuk) as coefficients?
  Problem:
  expensive handling of coefficients (normalization etc.)
  develop intrinsics / elementary functions
  loss of time to develop and to execute

Remark: implementing a high precision arithmetic

- implement double-double as coefficients (already done when we used TwoSum and TwoMult)
- implement "multiple double", better known as expansions (cf. Shewchuk) as coefficients?
  advice: do not develop expansions (cf. Shewchuk)

Remark: implementing a high precision arithmetic

- implement double-double as coefficients (already done when we used TwoSum and TwoMult)
- implement "multiple double", better known as expansions (cf. Shewchuk) as coefficients?
  advice: do not develop expansions (cf. Shewchuk)
- use arbitrary precision arithmetic library for coefficients, such as MPFR

Remark: implementing a high precision arithmetic

- implement double-double as coefficients (already done when we used TwoSum and TwoMult)
- implement "multiple double", better known as expansions (cf. Shewchuk) as coefficients?
  advice: do not develop expansions (cf. Shewchuk)
- use arbitrary precision arithmetic library for coefficients, such as MPFR

Consider a tradeoff between the price to pay to develop the library, the overhead in memory and time for large precision, and the extra accuracy you get.

Remark: implementing a high precision arithmetic

- implement double-double as coefficients (already done when we used TwoSum and TwoMult)
- implement "multiple double", better known as expansions (cf. Shewchuk) as coefficients?
  advice: do not develop expansions (cf. Shewchuk)
- use arbitrary precision arithmetic library for coefficients, such as MPFR

Consider a tradeoff between the price to pay to develop the library, the overhead in memory and time for large precision, and the extra accuracy you get.

Advice: use double-double or otherwise switch to MPFR (but not hand-coded expansions: hard to code and unlikely to be efficient).

Change the polynomial basis?

For instance:
Bernstein polynomials for better numerical accuracy and use in bounding $P(I)$;

Chebychev which are better suited for approximation problems on a given domain (serve as starting point for Remes algorithm to determine minimax polynomial $\min_P \max_{x \in I} |f(x) - P(x)|$), thus smaller interval remainder?...

Problem: re-develop your library for Taylor models from scratch.

Conclusion and future work

More accurate implementation using FP arithmetic: to be done.

Conclusion and future work

More accurate implementation using FP arithmetic: to be done.

Self adaptation to get a prescribed accuracy: several possibilities, hints...

Conclusion and future work

More accurate implementation using FP arithmetic: to be done.

Self adaptation to get a prescribed accuracy: several possibilities, hints...
but more questions than answers!

Conclusion and future work

More accurate implementation using FP arithmetic: to be done.

Self adaptation to get a prescribed accuracy: several possibilities, hints...
but more questions than answers!

Ongoing work: Alexander Wittig implements higher precision in COSY.

I would be happy to receive your answers to my questions!