10-1-2003

Greedy Algorithms for Optimizing Multivariate Horner Schemes

Martine Ceberio
University of Texas at El Paso, mceberio@utep.edu

Vladik Kreinovich
University of Texas at El Paso, vladik@utep.edu

Comments:
Technical Report: UTEP-CS-03-26
Published in: *ACM SIGSAM Bulletin*, 2004, Vol. 38, No. 1 (147), pp. 8-15.

Greedy Algorithms for Optimizing Multivariate Horner Schemes

Martine Ceberio and Vladik Kreinovich
Department of Computer Science, University of Texas at El Paso
El Paso, TX 79968, USA, {mceberio,vladik}@cs.utep.edu

Abstract

For univariate polynomials $f(x_1)$, Horner scheme provides the fastest way to compute the value. For multivariate polynomials, several different version of Horner scheme are possible; it is not clear which of them is optimal. In this paper, we propose a greedy algorithm that will hopefully lead to good computation times.

A univariate Horner scheme has another advantage: if the value x_1 is known with uncertainty, and we are interested in the resulting uncertainty in $f(x_1)$, then Horner scheme leads to a better estimate for this uncertainty that many other ways of computing $f(x_1)$. The second greedy algorithm that we propose tries to find the multivariate Horner scheme that leads to the best estimate for the uncertainty in $f(x_1, \ldots, x_n)$.

1 Introduction

It is well known (see, e.g., [11]) that if we want to compute the value of a *univariate* polynomial

$$f(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_{d-1} \cdot x^{d-1} + a_d \cdot x^d, \tag{1}$$

then the fastest way to compute it is to use the Horner scheme

$$f(x) = a_0 + x \cdot (a_1 + x \cdot (\ldots (a_{d-1} + x \cdot a_d) \ldots)). \tag{2}$$

For univariate polynomials, Horner's scheme is the one with the smallest overall number of arithmetic operations (additions and multiplications).

For *multivariate* polynomials, it is not known which scheme leads to the smallest possible number of arithmetic operations. One approach – implemented in MatLab and Mathematica – is to use multivariate Horner scheme. In this scheme, we choose a variable, say x_1 and apply Horner scheme as if all the other variables were constants. The resulting coefficients a_i are then functions of all the other variables $x_2, \ldots, x_n$. To compute these coefficients, we select one of the remaining variables and apply Horner scheme with this variable, etc. For example, for

$$f(x_1, x_2, x_3) = x_1^3 \cdot x_2 + x_1^2 \cdot x_3 + x_1^2 \cdot x_2 \cdot x_3, \tag{3}$$

if we first select x_1, then we get

$$f(x_1, x_2, x_3) = x_1^2 \cdot (a_2 + x_1 \cdot a_3), \tag{4}$$

where

$$a_2 = x_2 \cdot x_3 + x_3; \quad a_3 = x_2, \tag{5}$$

i.e.,

$$f(x_1, x_2, x_3) = x_1^2 \cdot (x_2 \cdot x_3 + x_3 + x_1 \cdot x_2). \tag{6}$$

If we now select x_3 as the next variable, we get

$$a_2 = x_3 \cdot (x_2 + 1), \tag{7}$$

i.e.,

$$f(x_1, x_2, x_3) = x_1^2 \cdot (x_3 \cdot (x_2 + 1) + x_1 \cdot x_2). \tag{8}$$

It is known (see, e.g., [8]) that for some polynomials, there are faster computation schemes than multivariate Horner, but in many practical cases, for an appropriately selected sequence of variables, the multivariate Horner scheme works reasonably well; see, e.g., [12, 13].

The problem is that it is sometimes difficult to select the appropriate order of variables, and with the wrong order, we may end up performing much more arithmetic operations to compute the value of the same polynomial. For example, for the polynomial (3), the above scheme (8) requires 4 multiplications and 2 additions. However, if we selected x_2 first, and x_3 next, then we would end up with the scheme

$$f(x_1, x_2, x_3) = x_2 \cdot (x_1^3 + x_1^2 \cdot x_3) + x_1^2 \cdot x_3, \tag{9}$$

that requires the same number (2) of additions, but 7 multiplications: 2 to compute x_1^3, 1 to compute x_1^2, one to compute $x_1^2 \cdot x_3$, one to compute $x_2 \cdot (x_1^3 + x_1^2 \cdot x_3)$, 1 to compute x_1^2, and 1 to compute $x_1^2 \cdot x_3$. Even if we take into consideration that in this procedure, we compute x_1^2 twice, we still end up with $6 > 4$ multiplications.

How can we select the order of the variables so as to minimize the overall number of arithmetic operations?

2 First Greedy Algorithm: Motivation

There is no known method for selecting an optimal order. Let us therefore describe a (heuristic) *greedy* algorithm for finding such an order. The main idea behind greedy algorithms (see, e.g., [4]) is that if we have a sequence of choices, then we make each choice in such a way that after this choice, the minimized function gets the largest possible decrease.

What does this general idea translate into for multivariate Horner scheme? Let us first describe its application to the selection of the first variable. We start with a polynomial f that is a sum of monomials M_1, M_2, ...:

$$f(x_1, \ldots, x_n) = \sum_k M_k. \tag{10}$$

If we select x_i as the first variable, then we group together all monomials that contain x_i – and have x_i as a common factor – and keep all other monomials intact. In other words, we replace the expression (10) by a new expression

$$f(x_1, \ldots, x_n) = x_i \cdot \left(\sum_{k: x_i \in M_k} M_k' \right) + \sum_{k: x_i \notin M_k} M_k, \tag{11}$$

where $M_k' \stackrel{\text{def}}{=} M_k / x_i$ is the result of taking x_i out as a factor.

With this transformation, the number of additions does not change, but the number of multiplication decreases. Indeed, If we use the original formula (10) as the computation scheme,

then we have to we compute all the terms M_k that do not contain x_i, and we also compute all the terms M_k that contain x_i. Computing a monomial $M_k = x_1^{d_{k1}} \cdot \ldots x_i^{d_{ki}} \cdot \ldots \cdot x_n^{d_{kn}}$ of order $d_k \stackrel{\text{def}}{=} d_{k1} + \ldots + d_{ki} + \ldots + d_{kn}$ requires $d_k - 1$ multiplications.

If we use the formula (11) as the computation scheme, then we still have to compute all the terms M_k that do not contain x_i. However, for each term M_k that contains x_i, we do not compute the monomial M_k of degree d_k anymore: instead, we compute the auxiliary monomial $M'_k = M_k/x_i$ of degree $d'_k = d_k - 1$, and this computations requires $d'_k - 1 = d_k - 2$ multiplications. After that, we need an extra multiplication by x_i.

Thus, we use one extra multiplication, but we save one multiplication for each monomial that contains x_i. So, if we denote the total number of such monomials by N_i, then we save $N_i - 1$ multiplications if we choose x_i.

According to the idea of a greedy algorithm, it is therefore reasonable to select x_i for which this decrease is the largest, i.e., for which the value N_i is the largest. Thus, we arrive at the following algorithm.

3 First Greedy Algorithm: Description and Example

In the proposed algorithm, given a polynomial $f(x_1, \ldots, x_n)$, we count, for each variable x_i, the number of monomials N_i that contain this variable. We then select a variable with the largest value of N_i, and represent the original polynomial as $f = A_0 + A_1 \cdot x_i$, where A_0 is the sum of all monomials that do not contain x_i, and A_i is the sum of all the terms $M'_k = M_k/x_i$ corresponding to all the monomials M_k that do contain x_i.

Then, we apply the same algorithm to each of the resulting terms A_0 and A_1, etc. – until we get a computation scheme.

Let us show how this algorithm will work on the above polynomial (3). For this polynomial, $N_1 = 2$, $N_2 = 2$, and $N_3 = 2$. The variable with the largest value of N_i is x_1, so we select x_1 for the first step of our multivariate Horner scheme. Here, all three monomials contain x_1, so $A_0 = 0$, and we get the decomposition $f = x_1 \cdot A_1$, where

$$A_1 = x_1^2 \cdot x_2 + x_1 \cdot x_3 + x_1 \cdot x_2 \cdot x_3. \tag{12}$$

Similarly, for A_1, we have $N_1 = 3 > N_2 = N_3 = 2$, so we again select x_1, and get the following expression:

$$f = x_1 \cdot x_1 \cdot A'_1, \tag{13}$$

where

$$A'_1 \stackrel{\text{def}}{=} x_1 \cdot x_2 + x_3 + x_2 \cdot x_3. \tag{14}$$

For A'_1, we select either x_2 or x_3. If we select x_2, we get

$$f = x_1 \cdot x_1 \cdot (x_2 \cdot (x_1 + x_3) + x_3). \tag{15}$$

If we select x_3, we get

$$f = x_1 \cdot x_1 \cdot (x_3 \cdot (x_2 + 1) + x_1 \cdot x_2). \tag{16}$$

In both cases, we get no more multiplications than in the above form (9), but if we select x_2, we get even fewer multiplications: 3 instead of 4.

This example also shows why selecting x_2 as the first variable led to too many multiplications: for x_2, we have $N_2 = 2 < N_1$.

4 How Good Is This Method?

The above algorithm is based on the heuristic idea of greedy algorithms. In general, heuristic methods are not always optimal. How good is our method? Our preliminary tests show that this method works well. It is desirable to test this method further.

What we can do ourselves is to test this new method as follows: we can design a generator for generating "random" polynomials (corresponding to some reasonable probability distribution on the set of all polynomials), and then test our method on the polynomials generated by this generator. However, the problem with this test is that its results may depend on the choice of the probability distribution used in this generation. Thus, even if we get good results for some probability distribution, it will always be possible that the method does not work as well on polynomials from practical problems.

We would therefore like to test this algorithm on polynomials encountered in real-life problems, and the only way to do it is to publicize our algorithm and to ask the readers to try it. We would greatly appreciate the results.

5 Error Propagation: Another Reason Why Horner Schemes Are Good

The Horner scheme not only saves on computation time, it also helps in error propagation: namely, if we only know the value of the input with some uncertainty, it helps to estimate the resulting uncertainty in the output. In precise terms, the uncertainty in the input x_i means that instead of knowing the exact value of x_i, we know the approximate value $\widetilde{x}_i$ of the corresponding quantity, and we know the accuracy Δ_i of this approximation, i.e., we know that the difference between the actual (unknown) value x_i and this approximation cannot exceed Δ_i: $|\widetilde{x}_i - x_i| \le \Delta$. This means that the only information that we have about x_i is that it belongs to the interval $\mathbf{x}_i = [\underline{x}_i, \overline{x}_i]$, where $\underline{x}_i = \widetilde{x}_i - \Delta_i$ and $\overline{x}_i = \widetilde{x}_i + \Delta_i$.

Different values $x_i \in \mathbf{x}_i$ lead to different values of the polynomial $y = f(x_1, \ldots, x_n)$. It is therefore to estimate the range of the possible values of this polynomial, i.e., the interval

$$\mathbf{y} \stackrel{\text{def}}{=} \{f(x_1, \ldots, x_n) \,|\, x_1 \in \mathbf{x}_1 \,\&\, \ldots \,\&\, x_n \in \mathbf{x}_n\}. \tag{17}$$

The problem of computing the exact range is NP-hard; see, e.g., [9, 17], so, in general, we cannot feasibly compute the exact bounds on the accuracy of y. We should therefore aim for reasonable upper bounds, i.e., for a reasonable *enclosure* $\mathbf{Y} \supseteq \mathbf{y}$.

A natural way to compute such an enclosure is to use a technique called straightforward interval computations (see, e.g., [5, 6, 7, 10, 11]). This technique uses the fact that for the polynomials that represent simple arithmetic operations, we get explicit expressions for $\mathbf{y}$:

- for $f(x_1, x_2) = x_1 + x_2$, we have

$$[\underline{y}, \overline{y}] = [\underline{x}_1 + \underline{x}_2, \overline{x}_1 + \overline{x}_2]; \tag{18}$$

- for $f(x_1, x_2) = x_1 - x_2$, we have

$$[\underline{y}, \overline{y}] = [\underline{x}_1 - \overline{x}_2, \overline{x}_1 - \underline{x}_2]; \tag{19}$$

- for $f(x_1, x_2) = x_1 \cdot x_2$, we have

$$[\underline{y}, \overline{y}] = [\min(\underline{x}_1 \cdot \underline{x}_2, \underline{x}_1 \cdot \overline{x}_2, \overline{x}_1 \cdot \underline{x}_2, \overline{x}_1 \cdot \overline{x}_2), \max(\underline{x}_1 \cdot \underline{x}_2, \underline{x}_1 \cdot \overline{x}_2, \overline{x}_1 \cdot \underline{x}_2, \overline{x}_1 \cdot \overline{x}_2)]. \tag{20}$$

These formulas are called *interval arithmetic*; they can be used to compute the desired enclosure for an arbitrary polynomial $f(x_1, \ldots, x_n)$ as follows. In the computer, an arbitrary computation is eventually translated ("parsed") into a *code list*, which is a sequence of elementary arithmetic operations. In straightforward interval computations, we replace each real operand with the corresponding interval $\mathbf{x}_i$, and replace each arithmetic operation with real numbers with the corresponding interval-arithmetic operation.

It is known that, as a result, we always obtain an *enclosure* $\mathbf{Y}$ of the desired interval $\mathbf{y}$, i.e., an interval for which $\mathbf{Y} \supseteq \mathbf{y}$. For example, if $f(x_1) = x_1 - x_1^2$, a natural parsing leads to $r_1 := x_1^2$ and $y := x_1 - r_1$. Then, for $\mathbf{x}_1 = [0, 1]$, straightforward interval evaluation leads to the estimates

$$\mathbf{R}_1 := [0,1] \cdot [0,1] = [\min(0 \cdot 0, 0 \cdot 1, 1 \cdot 0, 1 \cdot 1), \max(0 \cdot 0, 0 \cdot 1, 1 \cdot 0, 1 \cdot 1)] \tag{21}$$

and $\mathbf{Y} := [0, 1] - [0, 1] = [-1, 1]$.

The problem with this technique is that, as we have mentioned, this enclosure can be wider than the actual (desired) range. For example, in the above example, the actual range is $\mathbf{y} = [0, 0.25]$.

One way to make the enclosure more accurate is to use the fact that interval arithmetic is *semi-distributive*, i.e., that for every three intervals $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$, we have

$$\mathbf{a} \cdot (\mathbf{b} + \mathbf{c}) \subseteq \mathbf{a} \cdot \mathbf{b} + \mathbf{a} \cdot \mathbf{c}. \tag{22}$$

In many cases, the inclusion is proper. For example, if $\mathbf{a} = [0, 1]$, $\mathbf{b} = [1, 1]$, and $\mathbf{c} = [-1, -1]$, then $\mathbf{b} + \mathbf{c} = [1, 1] + [-1, -1] = [0, 0]$, hence

$$\mathbf{a} \cdot (\mathbf{b} + \mathbf{c}) = [0,1] \cdot [0,0] = [0,0]. \tag{23}$$

On the other hand, $\mathbf{a} \cdot \mathbf{b} = [0, 1] \cdot [1, 1] = [0, 1]$ and $\mathbf{a} \cdot \mathbf{c} = [0, 1] \cdot [-1, -1] = [-1, 0]$, hence

$$\mathbf{a} \cdot \mathbf{b} + \mathbf{a} \cdot \mathbf{c} = [0,1] + [-1,0] = [-1,1] \supset [0,0]. \tag{24}$$

Due to this property, if two terms in the description of a polynomial have a common factor, then we get a narrower interval if we apply interval computations to the expression in which this factor has been explicitly factored out.

For univariate polynomials $f(x_1)$, the natural common factor is x_1, and factoring out this common factor is exactly what the Horner scheme is doing. Thus, for univariate polynomials, the Horner scheme does lead to narrower intervals than an application of straightforward interval computation to the original expression (1).

We can apply the same idea for multivariate polynomials. For example, in [1, 2], we have shown that an algorithm similar to the first greedy algorithm often leads to a drastic decrease in the width of the corresponding interval enclosure. General formulas for the numerical accuracy resulting from different Horner schemes are also given in [12, 13].

The question is: which of the possible multivariate Horner schemes is the best? In the following section, we will describe a greedy algorithm that tries to minimize the width of the resulting interval enclosure for the range $f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

6 Second Greedy Algorithm: Motivation

As we have mentioned, the main idea behind a greedy algorithm is that at each step, we maximize the effect of this step on the final result. To apply this idea to interval estimates, let us estimate the gain resulting when we move from the original expression $a \cdot b + a \cdot c$ to the factored expression $a \cdot (b + c)$.

In this estimate, instead of the above formulas for interval computations – formulas that describe, for arithmetic expressions, the exact range – we will use the simplified formulas first proposed by S. Rump (see, e.g., [14]). In these formulas, every interval $\mathbf{a} = [\underline{a}, \overline{a}]$ is represented by its midpoint $\widetilde{a} = (\underline{a} + \overline{a})/2$ and its half-width (radius) $\Delta_a = (\overline{a} - \underline{a})/2$, so that $\mathbf{a} = [\widetilde{a} - \Delta_a, \widetilde{a} + \Delta_a]$, and the corresponding arithmetic operations take the following form:

$$[\widetilde{a} - \Delta_a, \widetilde{a} + \Delta_a] + [\widetilde{b} - \Delta_b, \widetilde{b} + \Delta_b] = [\widetilde{c} - \Delta_c, \widetilde{c} + \Delta_c],$$

$$\text{where} \quad \widetilde{c} = \widetilde{a} + \widetilde{b} \quad \text{and} \quad \Delta_c = \Delta_a + \Delta_b; \tag{25}$$

$$[\widetilde{a} - \Delta_a, \widetilde{a} + \Delta_a] - [\widetilde{b} - \Delta_b, \widetilde{b} + \Delta_b] = [\widetilde{c} - \Delta_c, \widetilde{c} + \Delta_c],$$

$$\text{where} \quad \widetilde{c} = \widetilde{a} - \widetilde{b} \quad \text{and} \quad \Delta_c = \Delta_a + \Delta_b; \tag{26}$$

$$[\widetilde{a} - \Delta_a, \widetilde{a} + \Delta_a] \cdot [\widetilde{b} - \Delta_b, \widetilde{b} + \Delta_b] = [\widetilde{c} - \Delta_c, \widetilde{c} + \Delta_c],$$

$$\text{where} \quad \widetilde{c} = \widetilde{a} \cdot \widetilde{b} \quad \text{and} \quad \Delta_c = |\widetilde{a}| \cdot \Delta_b + |\widetilde{b}| \cdot \Delta_a + \Delta_a \cdot \Delta_b. \tag{27}$$

For addition and subtraction, these formulas are the same as the above ones (and thus, lead to the exact interval range). For multiplication, if we only consider the first order terms in terms of the half-widths Δ_a and Δ_b of the intervals $\mathbf{a}$ and $\mathbf{b}$, the new formula is exact; it does, however, lead to excess width if we take second order terms into account. We are using these not very exact formulas because they are faster (as the very title of Rump's paper [14] shows), and they are exact when it comes to first order terms – and these are the terms that we will be minimizing.

By using formulas of Rump's arithmetic, we conclude that $\mathbf{b} + \mathbf{c} = [\widetilde{d} - \Delta_d, \widetilde{d} + \Delta_d]$, where $\widetilde{d} = \widetilde{b} + \widetilde{c}$ and $\Delta_d = \Delta_b + \Delta_c$ and therefore, if we ignore second order terms:

$$\mathbf{a} \cdot (\mathbf{b} + \mathbf{c}) = [\widetilde{\ell} - \Delta_\ell, \widetilde{\ell} + \Delta_\ell], \tag{28}$$

where

$$\widetilde{\ell} = \widetilde{a} \cdot (\widetilde{b} + \widetilde{c}) \tag{29}$$

and

$$\Delta_\ell = |\widetilde{a}| \cdot (\Delta_b + \Delta_c) + |\widetilde{b} + \widetilde{c}| \cdot \Delta_a. \tag{30}$$

On the other hand, $\mathbf{a} \cdot \mathbf{b} = [\widetilde{e} - \Delta_e, \widetilde{e} + \Delta_e]$ and $\mathbf{a} \cdot \mathbf{c} = [\widetilde{f} - \Delta_f, \widetilde{f} + \Delta_f]$, where $\widetilde{e} = \widetilde{a} \cdot \widetilde{b}$, $\widetilde{f} = \widetilde{a} \cdot \widetilde{c}$, $\Delta_e = |\widetilde{a}| \cdot \Delta_b + |\widetilde{b}| \cdot \Delta_a$, and $\Delta_f = |\widetilde{a}| \cdot \Delta_c + |\widetilde{c}| \cdot \Delta_a$. Therefore,

$$\mathbf{a} \cdot \mathbf{b} + \mathbf{a} \cdot \mathbf{c} = [\widetilde{r} - \Delta_r, \widetilde{r} + \Delta_r], \tag{31}$$

where

$$\widetilde{r} = \widetilde{a} \cdot \widetilde{b} + \widetilde{a} \cdot \widetilde{c} \tag{32}$$

and

$$\Delta_r = |\widetilde{a}| \cdot \Delta_b + |\widetilde{a}| \cdot \Delta_c + |\widetilde{b}| \cdot \Delta_a + |\widetilde{c}| \cdot \Delta_a. \tag{33}$$

By comparing (29) with (32) and (30) with (33), we conclude that the two resulting intervals have the same midpoint $\widetilde{\ell} = \widetilde{r}$, and that the radius Δ_ℓ of the left-hand side interval is narrower by the amount

$$\Delta_r - \Delta_\ell = (|\widetilde{b}| + |\widetilde{c}| - |\widetilde{b} + \widetilde{c}|) \cdot \Delta_a. \tag{34}$$

In general, if we replace $a \cdot b_1 + \ldots + a \cdot b_n$ with $a \cdot (b_1 + \ldots + b_n)$, the radius of the resulting interval is narrower by the value

$$(|\widetilde{b}_1| + \ldots + |\widetilde{b}_n| - |\widetilde{b}_1 + \ldots + \widetilde{b}_n|) \cdot \Delta_a. \tag{35}$$

The expression (35) can be further simplified if we separate positive and negative terms $\widetilde{b}_i$. Then, the sum $|\widetilde{b}_1| + \ldots + |\widetilde{b}_n|$ is equal to $S^+ + S^-$, where S^+ is the sum of all the positive values $\widetilde{b}_i$ and S^- is the sum of the absolute values of all the negative values $\widetilde{b}_i$. Similarly, the expression $\widetilde{b}_1 + \ldots + \widetilde{b}_n$ is equal to $S^+ - S^-$, so the coefficient at Δ_a in (35) is equal to $S^+ + S^- - |S^+ - S^-|$. By considering both possible cases $S^+ \geq S^-$ and $S^+ < S^-$, we can conclude that $S^+ + S^- - |S^+ - S^-| = 2\min(S^+, S^-)$. Thus, when we replace $a \cdot b_1 + \ldots + a \cdot b_n$ with $a \cdot (b_1 + \ldots + b_n)$, we gain $2\Delta_a \cdot \min(S^+, S^-)$.

For multivariate Horner scheme, when we select x_i as a common factor, the gain can be expressed as follows:

$$2\Delta_i \cdot \min\left(\sum_{k:\widetilde{M}'_k > 0} \widetilde{M}'_k, \sum_{k:\widetilde{M}'_k < 0} \left|\widetilde{M}'_k\right|\right), \tag{36}$$

where M_k runs over all monomials that contain x_i, $\widetilde{M}_k$ is the value of the monomial M_k when $x_1 = \widetilde{x}_1$, $\ldots$, $x_n = \widetilde{x}_n$, and $\widetilde{M}'_k = \widetilde{M}_k / \widetilde{x}_i$. This expression can be further simplified into the following:

$$2\Delta_i \cdot \min\left(\sum_{k:x_i \in M_k \,\&\, \widetilde{M}_k > 0} \widetilde{M}_k, \sum_{k:x_i \in M_k \,\&\, \widetilde{M}_k < 0} |\widetilde{M}_k|\right). \tag{37}$$

According to the main idea behind a greedy algorithm, at each stage, we must select a variable x_i that leads to the largest decrease in the resulting interval range. Thus, at each stage, for factoring, we must select the variable x_i for which the value (37) is the largest.

7 Second Greedy Algorithm: Description

In the proposed algorithm, given a polynomial $f(x_1, \ldots, x_n)$ and the intervals $[\widetilde{x}_1 - \Delta_1, \widetilde{x}_1 + \Delta_1]$, $\ldots$, $[\widetilde{x}_n - \Delta_n, \widetilde{x}_n + \Delta_n]$, we compute, for each monomial M_k, its value $\widetilde{M}_k$ for $x_1 = \widetilde{x}_1$, $\ldots$, $x_n = \widetilde{x}_n$. Then, for each variable x_i, we compute the value (37), select a variable x_i with the largest value of this quantity, and represent the original polynomial as $f = A_0 + A_1 \cdot x_i$, where A_0 is the sum of all monomials that do not contain x_i, and A_i is the sum of all the terms $M'_k = M_k / x_i$ corresponding to all the monomials M_k that do contain x_i.

Then, we apply the same algorithms to each of the resulting terms A_0 and A_1, etc. – until we get a computation scheme.

8 Discussion

Due to semi-distributivity, this method always leads to a better (or at least not worse) estimate than a simple application of straightforward interval computations to the original polynomial. However, since – just like the previous greedy algorithm – this algorithm is based on a heuristic idea, we do not have any guarantee that this method will always lead to an optimal estimate. To see how good

it is for polynomials coming from practical problems, it is desirable to test this method on such polynomials. We would welcome all results of such testing.

What we know for sure is that even if the resulting range estimates are the best among all possible Horner scheme, these estimates cannot always be optimal – because the Horner scheme is a feasible (polynomial-time) algorithm, and the problem of computing the exact range of a polynomial is (as we have mentioned) NP-hard. It is worth mentioning that, as we have shown in [16], this NP-hardness – i.e., the possibility to reduce any NP problem to the problem of range estimation – is not just a negative result, it can be used in a positive way: a (similar) greedy algorithm for estimating an interval range leads to a reasonably efficient greedy-type algorithm for solving the original NP-hard problem of checking satisfiability of propositional formulas.

It is also desirable to see what happens if, in addition to the above-described formulas of interval arithmetic, we use expressions for the interval of the power:

- For $f(x_1) = x_1^k$ with odd k,
$$[\underline{y}, \overline{y}] = [\underline{x}_1^k, \overline{x}_1^k]; \tag{38}$$
- for $f(x_1) = x_1^k$ with even k,
$$[\underline{y}, \overline{y}] = [\underline{x}_1^k, \overline{x}_1^k] \text{ if } 0 \le \underline{x}_1; \tag{39}$$
$$[\underline{y}, \overline{y}] = [\overline{x}_1^k, \underline{x}_1^k] \text{ if } \overline{x}_1 \le 0; \tag{40}$$
$$[\underline{y}, \overline{y}] = [0, \max(\underline{x}_1^k, \overline{x}_1^k)] \text{ if } \underline{x}_1 < 0 < \overline{x}_1. \tag{41}$$

The use of these formulas leads to narrower intervals: e.g., when we estimate the range of the function $f(x_1) = x_1^2$ on the interval $\mathbf{x}_1 = [-1, 1]$, straightforward interval computations lead to $[-1, 1] \cdot [-1, 1] = [-1, 1]$, while the use of the formula for x_1^2 leads to the exact range $[0, 1]$. However, Horner scheme does not necessarily narrow the interval, because for x_1^2, Horner scheme would lead exactly to $x_1 \cdot x_1$. In algebraic terms, if we add these functions, semi-distributivity is no longer always true.

In [3], we showed, for univariate polynomials, how we can get narrow bounds in the presence of interval power functions. It is desirable to extend this approach to multivariate polynomials.

Acknowledgments

This work was supported in part by NASA under cooperative agreement NCC5-209, by the Future Aerospace Science and Technology Program (FAST) Center for Structural Integrity of Aerospace Systems, effort sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-00-1-0365, by NSF grants EAR-0112968 and EAR-0225670, and by Army Research Laboratories grant DATM-05-02-C-0046.

References

[1] M. Ceberio and L. Granvilliers, "Solving nonlinear systems by constraint inversion and interval arithmetic", *Proceedings of the AISC'2000, 5th International Conference on Artificial Intelligence and Symbolic Computations*, Springer Lecture Notes in Artificial Intelligence, Vol. 1930, 2000, pp. 127–141.

[2] M. Ceberio and L. Granvilliers, "Résolution de systèmes non linéaires par inversion des contraintes et analyse des intervalles", *Proceedings of the 9th French Conference on Logic Programming and Constraint Programming JFPLC'2000*, Hermès, 2000, pp. 205–220.

[3] M. Ceberio and L. Granvilliers, "Horner's rule for interval evaluation revisited", *Computing*, 2002, Vol. 69, No. 1, pp. 51–81.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, MA, and Mc-Graw Hill Co., N.Y., 2001.

[5] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter, *Applied Interval Analysis*, Springer-Verlag, Berlin, 2001.

[6] R. B. Kearfott, *Rigorous Global Search: Continuous Problems*, Kluwer, Dordrecht, 1996.

[7] R. B. Kearfott and V. Kreinovich (eds.), *Applications of Interval Computations*, Kluwer, Dordrecht, 1996.

[8] D. E. Knuth, *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms*, 2nd edition, Addison Wesley, London, 1981.

[9] V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational Complexity and Feasibility of Data Processing and Interval Computations*, Kluwer, Dordrecht, 1997.

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

[11] A. Neumaier, *Introduction to Numerical Analysis*, Cambridge Univ. Press, Cambridge, UK, 2001.

[12] J. M. Peña and T. Sauer, "On the multivariate Horner scheme", *SIAM J. Numer. Anal.*, 2000, Vol. 37, pp. 1186–1197.

[13] J. M. Peña and T. Sauer, "On the multivariate Horner scheme. II: Running error analysis", *Computing*, 2000, Vol. 65, pp. 313–322.

[14] S. M. Rump, "Fast and parallel interval arithmetic", *BIT Numerical Mathematics*, 1999, Vol. 39, No. 3, pp. 534–554.

[15] V. Stahl, *Interval methods for bounding the range of polynomials and solving systems of nonlinear equations*, Ph.D. Dissertation, University of Linz, Austria, 1995.

[16] B. Traylor and V. Kreinovich, "A bright side of NP-hardness of interval computations: interval heuristics applied to NP-problems", *Reliable Computing*, 1995, Vol. 1, No. 3, pp. 343–360.

[17] S. A. Vavasis, *Nonlinear Optimization: Complexity Issues*, Oxford University Press, N.Y., 1991.