Reliable Computing (2006) **12:** 171–192
DOI: 10.1007/s11155-006-7217-4
© Springer 2006

A General Reliable Quadratic Form: An Extension of Affine Arithmetic

FRÉDÉRIC MESSINE* and AHMED TOUHAMI
ENSEEIHT–IRIT, CNRS–UMR 5505, 2 rue C. Camichel, 31071 Toulouse Cedex, France,
e-mail: {Frederic.Messine, Ahmed.Touhami}@enseeiht.fr

(Received: 23 June 2003; accepted: 18 June 2005)

Abstract. In this article, a new extension of *affine arithmetic* is introduced. This technique is based on a *quadratic form* named *general quadratic form*. We focus here on the computation of reliable bounds of a function over a hypercube by using this new tool. Some properties of first quadratic functions and then polynomial ones are reported. In order to show the efficiency of such a method, ten polynomial global optimization problems are presented and solved by using an interval branch-and-bound based algorithm.

1. Introduction

This paper is dedicated to the study of a new reliable automatic way to generate bounds of a (polynomial) function f over a hypercube $X \subseteq \mathbb{R}^n$. We denote by $f(X) = [\min_{x\in X} f(x), \max_{x\in X} f(x)]$ the range of f over the box X.

Interval arithmetic was developed in the 1960s by Moore [11]. This is a basic tool which allows to construct *interval inclusion functions*, denoted by F (corresponding to a function f), such that $f(X) \subseteq F(X) = [F^L(X), F^U(X)]$. These bounds are said to be reliable if no numerical error can produce a wrong bound.

The *natural extension* of a function into interval is an inclusion function. Using interval arithmetic and Taylor expansions (for instance, of the first order), it is possible to construct more efficient inclusion functions, [10]–[12].

Affine arithmetic was proposed and developed recently by Stolfi et al. [1]–[5], although a similar tool, the *generalized interval arithmetic*, has been developed in 1975 by Hansen [7]. Affine arithmetic was first applied to infographic problems and surface intersection problems [1]; the principle is to keep affine information during the computations of bounds. Like generalized interval arithmetic [7], affine arithmetic was developed to take into account the problems of the dependencies of the variables generated by interval computations. This tool permits to limit some negative effects of interval arithmetic, namely the link between different occurrences of the same variable. For example, $f(x) = x - x$ with $x \in X = [0, 1]$

* The work of the first author was also supported by the Laboratoire de Mathématiques Appliquées CNRS–FRE 2570, Université de Pau et des Pays de l'Adour, France, and by the Laboratoire d'Electrotechnique et d'Electronique Industrielle CNRS–UMR 5828, Group EM^3, INPT–ENSEEIHT.

gives $F(X) = [-1, 1] \neq 0$ using interval arithmetic. In that case, $[0, 0]$ is obtained using affine and generalized interval arithmetics. Affine arithmetic as defined in the first paper [1] was not reliable. However, by rounding some coefficients used in the affine operations, Stolfi et al. have obtained a reliable affine arithmetic; it is explained in detail in [5].

In a previous work of the first author [9], two new affine forms and one *quadratic form* were introduced. All these techniques (including affine arithmetic) were used in an *interval branch-and-bound algorithm* in order to improve its efficiency at finding the global minimum, see [3], [9]. The purpose here is to extend [9] by considering the *complete quadratic form* and by developing the associated operations. In this work, only the operations $+$, $-$, $\times$ are developed and then only polynomial functions can be considered. However, it should be possible and rather simple to extend this work in order to consider more general functions; this is due to the conversions of the general quadratic form into another one (or into an interval) and to the combination of these conversions and the calculus using the corresponding different forms.

This new form, named *general quadratic form*, has intrinsic properties mainly concerning quadratic functions. The purpose of this technique is to conserve affine information as well as quadratic information (about the error due to a non-affine operation) during the computations. Therefore, this added information permits to improve the quality of the computed bounds, in spite of the expansion of the complexity of such an algorithm. The gain of this technique is validated on some *global optimization polynomial problems*.

Section 2 is dedicated to describing affine arithmetic introduced by Stolfi et al. [1]–[5], and the forms proposed by the first author in [9]. Then, the general quadratic form and its associated operations are detailed in Section 3. Some theoretical properties of the efficiency of the bounds are reported in Section 4. Section 5 describes the way to render the bounds computed by this technique reliable. Section 6 is devoted to applying this technique to global optimization, by inserting it into one of the classical interval branch-and-bound algorithms. Therefore, some techniques based on this interval branch-and-bound algorithm which uses some different inclusion functions are compared on ten polynomial examples to show the efficiency of the general quadratic form in global optimization.

2. Affine Arithmetic and Associated Known Forms

The purpose of this section is to recall the affine arithmetic developed by Stolfi et al. [1]–[5] and the three affine and quadratic forms introduced in [9].

2.1. Standard Affine Arithmetic

A standard affine form, as defined in [1], is denoted by:

$$\widehat{x} = x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n = x_0 + \sum_{i=1}^{n} x_i\varepsilon_i, \tag{2.1}$$

where x_i are the real known coefficients (stored as floating point numbers) and ε_i are symbolic variables which are called noise-symbolic variables. The values of these variables are unknown but belong to the interval $[-1, 1]$.

The conversions between affine forms and intervals are performed as follows:

- **Interval $\longrightarrow$ Affine Form:**

$$\begin{aligned} X &= [x^L, x^U] \\ \longrightarrow \widehat{x} &= \frac{x^L + x^U}{2} + \frac{x^L - x^U}{2}\varepsilon_k, \end{aligned} \tag{2.2}$$

  where ε_k represents the uncertainty of the value x, and where k is the new increment of the new symbolic variable ($k \leftarrow k + 1$ after each conversion).

- **Affine Form $\longrightarrow$ Interval:**

$$\begin{aligned} \widehat{x} &= x_0 + \sum_{i=1}^{n} x_i \varepsilon_i \\ \longrightarrow X &= x_0 + \left(\sum_{i=1}^{n} |x_i| \right) \times [-1, 1]. \end{aligned} \tag{2.3}$$

All the standard operations such as $+$, $-$, $\times$, as well as other classical functions such as x^2, $\sqrt{x}$, $\sin(x)$, $\cos(x)$, are redefined for affine forms. In this work, only polynomial functions are considered, so only the operations $+$, $-$, $\times$ are necessary; for details on other operations and functions see [1]–[5].

Affine operations on affine forms are performed as follows:

$$\begin{aligned} \widehat{x} \pm \widehat{y} &= (x_0 \pm y_0) + \sum_{i=1}^{n} (x_i \pm y_i)\varepsilon_i, \\ a \pm \widehat{x} &= (a \pm x_0) \pm \sum_{i=1}^{n} x_i \varepsilon_i, \\ a \times \widehat{x} &= a x_0 + \sum_{i=1}^{n} a x_i \varepsilon_i, \end{aligned} \tag{2.4}$$

where $\widehat{x}$ and $\widehat{y}$ are affine forms and a is a real number.

It is obvious that multiplication, division, square and square root functions are not affine functions. Consequently, these non-affine results should be approximated by an affine form. Thus, another supplementary term $z_k \varepsilon_k$ must be inserted in order to represent the error due to the affine approximation [1]–[5]. For multiplication, we have:

$$\begin{aligned} \widehat{x} \times \widehat{y} &= \left(x_0 + \sum_{i=1}^{n} x_i \varepsilon_i \right) \times \left(y_0 + \sum_{i=1}^{n} y_i \varepsilon_i \right) \\ &= x_0 y_0 + \sum_{i=1}^{n} (x_0 y_i + x_i y_0)\varepsilon_i + \left(\sum_{i=1}^{n} x_i \varepsilon_i \right) \left(\sum_{i=1}^{n} y_i \varepsilon_i \right). \end{aligned}$$

An efficient approximation which keeps the inclusion property is:

$$\widehat{x} \times \widehat{y} = x_0 y_0 + \sum_{i=1}^{n} (x_0 y_i + x_i y_0)\varepsilon_i + \left(\sum_{i=1}^{n} |x_i| \times \sum_{i=1}^{n} |y_i| \right) \varepsilon_{n+1}. \tag{2.5}$$

This approximation is not the best one, but it seems to be the most efficient (a compromise between the quality of bounds and the complexity of the induced algorithm). It is the approximation used in affine arithmetic [1]–[5]. Note that a new symbolic variable ε_{n+1} ($k = n+1$) is included in the affine form with the coefficient $\sum_{i=1}^{n} |x_i| \times \sum_{i=1}^{n} |y_i|$, which is the upper limit due to the affine approximation.

Remark 2.1. In the process of performing a non-affine operation, such as a multiplication, the affine form increases in size by one supplementary new term. This technical difficulty is discussed and solved in [9].

Therefore, by using affine arithmetic, it is possible to define a new *inclusion function* of f, denoted by **AF**:

1. Let us consider the hypercube $X \subseteq \mathbb{R}^n$ and the n intervals X_i converted into n affine forms:
$$X_i \longrightarrow \widehat{x}_i = \frac{{x_i}^L + {x_i}^U}{2} + \frac{{x_i}^L - {x_i}^U}{2}\varepsilon_i.$$
A new symbolic variable ε_i is introduced for each conversion.
2. Each occurrence of a variable x_i in an expression of f is replaced by the corresponding affine form $\widehat{x}_i$ and all the operations $+, -, \times$ are replaced by the corresponding affine operations.
3. The affine computations are performed and the affine result $\widehat{r}$ is obtained.
4. The resulting affine form $\widehat{r}$ is then converted into the interval $[\widehat{r}\,] = [r^L, r^U]$ which is an enclosure of the direct image of f over the considered box X.

Hence, $f(X) \subseteq \mathbf{AF}(X) = [\widehat{r}\,]$.

The affine arithmetic so defined is not reliable. However by performing efficient rounding operations inside the affine computations, a reliable affine arithmetic was defined; see [5] for details.

2.2. Extended Forms

These forms are extensions of the standard affine form. They allow to reduce the difficulties induced by the use of this tool: the fact that affine forms grow by performing non-affine operations and the fact that positive and negative errors cannot be computed separately. The three forms presented in [9] are recalled hereafter; for the arithmetic using these three forms, see [9].

- **First Affine Form.** The main idea is to conserve all the errors in a single symbolic variable ε_{n+1}. This form has the advantage that the resulting form does

not grow by performing a non-affine operation. The bounds produced in this way are equal to those produced by the inclusion function **AF**. The first affine form can be formulated as follows:

$$\widehat{x} = x_0 + \sum_{i=1}^{n} x_i \varepsilon_i + x_{n+1}\varepsilon_{n+1},$$

where the new variable $\varepsilon_{n+1} \in [-1, 1]$ represents the totality of the errors introduced by performing non-affine operations; only one supplementary term is added. This allows us to control the size of the resulting affine forms during the computations. The corresponding inclusion function based on this form is denoted by $\mathbf{AF}_1$.

- **Second Affine Form.** Starting from $\mathbf{AF}_1$, a second form is generated by introducing two new symbolic variables in order to control the positive and negative errors. This second affine form is formulated as follows:

$$\widehat{x} = x_0 + \sum_{i=1}^{n} x_i \varepsilon_i + x_{n+1}\varepsilon_{n+1} + x_{n+2}\varepsilon_{n+2} + x_{n+3}\varepsilon_{n+3},$$

where the positive errors are represented by $\varepsilon_{n+1} \in [0, 1]$ and the negative ones by $\varepsilon_{n+2} \in [-1, 0]$; $\varepsilon_{n+3} \in [-1, 1]$ represents the other possible errors. All the coefficients x_{n+1}, x_{n+2}, and x_{n+3} are always positive numbers. The corresponding inclusion function based on this form is denoted by $\mathbf{AF}_2$.

- **First Quadratic Form.** A first quadratic affine form was also generated in [9] by introducing symbolic variables in order to keep the information about the square terms during some quadratic or polynomial computations:

$$\widehat{\widehat{x}} = x_0 + \sum_{i=1}^{n} (x_i \varepsilon_i + x_{i+n}\varepsilon_i^2) + x_{2n+1}\varepsilon_{2n+1} + x_{2n+2}\varepsilon_{2n+2} + x_{2n+3}\varepsilon_{2n+3},$$

where $\varepsilon_i^2 = \varepsilon_{i+n} \in [0, 1]$, $\varepsilon_{2n+1} \in [0, 1]$, $\varepsilon_{2n+2} \in [-1, 0]$, and $\varepsilon_{2n+3} \in [-1, 1]$. In this form, a quadratic supplementary information is kept by using the terms ε_i^2. The corresponding inclusion function based on this quadratic form is denoted by **QF**.

3. General Quadratic Form

In this section, the general quadratic form is defined. It is an extension of the previous extended forms which generated the inclusion functions **AF**, $\mathbf{AF}_1$, $\mathbf{AF}_2$, and **QF**. In this paper, the complete quadratic form is considered, in contrast to **QF** which is a partial quadratic form.

A general quadratic form is represented by $\widehat{\widehat{x}}$:

$$\begin{aligned} \widehat{\widehat{x}} &= \varepsilon^T A \varepsilon + b^T \varepsilon + c + e^+ \varepsilon_{n+1} + e^- \varepsilon_{n+2} + e\varepsilon_{n+3} \\ &= \sum_{i,j=1}^{n} a_{ij}\varepsilon_i\varepsilon_j + \sum_{i=1}^{n} b_i\varepsilon_i + c + e^+\varepsilon_{n+1} + e^-\varepsilon_{n+2} + e\varepsilon_{n+3}, \end{aligned} \tag{3.1}$$

where $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$, $c \in \mathbb{R}$, $(e^+, e^-, e) \in (\mathbb{R}^+)^3$ and $\varepsilon_i \in [-1,1]$, $\forall i \in \{1,\ldots,n\}$, $\varepsilon_{n+1} \in [0,1]$, $\varepsilon_{n+2} \in [-1,0]$, and $\varepsilon_{n+3} \in [-1,1]$. The symbolic variables ε_{n+1}, ε_{n+2}, and ε_{n+3} represent the noise for the errors during non-affine computations.

The conversions between intervals, affine forms, and general quadratic forms are performed as follows:

- **Interval $\longrightarrow$ General Quadratic Form:**

$$\begin{aligned} X &= [x^L, x^U] \\ \longrightarrow \widehat{\widehat{x}} &= \varepsilon^T A \varepsilon + b^T \varepsilon + c + e^+\varepsilon_{n+1} + e^-\varepsilon_{n+2} + e\varepsilon_{n+3} \end{aligned}$$

$$\text{with } \begin{cases} A &= 0, \\ b &= \left(0,\ldots,0,\dfrac{x^L - x^U}{2},0,\ldots,0\right), \\ c &= \dfrac{x^L + x^U}{2}, \\ e^+ &= 0, \\ e^- &= 0, \\ e &= 0. \end{cases}$$

  The rank i where $b_i = \frac{x^L - x^U}{2}$ is generally determined by the variable x. When a vector $x \in \mathbb{R}^n$ is considered, then the rank i corresponds to x_i.

- **General Quadratic Form $\longrightarrow$ Interval:**

$$\begin{aligned} \widehat{\widehat{x}} &= \varepsilon^T A \varepsilon + b^T \varepsilon + c + e^+\varepsilon_{n+1} + e^-\varepsilon_{n+2} + e\varepsilon_{n+3} \\ &= \sum_{i,j=1}^{n} a_{ij}\varepsilon_i\varepsilon_j + \sum_{i=1}^{n} b_i\varepsilon_i + c + e^+\varepsilon_{n+1} + e^-\varepsilon_{n+2} + e\varepsilon_{n+3} \\ \longrightarrow X &= \sum_{\substack{i,j=1 \\ i\neq j}}^{n} [-|a_{ij}|, |a_{ij}|] + \sum_{i=1}^{n}[-|b_i|, |b_i|] + \sum_{i=1}^{n} a_{ii}[0,1] + [|c|, |c|] \\ &\quad + [0, e^+] + [-e^-, 0] + [-e, e]. \end{aligned}$$

- **General Quadratic Form $\longrightarrow$ Affine Form:**

$$\begin{aligned} \widehat{\widehat{x}} &= \varepsilon^T A \varepsilon + b^T \varepsilon + c + e^+\varepsilon_{n+1} + e^-\varepsilon_{n+2} + e\varepsilon_{n+3} \\ &= \sum_{i,j=1}^{n} a_{ij}\varepsilon_i\varepsilon_j + \sum_{i=1}^{n} b_i\varepsilon_i + c + e^+\varepsilon_{n+1} + e^-\varepsilon_{n+2} + e\varepsilon_{n+3} \\ \longrightarrow \widehat{x} &= c + \sum_{i=1}^{n} b_i\varepsilon_i + \alpha\varepsilon_{n+1}, \end{aligned}$$

  with $\alpha = \max\{|I^L|, |I^U|\}$, where $I = [-(e+e^-), (e+e^+)] + \sum_{i=1}^{n}\sum_{j=1}^{n}[-|a_{ij}|, |a_{ij}|]$.

- **Affine Form $\longrightarrow$ General Quadratic Form:**

$$\begin{aligned}
\hat{x} &= x_0 + \sum_{i=1}^{n} x_i \varepsilon_i \\
\longrightarrow \hat{\hat{x}} &= \varepsilon^T A \varepsilon + b^T \varepsilon + c + e^+ \varepsilon_{m+1} + e^- \varepsilon_{m+2} + e \varepsilon_{m+3}, \\
&\text{with} \begin{cases}
A = 0, \\
b_i = x_i, & \forall i \in \{1, \ldots, m\}, \\
b_i = 0, & \forall i \in \{m+1, \ldots, n\}, \\
c = x_0, \\
e = \sum_{i=n+1}^{m} |x_i|, \\
e^+ = 0, \\
e^- = 0.
\end{cases}
\end{aligned}$$

Remark 3.1. Note that m can be different from n. If $m = n$, then $e = 0$. Otherwise, it is interesting to consider $m < n$ because the affine form $\hat{x}$, generated after some non-affine operations, will be longer than the number of variables (by adding some extra error terms) whereas generally, the size of our general quadratic form must correspond to the number of variables.

The affine operations on general quadratic forms are performed as follows. Let $\hat{\hat{x}}$ and $\hat{\hat{y}}$ be two general quadratic forms and a a real number, with

$$\begin{aligned}
\hat{\hat{x}} &= \varepsilon^T A_x \varepsilon + b_x^T \varepsilon + c_x + e_x^+ \varepsilon_{n+1} + e_x^- \varepsilon_{n+2} + e_x \varepsilon_{n+3}, \\
\hat{\hat{y}} &= \varepsilon^T A_y \varepsilon + b_y^T \varepsilon + c_y + e_y^+ \varepsilon_{n+1} + e_y^- \varepsilon_{n+2} + e_y \varepsilon_{n+3}.
\end{aligned}$$

We obtain:

$$\begin{aligned}
-\hat{\hat{x}} &= -\varepsilon^T A_x \varepsilon - b_x^T \varepsilon - c_x + e_x^- \varepsilon_{n+1} + e_x^+ \varepsilon_{n+2} + e_x \varepsilon_{n+3}, \\
\hat{\hat{x}} + \hat{\hat{y}} &= \varepsilon^T (A_x + A_y) \varepsilon + (b_x + b_y)^T \varepsilon + (c_x + c_y) + (e_x^+ + e_y^+) \varepsilon_{n+1} \\
&\quad + (e_x^- + e_y^-) \varepsilon_{n+2} + (e_x + e_y) \varepsilon_{n+3}, \\
\hat{\hat{x}} - \hat{\hat{y}} &= \varepsilon^T (A_x - A_y) \varepsilon + (b_x - b_y)^T \varepsilon + (c_x - c_y) + (e_x^+ + e_y^-) \varepsilon_{n+1} \\
&\quad + (e_x^- + e_y^+) \varepsilon_{n+2} - (e_x + e_y) \varepsilon_{n+3}, \\
\hat{\hat{x}} \pm a &= \varepsilon^T A_x \varepsilon + b_x^T \varepsilon + (c_x \pm a) + e_x^+ \varepsilon_{n+1} + e_x^- \varepsilon_{n+2} + e_x \varepsilon_{n+3}, \\
a \times \hat{\hat{x}} &= \begin{cases}
\varepsilon^T (a \times A_x) \varepsilon + (a \times b_x)^T \varepsilon + a \times c_x \\
\qquad + a \times e_x^+ \varepsilon_{n+1} + a \times e_x^- \varepsilon_{n+2} + a \times e_x \varepsilon_{n+3}, & \text{if } a > 0, \\
\varepsilon^T (a \times A_x) \varepsilon + (a \times b_x)^T \varepsilon + a \times c_x \\
\qquad + |a| \times e_x^- \varepsilon_{n+1} + |a| \times e_x^+ \varepsilon_{n+2} + |a| \times e_x \varepsilon_{n+3}, & \text{if } a < 0.
\end{cases}
\end{aligned}$$

For multiplication (a non-affine operation), the following computations are performed:

$$\widehat{\widehat{x}} \times \widehat{\widehat{y}} = \varepsilon^T A_\times \varepsilon + b_\times^T \varepsilon + c_\times + e_\times^+ \varepsilon_{n+1} + e_\times^- \varepsilon_{n+2} + e_\times \varepsilon_{n+3}, \tag{3.2}$$

$$\text{with} \begin{cases} A_\times = c_y A_x + c_x A_y + b_x b_y^T, \\ b_\times = c_y b_x + c_x b_y, \\ c_\times = c_x c_y. \end{cases}$$

It remains to explain how the errors $e_\times^+$, $e_\times^-$, $e_\times$ are computed. These computations are done in the following four steps. In order to simplify the equations, we use the following symbols, for all real x:

$$[x]^+ = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise} \end{cases} \qquad \text{and} \qquad [x]^- = \begin{cases} |x| & \text{if } x < 0, \\ 0 & \text{otherwise.} \end{cases}$$

1. Errors of the type $\varepsilon^T A_x \varepsilon \times \varepsilon^T A_y \varepsilon$.

   The positive terms are:

   $$e_\times^+ := \sum_{i=1}^{n} [(a_x)_{ii}(a_y)_{ii}]^+ + \sum_{\substack{i,j=1 \\ j \neq i}}^{n} \left([(a_x)_{ii}(a_y)_{jj}]^+ + [(a_x)_{ij}(a_y)_{ij}]^+ + [(a_x)_{ij}(a_y)_{ji}]^+ \right).$$

   The negative terms are:

   $$e_\times^- := \sum_{i=1}^{n} [(a_x)_{ii}(a_y)_{ii}]^- + \sum_{\substack{i,j=1 \\ j \neq i}}^{n} \left([(a_x)_{ii}(a_y)_{jj}]^- + [(a_x)_{ij}(a_y)_{ij}]^- + [(a_x)_{ij}(a_y)_{ji}]^- \right).$$

   The other terms are:

   $$e_\times := \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} |(a_x)_{ij}(a_y)_{kl}|,$$

   where i, j, k, l are not in the previous four cases, i.e. $i = j = k = l$, $i = j$ and $k = l$, $i = k$ and $j = l$, $i = l$ and $j = k$.

2. Errors of the type $b_x^T \varepsilon \times \varepsilon^T A_y \varepsilon$ and $b_y^T \varepsilon \times \varepsilon^T A_x \varepsilon$.

   In that case, there is no positive or negative error, the other terms are:

   $$e_\times := e_\times + \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \left(|(b_x)_i (a_y)_{kl}| + |(b_y)_i (a_x)_{kl}| \right).$$

3. Errors due to the multiplication of an element of the matrix A by an error e^+, e^-, or e.

   The positive terms are:

   $$e_\times^+ := e_\times^+ + \sum_{i=1}^{n} \left(e_x^+ [(a_y)_{ii}]^+ + e_y^+ [(a_x)_{ii}]^+ + e_x^- [(a_y)_{ii}]^- + e_y^- [(a_x)_{ii}]^- \right).$$

The negative terms are:

$$e_\times^- := e_\times^- + \sum_{i=1}^{n} \left(e_x^-[(a_y)_{ii}]^+ + e_y^-[(a_x)_{ii}]^+ + e_x^+[(a_y)_{ii}]^- + e_y^+[(a_x)_{ii}]^- \right).$$

The other terms are:

$$e_\times := e_\times + (e_y^+ + e_y^- + e_y) \times \left(\sum_{i=1}^{n} \left(\left(|(b_x)_i| + |(b_y)_i| \right) + \sum_{\substack{j=1 \\ j \neq i}}^{n} \left(|(a_x)_{ij}| + |(a_y)_{ij}| \right) \right) \right).$$

4. Errors due to the multiplication of the errors, or the multiplication of an error and a constant.

   The positive terms are:

   $$e_\times^+ := e_\times^+ + e_x^+[c_y]^+ + e_y^+[c_x]^+ + e_x^-[c_y]^- + e_y^-[c_x]^- + e_x^+ e_y^+ + e_x^- e_y^-.$$

   The negative terms are:

   $$e_\times^- := e_\times^- + e_x^+[c_y]^- + e_y^+[c_x]^- + e_x^-[c_y]^+ + e_y^-[c_x]^+ + e_x^+ e_y^- + e_x^- e_y^+.$$

   The other terms are:

   $$e_\times := e_\times + e_x e_y^+ + e_x e_y^- + e_y e_x^+ + e_y e_x^- + 2 e_x e_y.$$

The construction of the corresponding inclusion function proceeds, as in the standard affine arithmetic, by replacing the forms, the operations, and the conversions. We denote the inclusion function induced by the general quadratic form by **GQF**.

4. Properties and Comparisons

This section discusses some properties of the bounds produced by **GQF** for quadratic and polynomial functions. This leads us to some comparisons between the different forms.

X represents the hypercube where the function is studied. The following notations are used during this section: $X = X_1 \times \cdots \times X_i \times \cdots \times X_n$, with $X_i = [x_i^L, x_i^U] \subseteq \mathbb{R}$ and $m_i = x_i^L + x_i^U$, $w_i = x_i^L - x_i^U$, such that $X_i = \left[\frac{m_i + w_i}{2}, \frac{m_i - w_i}{2} \right]$.

4.1. Properties for Quadratic Functions

Let us denote $f(x) = x^T A x + b^T x + c$, a quadratic function from $X \subseteq \mathbb{R}^n$ to $\mathbb{R}$. Change the variables $x_i \in X_i$ by $\varepsilon_i \in [-1, 1]$, such that $x_i = \frac{m_i}{2} + \frac{w_i}{2}\varepsilon_i$. Therefore a new quadratic function depending on variables ε_i is obtained. Let us denote this

quadratic function by $g(\varepsilon)$. By the construction of g, we have that $f(x) = g(\varepsilon)$ for all $x \in X$ (for each variable $x_i \in X_i$ we have a corresponding variable $\varepsilon_i = \frac{2x_i - m_i}{w_i}$).

Let us denote g by:

$$g(\varepsilon) = \varepsilon^T A_x \varepsilon + b_x^T \varepsilon + c_x = f(x).$$

We obtain:

$$\begin{cases} \varepsilon^T A_x \varepsilon &= \dfrac{1}{4} \displaystyle\sum_{i,j=1}^{n} a_{ij} w_i w_j \varepsilon_i \varepsilon_j, \\ b_x^T \varepsilon &= \displaystyle\sum_{i=1}^{n} \left\{ \frac{1}{2} b_i w_i + \frac{1}{4} \sum_{j=1}^{n} (a_{ij} + a_{ji}) m_i w_j \right\} \varepsilon_i, \\ c_x &= \dfrac{1}{2} \displaystyle\sum_{i=1}^{n} \left[\frac{1}{2} \sum_{j=1}^{n} a_{ij} m_j + b_i \right] m_i + c. \end{cases}$$

PROPOSITION 4.1. *Considering a quadratic function f and its associated quadratic function g as explained above, we have:*

$$\min_{x \in X} f(x) = \min_{\varepsilon \in [-1,1]^n} g(\varepsilon) \qquad \textit{and} \qquad \max_{x \in X} f(x) = \max_{\varepsilon \in [-1,1]^n} g(\varepsilon).$$

Proof. The proof is evident, because $f(x) = g(\varepsilon)$ by definition and $x \in X$ implies $\varepsilon \in [-1,1]^n$. □

THEOREM 4.1. *If matrix A is symmetric and positive definite and $x_i^L \neq x_i^U$, $\forall i \in \{1, \ldots, n\}$ (equivalent to $w_i \neq 0$, $\forall i$), then $A_x = \frac{1}{4}(a_{ij} w_i w_j)_{1 \leq i,j \leq n}$ is also symmetric and positive definite, and vice versa.*

Proof. Because $\forall (i,j) \in \{1, \ldots, n\}^2$, $w_j w_i > 0$, and the set of symmetric matrices $\mathbb{S}^n$ such that $\mathbb{S}^n = \{A \in \mathbb{R}^{n \times n} \mid A = A^T\}$ is a subspace of $\mathbb{R}^{n \times n}$. □

Remark 4.1. If the matrix A is symmetric then $b_x^T \varepsilon = \frac{1}{2} \displaystyle\sum_{i=1}^{n} \left[\sum_{j=1}^{n} a_{ij} m_j w_i + b_i w_i \right] \varepsilon_i$.

By denoting $\mathbf{QNE}_f$ the inclusion function obtained by the natural extension into interval of the expression f written as a quadratic standard form $f(x) = x^T A x + b^T x + c$, we obtain:

$$\mathbf{QNE}_f(X) = \sum_{\substack{i,j=1 \\ i \neq j}}^{n} a_{ij} X_i X_j + \sum_{i=1}^{n} (a_{ii} X_i^2 + b_i X_i) + c.$$

Attention must be paid to the way this interval result is obtained because $X_i^2 \subseteq X_i \times X_i$. Similarly, $\mathbf{QNE}_g$ denotes the inclusion function when g is considered instead of f.

PROPOSITION 4.2. *For all* $X \subseteq \mathbb{R}^n$, $\mathbf{GQF}(X) = \mathbf{QNE}_g([-1,1]^n)$.

Proof. This is due to the construction of the arithmetic of the general quadratic form. □

PROPOSITION 4.3. *If each component of the center of* X *is equal to* 0 *then* $\mathbf{GQF}(X) = \mathbf{QNE}_f(X)$.

Proof. If each component of the center of X is equal to 0 then $m_i = 0$ and then, $w_i\varepsilon_i = 2x_i \ \forall i \in \{1,\ldots,n\}$. Furthermore, $x_i^L = -x_i^U$ and then $\frac{w_i}{2} \times [-1,1] = X_i$. This implies that $\mathbf{QNE}_f(X) = \mathbf{QNE}_g([-1,1]^n)$ and the result follows. □

It is interesting to compare **GQF** with $\mathbf{QNE}_f$ for quadratic functions. We have performed a lot of numerical tests (> 500) using the MATLAB toolbox INTLAB, see [8]. In all cases, the two bounds were better using the **GQF** inclusion function than directly $\mathbf{QNE}_f$. Our quadratic function f has been generated using the command `rand` which yields uniformly distributed random numbers. Concerning the hypercube X, each interval component is defined by the midpoint and the radius which are produced randomly, using the commands `midrad` and `rand`; for instance, $X_i =$ `midrad(rand(1),5*rand(1))` $\subseteq [-5,6]$. The following randomly generated example of size 6 illustrates this:

$$A = \begin{pmatrix} -1.471 & 1.037 & 4.318 & 1.721 & 0.028 & 1.822 \\ 3.131 & -2.278 & -0.340 & 3.381 & 2.094 & -1.972 \\ -4.901 & -3.011 & -0.813 & -4.803 & -0.711 & 0.416 \\ -3.611 & -4.847 & 3.462 & 1.812 & -1.953 & -3.491 \\ -2.972 & 2.467 & 0.251 & -1.205 & -3.103 & 1.979 \\ -3.012 & -0.549 & -2.973 & 3.318 & -3.065 & -1.216 \end{pmatrix},$$

$$b = \begin{pmatrix} 3.600 \\ 3.536 \\ 0.935 \\ -0.034 \\ 3.990 \\ 3.216 \end{pmatrix},$$

$$c = 0,$$

$$X = ([-2.640, 5.539],\ [-0.107, 3.312],\ [-3.808, -0.396],\ [-3.294, 3.976],\ [-6.099, 2.285],\ [-1.171, 2.532]).$$

The bounds are $\mathbf{GQF}(X) = [-797.6, 722.8]$ and $\mathbf{QNE}_f(X) = [-1181.1, 1094.5]$.

In the above examples, we obtained that $\mathbf{QNE}_f(X) \not\subseteq \mathbf{GQF}(X)$. Furthermore, we had $\mathbf{GQF}(X) \subseteq \mathbf{QNE}_f(X)$ for all the randomly generated examples.

PROPOSITION 4.4. *There exist such quadratic functions f and such X that are boxes of $\mathbb{R}^n$ for which* $\mathbf{GQF}(X) \not\subseteq \mathbf{QNE}_f(X)$.

Proof. Let us consider the following quadratic function:

$$f(x) = x^T \begin{pmatrix} 9 & 5 & -6 & 5 \\ -1 & -8 & 1 & 8 \\ 8 & 0 & 9 & -6 \\ 8 & 5 & 6 & 0 \end{pmatrix} x + \begin{pmatrix} 1 \\ 4 \\ -8 \\ 8 \end{pmatrix}^T x - 3, \qquad X = [-0.1, 1.7]^4.$$

The natural extension into intervals gives $\mathbf{QNE}_f(X) = [-86.411, 207.071]$, and the inclusion function based on the general quadratic form, $\mathbf{GQF}(X) = [-92.171, 157.311]$.

In this case, $\mathbf{QNE}_f(X)^L > \mathbf{GQF}(X)^L$ and $\mathbf{QNE}_f(X)^U > \mathbf{GQF}(X)^U$. □

Remark 4.2. The efficiency of the two inclusion functions $\mathbf{QNE}_f$ and **GQF** is difficult to compare. Sometimes $\mathbf{QNE}_f$ is the most efficient. Nevertheless, **GQF** produces the best bounds in general. The counter-example in Proposition 4.4 was difficult to construct, and actually we don't have an example where both bounds are more efficient using directly $\mathbf{QNE}_f$.

THEOREM 4.2. *If f is a quadratic function then the inclusion functions* $\mathbf{AF}_2$, **QF**, *and* **GQF** *will produce exactly the same bounds:*

$$\mathbf{AF}_2(X) = \mathbf{QF}(X) = \mathbf{GQF}(X) \subseteq \mathbf{AF}(X) = \mathbf{AF}_1(X), \qquad \forall X \subseteq \mathbb{R}^n.$$

Proof. By construction $\mathbf{AF}(X) = \mathbf{AF}_1(X)$ for all $X \subseteq \mathbb{R}^n$, see [9].

By assumption, f is a quadratic function. Denote by $g_{\mathbf{AF}_1}$, $g_{\mathbf{AF}_2}$, $g_{\mathbf{QF}}$, and $g_{\mathbf{GQF}}$ the resulting forms before their conversion into intervals after the computation of the inclusion functions $\mathbf{AF}_1$, $\mathbf{AF}_2$, **QF**, **GQF**.

Hence, we obtain:

$$\begin{aligned} g_{\mathbf{GQF}}(\varepsilon) &= g(\varepsilon) = \varepsilon^T A_x \varepsilon + b_x^T \varepsilon + c_x, \\ g_{\mathbf{QF}}(\varepsilon) &= \sum_{i=1}^{n} \left((b_x)_i \varepsilon_i + (a_x)_{ii} \varepsilon_i^2 \right) + 0\varepsilon_{n+1} + 0\varepsilon_{n+2} + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ i \neq j}}^{n} |a_{ij}| \varepsilon_{n+3}, \\ g_{\mathbf{AF}_2}(\varepsilon) &= \sum_{i=1}^{n} (b_x)_i \varepsilon_i + \sum_{i=1}^{n} [(a_x)_{ii}]^+ \varepsilon_{n+1} + \sum_{i=1}^{n} [(a_x)_{ii}]^- \varepsilon_{n+2} + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ i \neq j}}^{n} |a_{ij}| \varepsilon_{n+3}. \end{aligned}$$

Denoting by $\mathbf{NE}_f$ the extension of a function f into interval, we get:

$$\begin{aligned}\mathbf{NE}_{g_{\mathbf{GQF}}}([-1,1]^n) &= \mathbf{NE}_{g_{\mathbf{GQF}}}([-1,1]^n \times [0,1] \times [-1,0] \times [-1,1]) \\ &= \mathbf{NE}_{g_{\mathbf{AF}_2}}([-1,1]^n \times [0,1] \times [-1,0] \times [-1,1]),\end{aligned}$$

and then $\mathbf{GQF}(X) = \mathbf{QF}(X) = \mathbf{AF}_2(X)$. □

4.2. General Properties

Let us now consider a polynomial function f from $\mathbb{R}^n$ to $\mathbb{R}$. The following properties could be extended to other types of functions, but this is not considered in this work.

PROPOSITION 4.5.

$$\mathbf{GQF}(X) \subseteq \mathbf{QF}(X) \subseteq \mathbf{AF}_2(X) \subseteq \mathbf{AF}_1(X) = \mathbf{AF}(X).$$

Proof. This property is due to the fact that the form of $\mathbf{AF}_1$ is inside the form of $\mathbf{AF}_2$. The form of **QF** includes the form of $\mathbf{AF}_2$ and is inside the form of **GQF**. Therefore, the inclusion of the corresponding inclusion functions follows. □

The function g denotes the corresponding quadratic function due to the computations of the inclusion function **GQF**, and h is the complete function:

$$\begin{aligned}h(\varepsilon) &= \varepsilon^T A_x \varepsilon + b_x^T \varepsilon + c_x + e^+ \varepsilon_{n+1} + e^- \varepsilon_{n+2} + e\varepsilon_{n+3} \\ &= f(x) = g(\varepsilon) + e^+ \varepsilon_{n+1} + e^- \varepsilon_{n+1} + e\varepsilon_{n+3},\end{aligned}$$

where $h : \mathbb{R}^{n+3} \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}$.

The problem comes from the fact that in the general case, the coefficients e^+, e^-, and e are not necessarily equal to 0 and therefore, there is no direct link between the variables x and the three symbolic variables ε_{n+1}, ε_{n+2}, and ε_{n+3} which are generated during non-affine and non-quadratic computations. However in such case, we obtain the following theorem:

THEOREM 4.3.

$$\begin{aligned}\min_{x \in X} f(x) \geq \min_{\substack{\varepsilon \in [-1,1]^n,\, \varepsilon_{n+1} \in [0,1], \\ \varepsilon_{n+2} \in [-1,0],\, \varepsilon_{n+3} \in [-1,1]}} h(\varepsilon) &= \min_{\varepsilon \in [-1,1]^n} g(\varepsilon) - e^- - e \\ &\geq \mathbf{NE}_h^L([-1,1]^n \times [0,1] \times [-1,0] \times [-1,1])\end{aligned}$$

and

$$\begin{aligned}\max_{x \in X} f(x) \leq \max_{\substack{\varepsilon \in [-1,1]^n,\, \varepsilon_{n+1} \in [0,1], \\ \varepsilon_{n+2} \in [-1,0],\, \varepsilon_{n+3} \in [-1,1]}} h(\varepsilon) &= \max_{\varepsilon \in [-1,1]^n} g(\varepsilon) + e^+ + e \\ &\leq \mathbf{NE}_h^U([-1,1]^n \times [0,1] \times [-1,0] \times [-1,1]).\end{aligned}$$

Proof. The demonstration is obvious because it follows from the definitions of h and g. □

It is interesting to consider $g(\varepsilon)$ because if the matrix A_x is symmetric and positive definite then the unique solution will produce efficient bounds only by solving a linear system.

Remark 4.3. We note that

$$\begin{aligned}\mathbf{NE}_h([-1,1]^n \times [0,1] \times [-1,0] \times [-1,1]) \\ = [\mathbf{NE}_g^L([-1,1])^n - e^- - e,\ \mathbf{NE}_g^U([-1,1])^n + e^+ + e].\end{aligned}$$

EXAMPLE 4.1. The properties of this general quadratic form can be illustrated by this simple example: $f(x_1, x_2) = x_1^2 x_2 - x_1 x_2^2$ over $[-1,3]^2$. We have:

$$\begin{aligned} x_1^2 x_2 - x_1 x_2^2 &= (1-2\varepsilon_1)^2(1-2\varepsilon_2) - (1-2\varepsilon_1)(1-2\varepsilon_2)^2 \\ &= (1-4\varepsilon_1+4\varepsilon_1^2)(1-2\varepsilon_2) - (1-4\varepsilon_2+4\varepsilon_2^2)(1-2\varepsilon_1) \\ &= (1-4\varepsilon_1+4\varepsilon_1^2-2\varepsilon_2+8\varepsilon_1\varepsilon_2-8\varepsilon_1^2\varepsilon_2) \\ &\quad +(-1+4\varepsilon_2-4\varepsilon_2^2+2\varepsilon_1-8\varepsilon_1\varepsilon_2+8\varepsilon_1\varepsilon_2^2) \\ &= -2\varepsilon_1+2\varepsilon_2+4\varepsilon_1^2-4\varepsilon_2^2-8\varepsilon_1^2\varepsilon_2+8\varepsilon_1\varepsilon_2^2. \end{aligned}$$

- $\mathbf{AF}(X) = \mathbf{AF}_1(X) = [-2\varepsilon_1 + 2\varepsilon_2 + 40\varepsilon_3] = [-44, 44]$, with $\varepsilon_{n+1} = \varepsilon_3 \in [-1,1]$.
- $\mathbf{AF}_2(X) = [-2\varepsilon_1 + 2\varepsilon_2 + 4\varepsilon_1^2 - 4\varepsilon_2^2 + 32\varepsilon_5] = [-40, 40] = \mathbf{QF}(X)$, with $\varepsilon_{n+3} = \varepsilon_5 = \varepsilon_{2n+3}\varepsilon_7 \in [-1,1]$.
- $\mathbf{GQF}(X) = [-2\varepsilon_1 + 2\varepsilon_2 + 4\varepsilon_1^2 - 4\varepsilon_2^2 + 16\varepsilon_5] = [-24, 24]$, with $\varepsilon_{n+3} = \varepsilon_5 \in [-1,1]$.
- $\mathbf{QNE}_f(X) = X_1^2 X_2 - X_1 X_2^2 = [-1,3]^2[-1,3] - [-1,3]^2[-1,3] = [-9,27] - [-9,27] = [-36,36]$.

In that case, $\mathbf{QNE}_f$ is better than $\mathbf{AF}$, $\mathbf{AF}_1$, $\mathbf{AF}_2$, and $\mathbf{QF}$ but $\mathbf{GQF}$ is the most efficient.

5. Reliable General Quadratic Form

In [5], Stolfi et al. have developed a reliable arithmetic based on affine forms. The way they proceed is based on some rounding of the floating point coefficients of the intermediate resulting affine forms for each operation (affine or not). A similar way is to convert all the floating point components into floating point intervals which enclose the real values, i.e. use the rounded interval arithmetic inside the operations of the affine forms. By this, the generalized interval analysis is constructed [7]. In this work, only the second way is considered for affine forms; for a thorough survey on reliable affine arithmetic refer to [5].

This section is dedicated to the representation of the general quadratic form in such a way that it becomes reliable with regard to numerical errors. The basic idea is to replace the floating point coefficients in quadratic forms by floating point interval coefficients:

- the real matrix A is replaced by an interval matrix $\mathbf{A}$, where each element $\mathbf{a}_{ij} = [\underline{a_{ij}}, \overline{a_{ij}}]$ and where $\underline{a_{ij}}, \overline{a_{ij}}$ are the closest inferior and superior floating point numbers to the real coefficients a_{ij},
- the real vector b is replaced by an interval vector $\mathbf{b}$ where each component $\mathbf{b}_i = [\underline{b_i}, \overline{b_i}]$ and where $\underline{b_i}, \overline{b_i}$ are the closest inferior and superior floating point numbers to the real coefficients b_i,
- the real constant c is replaced by an interval $\mathbf{c} = [\underline{c}, \overline{c}]$ where $\underline{c}$ and $\overline{c}$ are the closest inferior and superior floating point numbers to c,
- the real error e is replaced by an interval $\mathbf{E} = [e^L, e^U]$ where e^L and e^U are the floating point numbers enclosing all the errors due to the non-quadratic operations.

The reliable general quadratic form is represented by $\hat{\hat{\mathbf{x}}}$:

$$\begin{aligned}\hat{\hat{\mathbf{x}}} &= \varepsilon^T \mathbf{A} \varepsilon + \mathbf{b}^T \varepsilon + \mathbf{c} + \mathbf{E} \\ &= \sum_{i,j=1}^{n} [\underline{a_{ij}}, \overline{a_{ij}}] \varepsilon_i \varepsilon_j + \sum_{i=1}^{n} [\underline{b_i}, \overline{b_i}] \varepsilon_i + [\underline{c}, \overline{c}] + [e^L, e^U], \end{aligned} \tag{5.1}$$

where ε_i, for all $i \in \{1, \ldots, n+1\}$, are symbolic variables whose values are unknown but belong to $[-1, 1]$.

The affine operations on reliable general quadratic forms are performed as follows. Consider $\hat{\hat{\mathbf{x}}}$ and $\hat{\hat{\mathbf{y}}}$ to be two reliable general quadratic forms and $\mathbf{a}$ a floating interval:

$$\begin{aligned}\hat{\hat{\mathbf{x}}} &= \varepsilon^T \mathbf{A}_x \varepsilon + \mathbf{b}_x^T \varepsilon + \mathbf{c}_x + \mathbf{E}_x, \\ \hat{\hat{\mathbf{y}}} &= \varepsilon^T \mathbf{A}_y \varepsilon + \mathbf{b}_y^T \varepsilon + \mathbf{c}_y + \mathbf{E}_y. \end{aligned}$$

We obtain:

$$\begin{aligned}\hat{\hat{\mathbf{x}}} \pm \hat{\hat{\mathbf{y}}} &= \varepsilon^T (\mathbf{A}_x \pm \mathbf{A}_y) \varepsilon + (\mathbf{b}_x \pm \mathbf{b}_y)^T \varepsilon + \mathbf{c}_x \pm \mathbf{c}_y + (\mathbf{E}_x + \mathbf{E}_y), \\ \hat{\hat{\mathbf{y}}} \pm \mathbf{a} &= \varepsilon^T \mathbf{A}_y \varepsilon + \mathbf{b}_y^T \varepsilon + (\mathbf{c}_y \pm \mathbf{a}) + \mathbf{E}_y, \\ \mathbf{a} \times \hat{\hat{\mathbf{y}}} &= \mathbf{a} \times \varepsilon^T \mathbf{A}_y \varepsilon + \mathbf{a} \times \mathbf{b}_y^T \varepsilon + \mathbf{a} \times \mathbf{c}_y + \mathbf{a} \times \mathbf{E}_y. \end{aligned} \tag{5.2}$$

Concerning non-affine operations, for multiplying two reliable quadratic forms we have:

$$\widehat{\widehat{\mathbf{x}}} \times \widehat{\widehat{\mathbf{y}}} = \varepsilon^T \mathbf{A}_\times \varepsilon + \mathbf{b}_\times^T \varepsilon + \mathbf{c}_\times + \mathbf{E}_\times, \tag{5.3}$$

$$\text{with} \begin{cases} \mathbf{A}_\times = \mathbf{c}_y \mathbf{A}_x + \mathbf{c}_x \mathbf{A}_y + \mathbf{b}_x \mathbf{b}_y^T, \\ \mathbf{b}_\times = \mathbf{c}_y \mathbf{b}_x + \mathbf{c}_x \mathbf{b}_y, \\ \mathbf{c}_\times = \mathbf{c}_y \mathbf{c}_x, \\ \mathbf{E}_\times = (\mathbf{E}_y \times \widehat{\widehat{\mathbf{x}}}) + (\mathbf{E}_x \times \widehat{\widehat{\mathbf{y}}}) + \big((\varepsilon^T \mathbf{A}_x \varepsilon)(\varepsilon^T \mathbf{A}_y \varepsilon) \\ \qquad + (\mathbf{b}_x^T \varepsilon)(\varepsilon^T \mathbf{A}_y \varepsilon) + (\mathbf{b}_y^T \varepsilon)(\varepsilon^T \mathbf{A}_x \varepsilon)\big), \\ \qquad = (\mathbf{E}_y \widehat{\widehat{\mathbf{x}}}) + (\mathbf{E}_x \widehat{\widehat{\mathbf{y}}}) + \left(\sum_{i=1}^{n} (\mathbf{b}_y)_i \varepsilon_i \right) \left(\sum_{i,j=1}^{n} (\mathbf{a}_x)_{ij} \varepsilon_j \varepsilon_i \right) \\ \qquad + \left(\sum_{i,j=1}^{n} (\mathbf{a}_x)_{ij} \varepsilon_j \varepsilon_i + \sum_{i=1}^{n} (\mathbf{b}_x)_i \varepsilon_i \right) \sum_{i,j=1}^{n} (\mathbf{a}_y)_{ij} \varepsilon_j \varepsilon_i. \end{cases}$$

The construction of the corresponding inclusion function proceeds in the same way as for standard affine arithmetic, by only replacing the forms, the operations, and the conversion subroutines. This new reliable inclusion function is denoted by **RGQF**.

6. Application to Rigorous Global Optimization

In order to show the efficiency of inclusion functions based on the general quadratic form, 10 polynomial problems (see Table 1) of the following form are considered:

$$\min_{x \in X \subseteq \mathbb{R}^n} f(x), \tag{6.1}$$

where X is a hypercube in $\mathbb{R}^n$ and f is a polynomial function.

All these problems are solved by a basic interval branch-and-bound algorithm of Moore and Skelboe, see [12]. These techniques are said to be rigorous in the sense that no numerical error can generate wrong bounds. The bounds are computed using the natural extension into interval, denoted by **NE**, a technique based on Taylor expansions of the first order $\mathbf{T}_1$, and methods based on affine forms **AF** and general quadratic forms **GQF**.

This algorithm is based on an interval branch-and-bound principle [12], but it is modified in order to determine the global minimum f^* with maximum accuracy. In fact, this algorithm stops when the precision of the global solution cannot be improved by further computations; see the following algorithm named MAX-PREC. Here, `#its` represents the number of loops, and mid(X) is a real vector (in fact it is an interval vector which encloses the resulting real vector) whose components are the center of the interval vector X, i.e. the middle of the hypercube X.

The MAX-PREC Algorithm.

ε_p, MAX are fixed by the users of the algorithm.

Begin

1. $Y := X$, initial box
2. Compute lower bound of f over X: $\underline{f}$, and $\tilde{f} := f(\text{mid}(X))$
3. Repeat
   a) #its := 1
   b) While $(\tilde{f} - \underline{f} > \varepsilon_p)$ Do
      A standard Moore-Skelboe interval branch-and-bound algorithm is used including the midpoint test and different inclusion functions, see [12]
      ⋮
      #its := #its+1
   c) End Do
   d) $\varepsilon_p := \varepsilon_p \div 10$
4. Until $(\tilde{f} - \underline{f} \leq \varepsilon_p)$ or (#its = MAX)
5. The algorithm stops with the result: $[\underline{f}, \tilde{f}]$ with the accuracy $10 \times \varepsilon_p$.

End

6.1. Numerical Experiments

In this paragraph, we show some numerical results which illustrate the effectiveness of inclusion functions based on the general quadratic forms by comparing them to the well known classical ones. These inclusion functions are iteratively used in the MAX-PREC algorithm.

In Table 1, we summarize, for each polynomial problem, the initial domain of research and the global minimum f^*. In particular, some of these functions came from the literature [6], [7], [9], [10], [12] while others are randomly generated or constructed specially for this paper.

All these numerical tests have been performed on a HP-UX 9000/800 with 4 GB memory; it is a 64-bit quadri-processor computer from the Laboratoire d'Electrotechnique et d'Electronique Industrielle du CNRS/UMR 5828 ENSEEIHT-INP Toulouse. The code has been developed in Fortran 90. The type *real* in Fortran 90 is defined as a single precision floating point which uses 32 bits. The *double precision* type is then defined using 64 bits. This corresponds to Fortran 77 and the standard floating point IEEE norms. Nevertheless, it is possible using Fortran 90 to define the number of bits to be used for the floating point representation of real numbers; see the Fortran 90/95 specification for details.

Table 1. Test functions.

Functions	Initial Domain of Research
$f_1(x) = 1 + (x_1^2 + 2)x_2 + x_1x_2^2$	$X = [1,2] \times [-10,10]$ $f_1^* = -3.5$
$f_2(x) = x_1^3x_2 + x_2^2x_3x_4^2 - 2x_5^2x_1 + 3x_2x_4^2x_5$	$X = [-10,10]^5$ $f_2^* = -1042000$
$f_3(x) = (x_1 - 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$	$X = [-2.5,3.5] \times [-1.5,4.5]$ $f_3^* = 0.45$
$f_4(x) = [1 + (x_1 + x_2 + 1)^2$ $(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]$ $\times [30 + (2x_1 - 3x_2)^2$ $(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	$X = [-2,2]^2$ **Golstein Price function** $f_4^* = 3$
$f_5(x) = (x_1 - 1)(x_1 + 2)(x_2 + 1)(x_2 - 2)x_3^2$	$X = [-2,2]^3$ $f_5^* = -36$
$f_6(x) = 4x_1^2 - 2x_1x_2 + 4x_2^2 - 2x_2x_3 + 4x_3^2 - 2x_3x_4$ $+ 4x_4^2 + 2x_1 - x_2 + 3x_3 + 5x_4$	$X = [-1,3]\times[-10,10]\times[1,4]\times[-1,5]$ $f_6^* = 5.77$
$f_7(x) = x_1^3x_2 + x_2^2x_3x_4^2 - 2x_5^2x_1 + 3x_2x_4^2x_5$ $-\frac{1}{6}x_5^5x_4^3x_3^2$	$X = [-10,10]^5$ $f_7^* = -1667708716.3372$
$f_8(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	$X = [-1000,1000]^2$ **Ratschek function** $f_8^* = -1.03162845348366$
$f_9(x) = 4x_1^2 + 2x_2^2 - 5x_1^2x_3 + 6x_3x_4^2$ $-x_4^3 + 3x_4x_2 - x_3x_4 + 2x_1x_5 + 5x_5^2x_2$	$X = [-10,10]^5$ $f_9^* = -12001.8518518519$
$f_{10}(x) = 6.94x_1^4 + 0.96x_1^3 + 9.68x_1^2 + 4.16x_1$ $+ 7.53x_2^4 - 7.68x_2^3 + 8.21x_2^2 - 1.75x_2$ $-7.45x_1x_2 + 9.15x_1x_2^2 + 3.70x_1x_2^3 - 4.81x_1^2x_2$ $-3.06x_1^2x_2^2 - 0.79x_1^3x_2 - 0.18$	$X = [-50,50]^2$ **A random polynomial function** $f_{10}^* = -0.61585524178857$

The MAX-PREC algorithm uses iteratively the following inclusion functions: the natural extension into interval **NE**, a technique based on Taylor expansions of the first order $\mathbf{T}_1$, **AF**, and **GQF**. The reliable inclusion functions based on affine forms and general quadratic forms are denoted by **RAF** and **RGQF**. The following discussion is based on reliable techniques, i.e. the first four columns of Table 2 and Table 4. The two other results for **AF** and **GQF** are given to underline the efficiency of such techniques in global optimization: Each optimization problem described in Table 1 is solved in less than 3 seconds (the average is 1 second) with a very high precision, see Table 2. The best results among the reliable methods are in bold in Tables 2 and 4.

Table 2. Comparative tests of different inclusion functions.

Pbs	**NEDP**		**T$_1$DP**		**RAFDP**		**RGQFDP**		**AFDP**		**GQFDP**	
	CPU	ε_p	CPU	ε_p	CPU	ε_p	CPU	ε_p	CPU	ε_p	CPU	ε_p
f_1	49.9	10^{-5}	**0.3**	$\mathbf{10^{-13}}$	0.4	$\mathbf{10^{-13}}$	0.4	$\mathbf{10^{-13}}$	0.5	10^{-13}	0.5	10^{-13}
f_2	**0.6**	$\mathbf{10^{-8}}$	146.1	$\mathbf{10^{-8}}$	1.0	$\mathbf{10^{-8}}$	2.0	$\mathbf{10^{-8}}$	0.9	10^{-8}	1.0	10^{-8}
f_3	13.9	10^{-7}	**0.4**	$\mathbf{10^{-14}}$	0.5	$\mathbf{10^{-14}}$	0.6	$\mathbf{10^{-14}}$	0.5	10^{-14}	0.7	10^{-14}
f_4	612.0	1	9.9	$\mathbf{10^{-12}}$	6.1	$\mathbf{10^{-12}}$	**3.2**	$\mathbf{10^{-12}}$	1.5	10^{-12}	1.4	10^{-12}
f_5	130.1	10^{-4}	**1.0**	$\mathbf{10^{-12}}$	**0.9**	$\mathbf{10^{-12}}$	1.2	$\mathbf{10^{-12}}$	0.9	10^{-12}	0.8	10^{-12}
f_6	307.6	10^{-1}	135.8	$\mathbf{10^{-12}}$	**13.4**	$\mathbf{10^{-12}}$	15.8	$\mathbf{10^{-12}}$	2.0	10^{-13}	1.8	10^{-13}
f_7	**0.6**	$\mathbf{10^{-5}}$	—	—	17.1	10^{-4}	7.6	$\mathbf{10^{-5}}$	2.9	10^{-5}	2.3	10^{-5}
f_8	3.9	10^{-2}	**2.6**	$\mathbf{10^{-14}}$	8.5	$\mathbf{10^{-14}}$	4.2	$\mathbf{10^{-14}}$	1.1	10^{-14}	1.8	10^{-14}
f_9	3.7	10^{-1}	8.6	$\mathbf{10^{-10}}$	**3.3**	$\mathbf{10^{-10}}$	7.4	$\mathbf{10^{-10}}$	1.1	10^{-10}	1.8	10^{-10}
f_{10}	64.8	10^{-3}	**2.5**	$\mathbf{10^{-14}}$	8.1	$\mathbf{10^{-14}}$	3.3	$\mathbf{10^{-14}}$	0.8	10^{-14}	1.3	10^{-14}
avg	118.7s		34.1s (without f_7)		5.9s		**4.5**s		1.2s		1.3s	

Table 3. Use of single precision arithmetic.

Pbs	**NE**		**AF**		**RAF**		**GQF**		**RGQF**	
	CPU	ε_p	CPU	ε_p	CPU	ε_p	CPU	ε_p	CPU	ε_p
f_4	561.9	1	0.5	10^{-3}	7.5	10^{-3}	0.8	10^{-3}	2.2	10^{-3}
f_5	5.4	10^{-3}	0.3	10^{-4}	114.2	10^{-4}	0.3	10^{-4}	23.8	10^{-4}
f_6	295.5	10^{-1}	0.8	10^{-4}	8.2	10^{-4}	1.6	10^{-5}	5.2	10^{-4}

The first thing that we must compare is the accuracy (ε_p) obtained by each method. Therefore, assuming equal accuracy, the CPU times (CPU), the number of iterations (`#its`) and the number of elements remaining in the list at the end of the MAX-PREC algorithm (`#clst`) (similar to the number of clusters, [6]) can then be compared.

We would like first to underline the importance of using double precision **DP** representation for the floating point numbers in affine and quadratic forms. In Tables 2 and 3, we see that the gains in the obtained accuracy (ε_p) are considerable when the double representation is used inside affine and quadratic forms. These results are obtained without considering the CPU time differences (even though some gains are also noted). Furthermore, it yields no gain for **NE** to change the single precision into double; only for f_5 one extra order of accuracy is obtained, but the CPU time increases by a factor of 25. For all the other inclusion functions, important gains are noted. Consequently, we consider all the inclusion functions using the double precision floating point representation **DP**, and we compare their efficiency for the ten polynomial functions presented in Table 1.

Table 2 shows that the CPU times and the obtained accuracy (ε_p) are similar for all the considered inclusion functions except for the natural extension **NEDP** which

Table 4. Number of iterations and number of clusters.

Pbs	NEDP		T_1DP		RAFDP		RGQFDP		AFDP		GQFDP	
	#its	#clst	#its	#clst	#its	#clst	#its	#clst	#its	#clst	#its	#clst
f_1	31501	16193	287	35	227	34	**168**	**28**	226	34	169	28
f_2	**248**	**21**	9148	**21**	314	25	269	36	252	24	251	21
f_3	26453	10597	335	48	280	42	**206**	**38**	268	26	206	33
f_4	117552	14231	6536	114	1849	39	**1071**	**22**	1838	29	1066	15
f_5	49432	20783	1832	130	919	81	**698**	**64**	907	70	697	60
f_6	47738	40580	10980	3096	5495	**1057**	**4121**	1309	5322	660	1409	298
f_7	257	37	—	—	3278	827	**626**	**166**	3513	786	646	104
f_8	13125	7089	4431	91	3009	93	**1553**	**31**	2979	49	1547	23
f_9	10190	4770	2214	98	979	76	**797**	**58**	1402	141	744	62
f_{10}	26419	19732	1724	31	1180	35	**632**	**12**	1178	34	631	8
avg	32292	50400	4165 (without f_7)	407	1753	231	**1004**	**176**	1789	185	737	65

is clearly less efficient. This remark is not true when the functions f_2, f_6, and f_7 are considered, because those are examples constructed to show that sometimes the inclusion function **T_1DP** becomes inefficient (when the global optimum is at one end of the initial domain). This can be explained by noting that these three problems have a relatively high number of variables (4 or 5) and thus, the computation of enclosures of the gradient becomes very expensive with respect to the CPU time. However, note that **RGQF** is the most efficient method because it always produces the best accuracy and the best average CPU time; the gain between **RAFDP** and **RGQFDP** is about 25% on these ten examples. The differences noted between the two methods **RAFDP** and **RGQFDP** are noticeable for the functions f_4, f_7, f_8, and f_{10}. It seems to be due to the fact that the expressions of these four functions have multiple occurrences of the same variables, and that these polynomial functions have a high degree (see f_4 for example; in this case the quadratic form clearly shows its efficiency).

Now, if we consider the number of iterations (`#its`) and the number of elements remaining in the list (`#clst`) at the end of the algorithm then it is clearly visible from Table 4 that **RGQF** is the most efficient method; it always gives the best number of iterations (except for f_2 which is a particular case) and also the smallest number of elements remaining in the list at the end of the program (except for f_2 and f_6). These gains are important even if a polynomial function is quite simple in the sense that its quadratic form is not dense (a lot of elements of the matrix are equal to 0). Comparing **RAFDP** and **RGQFDP**, we notice an impressive gain of about 40%.

To summarize, these numerical results demonstrate that for these kinds of problems, the inclusion functions based on a natural extension of an expression of f must never be used when a great accuracy is required (except in a few special cases

such as f_2 and f_7). Furthermore, it can be noted that, even if the inclusion function $\mathbf{T}_1$ produces some good results, those based on affine or general quadratic forms are clearly the most efficient; this is emphasized by these numerical tests when a problem reaches a critical size (4 or 5 variables).

In Table 4, it is clearly shown that **RGQFDP** produces the best bounds compared to all the other methods; for (`#its`) a ratio of 1.7 is noted compared to **RAFDP**, even if the examples are simple. To conclude, we have numerically shown on some examples the gains from the association of **RGQF** with an interval branch-and-bound algorithm; it gives the best accuracy and the best average CPU time (see Table 2) and also the best numbers of iterations and of elements remaining in the list at the end of the program (see Table 4).

7. Conclusion

In this paper, a new method of computing bounds of a function over a box is presented. It is based on a general quadratic form. Some properties of quadratic and polynomial (or more general) functions are reported. The way to make this technique reliable is detailed along with the corresponding algorithm. In order to validate this approach, it was used with the classical interval branch-and-bound algorithm. In ten numerical tests, the general quadratic form technique has clearly shown its intrinsic efficiency by decreasing the number of iterations and consequently the CPU times of such a branch-and-bound algorithm. This efficiency gains seems to grow when the number of variables exceeds 4 and when the degree of the polynomial function is high.

References

1. Comba, J. and Stolfi, J.: Affine Arithmetic and Its Applications to Computer Graphics, in: *Proceedings of VI SIBGRAPI (Brazilian Symposium on Computer Graphics and Image Processing)*, 1993, pp. 9–18.
2. de Figueiredo, L.: Surface Intersection Using Affine Arithmetic, in: *Proceedings of Graphics Interface'96*, 1996, pp. 168–175.
3. de Figueiredo, L., Iwaarden, R. V., and Stolfi, J.: *Fast Interval Branch and Bound Methods for Unconstrained Global Optimization*, Technical Report 97–08, Institute of Computing, UNICAMP, Campinas, SP, Brazil, 1997.
4. de Figueiredo, L. and Stolfi, J.: Affine Arithmetic: Concepts and Applications, *Numerical Algorithms* **37** (2004), pp. 147–158.
5. de Figueiredo, L. and Stolfi, J.: *Self-Validated Numerical Methods and Applications*, Brazilian Mathematics Colloquium monographs, IMPA/CNPq, Rio de Janeiro, Brazil, 1997.
6. Du, K. and Kearfott, R.: The Cluster Problem in Multivariate Global Optimization, *Journal of Global Optimization* **10** (1996), pp. 27–32.
7. Hansen, E.: A Generalized Interval Arithmetic, in: Nickel, K. (ed.), *Interval Mathematics*, *Lecture Notes in Computer Science* **29**, Springer Verlag, 1975, pp. 7–18.
8. Hargreaves, G.: *Interval Analysis in MATLAB*, Numerical Analysis Report No. 416, Manchester Centre for Computational Mathematics, Manchester, 2002.
9. Messine, F.: Extension of Affine Arithmetic: Application to Unconstraineded Global Optimisation, *Journal of Universal Computer Science* **8** (2002), pp. 992–1015.

10. Messine, F. and Lagouanelle, J.-L.: Enclosure Methods for Multivariate Differentiable Functions and Application to Global Optimization, *Journal of Universal Computer Science* **4** (1998), pp. 589–603.
11. Moore, R.: *Interval Analysis*, Prentice Hall, 1966.
12. Ratschek, H. and Rokne, J.: *New Computer Methods for Global Optimisation*, Ellis Horwood, 1988.