Hansen’s Generalized Interval Arithmetic Realized in C-XSC

Hassan El-Owny

Preprint

BUW-WRSWT 2006/2

Wissenschaftliches Rechnen/
Softwaretechnologie

Impressum

Herausgeber:	Prof. Dr. W. Krämer, Dr. W. Hofschuster Wissenschaftliches Rechnen/Softwaretechnologie Fachbereich C (Mathematik und Naturwissenschaften) Bergische Universität Wuppertal Gaußstr. 20 D-42097 Wuppertal

Internet-Zugriff

Die Berichte sind in elektronischer Form erhältlich über die World Wide Web Seiten

`http://www.math.uni-wuppertal.de/wrswt/literatur.html`

Autoren-Kontaktadresse

M.Sc. Hassan El-Owny
Bergische Universität Wuppertal
Gaußstr. 20
D-42097 Wuppertal

E-mail: `hassan.el-owny@math.uni-wuppertal.de`

Hansen's Generalized Interval Arithmetic Realized in C-XSC

Hasssan El-Owny
Department of Mathematics and Computer Science, University of Wuppertal
hassan.el-owny@math.uni-wuppertal.de

Abstract

Generalized interval arithmetic reduces the dependency problem in the interval arithmetic. We present a C-XSC [5] implementation of a generalized interval arithmetic, which has been proposed by Hansen [2] in 1975. The work is still in progress

Key Words: generalized intervals, Hansen form, dependency problem, C-XSC, interval arithmetic.
MSC (2000): 65G20, 65G30, 65G99, 65Y99.

1 Introduction

One of the main obstacles in the wide-spread use of interval methods in numerical computing is that the range estimates computed with ordinary interval arithmetic (IA) tend to be too large, especially in complicated expression or long iterative computations. This overestimation is mainly due to IA's underlying assumption that the (unknown) values of the arguments to primitive operations may vary independently over their given intervals. If this assumption is not valid - that is, if there are any mathematical constraints between those quantities - then not all combinations of values in the given intervals will be valid. In that case, the interval obtained by interval arithmetic may be much wider than the exact range of the result quantity. This problem is known as the "dependency problem" [4]. The goal of this work is to provide a free, open-source software for the generalized interval arithmetic (Hansen Arithmetic) which reduces this effect in the environment of C-XSC [8], [5], [6]. The paper is organized as follows. In Section 2 we review interval arithmetic and the dependency problem. In Section 3 we describe Hansen forms. In Section 4 we introduce generalized interval arithmetic (Hansen Arithmetic). In Section 5 two arithmetic operations (multiplication and division) are discussed in more details with examples of how Hansen arithmetic handles the dependency problem. The elementary functions (exp(), sin(), ln(),......) are considered in Section 6. In Section 7 we introduce the algorithmic description. In Section 8 C-XSC program code and some numerical examples are considered. Conclusion are given in the last section.
We use the following notations. $\mathbb{R}$, $\mathbb{R}^n$, $I\mathbb{R}$, $I\mathbb{R}^n$, to denote the set of real numbers, the set of real vectors with n components, the set of intervals and the set of interval vectors with n

components, respectively.
Some formulation of this paper are taken directly from [4], [2].

2 The dependency problem in interval arithmetic

In interval arithmetic, real numbers are bounded by intervals; and basic operations and functions are extended to operate on intervals. More precisely, a real number $x \in \mathbb{R}$ is bounded by an interval $[a, b]$, where a and b are floating-point numbers, with understanding that $a \leq x \leq b$.
Basic arithmetic operations can be extended to intervals:

$$\left.\begin{array}{l} [a,b]+[c,d]=[a+c,b+d], \\ [a,b]-[c,d]=[a-d,b-c], \\ [a,b]\cdot[c,d]=[\text{min}(ac,ad,bc,bd),\text{max}(ac,ad,bc,bd)], \\ \frac{[a,b]}{[c,d]}=[a,b]\cdot[\frac{1}{d},\frac{1}{c}],(0\notin[c,d]), \end{array}\right\} \tag{2.1}$$

with special care to round lower bounds downwards and upper bounds upwards.
Once we have interval formulas for all primitive operations and functions, we then automatically have formulas for all functions that can be obtained by combining these primitives. As a consequence, it is easy to extend any function $f : \Omega \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$ to an interval function F that operates on the interval boxes $[x] \subseteq \Omega$ and produces an interval $F([x]) \subset \mathbb{R}$ such that

$$F([x]) \supseteq f([x]) = \{f(x) : x \in [x]\}.$$

Such a function F is called an inclusion function for f. The interval estimate $F([x])$ often overestimates the exact range $f([x])$, that is, $F([x])$ is often strictly larger than the smallest interval containing $f([x])$.
Overestimation occurs frequently when f is given by an expression that contains multiple instances of one or more variables because the formulas for the primitive arithmetic operations given above assume that the operands are independent. When the operands are partially dependent on each other, not all combinations of values in the given intervals will be valid and the exact result interval will probably be smaller than the one produced by the formulas. This is the ”dependency problem”. The more complex the function f is, the worse the dependency problem becomes. In particular, for long iterative computations, we often see a rapid growth of the result interval at each stage.
As an extreme example of the dependency problem, take $f(x) = x - x$. Then of course $f([x]) = \{0\}$ for every interval $[x]$. However, the interval subtraction formula gives
$[a, b] - [a, b] = [a - b, b - a]$, whose diameter is twice the diameter of $[a, b]$, instead of being zero.
For a less extreme example, take $f(x) = (10 + x) \cdot (10 - x)$ for $x \in [x] = [-1, 1]$. Using the basic formulas (2.1) , we get

$$\begin{array}{c} 10+[x]=[9,11], \\ 10-[x]=[9,11], \\ (10+[x])\cdot(10-[x])=[81,121]. \end{array}$$

The interval formulas give an interval whose diameter is 40, whereas the exact interval result $f([x]) = [99, 100]$ has a diameter of only 1. Note that when one operand in the product $(10 + x) \cdot (10 - x)$ is at the maximum value 11, the other must be at the minimum value 9; the combination $9 \cdot 9$ and $11 \cdot 11$, which gave the extreme values of $F([x])$, never occur.
A simple remedy for this example is to rewrite $(10 + x) \cdot (10 - x) = 100 - x^2$, which has only one occurrence of the variable x. An interval computation of this new expression will give the exact result. Unfortunately, this remedy is often impossible to apply in practice.
Several other methods have been proposed to attack the dependency problem. The main class of methods is known as centered forms [9], in several incarnations and generalizations, such as mean-value form [9], [10] and slopes [7].

3 Representation of a Generalized Interval (Hansen Interval)

For our purposes, it is convenient to represent an interval $[x] = [a, b]$ in the form $[x] = c + [u]$ where c is the mid-point of $[x]$, $[u] = [-r, r]$ is symmetrical interval, and r is the radius of $[x]$. Thus an arbitrary point $x \in [x]$ may be expressed as $x = c + \zeta$ where $\zeta \in [-r, r]$ and $r \geq 0$.

Definition 3.1. *[2] A generalized interval $[\hat{x}]$ is given by*

$$[\hat{x}] = [c^x] + \sum_{i=1}^{n} \zeta_i [v_i^x] \tag{3.1}$$

where $[c^x]$ and $[v_i^x]$ $(i = 1, 2, \cdots, n)$ are (computed numerical) intervals and $\zeta_i \in [-r_i, r_i]$.

From the above definition, it is clear that
$\hat{x} \in [\hat{x}] \Longleftrightarrow \hat{x} = c^x + \sum_{i=1}^{n} \zeta_i v_i^x$ with $c^x \in [c^x]$, $v_i^x \in [v_i^x]$ and $-r_i \leq \zeta_i \leq r_i$.
When we reduce the generalized interval in (3.1) to an ordinary interval, we obtain

$$\begin{aligned} \text{reduce}([\hat{x}]) &= \text{reduce}([c^x] + \sum_{i=1}^{n} [-r_i, r_i][v_i^x]) \\ &:= [c^x] + [-1, 1] \sum_{i=1}^{n} r_i v_i^x \end{aligned}$$

where $v_i^x := |[v_i^x]|$. In general, the absolute value of an interval $[x] = [a, b]$ is defined as $|[x]|$:=max$(|a|, |b|)$. Conversely, any ordinary interval can be represented by a generalized interval. The ordinary interval $[x] = [a, b]$ can be represented as the generalized interval $[\hat{x}] = [c^x] + \zeta_1 [v_1^x]$, where $[c^x] := [(b+a)/2, (b+a)/2]$, $\zeta_1 \in [-(b-a)/2, (b-a)/2]$ and $[v_1^x] := [1, 1]$.
In general, if we have an interval vector $[x] := ([x_1], \cdots, [x_n])^{\mathrm{T}} \in I\mathbb{R}^n$, the j-th interval $[x_j]$ can be represented with the generalized interval form

$$\begin{aligned} [\hat{x}_j] &= [c^{x_j}] + [0, 0]\zeta_1 + \cdots + [0, 0]\zeta_{j-1} + [1, 1]\zeta_j + [0, 0]\zeta_{j+1} + \cdots + [0, 0]\zeta_n \\ &= [c^{x_j}] + [1, 1]\zeta_j. \end{aligned}$$

4 Generalized Interval Arithmetic (Hansen Arithmetic)

Assume two generalized intervals $[\hat{x}]$ and $[\hat{y}]$ are expressed as

$$[\hat{x}] = [c^x] + \sum_{i=1}^{n} \zeta_i [v_i^x] \tag{4.1}$$

and

$$[\hat{y}] = [c^y] + \sum_{i=1}^{n} \zeta_i [v_i^y], \tag{4.2}$$

respectively.
We now consider the four arithmetic operations applied to these intervals.
Addition or subtraction:
The sum (difference) of $[\hat{x}]$ and $[\hat{y}]$ is another generalized interval $[\hat{z}] = [c^z] + \sum_{i=1}^{n} \zeta_i [v_i^z]$.
It holds

$$[\hat{x}] \pm [\hat{y}] = [c^x] \pm [c^y] + \sum_{i=1}^{n} \zeta_i ([v_i^x] \pm [v_i^y]). \tag{4.3}$$

Thus we have to define

$$[c^z] := [c^x] \pm [c^y], \quad [v_i^z] := [v_i^x] \pm [v_i^y], \quad (i = 1, 2, \cdots, n). \tag{4.4}$$

Lemma 4.1. $\hat{x} \in [\hat{x}]$ *and* $\hat{y} \in [\hat{y}] \Longleftrightarrow \hat{x} \pm \hat{y} = c^x \pm c^y + \sum_{i=1}^{n} \zeta_i (v_i^x \pm v_i^y) \in [\hat{z}]$.

Proof: (Addition)
($\Longrightarrow$) $\hat{x} \in [\hat{x}]$ and $\hat{y} \in [\hat{y}]$ $\xrightarrow{\text{Def. 1}}$
$\hat{x} = c^x + \sum_{i=1}^{n} \zeta_i v_i^x$ with $c^x \in [c^x]$, $v_i^x \in [v_i^x]$ and $-r_i \le \zeta_i \le r_i$
$\hat{y} = c^y + \sum_{i=1}^{n} \zeta_i v_i^y$ with $c^y \in [c^y]$, $v_i^y \in [v_i^y]$ and $-r_i \le \zeta_i \le r_i$, $(i = 1, 2, \cdots, n)$
hence,

$$\begin{aligned}
\hat{x} + \hat{y} &= c^x + \sum_{i=1}^{n} \zeta_i v_i^x + c^y + \sum_{i=1}^{n} \zeta_i v_i^y \\
&= c^x + c^y + \sum_{i=1}^{n} \zeta_i (v_i^x + v_i^y) \\
&\in [c^x] + [c^y] + \sum_{i=1}^{n} \zeta_i ([v_i^x] + [v_i^x]) \\
&= [\hat{x}] + [\hat{y}] = [\hat{z}].
\end{aligned}$$

($\Longleftarrow$)
$\hat{z} \in [\hat{z}] \quad \xrightarrow{\text{Def. 1, } (4.3) \text{ and } (4.4)}$

$$\begin{aligned}
\hat{z} &= c^x + c^y + \sum_{i=1}^{n} \zeta_i (v_i^x + v_i^y) \\
&= c^x + c^y + \sum_{i=1}^{n} \zeta_i v_i^x + \sum_{i=1}^{n} \zeta_i v_i^y \\
&= \underbrace{c^x + \sum_{i=1}^{n} \zeta_i v_i^x}_{\in[\hat{x}]} + \underbrace{c^y + \sum_{i=1}^{n} \zeta_i v_i^y}_{\in[\hat{y}]} .
\end{aligned}$$

The subtraction is proved in a similar manner.

Multiplication:
To obtain a rule for multiplication of two generalized intervals, note that

$$\begin{aligned}
[\hat{x}] \cdot [\hat{y}] &= \{\hat{x} \cdot \hat{y} | \ \hat{x} \in [\hat{x}], \ \hat{y} \in [\hat{y}]\} \\
&\subseteq [c^x] \cdot [c^y] + \sum_{i=1}^{n} \zeta_i ([c^x][v_i^y] + [c^y][v_i^x]) + \underbrace{\sum_{i=1}^{n} \sum_{j=1}^{n} \zeta_i \zeta_j [v_i^x][v_j^y]}_{(*)} .
\end{aligned}$$

We shall choose to retain only linear terms in ζ_i ($i = 1, 2, \cdots, n$) although higher order terms could be kept.
Note that in ($*$) the terms for $i = j$ involve ζ_i^2 which can be replaced by $[-r_i, r_i]^2 = [0, r_i^2]$. For $i \neq j$, we cannot take advantage of the special result that the square of an interval must be positive. We replace $\zeta_i \zeta_j$ by $\zeta_i [-r_j, r_j]$ since $\zeta_j \in [-r_j, r_j]$. Then

$$\begin{aligned}
[\hat{x}] \cdot [\hat{y}] &\subseteq [c^x] \cdot [c^y] + \sum_{i=1}^{n} \zeta_i ([c^x][v_i^y] + [c^y][v_i^x]) + \sum_{i=1}^{n} \sum_{j=1}^{n} \zeta_i \zeta_j [v_i^x][v_j^y] \\
&\subseteq [c^x] \cdot [c^y] + \sum_{i=1}^{n} \zeta_i ([c^x][v_i^y] + [c^y][v_i^x]) \\
&\quad + \sum_{i=1}^{n} [0, r_i^2][v_i^x][v_i^y] + \sum_{i=1}^{n} \zeta_i [v_i^x] \sum_{\substack{j=1 \\ j \neq i}}^{n} [-r_j, r_j][v_j^y] \\
&=: [\hat{z}] = [c^z] + \sum_{i=1}^{n} \zeta_i [v_i^z] \qquad (4.5)
\end{aligned}$$

where

$$[c^z] := [c^x][c^y] + \sum_{i=1}^{n} [0, r_i^2][v_i^x][v_i^y] \qquad (4.6)$$

and

$$\begin{aligned}[v_i^z] &:= [c^x][v_i^y] + [c^y][v_i^x] + [v_i^x]\sum_{\substack{j=1\\j\neq i}}^{n}[-r_j, r_j][v_j^y] \\ &= [c^x][v_i^y] + [c^y][v_i^x] + [-1,1]v_i^x\sum_{\substack{j=1\\j\neq i}}^{n} r_j v_j^y \qquad (4.7)\end{aligned}$$

where as before $v_i^x := |[v_i^x]|$ and $v_i^y := |[v_i^y]|$. Thus we define the product of two generalized intervals $[\hat{x}]$ and $[\hat{y}]$ to be that given by (4.5) with $[c^z]$ defined by (4.6) and $[v_i^z]$ defined by (4.7).

Lemma 4.2. $\hat{x} \in [\hat{x}]$ *and* $\hat{y} \in [\hat{y}] \Longrightarrow$
$\hat{x}\cdot\hat{y} = c^x c^y + \sum_{i=1}^n \zeta_i^2 v_i^x v_i^y + \sum_{i=1}^n \zeta_i(c^x v_i^y + c^y v_i^x) + \sum_{i=1}^n \zeta_i v_i^x \sum_{\substack{j=1\\j\neq i}}^n \zeta_j v_j^y \in [\hat{z}]$.

Proof:
$\hat{x} \in [\hat{x}]$ and $\hat{y} \in [\hat{y}]$ $\xrightarrow{\text{Def. 1}}$
$\hat{x} = c^x + \sum_{i=1}^n \zeta_i v_i^x$ with $c^x \in [c^x]$, $v_i^x \in [v_i^x]$ and $-r_i \le \zeta_i \le r_i$,
$\hat{y} = c^y + \sum_{i=1}^n \zeta_i v_i^y$ with $c^y \in [c^y]$, $v_i^y \in [v_i^y]$ and $-r_i \le \zeta_i \le r_i$, $(i = 1, 2, \cdots, n)$,
hence

$$\begin{aligned}\hat{x}\cdot\hat{y} &= (c^x + \sum_{i=1}^n \zeta_i v_i^x)\cdot(c^y + \sum_{i=1}^n \zeta_i v_i^y) \\ &= c^x c^y + c^x\sum_{i=1}^n \zeta_i v_i^y + c^y\sum_{i=1}^n \zeta_i v_i^x + \sum_{i=1}^n \zeta_i v_i^x \sum_{i=1}^n \zeta_i v_i^y \\ &= c^x c^y + \sum_{i=1}^n \zeta_i(c^x v_i^y + c^y v_i^x) + \sum_{i=1}^n \zeta_i v_i^x \zeta_i v_i^y + \sum_{i=1}^n \zeta_i v_i^x \sum_{\substack{j=1\\j\neq i}}^n \zeta_j v_j^y \\ &= c^x c^y + \sum_{i=1}^n \zeta_i^2 v_i^x v_i^y + \sum_{i=1}^n \zeta_i(c^x v_i^y + c^y v_i^x) + \sum_{i=1}^n \zeta_i v_i^x \sum_{\substack{j=1\\j\neq i}}^n \zeta_j v_j^y \\ &= c^x c^y + \sum_{i=1}^n \zeta_i^2 v_i^x v_i^y + \sum_{i=1}^n \zeta_i(c^x v_i^y + c^y v_i^x + v_i^x \sum_{\substack{j=1\\j\neq i}}^n \zeta_j v_j^y), \quad \text{i.e.} \\ \hat{x}\cdot\hat{y} &\in \{c^x c^y + \sum_{i=1}^n \zeta_i^2 v_i^x v_i^y + \sum_{i=1}^n \zeta_i(c^x v_i^y + c^y v_i^x + v_i^x \sum_{\substack{j=1\\j\neq i}}^n \zeta_j v_j^y) \text{ with } c^x \in [c^x], \\ &\quad v_i^x \in [v_i^x],\ c^y \in [c^y],\ v_i^y \in [v_i^y] \text{ and } -r_i \le \zeta_i \le r_i\} \\ &\subseteq [c^x][c^y] + \sum_{i=1}^n \zeta_i^2 [v_i^x][v_i^y] + \sum_{i=1}^n \zeta_i([c^x][v_i^y] + [c^y][v_i^x] + [v_i^x]\sum_{\substack{j=1\\j\neq i}}^n \zeta_j [v_j^y]). \\ &=: [\hat{z}] = [c^z] + \sum_{i=1}^n \zeta_i [v_i^z].\end{aligned}$$

Example 4.1. *Consider the expression* $f = x \cdot y$*, with* $x \in [1,2]$ *and* $y \in [3,4]$.
Ordinary interval computation give $F = [1,2] \cdot [3,4] = [3,8]$.
Using Hansen forms and using (4.6) and (4.7) gives
$F_{\mathit{Hansen}} = [5.25, 5.25] + [3,4]\zeta_1 + [1.5, 1.5]\zeta_2$ *which reduces to* $[2.5, 8]$.
This means:
$\hat{x} \in [\hat{x}] = [1.5, 1.5] + [1,1]\zeta_1 + [0,0]\zeta_2$ *and* $\hat{y} \in [\hat{y}] = [3.5, 3.5] + [0,0]\zeta_1 + [1,1]\zeta_2$*, where* $\zeta_1 \in [-0.5, 0.5]$ *and* $\zeta_2 \in [-0.5, 0.5] \longrightarrow \hat{x} \cdot \hat{y} \in [\hat{x}] \cdot [\hat{y}] = [2.5, 8]$.
But the converse is not correct, this means if we choose the point $2.75 \in [2.5, 8]$*, then we see that there is no* $\hat{x} \in [\hat{x}]$ *and* $\hat{y} \in [\hat{y}]$ *such that* $\hat{x} \cdot \hat{y} = 2.75$.
The reduced Hansen form overestimates the (ordinary) interval result.

Division:
Division of two generalized intervals can also be done, Note that

$$\{\frac{\hat{x}}{\hat{y}} | \ \hat{x} \in [\hat{x}], \ \ \hat{y} \in [\hat{y}]\} \quad \subseteq \quad [c^z] + \sum_{i=1}^{n} \zeta_i [v_i^z] = [\hat{z}] \tag{4.8}$$

with

$$[c^z] := \frac{[c^x]}{[c^y]} \tag{4.9}$$

and

$$[v_i^z] := \frac{[c^y][v_i^x] - [c^x][v_i^y]}{[c^y]([c^y] + [-1,1]\sum_{j=1}^{n} r_j v_j^y)} \tag{4.10}$$

The denominator in (4.10) should not be written as

$$[c^y]^2 + [c^y][-1,1]\sum_{j=1}^{n} r_j v_j^y$$

since this form will always yield a wider interval unless the width of $[c^y]$ is zero. No advantage can be gained by using the special definition of the square of an interval to compute $[c^y]^2$ since $0 \notin [c^y]$. For $0 \in [c^y]$, we have $0 \in [\hat{y}]$ and we cannot perform the division.

Lemma 4.3. $\hat{x} \in [\hat{x}]$ *and* $\hat{y} \in [\hat{y}]$ *with* $0 \notin [\hat{y}] \Longrightarrow$

$$\frac{\hat{x}}{\hat{y}} \quad = \quad \frac{c^x}{c^y} + \sum_{i=1}^{n} \zeta_i \frac{c^y v_i^x - c^x v_i^y}{c^y(c^y + \sum_{j=1}^{n} \zeta_j v_j^y)} \in [\hat{z}] = [c^z] + \sum_{i=1}^{n} \zeta_i [v_i^z]$$

Proof:
$\hat{x} \in [\hat{x}]$ and $\hat{y} \in [\hat{y}]$ $\xrightarrow{\text{Def. 1}}$

$\hat{x} = c^x + \sum_{i=1}^n \zeta_i v_i^x$ with $c^x \in [c^x]$, $v_i^x \in [v_i^x]$ and $-r_i \leq \zeta_i \leq r_i$,
$0 \neq \hat{y} = c^y + \sum_{i=1}^n \zeta_i v_i^y$ with $c^y \in [c^y]$, $v_i^y \in [v_i^y]$ and $-r_i \leq \zeta_i \leq r_i$, $(i = 1, 2, \cdots, n)$,

$$\begin{aligned}
\frac{\hat{x}}{\hat{y}} &= \frac{c^x + \sum_{i=1}^n \zeta_i v_i^x}{c^y + \sum_{j=1}^n \zeta_j v_j^y} \\
&= \frac{c^y(c^x + \sum_{i=1}^n \zeta_i v_i^x)}{c^y(c^y + \sum_{j=1}^n \zeta_j v_j^y)} \\
&= \frac{c^x(c^y + \sum_{j=1}^n \zeta_j v_j^y) + \sum_{i=1}^n \zeta_i (c^y v_i^x - c^x v_i^y)}{c^y(c^y + \sum_{j=1}^n \zeta_j v_j^y)} \\
&= \frac{c^x}{c^y} + \frac{\sum_{i=1}^n \zeta_i (c^y v_i^x - c^x v_i^y)}{c^y(c^y + \sum_{j=1}^n \zeta_j v_j^y)} \\
&= \frac{c^x}{c^y} + \sum_{i=1}^n \zeta_i \frac{c^y v_i^x - c^x v_i^y}{c^y(c^y + \sum_{j=1}^n \zeta_j v_j^y)} \\
&\in \{\frac{c^x}{c^y} + \sum_{i=1}^n \zeta_i \frac{c^y v_i^x - c^x v_i^y}{c^y(c^y + \sum_{j=1}^n \zeta_j v_j^y)} \text{ with } c^x \in [c^x],\ v_i^x \in [v_i^x],\ c^y \in [c^y], \\
&\quad v_i^y \in [v_i^y] \text{ and } - r_i \leq \zeta_i \leq r_i\} \\
&\subseteq \frac{[c^x]}{[c^y]} + \sum_{i=1}^n \zeta_i \frac{([c^y][v_i^x] - [c^x][v_i^y])}{[c^y]([c^y] + \sum_{j=1}^n \zeta_j [v_j^y])} \xrightarrow{(4.9) \text{ and } (4.10)} \\
&=: [\hat{z}] = [c^z] + \sum_{i=1}^n \zeta_i [v_i^z].
\end{aligned}$$

Example 4.2. *Consider the expression* $f = x/y$, *with* $x \in [1, 2]$ *and* $y \in [3, 4]$.
Ordinary interval computation give $F = [1, 2]/[3, 4] = [0.25, 0.66667]$.
Using Hansen forms and using (4.9) and (4.10) gives
$F_{\text{Hansen}} = [0.428571, 0.428572] + [0.25, 0.33334]\zeta_1 + [-0.142858, 0.107142]\zeta_2$ *which reduces to* $[0.190476, 0.666667]$.
This means:
$\hat{x} \in [\hat{x}] = [1.5, 1.5] + [1, 1]\zeta_1 + [0, 0]\zeta_2$ *and* $\hat{y} \in [\hat{y}] = [3.5, 3.5] + [0, 0]\zeta_1 + [1, 1]\zeta_2$, *where* $\zeta_1 \in [-0.5, 0.5]$ *and* $\zeta_2 \in [-0.5, 0.5] \longrightarrow \hat{x}/\hat{y} \in [\hat{x}]/[\hat{y}] = [0.190476, 0.666667]$.
But the converse is not correct, this means if we choose the point $0.2 \in [0.19048, 0.66667]$, *then we see that there is no* $\hat{x} \in [\hat{x}]$ *and* $\hat{y} \in [\hat{y}]$ *such that* $\hat{x}/\hat{y} = 0.2$.
The reduced Hansen form overestimates the (ordinary) interval result.

In the next section, we shall consider the multiplication and division for generalized intervals (Hansen Arithmetic) in more detail and present some examples.

5 Multiplication and Division

5.1 Multiplication:

We first note that to obtain the square of a generalized interval, we can use a special definition as in the case for ordinary interval arithmetic. For $[\hat{x}] = [\hat{y}]$, equation (4.6) becomes

$$\begin{aligned} [c^z] &:= [c^x]^2 + \sum_{i=1}^{n} [0, r_i^2][v_i^x]^2 \\ &= [c^x]^2 + \sum_{i=1}^{n} [0, r_i^2](v_i^x)^2. \end{aligned} \tag{5.1}$$

The term $[c^x]^2$ should be computed using the special definition for the square of an interval. Equation (4.7) becomes

$$[v_i^z] = 2[c^x][v_i^x] + [-1, 1]v_i^x \sum_{\substack{j=1 \\ j\neq i}}^{n} r_j v_j^y. \tag{5.2}$$

Consider the square of an interval $[\hat{x}] = c^x + \zeta$ with $\zeta \in [-r, r]$. In this case, c^x is a real number and (5.1) and (5.2) yields

$$[\hat{x}]^2 = (c^x)^2 + [0, r^2] + 2\zeta c^x.$$

Reducing to an interval,

$$\begin{aligned} [\hat{x}]^2 &= [(c^x)^2 - 2r|c^x|, (c^x)^2 + r^2 + 2r|c^x|] \\ &= [(c^x)^2 - 2r|c^x|, (|c^x| + r)^2]. \end{aligned}$$

The right endpoint is correct. However, the left endpoint should be

$$\begin{array}{cl} 0 & \text{if} \quad 0 \in [\hat{x}], \\ (|c^x| - r)^2 & \text{if} \quad 0 \notin [\hat{x}]. \end{array}$$

Hence we will obtain an incorrect left endpoint for our result unless $c^x = 0$.

The magnitude of the error is

$$\begin{array}{cl} |(c^x)^2 - 2r|c^x|| & \text{if} \quad 0 \in [\hat{x}], \\ r^2 & \text{if} \quad 0 \notin [\hat{x}]. \end{array}$$

Thus if r is small, the error is small. In fact, the error is O(r^2) since in the case $0 \in [\hat{x}]$, we must have $|c^x| \leq r$. If r is much greater than 1, the error can be unacceptably large.

Example 5.1. *Consider $F = [x]^2$, with $[x] = [-0.2, 0.3]$.*
Using ordinary interval arithmetic gives $F = [0, 0.09]$
Using generalized interval arithmetic, where $[\hat{x}] = [0.05, 0.05] + [1, 1]\zeta$, $\zeta \in [-0.25, 0.25]$ gives

$$F_{\textit{Hansen}} = [-0.0025, 0.065] + [0.1, 0.1]\zeta$$

which reduces to $[-0.0225, 0.09]$. *The reduced Hansen form overestimates the (ordinary) interval result.*

However, let $F = [x]^2 - [x]^2$, *with* $[x] = [-0.2, 0.3]$.
Using ordinary interval arithmetic gives $F = [-0.09, 0.09]$
Using generalized interval arithmetic gives

$$F_{\textit{Hansen}} = [-0.0625, 0.0625] + [0, 0]\zeta$$

which reduces to $[-0.0625, 0.0625]$. *This is an improvement over the ordinary interval arithmetic result* $F = [-0.09, 0.09]$.

As a final note on multiplication we consider multiplication of a generalized interval by a real number or by an interval which we choose not to be represented as a generalized interval. Let B be such a number or interval and

$$[\hat{x}] = [c^x] + \sum_{i=1}^{n} \zeta_i [v_i^x].$$

Then

$$\{B \cdot \hat{x} |\ \hat{x} \in [\hat{x}]\} \subseteq B \cdot [\hat{x}] := [\bar{c}^x] + \sum_{i=1}^{n} \zeta_i [\bar{v}_i^x],$$

where

$$[\bar{c}^x] := B \cdot [c^x], \quad [\bar{v}_i^x] := B \cdot [v_i^x].$$

5.2 Division:

For an interval $[\hat{x}] = c^x + \zeta v^x$ depending on only one datum interval, if the quantities c and v are real numbers, then from the forms (4.9) and (4.10) we will find $[\hat{x}]/[\hat{x}] = 1$. This will never be true for interval arithmetic if the width of $[x]$ is nonzero.

In general, a single division in generalized interval arithmetic introduces errors which are of second order in the interval widths. We now show this for an interval $[\hat{x}] = c^x + \zeta v^x$ where again $c^x > 0$ and $v^x > 0$ are real numbers and $\zeta \in [-r, r]$. Consider $[x'] = 1/[\hat{x}]$.

From (4.9) and (4.10),

$$[x'] = \frac{1}{c^x} - \frac{v^x}{c^x(c^x + [-1, 1]rv^x)}\zeta$$

which reduces to

$$[x'] = \left[\frac{1}{c^x} - \frac{rv^x}{c^x(c^x - rv^x)}, \frac{1}{c^x} + \frac{rv^x}{c^x(c^x - rv^x)}\right].$$

The width of this interval is

$$w' = \frac{2rv^x}{c^x(c^x - rv^x)}.$$

The correct result is

$$[\frac{1}{c^x + rv^x}, \frac{1}{c^x - rv^x}],$$

which has width

$$w = \frac{2rv^x}{(c^x)^2 - r^2(v^x)^2}.$$

The error of the width is of amount

$$w' - w = \frac{2r^2(v^x)^2}{c^x((c^x)^2 - r^2(v^x)^2)}$$

which is of second order in r.

Example 5.2. *[9] Consider*

$$F = \frac{[x_1] + [x_2]}{[x_1] - [x_2]},$$

with $[x_1] = [1, 2]$ *and* $[x_2] = [5, 10]$.
Using (4.4) gives $[\hat{x}_1] + [\hat{x}_2] = 9 + \zeta_1 + \zeta_2$ *and* $[\hat{x}_1] - [\hat{x}_2] = -6 + \zeta_1 - \zeta_2$*, where* $[\hat{x}_1] = 1.5 + \zeta_1$, $[\hat{x}_2] = 7.5 + \zeta_2$ *with* $\zeta_1 \in [-0.5, 0.5]$ *and* $\zeta_2 \in [-2.5, 2.5]$. *Using (4.9) and (4.10) we get*

$$F_{Hansen} = -\frac{9}{6} + \zeta_1[-\frac{5}{6}, -\frac{5}{18}] + \zeta_2[\frac{1}{18}, \frac{1}{6}],$$

which reduces to $[-\frac{7}{3}, -\frac{2}{3}] \subset [-2.334, -0.666]$.
This is the same result obtained by Moore ([9]) using the centered form with interval arithmetic and better than the result $[-\frac{67}{18}, \frac{13}{18}] \subset [-3.723, 0.7223]$ *he obtained using the mean value theorem. Incidentally, his latter result can be sharpened to* $[-\frac{1073}{392}, -\frac{103}{392}] \subset [-2.738, -0.2627]$ *which obtained by Hansen ([3]).*
Direct use of interval arithmetic yields $[-4, -\frac{2}{3}]$.
We obtain an exact result using interval arithmetic by rewriting F *as* $F = 1 + 2/([x_1]/[x_2] - 1)$ *since each variable occurs only once. We find* $F = [-\frac{7}{3}, -\frac{11}{9}] \subset [-2.334, -1.222]$. *Thus the result using generalized interval arithmetic has a sharp left endpoint but not a sharp right endpoint.*

Example 5.3. *Let* $[x] = [0.001, 0.003]$. *Evaluate*

$$F = \frac{1 + [x] + [x]^2}{1 + [x] + 2[x]^2}.$$

Using generalized interval arithmetic, where

$$[\hat{x}] = [0.002, 0.002] + [1, 1]\zeta, \ \ \zeta \in [-0.001, 0.001]$$

we obtain

$$1 + [\hat{x}] + [\hat{x}]^2 = [1.002003, 1.002006] + [1.003999, 1.00400]\zeta,$$

and

$$1 + [\hat{x}] + [\hat{x}]^2 = [1.002007, 1.002011] + [1.007999, 1.00800]\zeta,$$

with $\zeta \in [-0.001, 0.001]$*, so that*

$$F_{\textit{Hansen}} = [0.999994, 0.999998] + [-0.003993, -0.003981]\zeta$$

which reduces to $[0.999990, 1.000001]$.
In interval arithmetic, we obtain the result $[0.997989, 1.002005]$. *We obtain an exact result using interval arithmetic by rewriting* F *as* $F = 1 - 1/(([x]+0.5)^2 + 1.75)$ *since each variable occurs only once. We find* $F = [0.999991, 0.999999]$.

6 Elementary Functions

Elementary functions can be evaluated in generalized interval arithmetic by making use of Taylor series (only the first order).
We first describe the Taylor-form of order 1 for a real function f of one variable.
Let $S \subseteq \mathbb{R}$ be open, $x^*, c \in S$, and let $f : S \subseteq \mathbb{R} \longrightarrow \mathbb{R}$ be an once differentiable function on the open set S. Then, there exists $\eta = x^* + \theta(x^* - c)$, with $0 \le \theta \le 1$, such that

$$f(x^*) = f(c) + f'(\eta)(x^* - c).$$

Let $F'([x])$ be an inclusion function for f' of f. If $\eta, x^* \in [x]$, then

$$f(x^*) \in f(c) + F'([x])([x] - c) =: F([x], c).$$

In the practice, the optimal choice for c is the mid-point of $[x]$, $c = \text{mid}([x])$.

In generalized interval arithmetic, we can expanded the function f as

$$\begin{aligned} f(\hat{x}) &\in F([c^x]) + F'([\hat{x}])\sum_{i=1}^{n}\zeta_i[v_i^x] \\ &= F([c^x]) + \sum_{i=1}^{n}\zeta_i[v_i^z] =: F([\hat{x}], \zeta), \end{aligned}$$

where $[v_i^z] := F'([\hat{x}])[v_i^x]$, $(i = 1, 2, \cdots, n)$.

Example 6.1. *Let* $[x] = [1, 1.1]$. *Evaluate*

$$F = \exp([x]).$$

Using generalized interval arithmetic, where $[\hat{x}] = [1.05, 1.05] + [1, 1]\zeta$, $\zeta \in [-0.05, 0.05]$ *we obtain*

$$\begin{aligned} F_{\textit{Hansen}} &= \exp([1.05, 1.05]) + \underbrace{[2.7182818, 3.0041661]}_{F'([\hat{x}])}[1, 1]\zeta \\ &= [2.8576511, 2.8576512] + [2.7182818, 3.0041661]\zeta \end{aligned}$$

which reduces to $[2.7074428, 3.0078595]$. *In interval arithmetic, we obtain the result* $[2.7182818, 3.0041661]$

In case of the function f is a real function of multi-variable, we can extended a case of one variable, let $f : S \subseteq \mathbb{R}^n \longrightarrow \mathbb{R}$, be an once differentiable function, $x^*, c \in S$, where S is an open set. Then, there exists $\eta = x^* + \theta(x^* - c)$, with $0 \leq \theta \leq 1$, sucht that

$$f(x^*) = f(c) + \sum_{j=1}^{n} \frac{\partial f}{\partial x_j}(\eta)(x_j^* - c_j).$$

Let $F_j'([x])$ be an inclusion function for $\partial f/\partial x_j =: f_j'$, $(j = 1, 2, \cdots, n)$. If $\eta, x^* \in [x]$, then

$$f(x^*) \in f(c) + \sum_{j=1}^{n} F_j'([x])([x_j] - c_j) =: F([x], c).$$

In generalized interval arithmetic, we can expanded a function of multi-variable as

$$f(\hat{x}_1, \cdots, \hat{x}_n) = f(c^x) + \sum_{j=1}^{n} \frac{\partial f}{\partial x_j}(c^x + \theta \sum_{k=1}^{n} \zeta_k v_k^x) \cdot \sum_{i=1}^{n} \zeta_i v_i^{x_j},$$

where $\hat{x}_i = c^{x_i} + \sum_{j=1}^{n} \zeta_j v_j^{x_i}$, $(i = 1, 2, \cdots, n)$, $c^x := (c^{x_1}, \cdots, c^{x_n})^{\mathrm{T}} \in \mathbb{R}^n$ and $v_k^x := (v_k^{x_1}, \cdots, v_k^{x_n})^{\mathrm{T}} \in \mathbb{R}^n$, $(k = 1, 2, \cdots, n)$.

Then for $\hat{x}, c^x + \theta \sum_{k=1}^{n} \zeta_k v_k^x \in [\hat{x}]$, it is obviously that

$$\begin{aligned} f(\hat{x}_1, \cdots, \hat{x}_n) &\in F([c^x]) + \sum_{j=1}^{n} F_j'([\hat{x}]) \sum_{i=1}^{n} \zeta_i [v_i^{x_j}] \\ &= F([c^x]) + \sum_{i=1}^{n} \zeta_i [v_i^z] =: F([\hat{x}], \zeta) \end{aligned}$$

where $[v_i^z] := \sum_{j=1}^{n} F_j'([\hat{x}])[v_i^{x_j}]$, $(i = 1, 2, \cdots, n)$.

Example 6.2. *Let* $[x_1] = [5, 10]$, $[x_2] = [1, 2]$. *Evaluate*

$$F = \sqrt{\frac{[x_1] + [x_2]}{[x_1] - [x_2]}}.$$

Using generalized interval arithmetic, where $[\hat{x}_1] = [7.5, 7.5] + [1, 1]\zeta_1$*,*
$[\hat{x}_2] = [1.5, 1.5] + [1, 1]\zeta_2$ *with* $\zeta_1 \in [-2.5, 2.5]$*,* $\zeta_2 \in [-0.5, 0.5]$ *we obtain*

$$\frac{[\hat{x}_1] + [\hat{x}_2]}{[\hat{x}_1] - [\hat{x}_2]} = [1.5, 1.5] + \underbrace{[-0.166667, -0.0555555]}_{[v_1^{x_1}]}\zeta_1 + \underbrace{[0.2777777, 0.833334]}_{[v_2^{x_2}]}\zeta_2$$

so that

$$\begin{aligned} F_{\mathit{Hansen}} &= \sqrt{[1.5, 1.5]} + \underbrace{[-0.6123725, 0.06804139]}_{\partial F([\hat{x}])/\partial x_1}\underbrace{[-0.166667, -0.0555555]}_{[v_1^{x_1}]}\zeta_1 \\ &\quad + \underbrace{[0.06061608, 1.0206207]}_{\partial F([\hat{x}])/\partial x_2}\underbrace{[0.2777777, 0.833334]}_{[v_2^{x_2}]}\zeta_2 \\ &= [1.2247448, 1.2247449] + [-0.0113402, 0.1020621]\zeta_1 \\ &\quad +[0.0168378, 0.85051728]\zeta_2 \end{aligned}$$

which reduces to $[0.54433105, 1.9051587]$*. In interval arithmetic, we obtain the result* $[0.81649658, 2.0]$.

Example 6.3. *Let* $[x_1] = [5, 10]$*,* $[x_2] = [1, 2]$*. Evaluate*

$$F = \sqrt{\frac{[x_1] + [x_2]}{[x_1] - [x_2]}} - \sqrt{\frac{[x_1] + [x_2]}{[x_1] - [x_2]}}.$$

From the above example, we get

$$\begin{aligned} \sqrt{\frac{[\hat{x}_1] + [\hat{x}_2]}{[\hat{x}_1] - [\hat{x}_2]}} &= [1.2247448, 1.2247449] + [-0.0113402, 0.1020621]\zeta_1 \\ &\quad +[0.0168378, 0.85051728]\zeta_2, \end{aligned}$$

where $\zeta_1 \in [-2.5, 2.5]$*,* $\zeta_2 \in [-0.5, 0.5]$*, so that*

$$\begin{aligned} F_{\mathit{Hansen}} &= [-4.4408921E - 016, 4.4408921E - 016] + [-0.11340231, 0.11340231]\zeta_1 \\ &\quad +[-0.83367948, 0.83367948]\zeta_2 \end{aligned}$$

which reduces to $[-0.70034550, 0.70034550]$*. In interval arithmetic, we obtain the result* $[-1.1835035, 1.1835035]$.

7 Algorithmic Description

We now describe the algorithms for the elementary operations $+$, $-$, $\cdot$ and $/$, and for elementary functions $s \in \{$sqr, sqrt, power, exp, ln, sin, cos, tan, cot, asin, acos, atan, acot, sinh, cosh, tanh, coth} of generalized interval arithmetic (Hansen Arithmetic) for a once continuously differentiable function. For Hansen forms, we use quadruples

$$X = ([x], [c^x], [v^x], [g^x]),$$

with $[x] \in I\mathbb{R}$, $[c^x] \in I\mathbb{R}^n$, $[v^x] \in I\mathbb{R}^n$, and $[g^x] \in I\mathbb{R}^n$ for the description of the arithmetic rules. Here $[x]$, $[c^x]$, $[v^x]$ and $[g^x]$ denote the function value, the mid-point value, the argument value of ζ_i, $(i = 1, \cdots, n)$, and the gradient value, respectively.

In all of the following algorithms, the first step means the computation of the result in ordinary interval arithmetic.

Addition

Algorithm 7.1. **Operator +** (X, Y)

```
1. $[z] \;= [x] + [y]$;      function value
2. $[c^z] = [c^x] + [c^y]$;   mid-point
3. $[v^z] = [v^x] + [v^y]$;   argument
4. $[g^z] = [g^x] + [g^y]$;   gradient
5. **return** Z := ($[z]$, $[c^z]$, $[v^z]$, $[g^z]$ );
```

Subtraction

Algorithm 7.2. **Operator —** (X, Y)

```
1. $[z] \;= [x] - [y]$;      function value
2. $[c^z] = [c^x] - [c^y]$;   mid-point
3. $[v^z] = [v^x] - [v^y]$;   argument
4. $[g^z] = [g^x] - [g^y]$;   gradient
5. **return** Z := ($[z]$, $[c^z]$, $[v^z]$, $[g^z]$ );
```

In Algorithms (7.3) and (7.4), $[sx]$, $[sy]$, $[sxy]$, $[s_{vxy}]$, $[sxg]$ and $[syg]$ denote real intervals.

Multiplication

Algorithm 7.3. Operator ● (X, Y)

```
1. $[z] = [x] \cdot [y]$;     function value
2. $[sx] = 0$; $[sy] = 0$; $[s_{vxy}] = 0$;
   $[sxy] = 0$; $[sxg] = 0$; $[syg] = 0$;
3. for $i = 1$ to $n$ do
       $[c^z_i] = 0$;     mid-point
       $[sx] = [sx] + [c^x_i]$;
       $[sy] = [sy] + [c^y_i]$;
       $[sxg] = [sxg] + [c^x_i] + [v^x_i]$·interval($-$rad($[x]$),rad($[x]$));
       $[syg] = [syg] + [c^y_i] + [v^y_i]$·interval($-$rad($[y]$),rad($[y]$));
       if (rad($[x]$) $= 0$)     $r_i$ =rad($[y]$);
         else     $r_i$ =rad($[x]$);
       $[s_{vxy}] = [s_{vxy}]$+interval$(0, r_i^2) \cdot [v^x_i] \cdot [v^y_i]$;
4. for $i = 1$ to $n$ do
       absu =AbsMax($[v^x_i]$);
       $[sxy] = 0$;
       for $j = 1$ to $n$ do
           if$(i \neq j)$
             absv =AbsMax($[v^y_j]$);
             if (rad($[x]$) $= 0$)     $r_j$ =rad($[y]$);
               else     $r_j$ =rad($[x]$);
             $[sxy] = [sxy]$+interval$(-1, 1)\cdot$ absu$\cdot r_j \cdot$absv;
       $[v^z_i] = [c^x_i] \cdot [v^y_i] + [c^y_i] \cdot [v^x_i] + [sxy]$; argument
       $[g^z_i] = [syg] \cdot [g^x_i] + [sxg] \cdot [g^y_i]$;     gradient
5. $[c^z_1] = [sx] \cdot [sy] + [s_{vxy}]$;     mid-point
6. return  Z = ($[z]$, $[c^z]$, $[v^z]$, $[g^z]$ );
```

In Algorithm (7.4), we do not take care of the case $0 \in [y]$, because it does not make sense to go any further in computations when this case occurs. In an implementation, the standard error handling (runtime error) should be invoked if a division by zero occurs.

Division

Algorithm 7.4. **Operator** $/$ (X, Y)

```
1. $[z] = [x]/[y]$;     function value
2. $[sx] = 0$; $[sy] = 0$; $[svy] = 0$;
   $[sxg] = 0$; $[syg] = 0$;
3. for $i = 1$ to $n$ do
       $[c_i^z] = 0$;     mid-point
       $[sx] = [sx] + [c_i^x]$;
       $[sy] = [sy] + [c_i^y]$;
       $[sxg] = [sxg] + [c_i^x] + [v_i^x] \cdot$interval$(-$rad$([x])$,rad$([x]))$;
       $[syg] = [syg] + [c_i^y] + [v_i^y] \cdot$interval$(-$rad$([y])$,rad$([y]))$;
       absv =AbsMax$([v_i^y])$;
       if (rad$([x]) = 0$) $r_i$ =rad$([y])$;
         else $r_i$ =rad$([x])$;
       $[svy] = [svy]+$interval$(-1, 1) \cdot r_i \cdot$absv;
3. for $i = 1$ to $n$ do
       $[v_i^z] = ([sy] \cdot [v_i^x] - [sx] \cdot [v_i^y])/([sy] \cdot ([sy] + [svy]))$; argument
       $[g_i^z] = ([g_i^x] - ([sxg]/[syg]) \cdot [g_i^y])/[syg]$; gradient
4. $[c_1^z] = [sx]/[sy]$; mid-point
5. return  Z = ($[z]$, $[c^z]$, $[v^z]$, $[g^z]$ );
```

Our implementation of Algorithm (7.5) uses the automatic differentiation module grad_ari (see [1], Chapter 12). [temp] and [sum] denote real intervals.

Elementary function

```
Algorithm 7.5. s(X)
1. $[z] := s([x])$; function value
2. [temp] $:= s'([\hat{x}])$; temporary value
3. [sum] $= 0$;
4. for $i = 1$ to $n$ do
     $[c_i^z] = 0$; mid-point
     [sum] $=$ [sum] $+ [c_i^x]$;
     $[g_i^z] =$ [temp] $\cdot [g_i^x]$; gradient
     $[v_i^z] = [v_i^x] \cdot [g_i^z]$; argument
5. $[c_1^z] = s($[sum]$)$; mid-point
6. return s:= Z = ($[z]$, $[c^z]$, $[v^z]$, $[g^z]$ );
```

8 Implementation and Examples

8.1 C-XSC Program Code

In the following section, we present a module for multi-dimensional generalized interval arithmetic.

8.1.1 Module hansen_arith.cpp

The header file hansen_arith.hpp of module hansen_arith.cpp supplies the definition of the two classes HansenType and HansenType_vector including operators and elementary functions for a generalized interval arithmetic. The type name HansenType_FctPtr can be used to declare a pointer to scalar-valued functions with an argument of type HansenType_vector.

```
//-------------------------------------------------------------------
// File: hansen_arith.hpp (header)
// Purpose: Definition of a one- and multi-dimensional generalized
//          interval arithmetic which allows function evaluation
//          with Hansen Arithmetic
//
// Types:
//    HansenType_FctPtr      : pointer for a scalar valued function
// Class HansenType:
//    HansenType()           : constructors
//    Resize()               : for resizing to a fixed dimension
//    operator =             : assignment operators for arguments of
//                             types HansenType, interval, and real
//    HansenVar()            : to define HansenType variables
//    get_fValue(),
//    get_midValue(),
//    get_hansenCoef()       : to get function value, mid-point
//                              value and Hansen coefficients
//    operators +, -, *, /   : operators of diff. arithmetic
//    sqr(), sqrt(), power(),
//    exp(), sin(), cos(), ..: elementary functions
//    print_hVal()           : to print Hansen value and the
//                             linear Form
//    print_fVal()           : to print function value only
//    print_fhVal()          : to print function, Hansen
//                             value and the linear form
// Class HansenType_vector:
//    HansenType_vector()    : constructors
//    ~HansenType_vector()   : destructor
//    operator =             : assignment operator
//    operator []            : component access
//-------------------------------------------------------------------
#ifndef __HANSEN_ARITH_HPP
#define __HANSEN_ARITH_HPP

// C++ standard headers
#include <iostream>     // I/O Handling

// cxsc headers
#include "imath.hpp"   // Interval mathematical functions
#include <imatrix.hpp>
#include <idot.hpp>

using namespace cxsc;
using namespace std;


class HansenType;
class HansenType_vector;

typedef HansenType (*HansenType_FctPtr)(const HansenType_vector&);

class HansenType {
    int              dim;  // the dimension of the interval vector
    interval         RIA;  // the Result value in
                           // (ordinary) Interval Arithmetic
                           //---------------------------------------

    ivector          C,    // the mid-point value
                     V,    // the coefficients of zeta
                           //---------------------------------------
                           // By using the mid-point value and the
                           // coefficients of zeta, we can get the
                           //linear form of the function, after that
                           //we can get the Hansen Value
                           //---------------------------------------

                     g;    // the gradient value, we will use it in
                           // the elementary function routines
                           //---------------------------------------
    rvector          r;    // the radius vector

    friend void Resize (HansenType&, int);

  public:
    // Constructors:
    explicit HansenType (int n=0);
    HansenType (const HansenType&); // copy constructors

    // assignment operators
    HansenType& operator= (const HansenType&);
    HansenType& operator= (const interval&);
    HansenType& operator= (const real&);

    friend HansenType HansenVar (int, const real&);
    friend HansenType HansenVar (int, const interval&);
    friend HansenType_vector HansenVar (const ivector&);
    friend HansenType_vector HansenVar (const rvector&);

    // access to the components:
    friend interval get_fValue     (const HansenType&);
    friend ivector  get_midvalue   (const HansenType&);
    friend ivector  get_hansenCoef (const HansenType&);

    // operator +, - :
    friend HansenType operator+ (HansenType&);
    friend HansenType operator- (const HansenType&);

    // operators +,-,*,/  for (HansenType, HansenType ):
    friend HansenType operator+ (const HansenType&,const HansenType&);
    friend HansenType operator- (const HansenType&,const HansenType&);
```

```
friend HansenType operator* (const HansenType&,const HansenType&);
friend HansenType operator/ (const HansenType&,const HansenType&);

// operators +,-,*,/ for (HansenType, interval):
friend HansenType operator+ (const HansenType&,const interval&);
friend HansenType operator- (const HansenType&,const interval&);
friend HansenType operator* (const HansenType&,const interval&);
friend HansenType operator/ (const HansenType&,const interval&);

// operators +,-,*,/ for (interval, HansenType):
friend HansenType operator+ (const interval&,const HansenType&);
friend HansenType operator- (const interval&,const HansenType&);
friend HansenType operator* (const interval&,const HansenType&);
friend HansenType operator/ (const interval&,const HansenType&);

// operators +,-,*,/ for (HansenType, real):
friend HansenType operator+ (const HansenType&,const real&);
friend HansenType operator- (const HansenType&,const real&);
friend HansenType operator* (const HansenType&,const real&);
friend HansenType operator/ (const HansenType&,const real&);

// operators +,-,*,/ for (real, HansenType):
friend HansenType operator+ (const real&,const HansenType&);
friend HansenType operator- (const real&,const HansenType&);
friend HansenType operator* (const real&,const HansenType&);
friend HansenType operator/ (const real&,const HansenType&);

// The standard functions:
friend HansenType sqr   (const HansenType&);
friend HansenType power (const HansenType&, const int);
friend HansenType sqrt  (const HansenType&);
friend HansenType exp   (const HansenType&);
friend HansenType ln    (const HansenType&);
friend HansenType sin   (const HansenType&);
friend HansenType cos   (const HansenType&);
friend HansenType tan   (const HansenType&);
friend HansenType cot   (const HansenType&);
friend HansenType asin  (const HansenType&);
friend HansenType acos  (const HansenType&);
friend HansenType atan  (const HansenType&);
friend HansenType acot  (const HansenType&);
friend HansenType sinh  (const HansenType&);
friend HansenType cosh  (const HansenType&);
friend HansenType tanh  (const HansenType&);
friend HansenType coth  (const HansenType&);
friend HansenType asinh (const HansenType&);
friend HansenType acosh (const HansenType&);
```

```
    friend HansenType atanh (const HansenType&);
    friend HansenType acoth (const HansenType&);

    // Help function
    friend HansenType For_Elementary_Functions(const HansenType&,
                                       interval&, interval&);
    // Output:
    friend void print_fhVal (HansenType_FctPtr, ivector);
    friend void print_hVal  (HansenType_FctPtr, ivector);
    friend void print_fVal  (HansenType_FctPtr, ivector);
};
//-------------------------------------------------------------------

class HansenType_vector {
    int vector_dim;         // dimension of the vector
                            //--------------------------
    HansenType *ht;         //pointer to a dynamic array of
                            //elements of type HansenType

  public:
     // Constructors and Destructors:
     explicit HansenType_vector (int n=0);
     HansenType_vector (const HansenType_vector&);//copy constructor
     ~HansenType_vector ( );

     HansenType_vector& operator= (const HansenType_vector&);
     HansenType& operator[] (int) const;

};

#endif
```

For a scalar-valued function of type HansenType, print_fVal() computes and returns only the function value (in interval arithmetic). The function print_fhVal() returns the value of $f(x)$ (in interval arithmetic), mid-point value, the argument value of ζ_i, ($i = 1, \cdots, n$), and Hansen value.

```
//-------------------------------------------------------------------
// File: hansen_arith.cpp (implementation)
// Purpose: Definition of a one- and multi-dimensional generalized
//          interval arithmetic which allows function evaluation
//          with Hansen Arithmetic
//
// Types:
//    HansenType_FctPtr      : pointer for a scalar valued function
// Class HansenType:
//    HansenType()           : constructors
//    Resize()               : for resizing to a fixed dimension
//    operator =             : assignment operators for arguments of
//                             types HansenType, interval, and real
//    HansenVar()            : to define HansenType variables
//    get_fValue(),
//    get_midValue(),
//    get_hansenCoef()       : to get function value, mid-point
//                             value and Hansen coefficients
//    operators +, -, *, /   : operators of diff. arithmetic
//    sqr(), sqrt(), power(),
//    exp(), sin(), cos(), ..: elementary functions
//    print_hVal()           : to print Hansen value and the
//                             linear Form
//    print_fVal()           : to print function value only
//    print_fhVal()          : to print function, Hansen
//                             value and the linear form
// Class HansenType_vector:
//    HansenType_vector()    : constructors
//    ~HansenType_vector()   : destructor
//    operator =             : assignment operator
//    operator []            : component access
//-------------------------------------------------------------------
#include <hansen_arith.hpp>

//*******************************************************************
//------------------------------------------------------------------*
// RIA: denote the result value (in interval Arithmetic)           *
// C  : denote the mid-point vector                                *
// V  : denote the coefficients value of \zeta_i (i=1,.....,n)     *
// g  : denote the gradient value                                  *
//------------------------------------------------------------------*
//*******************************************************************

//-------------------------------------------------------------------
// Constructors and resize function for the class 'HansenType'
//-------------------------------------------------------------------

void Resize (HansenType& X, int n)
{
  X.dim = n;
  Resize(X.g,0,n);
  Resize(X.C,0,n);
  Resize(X.V,0,n);
  Resize(X.r,0,n);
}
//-------------------------------------------------------------------
```

```
HansenType::HansenType (int n):dim(n)//Constructor: resizes internal
{
  if (dim <= 0) {dim = 0; return;}
  Resize(g,0,dim);
  Resize(C,0,dim);
  Resize(V,0,dim);
  Resize(r,0,dim);
}
//-------------------------------------------------------------------

//-------------------------------------------------------------------
// Assignment operators and copy constructors of the class
//  'HansenType'.
//-------------------------------------------------------------------

HansenType& HansenType::operator= (const HansenType& X)
{
  if (this == &X) return *this;
  dim = X.dim;
  if (dim > 0)
  {
    RIA = X.RIA;
    C   = X.C;
    V   = X.V;
    g   = X.g;
    r   = X.r;
  }
  return *this;
}
//-------------------------------------------------------------------

HansenType::HansenType (const HansenType& X)  // Copy constructor
{                                             //-----------------
  dim = X.dim;
  *this = X;   // use previously defined assignment operator
}
//-------------------------------------------------------------------

HansenType& HansenType::operator= (const interval& u)
{
  RIA = u;                                    // result value
  C = 0.0;
  C[0] = 0.5*(interval(Sup(u)+Inf(u)));       // mid-point
  if(mid(u)==0.0) C[0] =u;
  V = 0.0;                                    // argument
  g = 0.0;                                    // gradient
  r  =0.0;
  r[0] = 0.5*(subup(Sup(u),Inf(u)));          // radius
  return *this;
}
//--------------------------------------------------------------------

HansenType& HansenType::operator= (const real& c)
{
  *this = interval(c);
  return *this;
}
//--------------------------------------------------------------------

HansenType HansenVar(int n, const interval& u)
{
  HansenType  ht(n);
  real        RAD;

  ht.RIA = u ;
  for (int i = 0; i <= n; i++)
  {
    ht.C[i] = 0.0;
    ht.V[i] = 0.0;
    ht.g[i] = 0.0;
    ht.r[i] = 0.0;
  }
  ht.C[0] = mid(u);
  if(mid(u)==0.0) ht.C[0]=u;
  RAD = 0.5*(Sup(u)-Inf(u));
  ht.V[0] = interval(-RAD,RAD);
  if(mid(u)==0) ht.V[0]=0.0;
  return ht;
}
//--------------------------------------------------------------------

HansenType HansenVar (int n, const real& u) // Generate variable
{                                           //-----------------
  interval v;
  v=u;
  return HansenVar(n, v);
}
//--------------------------------------------------------------------

//--------------------------------------------------------------------
// Transfer functions for variables
//--------------------------------------------------------------------

HansenType_vector HansenVar (const ivector& u)  // Generate variable
```

```
{                                               //------------------
  int   d = Ub(u)-Lb(u)+1;
  HansenType_vector  htv(d);

  for (int i = 1; i <= d; i++)
  {
    htv[i].RIA = u[i];
    for (int k = 0; k <= d; k++)
    {
      if(i==k)
      {
       htv[i].C[k] = mid(u[i]);
       htv[i].V[k] = 1.0;
       htv[i].g[k] = 1.0;
       htv[i].r[k] = 0.5*(Sup(u[i])-Inf(u[i]));
      }
       else
       {
        htv[i].C[k] = 0.0;
        htv[i].V[k] = 0.0;
        htv[i].g[k] = 0.0;
        htv[i].r[k] = 0.0;
       }
    }
  }
 return htv;
}
//--------------------------------------------------------------------

HansenType_vector HansenVar (const rvector& v)  // Generate variable
{                                               //------------------
  int      k=Lb(v), m = Ub(v);
  ivector u(k,m);

  for (int i=k; i<=m; i++)
   u[i] = v[i];
  return HansenVar(u);
}
//--------------------------------------------------------------------

//--------------------------------------------------------------------
// Access functions for the function value, the mid-point values
// and  the Hansen coefficient values
//--------------------------------------------------------------------

interval get_fValue (const HansenType& X)     // Get function value
  { return X.RIA; }                           //------------------
```

```
ivector  get_midvalue (const HansenType& X)   //Get mid-point
 { return X.C; }                              //------------------


ivector get_hansenCoef (const HansenType& X)  //Get Hansen
 { return  X.V; }                             //coefficients of zeta
                                              //-------------------
//--------------------------------------------------------------------

//--------------------------------------------------------------------
// Unary operators + and - for HansenType operands
//--------------------------------------------------------------------

HansenType operator+ (HansenType& X)
  { return X; }
//--------------------------------------------------------------------

HansenType operator- (const HansenType& X)
{
  HansenType min(X.dim);

  min.RIA = -X.RIA;
  for (int i = 0; i <= X.dim; i++)
  {
    min.C[i] = -X.C[i];
    min.V[i] = -X.V[i];
    min.r[i] = X.r[i];
    min.g[i] = -X.g[i];
  }
  return min;
}
//--------------------------------------------------------------------

//--------------------------------------------------------------------
// Operators +, -, *, and / for two HansenType operands
//--------------------------------------------------------------------

HansenType operator+ (const HansenType& X, const HansenType& Y)
{
  HansenType res(X.dim);

  res.RIA = X.RIA + Y.RIA;

  //--------------------------------
  res.C = X.C + Y.C;
  res.V = X.V + Y.V;
  res.g = X.g + Y.g;
  //-----------------------------------

  for (int i = 0; i <= X.dim; i++)
  {
   if(X.r[i]==0.0)  res.r[i] = Y.r[i];
     else           res.r[i] = X.r[i];
  }
 return res;
}
//-------------------------------------------------------------------

HansenType operator- (const HansenType& X, const HansenType& Y)
{
  HansenType res(X.dim);

  res.RIA = X.RIA - Y.RIA;

  //----------------------------------
  res.C  = X.C - Y.C;
  res.V = X.V - Y.V;
  res.g = X.g - Y.g;
  //----------------------------------

  for (int i = 0; i <= X.dim; i++)
  {
    if(X.r[i]==0.0) res.r[i] = Y.r[i];
     else           res.r[i] = X.r[i];
  }

  return res;
}
//-------------------------------------------------------------------

HansenType operator* (const HansenType& X, const HansenType& Y)
{
  HansenType res(X.dim);
  interval   Sx, Sy, Sxy, S_vxy,
             Syg, Sxg;
  real       cr;

  Sx=0.0;  Sy=0.0; S_vxy=0.0;
  Sxg=0.0; Syg=0.0;

  res.RIA = X.RIA * Y.RIA;
  for (int i = 0; i <= X.dim; i++)
  {
    res.C[i]=0.0;
    Sx += X.C[i];
    Sy += Y.C[i];
    Sxg += X.C[i]+X.V[i]*interval(-X.r[i],X.r[i]);
    Syg += Y.C[i]+Y.V[i]*interval(-Y.r[i],Y.r[i]);
    if(X.r[i]==0.0)
    {
     if(Y.r[i]==0.0)  cr=X.r[i]*X.r[i];
      else            cr=Y.r[i]*Y.r[i];
    }
    else   cr=X.r[i]*X.r[i];
    S_vxy += interval(0.0,cr)*X.V[i]*Y.V[i];
  }

  for(int i=0;i<=X.dim;i++)
  {
    Sxy=0.0;
    for(int j=0;j<=Y.dim;j++)
    {
     if(j!=i)
      Sxy +=Y.r[j]*AbsMax(Y.V[j]);
    }
    Sxy = Sxy*interval(-1,1)*AbsMax(X.V[i]);
    res.V[i]= Sx*Y.V[i]+Sy*X.V[i]+Sxy;
    if(X.r[i]==0.0) res.r[i]=Y.r[i];
     else           res.r[i]=X.r[i];
    res.g[i] = Syg*X.g[i] + Sxg*Y.g[i];
  }
  res.C[1]= Sx*Sy+S_vxy;

  return res;
}
//-------------------------------------------------------------------

HansenType operator/ (const HansenType& X, const HansenType& Y)
{
  HansenType res(Y.dim);
  interval   Sx, Sy, Svy,
             Sxg, Syg;
  Sx=0.0;  Sy=0.0; Svy=0.0;
  Sxg=0.0; Syg=0.0;

  res.RIA = X.RIA / Y.RIA; // can propagate "division by zero" error
                           //-----------------------------------------
  for (int i = 0; i <= Y.dim; i++)
   {
     res.C[i]= 0.0;
     Sx  += X.C[i];
     Sy  += Y.C[i];
     Sxg += X.C[i]+X.V[i]*interval(-X.r[i],X.r[i]);
     Syg += Y.C[i]+Y.V[i]*interval(-Y.r[i],Y.r[i]);
     Svy += interval(-1,1)*Y.r[i]*AbsMax(Y.V[i]);
   }
  for (int i = 0; i <= Y.dim; i++)
  {
    res.V[i] =(Sy*X.V[i]- Sx*Y.V[i] )/ ( Sy*(Sy+Svy));
    res.g[i] = (X.g[i]- (Sxg/Syg)*Y.g[i])/Syg;
    if(X.r[i]==0.0) res.r[i]=Y.r[i];
     else           res.r[i]=X.r[i];
  }
  res.C[1]=Sx/Sy;
  return res ;
}
//-------------------------------------------------------------------

//-------------------------------------------------------------------
// Operators +, -, * and / for one interval and one
//  HansenType operand
//-------------------------------------------------------------------

HansenType operator+ (const HansenType& X, const interval& b)
{
  HansenType res(X.dim);
  int        n=X.dim;
  res=X+HansenVar(n,b);
  return res;
}
//-------------------------------------------------------------------

HansenType operator- (const HansenType& X, const interval& b)
{
  HansenType res(X.dim);
  int        n=X.dim;
  res=X-HansenVar(n,b);
  return res;
}
//-------------------------------------------------------------------

HansenType operator* (const HansenType& X, const interval& b)
{
  HansenType res(X.dim);
  int        n=X.dim;
  res=X*HansenVar(n,b);
   return res;
```

```
}
//-------------------------------------------------------------------

HansenType operator/ (const HansenType& X, const interval& b)
{
  HansenType res(X.dim);
  int        n=X.dim;
  res=X/HansenVar(n,b);
  return res;
}
//-------------------------------------------------------------------

HansenType operator+ (const interval& a, const HansenType& Y)
{
  HansenType res(Y.dim);
  int        n=Y.dim;
  res=HansenVar(n,a)+Y;
  return res;
}
//-------------------------------------------------------------------

HansenType operator- (const interval& a, const HansenType& Y)
{
  HansenType res(Y.dim);
  int        n=Y.dim;
  res=HansenVar(n,a)-Y;
  return res;
}
//-------------------------------------------------------------------

HansenType operator* (const interval& a, const HansenType& Y)
{
  HansenType res(Y.dim);
  int        n=Y.dim;
  res=HansenVar(n,a)*Y;
  return res;
}
//-------------------------------------------------------------------

HansenType operator/ (const interval& a, const HansenType& Y)
{
  HansenType res(Y.dim);
  int        n=Y.dim;
  res=HansenVar(n,a)/Y;
  return res ;
}
//-------------------------------------------------------------------
```

```
//-------------------------------------------------------------------
// Operators +, -, * and / for one real and one HansenType operand
//-------------------------------------------------------------------
HansenType operator+ (const HansenType& X, const real& b)
  { return X + interval(b); }

HansenType operator- (const HansenType& X, const real& b)
  { return X - interval(b); }

HansenType operator* (const HansenType& X, const real& b)
  { return X * interval(b); }

HansenType operator/ (const HansenType& X, const real& b)
  { return X/interval(b);}// Can propagate 'division by zero' error
                          //----------------------------------------
HansenType operator+ (const real& a, const HansenType& Y)
  { return interval(a) + Y; }

HansenType operator- (const real& a, const HansenType& Y)
  { return interval(a) - Y; }

HansenType operator* (const real& a, const HansenType& Y)
  { return interval(a) * Y; }

HansenType operator/ (const real& a, const HansenType& Y)
  {return interval(a)/Y;}  // Can propagate 'division by zero' error
                           //----------------------------------------
//-------------------------------------------------------------------

//-------------------------------------------------------------------
// Elementary function for HansenType arguments
//-------------------------------------------------------------------

HansenType sqr (const HansenType& X)
{
  HansenType res(X.dim);
  interval   S1, S2, S3,
             M1, R1;
  real       cr, abs_uz, abs_vz;

  res.RIA = sqr(X.RIA);
                            -
  S1=0.0; S2=0.0; M1=0.0;

  //////////////////////
  // calculate the gradient
  S3=0.0;
  for(int i=0;i<=X.dim;i++)
   S3 += X.C[i] + X.V[i]*interval(-X.r[i],X.r[i]);
  R1 = 2.0 * S3;
  ////////////////////////

  for (int i = 0; i <= X.dim; i++)
  {
    res.C[i] = 0.0;
    res.g[i] = R1 * X.g[i];
    S1   += X.C[i];
    cr = X.r[i]*X.r[i];
    S2 += interval(0.0,cr)*X.V[i]*X.V[i];
  }

  for(int i=0;i<=X.dim;i++)
  {
    abs_uz=AbsMax(X.V[i]);
    M1=0.0;
    for(int j=0;j<=X.dim;j++)
    {
     if(j!=i)
     {
      abs_vz=AbsMax(X.V[j]);
      M1 +=interval(-1,1)*abs_uz*X.r[j]*abs_vz;
     }
    }
    res.V[i] = 2.0*S1*X.V[i]+M1;
    res.r[i] = X.r[i];
  }
  res.C[1]= sqr(S1) + S2;

  return res;
}
//------------------------------------------------------------------

HansenType power (const HansenType& X, const int k)
{
  int         k1, i, l;
  interval    S1;
  HansenType  res(X.dim), help(X.dim);

  if (k == 0)
    { res = 1.0; return res; }
  if (k == 1)
    return X;
  if (k == 2)
    return sqr(X);
  if (k == -1)
    return (1.0/X);
  if (k == -2)
    return (1.0/sqr(X));
  if (k < -2)
    return (1.0/power(X,k));

  // compute how many can sqr
  // ( X^6 = (X^2)^3  this means three times)
  //-----------------------------------------
  k1=k-2;
  i=1;
  if(k1>=2)
  {
   do{
      k1=k1-2;
      i++;
    }while(k1>=2);
  }
  l=i;

  // calculate the gradient k*X^(k-1)
  //-------------------------------------------------
  help = double(k)*power(X,k-1);
  S1=0.0;
  for(int i=0;i<=X.dim;i++)
   S1+= help.C[i]+help.V[i]*interval(-help.r[i],help.r[i]);
  //-------------------------------------------------

  res=sqr(X);
  for(int j=2;j<=l;j++)
   res=res*sqr(X);
  if(k1==1)
   res=res*X;

  res.RIA=power(X.RIA,k);
  for(int i=0; i<=X.dim; i++)
   res.g[i]= S1*X.g[i];
  return res;
}
//-------------------------------------------------------------

HansenType sqrt (const HansenType& X)
{
  HansenType res(X.dim);
  interval   R1, S, S1;
  S1=0.0;
  for(int i=0;i<=X.dim;i++)
   S1 += X.C[i]+X.V[i]*interval(-X.r[i],X.r[i]);
  R1 = 0.5/sqrt(S1);

  res = For_Elementary_Functions( X , R1, S);
  res.RIA = sqrt(X.RIA);
  res.C[1]=sqrt(S);
  return res;
}
//-------------------------------------------------------------------

HansenType exp (const HansenType& X)
{
  HansenType res(X.dim);
  interval   R1, S, S1;

  S1=0.0;
  for(int i=0;i<=X.dim;i++)
   S1 += X.C[i]+X.V[i]*interval(-X.r[i],X.r[i]);
  R1= exp(S1);

  res = For_Elementary_Functions( X, R1, S );
  res.RIA = exp(X.RIA);
  res.C[1]=exp(S);
  return res;
}
//-------------------------------------------------------------------

HansenType ln (const HansenType& X)
{
  HansenType res(X.dim);
  interval   R1, S, S1;

  S1=0.0;
  for(int i=0;i<=X.dim;i++)
   S1 += X.C[i]+X.V[i]*interval(-X.r[i],X.r[i]);
  R1= 1.0/S1;

  res = For_Elementary_Functions( X, R1, S );
  res.RIA = ln(X.RIA);
  res.C[1]=ln(S);
  return res;
}
//-------------------------------------------------------------------

HansenType sin  (const HansenType& X)
```

```
{
  HansenType res(X.dim);
  interval   R1, S, S1;
  S1=0.0;
  for(int i=0;i<=X.dim;i++)
   S1 += X.C[i]+X.V[i]*interval(-X.r[i],X.r[i]);
  R1=cos(S1);

  res = For_Elementary_Functions( X, R1, S );
  res.RIA = sin(X.RIA);
  res.C[1]=sin(S);
  return res;
}
//-------------------------------------------------------------------

HansenType cos (const HansenType& X)
{
  HansenType res(X.dim);
  interval   R1, S, S1;
  S1=0.0;
  for(int i=0;i<=X.dim;i++)
   S1 += X.C[i]+X.V[i]*interval(-X.r[i],X.r[i]);
  R1=-sin(S1);

  res = For_Elementary_Functions( X, R1, S );
  res.RIA = cos(X.RIA);
  res.C[1]=cos(S);
  return res;
}
//-------------------------------------------------------------------

HansenType tan  (const HansenType& X)
{
  HansenType res(X.dim);
  interval   R1, S, S1;
  S1=0.0;
  for(int i=0;i<=X.dim;i++)
   S1 += X.C[i]+X.V[i]*interval(-X.r[i],X.r[i]);
  R1=sqr(tan(S1)) + 1.0;

  res = For_Elementary_Functions( X, R1, S );
  res.RIA = tan(X.RIA);
  res.C[1]=tan(S);
  return res;
}
//-------------------------------------------------------------------
```

```
HansenType cot (const HansenType& X)
{
  HansenType res(X.dim);
  interval   R1, S, S1;
  S1=0.0;
  for(int i=0;i<=X.dim;i++)
   S1 += X.C[i]+X.V[i]*interval(-X.r[i],X.r[i]);
  R1= -( sqr(cot(S1)) + 1.0 );

  res = For_Elementary_Functions( X, R1, S );
  res.RIA = cot(X.RIA);
  res.C[1]=cot(S);
  return res;
}
//-------------------------------------------------------------------

HansenType asin (const HansenType& X)
{
  HansenType res(X.dim);
  interval   R1, S, S1;
  S1=0.0;
  for(int i=0;i<=X.dim;i++)
   S1 += X.C[i]+X.V[i]*interval(-X.r[i],X.r[i]);
  R1= 1.0/sqrt(1.0-sqr(S1));

  res = For_Elementary_Functions( X, R1, S );
  res.RIA = asin(X.RIA);
  res.C[1]= asin(S);
  return res;
}
//-------------------------------------------------------------------

HansenType acos (const HansenType& X)
{
  HansenType res(X.dim);
  interval   R1, S, S1;
  S1=0.0;
  for(int i=0;i<=X.dim;i++)
   S1 += X.C[i]+X.V[i]*interval(-X.r[i],X.r[i]);
  R1 =  -1.0 / sqrt( 1.0 -sqr(S1));

  res   = For_Elementary_Functions( X, R1, S );
  res.RIA = acos(X.RIA);
  res.C[1]= acos(S);
  return res;
}
//-------------------------------------------------------------------
```

```
HansenType atan  (const HansenType& X)
{
  HansenType res(X.dim);
  interval   R1,S, S1;

  S1=0.0;
  for(int i=0;i<=X.dim;i++)
   S1 += X.C[i]+X.V[i]*interval(-X.r[i],X.r[i]);
  R1= 1.0/ (1.0+ sqr(S1));

  res   = For_Elementary_Functions( X, R1, S );
  res.RIA = atan(X.RIA);
  res.C[1]=atan(S);
  return res;
}
//-------------------------------------------------------------------

HansenType acot  (const HansenType& X)
{
  HansenType res(X.dim);
  interval   R1,S, S1;

  S1=0.0;
  for(int i=0;i<=X.dim;i++)
   S1 += X.C[i]+X.V[i]*interval(-X.r[i],X.r[i]);
  R1 = -1.0/ (1.0+sqr(S1));

  res   = For_Elementary_Functions( X, R1, S );
  res.RIA = acot(X.RIA);
  res.C[1]=acot(S);
  return res;
}
//-------------------------------------------------------------------

HansenType sinh  (const HansenType& X)
{
  HansenType res(X.dim);
  interval   R1, S, S1;
  S1=0.0;
  for(int i=0;i<=X.dim;i++)
   S1 += X.C[i]+X.V[i]*interval(-X.r[i],X.r[i]);
  R1= cosh(S1);

  res   = For_Elementary_Functions( X, R1, S );
  res.RIA = sinh(X.RIA);
  res.C[1]=sinh(S);
  return res;
}
//-------------------------------------------------------------------

HansenType cosh (const HansenType& X)
{
  HansenType res(X.dim);
  interval   R1, S, S1;
  S1=0.0;
  for(int i=0;i<=X.dim;i++)
   S1 += X.C[i]+X.V[i]*interval(-X.r[i],X.r[i]);
  R1 = sinh(S1);

  res   = For_Elementary_Functions( X, R1, S );
  res.RIA = cosh(X.RIA);
  res.C[1]=cosh(S);
  return res;
}
//-------------------------------------------------------------------

HansenType tanh  (const HansenType& X)
{
  HansenType res(X.dim);
  interval   R1, S, S1;
  S1=0.0;
  for(int i=0;i<=X.dim;i++)
   S1 += X.C[i]+X.V[i]*interval(-X.r[i],X.r[i]);
  R1= 1.0 - sqr( tanh(S1) );

  res   = For_Elementary_Functions( X, R1, S );
  res.RIA = tanh(X.RIA);
  res.C[1]=tanh(S);
  return res;
}
//-------------------------------------------------------------------

HansenType coth  (const HansenType& X)
{
  HansenType res(X.dim);
  interval   R1, S, S1;
  S1=0.0;
  for(int i=0;i<=X.dim;i++)
   S1 += X.C[i]+X.V[i]*interval(-X.r[i],X.r[i]);
  R1= 1.0 -sqr( coth(S1) );

  res   = For_Elementary_Functions( X, R1, S );
  res.RIA = coth(X.RIA);
  res.C[1]=coth(S);
  return res;
}
//-------------------------------------------------------------------

HansenType asinh (const HansenType& X)
{
  HansenType res(X.dim);
  interval   R1, S, S1;
  S1=0.0;
  for(int i=0;i<=X.dim;i++)
   S1 += X.C[i]+X.V[i]*interval(-X.r[i],X.r[i]);
  R1 = 1.0 / sqrt( 1.0 +sqr(S1) );

  res   = For_Elementary_Functions( X, R1, S );
  res.RIA = asinh(X.RIA);
  res.C[1]= asinh(S);
  return res;
}
//-------------------------------------------------------------------

HansenType acosh (const HansenType& X)
{
  HansenType res(X.dim);
  interval   R1, S, S1;
  S1=0.0;
  for(int i=0;i<=X.dim;i++)
   S1 += X.C[i]+X.V[i]*interval(-X.r[i],X.r[i]);
  R1 = 1.0 / sqrt( sqr(S1) -1.0 );

  res   = For_Elementary_Functions( X, R1, S );
  res.RIA = acosh(X.RIA);
  res.C[1]= acosh(S);
  return res;
}
//-------------------------------------------------------------------

HansenType atanh  (const HansenType& X)
{
  HansenType res(X.dim);
  interval   R1, S, S1;

  S1=0.0;
  for(int i=0;i<=X.dim;i++)
   S1 += X.C[i]+X.V[i]*interval(-X.r[i],X.r[i]);
  R1 = 1.0 / (1.0 -sqr(S1));
  res   = For_Elementary_Functions( X, R1, S );
  res.RIA = atanh(X.RIA);
  res.C[1]=atanh(S);
  return res;
}
//------------------------------------------------------------------

HansenType acoth  (const HansenType& X)
{
  HansenType res(X.dim);
  interval   R1, S, S1;

  S1=0.0;
  for(int i=0;i<=X.dim;i++)
   S1 += X.C[i]+X.V[i]*interval(-X.r[i],X.r[i]);
  R1 = 1.0 / (1.0 -sqr(S1));

  res   = For_Elementary_Functions( X, R1, S );
  res.RIA = acoth(X.RIA);
  res.C[1]=acoth(S);
  return res;
}
//------------------------------------------------------------------

HansenType For_Elementary_Functions(const HansenType& X,interval& R1,
                                     interval& S)
{
  HansenType res(X.dim);
  S=0.0;
  for(int i=0; i<=X.dim; i++)
  {
    res.C[i] = 0.0;
    S += X.C[i];
    res.g[i] = R1*X.g[i];
    res.V[i]= X.V[i]*res.g[i];
    res.r[i]=X.r[i];
  }
  return res;
}
//------------------------------------------------------------------

//------------------------------------------------------------------
// Constructors, copy constructors and Destructors for the
//  class 'HansenType_vector'
//------------------------------------------------------------------

HansenType_vector::HansenType_vector (int n): vector_dim(n)
```

```
{
  if (vector_dim <= 0)
  { vector_dim = 0; ht = NULL; return; }
  ht = new HansenType[vector_dim];
  for (int i= 0; i< vector_dim; i++) Resize(ht[i],vector_dim);
}
//------------------------------------------------------------------

HansenType_vector::HansenType_vector (const HansenType_vector& v)
{
  vector_dim = v.vector_dim;
  *this = v;   // use previously defined assignment operator
}
//------------------------------------------------------------------

HansenType_vector::~HansenType_vector ( ) // Destructor
{                                         //-----------
  delete [] ht;
  ht =NULL;
}
//------------------------------------------------------------------

//------------------------------------------------------------------
// Assignment operators the class 'HansenType_vector'.
//------------------------------------------------------------------

HansenType_vector& HansenType_vector::operator= (
                                        const HansenType_vector& v)
{
  if (this == &v) return *this;
  delete [] ht;
  vector_dim = v.vector_dim;
  if (vector_dim == 0) { ht = NULL; return *this; }
  ht = new HansenType[vector_dim];
  for (int i=0; i<vector_dim; i++)
    ht[i] = v.ht[i];
  return *this;
}
//------------------------------------------------------------------

//------------------------------------------------------------------
// Component access for the type 'HansenType_vector'.
//------------------------------------------------------------------

HansenType& HansenType_vector::operator[] (int k) const
{
  return ht[k-1];        // Index range starts at 1
```

```
}
//--------------------------------------------------------------------

//--------------------------------------------------------------------
// Purpose: Evaluation and print of function ’f(x)’ for argument ’x’
//          in Hansen arithmetic, computing the function ’f(x)’ in
//          ordinary interval arithmetic  too.
// Parameters:
//    In : ’f’ : function of ’HansenType’
//         ’xx’ : argument for evaluation of ’f(x)’
//    Out: ’fx’: the function value ’f(x)’
//         ’Cx’: the interval mid-point ’
//         ’Hfx’: the Hansen interval vector (the argument
//                of \zeta_i,(i=1,...,n)
//        ’fHans’: the Hansen value of the function ’f(x)’
//--------------------------------------------------------------------

void print_fhVal ( HansenType_FctPtr f, ivector xx)
{
  int            m = Ub(xx);
  interval       fx, Cx, fHans;
  ivector        Hfx(0,m), mvalue(0,m),
         ONE(0,m);
  HansenType     fxH(m+1);
  real           r_x;
  idotprecision  IAccu;
  ONE=1.0;

  fxH    = f(HansenVar(xx));
  fx     = get_fValue(fxH);
  mvalue = get_midvalue(fxH);
  Hfx    = get_hansenCoef(fxH);

  IAccu=0.0;
  for(int i=0;i<=m;i++)
    accumulate(IAccu,mvalue[i],ONE[i]);
  Cx=rnd(IAccu);

  fHans   = Cx;

  cout << SetPrecision(15,6) << Scientific;   // Output format

  cout<< " \n F([x]): " << endl << fx << endl;

  for(int i=1;i<=m;i++)
  {
   r_x= 0.5*(Sup(xx[i])-Inf(xx[i]));
   fHans += Hfx[i]*interval(-r_x,r_x);
  }
  cout<< "\n Hansen Arithmetic: "<<endl << fHans<<endl<<endl;

  cout<<"\n The linear form: "<<endl;
  cout<<Cx;
  for(int i=1;i<=m;i++)
   cout<<" + "<< Hfx[i] <<" r["<<i<<"] ";

  cout<<endl;
  cout<<"\n Where r is the interval radius of the interval vector x:"
      <<endl;
  for(int i=1;i<=m;i++)
  {
   r_x=0.5*(Sup(xx[i])-Inf(xx[i]));
   cout<<"\n r["<<i<<"] = "<<interval(-r_x,r_x)<<endl<<endl;
  }
}
//-------------------------------------------------------------------

//-------------------------------------------------------------------
// Purpose: Evaluation and print of function 'f(x)' for argument 'x'
//          in Hansen arithmetic.
// Parameters:
//    In : 'f' : function of 'HansenType'
//         'xx' : argument for evaluation of 'f(x)'
//    Out: 'Cx': the interval mid-point '
//         'Hfx': the Hansen interval vector (the argument
//                of \zeta_i, i=1,...,n)
//        'fHans': the Hansen value of the function 'f(x)'
//-------------------------------------------------------------------

void print_hVal ( HansenType_FctPtr f, ivector xx)
{
  int            m = Ub(xx);
  interval       Cx, fHans;
  ivector        Hfx(0,m), mvalue(0,m),
         ONE(0,m);
  HansenType     fxH(m+1);
  real           RAD;
  idotprecision  IAccu;
  ONE=1.0;

  fxH    = f(HansenVar(xx));
  mvalue = get_midvalue(fxH);
  Hfx    = get_hansenCoef(fxH);
  IAccu=0.0;
  for(int i=0;i<=m;i++)
    accumulate(IAccu,mvalue[i],ONE[i]);
  Cx=rnd(IAccu);

  fHans   = Cx;

  cout << SetPrecision(23,15) << Scientific;   // Output format

  for(int i=1;i<=m;i++)
  {
   RAD = 0.5*(Sup(xx[i])-Inf(xx[i]));
   fHans += Hfx[i]*interval(-RAD,RAD);
  }
  cout<< "\n Hansen Arithmetic: "<<endl << fHans<<endl<<endl;

  cout<<"\n The linear form: "<<endl;
  cout<<Cx;
  for(int i=1;i<=m;i++)
   cout<<" + "<< Hfx[i] <<" r["<<i<<"] ";

  cout<<endl;
  cout<<"\n Where r is the interval radius of the interval vector x:"
      <<endl;
  for(int i=1;i<=m;i++)
  {
   RAD=0.5*(Sup(xx[i])-Inf(xx[i]));
   cout<<"\n r["<<i<<"] = "<<interval(-RAD,RAD)<<endl<<endl;
  }
}
//-------------------------------------------------------------------

//-------------------------------------------------------------------
// Purpose: Evaluation and print of function ’f(x)’ in ordinary
//          interval arithmetic.
// Parameters:
//    In : ’f’ : function of ’HansenType’
//         ’xx’ : argument for evaluation of ’f(x)’
//    Out: ’fx’: the function value ’f(x)’
//-------------------------------------------------------------------

void print_fVal ( HansenType_FctPtr f, ivector xx)
{
  int         m = Ub(xx);
  HansenType  fxH(m+1);
  interval    fx;
  fxH    = f(HansenVar(xx));
  fx     = get_fValue(fxH);
  cout << SetPrecision(23,15) << Scientific;   // Output format
  cout << " \n F([x]): " << endl << fx << endl<<endl;

}
//------------------------------------------------------------------
```

8.2 Examples

Here, we demonstrate how to use the modules defined in the previous section. A sample program contained in the distribution file hansen_ex.cpp reads input data (starting interval vector) from the console.

```
//-----------------------------------------------------
// Example: Automatic Hansen Arithmetic
//-----------------------------------------------------
#include <hansen_arith.hpp>     // Hansen arithmetic

using namespace cxsc;
using namespace std;

// xDim the dimension of a vector.
const int xDim = 2;

ivector x(xDim);

HansenType f ( const HansenType_vector& x )
{
  //please don't forget changing the dimension
  //-------------------------------------------

  // here are some expressions for testing
  //----------------------------------

  return exp(x[1]+(x[2]/(1+x[2])));
  // return power(x[1],4)+sin(x[1]);
  // return -(x[1]+x[2])*x[3];
}

int main ( )
{

  cout << "\n expression:  f(x) = exp(x[1]+(x[2]/(1+x[2]))); "
       << endl << endl;
  cout << "Starting vector x =   ";  cin >> x;

  /*********************************************************
  //The following functions are to compute:                *
  // 1) The expression in ordinary interval arithmetic     *
  // 2) The expression in Hansen Arithmetic and The linear*
  //    form of the expression (the expression is,         *
  //    in general, nonlinear expression)                  *
  //-----------------------------                          *
  // we have written the goal of every function beside it *
  //********************************************************

  print_fhVal(f,x);   // function, Hansen and linear form
  //print_fVal(f,x);  // function only
  //print_hVal(f,x);  // Hansen and linear form

  return 0;
}
```

If we compile the above main program we will get the following result:

```
function:  f(x) =  exp(x[1]+(x[2]/(1+x[2])));

Starting vector x =   [1.2,1.3] [13,13.5]

F([x]):
[8.138059200571847E+000, 9.624247682729886E+000]

Hansen Arithmetic:
[8.379173510681724E+000, 9.310335311913617E+000]

The linear form:
[8.844754411297660E+000, 8.844754411297681E+000] +
[8.402854155649450E+000, 9.309893894037638E+000] r1 +
[1.001654451902579E-004, 3.448236562108838E-004] r2

Where r is the interval radius of the interval vector x:
[r1] = [-5.000000000000005E-002, 5.000000000000005E-002]
[r2] = [-2.500000000000000E-001, 2.500000000000000E-001]
```

Example 8.1. *Compute the function*

$$f(x) := x_1^5 - 25.2x_1^3 + 24x_1 - 6x_2$$

with $x_1 \in [1,2]$ *and* $x_2 \in [3,4]$.

We get the result

```
[x]:
[ 1.000000000000000E+000, 2.000000000000000E+000]
[ 3.000000000000000E+000, 4.000000000000000E+000]
F([x]):
[-2.006000000000000E+002, 3.680000000000002E+001]
Hansen Arithmetic:
[-1.573500000000001E+002, 1.299375000000001E+001]
The linear form:
[-90.806250001,-53.54999999]+[-127.08750000,-115.09999999] r1 +
[-6.000000000,-6.0000000000] r2
Where r is the interval radius of the interval vector x:
[r1]=[-0.5, 0.5], [r2]=[-0.5,0.5]
```

Example 8.2. *Compute the function*

$$f(x) := \sin^3 x + x^4 \quad \textit{with} \quad x \in [0.5, 0.51].$$

We get the result

```
[x]:
[5.000000000000000E-001, 5.100000000000000E-001]
F([x]):
[1.726954073021381E-001, 1.839929588443123E-001]
Hansen Arithmetic:
[1.726250626886523E-001, 1.840059044076734E-001]
The linear form:
[0.178282383679,0.178348583418]+[1.1279994763506,1.1314641979673] r1
Where r is the interval radius of the interval vector x:
[r1]= [-5.0000000000E-03, 5.0000000000E-03]
```

Example 8.3. *Compute the function*

$$f(x) := x_1(1 + \exp(x_3 + \frac{x_2}{1 + 0.02542x_2}))$$

with $x_1 \in [0.19, 0.25]$, $x_2 \in [13, 13.5]$ *and* $x_3 \in [-10.7, -10.6]$.

We get the result

```
[x]
[ 1.899999999999999E-001, 2.500000000000000E-001]
[ 1.300000000000000E+001, 1.350000000000000E+001]
[-1.070000000000001E+001,-1.059999999999999E+001]
F([x]):
[2.584169245534380E-001, 4.089083173012873E-001]
Hansen Arithmetic:
[2.620563818087385E-001, 3.882137431927534E-001]
```

```
The linear form:
[0.325135062501,0.325135062502]+[1.4029183226,1.5528549728] r1 +
[0.026736055843,0.040530934349] r2 +[0.086893581899,0.12720595843] r3
Where r is the interval radius of the interval vector x:
[r1]=[-0.03, 0.03], [r2]= [-0.25, 0.25], [r3]=[-0.05, 0.05]
```

Example 8.4. *Compute the function*

$$f(x) := \frac{x_1 x_4}{(1+0.02542x_2)^2}\exp(x_3 + \frac{x_2}{1+0.02542x_2})$$

with $x_1 \in [0.19, 0.25]$, $x_2 \in [13, 13.5]$, $x_3 \in [-10.7, -10.6]$ *and* $x_4 \in [0.39, 0.40]$.

We get the result

```
[x]
[ 1.899999999999999E-001, 2.500000000000000E-001]
[ 1.300000000000000E+001, 1.350000000000000E+001]
[-1.070000000000001E+001,-1.059999999999999E+001]
[ 3.899999999999999E-001, 4.000000000000001E-001]
F([x]):
[1.986539349139782E-002, 4.777545128791166E-002]
Hansen Arithmetic:
[2.087197960405794E-002, 4.124400622486913E-002]
The linear form:
[0.03105084241,0.03106514342] +[0.11621908695,0.16621468448] r1 +
[0.00727048500,0.01142400417] r2 +[0.02567517932,0.03758661714] r3 +
[0.06587734361,0.09141807647] r4
Where r is the interval radius of the interval vector x:
[r1]=[-0.03, 0.03], [r2]= [-0.25, 0.25]
[r3]=[-0.05, 0.05], [r4]= [-0.005, 0.005]
```

9 Conclusion

We have introduced algorithms and C-XSC programs for a generalized interval arithmetic which reduces the inherent lack of sharpness of interval arithmetic to a second order effect, in addition to enclosing ranges of nonlinear interval expressions by linear interval forms. This method may not be useful if second order quantities are not truly negligible. Moreover, this method is of little value if the original data for a problem are reals rather than intervals of non-zero width.

References

[1] Hammer, R.; Hocks, M.; Kulisch, U.; Ratz, D.: C++ Toolbox for Verified Computing. Springer-Verlag, Berlin, Heidelberg, 1995.

[2] Hansen, E. R. : Generalized Interval Arithmetic. In Nickel, K. L. (ed.), Interval Mathematics, Vol. 29 of lecture notes in computer science, page 7-18, Springer-Verlag, Berlin, 1975.

[3] Hansen, E. R. : On Solving Systems of Equations Using Interval Arithmetic. Math. Comp., 22, page 374-384, 1968.

[4] Henrique de Figueiredo, L. ; Stolfi, J. : Affine arithmetic: concepts and applications. Numerical Algorithms 37: 147-158, 2004.

[5] Hofschuster, W.; Krämer, W.: C-XSC 2.0 - A C++ Class Library for Extended Scientific Computing. In: Numerical Software with Result Verification, R. Alt, A. Frommer, B. Kearfott, W. Luther (eds), Springer Lecture Notes in Computer Science 2991, 2004, pp. 15-35.

[6] Krämer, W.: Generalized Intervals and the Dependency Problem. Preprint 2006/4, Universität Wuppertal, Accepted for publication in PAMM, 2006.

[7] Krawczyk, R.; Neumaier, A.: Interval slopes for rational functions and associated centered forms. SIAM J. Numer. Anal., Vol. 22, No. 3, 1985.

[8] Link to the C-XSC library

`http://www.math.uni-wuppertal.de/˜xsc/`

[9] Moore, R.: Interval analysis. Prentice-Hall, Inc. Englewood Cliffs, N. J. , 1966.

[10] Neumaier, A.: Interval Methods for Systems of Equations. Cambridge University Press, Cambridge, 1990.