Certified Roundoff Error Bounds using Bernstein Expansions and Sparse Krivine-Stengle Representations
Alexandre Rocca, Victor Magron, Thao Dang

To cite this version:
Alexandre Rocca, Victor Magron, Thao Dang. Certified Roundoff Error Bounds using Bernstein Expansions and Sparse Krivine-Stengle Representations. 2017. <hal-01448167>

HAL Id: hal-01448167
https://hal.archives-ouvertes.fr/hal-01448167
Submitted on 27 Jan 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Certified Roundoff Error Bounds using Bernstein Expansions and Sparse Krivine-Stengle Representations

Alexandre Rocca\textsuperscript{1,2}, Victor Magron\textsuperscript{1}, and Thao Dang\textsuperscript{1}

\textsuperscript{1} VERIMAG/CNRS
700 avenue Centrale, 38400 Saint Martin D’Hères, France
\textsuperscript{2} UJF-Grenoble 1/CNRS, TIMC-IMAG,
UMR 5525, Grenoble, F-38041, France

Abstract. Floating point error is an inevitable drawback of embedded systems implementation. Computing rigorous upper bounds of roundoff errors is absolutely necessary for the validation of critical software. This problem of computing rigorous upper bounds is even more challenging when addressing non-linear programs. In this paper, we propose and compare two new methods based on Bernstein expansions and sparse Krivine-Stengle representations, adapted from the field of the global optimization, to compute upper bounds of roundoff errors for programs implementing polynomial functions. We release two related software package FPBern and FPKiSten, and compare them with state of the art tools.
We show that these two methods achieve competitive performance, while computing accurate upper bounds by comparison with other tools.

Keywords: Polynomial Optimization, Floating Point Arithmetic, Roundoff Error Bounds, Linear Programming Relaxations, Bernstein Expansions, Krivine-Stengle Representations

1 Introduction

Theoretical models, algorithms, and programs are often reasoned and designed in real algebra. However, their implementation on computers uses floating point algebra: this conversion from real numbers and their operations to floating point is not without errors. Indeed, due to finite memory and binary encoding in computers, real numbers cannot be exactly represented by floating point numbers. Moreover, numerous properties of the real algebra are not conserved such as commutativity or associativity.

The consequences of such imprecisions become particularly significant in safety-critical systems, especially in embedded systems which often include control components implemented as computer programs. When implementing an algorithm designed in real algebra, and initially tested on computers with single or double floating point precision, one would like to ensure that the roundoff error is not too large on more limited platforms (small processor, low memory capacity) by computing their accurate upper bounds.
For programs implementing linear functions, SAT/SMT solvers, as well as affine arithmetic, are efficient tools to obtain good upper bounds. When extending to programs with non-linear polynomial functions, the problem of determining a precise upper bound becomes substantially more difficult, since polynomial optimization problems are in general NP-hard [17]. We can cite at least three closely related and recent frameworks designed to provide upper bounds of roundoff errors for non-linear programs. FPTaylor [25] is a tool based on Taylor-interval methods, while Rosa [5] combines SMT with interval arithmetic. Real2Float [19] relies on Putinar representations of positive polynomials while exploiting sparsity in a similar way as the second method that we propose in this paper.

We introduce two methods, coming from the field of polynomial optimization, to compute upper bounds on roundoff errors of polynomial programs. The first method is based on Bernstein expansions of polynomials, while the second relies on sparse Krivine-Stengle certificates for positive polynomials. In practice, these methods (presented in Section 3) provide accurate bounds at a reasonable computational cost. Indeed, the size of the Bernstein expansion used in the first method as well as the size of the LP relaxation problems considered in the second method are both linear w.r.t. the number of roundoff error variables.

1.1 Overview

Before explaining in detail each method, let us first illustrate the addressed problem on an example. Let \( f \) be the degree two polynomial defined by:

\[
 f(x) := x^2 - x, \quad \forall x \in X = [0, 1].
\]

When approximating the value of \( f \) at a given real number \( x \), one actually computes the floating point result \( \hat{f} = \hat{x} \odot \hat{x} \ominus \hat{x} \), with all the real operators \(+, -, \times\) being substituted by their associated floating point operators \( \oplus, \odot, \ominus \), and \( x \) being represented by the floating point number \( \hat{x} \) (see Section 2.1 for more details on floating point arithmetics). A first simple rounding model consists of introducing an error term \( e_i \) for each floating point operation, as well as for each floating point variable. For instance, \( \hat{x} \odot \hat{x} \) corresponds to \(((1 + e_1) x (1 + e_1) x)(1 + e_2)\), where \( e_1 \) is the error term between \( x \) and \( \hat{x} \), and \( e_2 \) is the one associated to the operation \( \odot \). Let \( e \) be the vector of all error terms \( e_i \). Given \( e_i \in [-\varepsilon, \varepsilon] \) for all \( i \), with \( \varepsilon \) being the machine precision, we can write the floating point approximation \( \hat{f} \) of \( f \) as follows:

\[
 \hat{f}(x, e) = (((1 + e_1) x (1 + e_1) x)(1 + e_2) - x(1 + e_1))(1 + e_3).
\]

Then, the absolute roundoff error is defined by:

\[
 r(x, e) := \max_{\hat{x} \in [0,1]} (|\hat{f}(x, e) - f(x)|).
\]

However, we can make this computation easier with a slight approximation:

\[
 |\hat{f}(x, e) - f(x)| \leq |l(x, e)| + |h(x, e)| \text{ with } l(x, e) \text{ being the sum of the terms of}
\]
Roundoff errors with Bernstein expansions & Krivine-Stengle representations

\[
\hat{f}(x, e) = f(x) + O(\|e\|) - f(x)
\]
which are linear in \(e\), and
\[
h(x, e) = \sum \text{terms non-linear in } e
\]
are the sum of the terms which are non-linear in \(e\). The term \(\|h(x, e)\|\) can then be over-approximated by \(O(\|e\|^2)\) which is in general negligible compared to \(\|l(x, e)\|\), and can be bounded using standard interval arithmetic. For this reason, we focus on computing an upper bound of \(\|l(x, e)\|\).

In the context of our example, \(l(x, e)\) is given by:

\[
l(x, e) = (2x^2 - x)e_1 + x^2 e_2 + (x^2 - x)e_3.
\]

We divide each error term \(e_j\) by \(e\), and then consider the (scaled) linear part \(l' := \frac{1}{e}\) of the roundoff error with the error terms \(e \in [-1, 1]^3\). For all \(x \in [0, 1]\) and \(e \in [-1, 1]^3\), one can easily compute a valid upper bound of \(\|l'(x, e)\|\) with interval arithmetic. Surcharging the notation for elementary operations +, −, \times in interval arithmetic, one has \(l'(x, e) \in (\ldots \times [-1, 1] \times [0, 1] \times [-1, 1] + \ldots) = [-2.25, 2.25]\), yielding \(\|l(x, e)\| \leq 2.25e\).

Using the first method based on Bernstein expansion detailed in Section 3.1, we compute 2\(\epsilon\) as an upper bound of \(\|l(x, e)\|\) in 0.23s using FPKern(b) a rational arithmetic implementation. With the second method based on sparse Krivine-Stengle representation detailed in Section 3.2, we also compute an upper bound of 2\(\epsilon\) in 0.03s.

Although on this particular example, the method based the sparse Krivine-Stengle representation appears to be more time-efficient, in general the computational cost of the method based on Bernstein expansions is lower. For this example, the bounds provided by both methods are tighter than the ones determined by interval arithmetic. We emphasize the fact that the bounds provided by our two methods can be certified. Indeed, in the first case, the Bernstein coefficients (see Sections 2.2 and 3.1) can be computed either with rational arithmetic or certified interval arithmetic to ensure guaranteed values of upper bounds. In the second case, the positivity certificates are directly provided by sparse Krivine-Stengle representations.

1.2 Related Works

We first mention two tools, based on positivity certificates, to compute roundoff error bounds. The first tool, related to [3], relies on a similar approach to our second method. It uses dense Krivine-Stengle representations of positive polynomials to cast the initial problem as a finite dimensional LP problem. To reduce the size of this possibly large LP, [3] provides heuristics to eliminate some variables and constraints involved in the dense representation. However, this approach has the main drawback of loosing the property of convergence toward optimal solutions of the initial problem. Our second method uses sparse representations and is based on the previous works by [2] and [27], allowing to ensure the convergence towards optimal solutions while greatly reducing the computational cost of LP problems. Another tool, Real2Float [19], exploits sparsity in the same way while using Putinar representations of positive polynomials, leading to solving semidefinite (SDP) problems. Bounds provided by such SDP relaxations are in general more precise than LP relaxations [15], but the solving
cost is higher. Several other tools are available to compute floating point roundoff errors. SMT solvers are efficient when handling linear programs, but often provide coarse bounds for non-linear programs, e.g., when the analysis is done in isolation \cite{5}. The Rosa \cite{5} tool is a solver mixing SMT and interval arithmetic which compiles functional \textsc{Scala} programs implementing non-linear functions (involving $/$, $\sqrt{}$, and polynomials) as well as conditional statements. SMT solvers are theoretically able to output certificates which can be validated externally afterwards. 

The \texttt{Rosa} tool \cite{5}, relies on Symbolic Taylor expansion method, which consists of a branch and bound algorithm based on interval arithmetic. Bernstein expansions have been extensively used to handle systems of polynomial equations \cite{21,23} as well as systems of polynomial inequalities (including polynomial optimization), see for example \cite{23,8,22}. Yet, to the best of our knowledge, there is no tool based on Bernstein expansions in the context of roundoff error computation. The \texttt{Gappa} tool provides certified bounds with elaborated interval arithmetic procedure relying on multiple-precision dyadic fractions. The static analysis tool \texttt{FLUCTUAT} \cite{7} performs forward computation (by contrast with optimization) to analyze floating point C programs. Both \texttt{FLUCTUAT} and \texttt{Gappa} use a different rounding model (see Section \ref{sec:background}), also available in \texttt{FPTaylor}, that we do not handle in our current implementation. Some tools also allow formal validation of certified bounds. \texttt{FPTaylor}, \texttt{Real2Float} \cite{19}, as well as \texttt{Gappa} \cite{6} provide formal proof certificates, with HOL-Light \cite{12} for the first case, and Coq \cite{4} for the two other ones.

1.3 Key Contributions

Here is a summary of our key contributions:

- We present two new methods to compute upper bounds of floating point roundoff errors for programs implementing multivariate polynomial functions with input variables constrained to boxes. The first one is based on Bernstein expansions and the second one relies on sparse Krivine-Stengle representations. We also propose a theoretical framework to guarantee the validity of upper bounds computed with both methods (see Section \ref{sec:theory}). In addition, we give an alternative shorter proof in Section \ref{sec:theory} for the existence of Krivine-Stengle representations for sparse positive polynomials (proof of Theorem \ref{thm:existence}).

- We release two software packages based on each method. The first one, called \texttt{FPBern}\footnote{https://github.com/roccaa/FPBern}, computes the bounds using the Bernstein expansions, with two modules built on top of the software related to \cite{8}: \texttt{FPBern(a)} is a C++ module using double precision floating point arithmetic while \texttt{FPBern(b)} is a Matlab module using rational arithmetic. The second one \texttt{FPKriSten}\footnote{https://github.com/roccaa/FPKriSten} computes the bounds using Krivine-Stengle representations in Matlab. \texttt{FPKriSten} is built on top of the implementation related to \cite{27}. 


We compare our two methods implemented in FPPern and FPKriSten to three state-of-the-art methods. Our new methods have similar precision with the compared tools (Real2Float, Rosa, FPTaylor). At the same time, FPPern(a) shows an important time performance improvement, while FPPern(b) and FPKriSten has similar time performances compared with the other tools, yielding promising results.

The rest of the paper is organized as follows: in Section 2 we give basic background on floating point arithmetic, the Bernstein expansions and Krivine-Stengle representations. In Section 3 we give the main contributions, that is the computation of roundoff error bounds using Bernstein expansions and sparse Krivine-Stengle representations. Finally, in Section 4 we compare the performance and precision of our two methods with the existing tools, and show the advantages of our tools.

2 Preliminaries

We first recall useful notation on multivariate calculus. For \( \mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n \) and the multi-index \( \mathbf{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}^n \), we denote by \( \mathbf{x}^{\mathbf{\alpha}} \) the product \( \prod_{i=1}^{n} x_i^{\alpha_i} \). We also define \( |\mathbf{\alpha}| = |\alpha_1| + \ldots + |\alpha_n| \), \( \mathbf{0} = (0, \ldots, 0) \) and \( \mathbf{1} = (1, \ldots, 1) \).

The notation \( \sum_{\mathbf{\alpha}} \) is the nested sum \( \sum_{\alpha_1} \ldots \sum_{\alpha_n} \). Equivalently we have \( \prod_{\mathbf{\alpha}} \) which is equal to the nested product \( \prod_{\alpha_1} \ldots \prod_{\alpha_n} \).

Given another multi-index \( \mathbf{d} = (d_1, \ldots, d_n) \in \mathbb{N}^n \), the inequality \( \mathbf{\alpha} < \mathbf{d} \) (resp. \( \mathbf{\alpha} \leq \mathbf{d} \)) means that the inequality holds for each sub-index: \( \alpha_1 < d_1, \ldots, \alpha_n < d_n \) (resp. \( \alpha_1 \leq d_1, \ldots, \alpha_n \leq d_n \)). Moreover, the binomial coefficient \( \binom{d}{\alpha} \) is the product \( \prod_{\alpha_i} \binom{d_i}{\alpha_i} \).

Let \( \mathbb{R}[x] \) be the vector space of multivariate polynomials. Given \( f \in \mathbb{R}[x] \), we associate a multi-degree \( \mathbf{d} = (d_1, \ldots, d_n) \) to \( f \), with each \( d_i \) standing for the degree of \( f \) with respect to the variable \( x_i \). Then, we can write \( f(\mathbf{x}) = \sum_{\gamma \leq \mathbf{d}} a_{\gamma} \mathbf{x}^{\gamma} \), with \( a_{\gamma} \) (also noted \( (f)_{\gamma} \)) being the coefficients of \( f \) in the monomial basis and each \( \gamma \in \mathbb{N}^n \) is a multi-index. The degree \( d \) of \( f \) is given by \( d := \max_{\gamma, a_\gamma \neq 0} |\gamma| \).

As an example, if \( f(x_1, x_2) = x_1^2 x_2 + x_1 x_2^2 \) then \( \mathbf{d} = (4, 3) \) and \( d = 5 \). For the polynomial \( l \) used in Section 4.1 one has \( \mathbf{d} = (2, 1, 1, 1) \) and \( d = 3 \).

2.1 Floating Point arithmetic

This section gives background on floating point arithmetic, inspired from material available in [25, Section 3]. The IEEE754 standard [25] defines a binary floating point number as a triple significant, sign, and exponent (denoted by \( \text{sig}, \text{sgn}, \text{exp} \)) which represents the numerical value of \( (-1)^{\text{sgn}} \times \text{sig} \times 2^{\exp} \). The standard describes 3 formats (32, 64, and 128 bits) which vary by the size of the significant and the exponent, as well as special values (such as NaN, the infinities). Denoting by \( \mathbb{F} \) the set of floating point numbers, we call rounding operator, the function \( \text{rnd} : \mathbb{R} \to \mathbb{F} \) which takes a real number and returns the
closest floating point number rounded to the nearest, toward zero, or toward ±∞. A simple model of rounding is given by the following formula:

\[ \text{rnd}(x) = x(1 + e) + u, \]

with \(|e| \leq \varepsilon, |u| \leq \mu \) and \(eu = 0\). The value \(\varepsilon\) is the maximal relative error (given by the machine precision) and \(\mu\) is the maximal absolute error for numbers very close to 0. For example, in the single (32 bits) format, \(\varepsilon\) is equal to \(2^{-24}\) while \(\mu\) equals \(2^{-150}\). It is clear that in general \(\mu\) is negligible compared to \(\varepsilon\), thus we neglect terms depending on \(u\) in the remainder of this paper.

Given an operation \(op : \mathbb{R}^n \rightarrow \mathbb{R}\), let \(op_{FP}\) be the corresponding floating point operation. An operation is exactly rounded when \(op_{FP}(x) = \text{rnd}(op(x))\), for all \(x \in \mathbb{R}^n\).

In the IEEE754 standard the following operations are defined as exactly rounded: +, -, ×, ÷, and the fma operation. It follows that for those operations we have the continuation of the simple rounding model \(op_{FP}(x) = op(x)(1 + e)\).

The previous rounding model is called “simple” in contrast with more improved rounding model. Given the function \(pc(x) = \max_{k \in \mathbb{Z}} \{2^k : 2^k < x\}\), then the improved rounding model is defined by: \(op_{FP}(x) = op(x) + pc(op(x))\), for all \(x \in \mathbb{R}^n\). As the function \(pc\) is piecewise constant, this rounding model needs design of algorithms based on successive subdivisions, which is not currently handled in our methods. Combining branch and bound algorithms with interval arithmetic is adapted to roundoff error computation with such rounding model, which is the case with FLUCTUAT, Gappa, and FPTaylor.

2.2 Bernstein Expansion of Polynomials

In this section we give mandatory background on the Bernstein expansion for the contribution detailed in Section 3.1. Given a multivariate polynomial \(f \in \mathbb{R}[x]\), we recall how to compute a lower bound of \(f^* := \min_{x \in [0,1]^n} f(x)\). The next result can be retrieved in [9, Theorem 2]:

**Theorem 1 (Multivariate Bernstein expansion).** Given a multivariate polynomial \(f\) and a degree \(k \geq d\) with \(d\) the multi-degree of \(f\), then the Bernstein expansion of multi-degree \(k\) of \(f\) is given by:

\[ f(x) = \sum_{\gamma} a_\gamma x^\gamma = \sum_{\alpha \leq k} b_{\alpha} f_{k,\alpha}(x). \]

where \(b_{\alpha}(f)\) (also denoted by \(b_{\alpha}\) when there is no confusion) are the Bernstein coefficients (of multi-degree \(k\)) of \(f\), and \(B_{k,\alpha}(x)\) are the Bernstein basis polynomials defined by \(B_{k,\alpha}(x) := \Pi_{i=1}^n B_{k_i,\alpha_i}(x_i)\) and \(B_{k_i,\alpha_i}(x_i) := \left(\begin{array}{c} k_i \\ \alpha_i \end{array}\right)(1 - x_i)^{k_i - \alpha_i}\). The Bernstein coefficients are given by the following formulas:

\[ b_{\alpha} = \sum_{\beta < \alpha} \left(\begin{array}{c} \frac{\alpha}{\beta} \\ \frac{\beta}{\beta} \end{array}\right) a_\beta, \quad 0 \leq \alpha \leq k. \]

\[ \text{fma} \] operator is defined by \(\text{fma}(x, y, z) = x \times y + z\).
The Bernstein expansion having numerous properties, we give only four of them which are useful for Section 3.1. For a more exhaustive introduction to Bernstein expansion, as well as some proof of the basic properties, we refer the interested reader to [23].

Property 1 (Cardinality [23, (3.14)]). The number of Bernstein coefficients in the Bernstein expansion (of multi-degree $k$) is equal to $(k+1)^n = \prod_{i=1}^{n} (k_i + 1)$.

Property 2 (Linearity [23, (3.2.3)]). Given two polynomials $p_1$ and $p_2$, one has:
$$b^{(c p_1 + p_2)}_\alpha = c b^{(p_1)}_\alpha + b^{(p_2)}_\alpha, \quad \forall c \in \mathbb{R},$$
where Bernstein expansions with same multi-degrees are considered.

Property 3 (Enclosure [23, (3.2.4)]). The minimum (resp. maximum) of a polynomial $f$ over $[0,1]^n$ can be lower bounded (resp. upper bounded) by the minimum (resp. maximum) of its Bernstein coefficients:
$$\min_{\alpha \leq k} b_\alpha \leq f(x) \leq \max_{\alpha \leq k} b_\alpha, \quad \forall x \in [0,1]^n.$$

Property 4 (Sharpness [23, (3.2.5)]). If the minimum (resp. maximum) of the $b_\alpha$ is reached for $\alpha$ in a corner of the box $[0,k_1] \times \cdots \times [0,k_n]$, then $b_\alpha$ is the minimum (resp. maximum) of $f$ over $[0,1]^n$.

Property 1 gives the maximal computational cost needed to find a lower bound of $f^*$ for a Bernstein expansion of fixed multi-degree $k$. Property 3 is used to bound from below optimal values, while Property 4 allows to determine if the lower bound is optimal.

### 2.3 Dense and Sparse Krivine-Stengle Representations

In this section, we first give the necessary background on Krivine-Stengle representations, used in the context of polynomial optimization. Then, we present a sparse version based on [11]. These notions are applied later in Section 3.2.

**Dense Krivine-Stengle representations.** Krivine-Stengle certificates for positive polynomials can first be found in [11,20] (see also [16, Theorem 1(b)]). Such certificates give representations of positive polynomials over a set $K = \{x \in \mathbb{R}^n : 0 \leq g_i(x) \leq 1, \ i = 1, \ldots, p\}$, with $g_1, \ldots, g_p \in \mathbb{R}[x]$. The compact set $K$ is a basic semialgebraic set, since it is defined as a conjunction of polynomial inequalities.

Given $\alpha = (\alpha_1, \ldots, \alpha_p)$ and $\beta = (\beta_1, \ldots, \beta_p)$, let us define the polynomial $h_{\alpha,\beta}(x) = g^\alpha(1-g)^\beta = \prod_{i=1}^{p} g_i^{\alpha_i}(1-g_i)^{\beta_i}$.

For instance on the two-dimensional unit box, one has $n = p = 2$, $K = [0,1]^2 = \{x \in \mathbb{R}^2 : 0 \leq x_1 \leq 1, \ 0 \leq x_2 \leq 1\}$. With $\alpha = (2,1)$ and $\beta = (1,3)$, one has $h_{\alpha,\beta}(x) = x_1^2 x_2 (1-x_1)(1-x_2)^3$. 

Theorem 2 (Dense Krivine-Stengle representations). Let \( \psi \in \mathbb{R}[x] \) be a positive polynomial over \( K \). Then there exist \( k \in \mathbb{N} \) and a finite number of nonnegative weights \( \lambda_{\alpha, \beta} \geq 0 \) such that:
\[
\psi(x) = \sum_{|\alpha + \beta| \leq k} \lambda_{\alpha, \beta} h_{\alpha, \beta}(x), \quad \forall x \in \mathbb{R}^n. \tag{4}
\]
It is possible to compute the weights \( \lambda_{\alpha, \beta} \) by identifying in the monomial basis the coefficients of the polynomials in the left and right sides of (4). Denoting by \( (\psi)_{\gamma} \) the monomial coefficients of \( \psi \), with \( \gamma \in \mathbb{N}_k^n := \{ \gamma \in \mathbb{N}^n : |\gamma| \leq k \} \), the \( \lambda_{\alpha, \beta} \) fulfill the following equalities:
\[
(\psi)_{\gamma} = \sum_{|\alpha + \beta| \leq k} \lambda_{\alpha, \beta} (h_{\alpha, \beta})_{\gamma}, \quad \forall \gamma \in \mathbb{N}_k^n. \tag{5}
\]

Global optimization using the dense Krivine-Stengle representations.
Here we consider the polynomial minimization problem \( f^* := \min_{x \in K} f(x) \), with \( f \) a polynomial of degree \( d \). We can rewrite this problem as the following infinite dimensional problem:
\[
f^* := \max_{t \in \mathbb{R}} t, \quad \text{s.t. } f(x) - t \geq 0, \quad \forall x \in K. \tag{6}
\]
The idea is to look for a hierarchy of finite dimensional linear programming (LP) relaxations by using Krivine-Stengle representations of the positive polynomial \( \psi = f - t \) involved in Problem (6). Applying Theorem 2 to this polynomial, we obtain the following LP problem for each \( k \geq d \):
\[
p_k^* := \max_{t, \lambda_{\alpha, \beta}} t, \quad \text{s.t. } (f - t)_{\gamma} = \sum_{|\alpha + \beta| \leq k} \lambda_{\alpha, \beta} (h_{\alpha, \beta})_{\gamma}, \quad \forall \gamma \in \mathbb{N}_k^n; \tag{7}
\]
\[
\lambda_{\alpha, \beta} \geq 0.
\]
As in [16, (4)], one has:

Theorem 3 (Dense Krivine-Stengle LP relaxations). The sequence of optimal values \( (p_k^*) \) satisfies \( p_k^* \to f^* \) as \( k \to +\infty \). Moreover each \( p_k^* \) is a lower bound of \( f^* \).

At fixed \( k \), the total number of variables of Problem (7) is given by the number of \( \lambda_{\alpha, \beta} \) and \( t \), that is \( (2n+k)+1 \). The number of constraints is equal to the cardinality of \( \mathbb{N}_k^n \), which is \( \binom{n+k}{k} \).

Sparse Krivine-Stengle representations. We now explain how to derive less computationally expensive LP relaxations, by relying on sparse Krivine-Stengle representations. For \( I \subseteq \{1, \ldots, n\} \), let \( \mathbb{R}[x, I] \) be the ring of polynomials restricted to the variables \( \{x_i : i \in I\} \). We borrow the notion of a sparsity pattern from [27, Assumption 1]:
Definition 1 (Sparsity Pattern). Given $m \in \mathbb{N}$, $I_j \subseteq \{1, \ldots, n\}$, and $J_j \subseteq \{1, \ldots, p\}$ for all $j = 1, \ldots, m$, a sparsity pattern is defined by the four following conditions:

- $f$ can be written as: $f = \sum_{j=1}^{m} f_j$ with $f_j \in \mathbb{R}[x, I_j]$,
- $g_j \in \mathbb{R}[x, I_j]$ for all $i \in J_j$, for all $j = 1, \ldots, m$,
- $\bigcup_{j=1}^{m} I_j = \{1, \ldots, n\}$ and $\bigcup_{j=1}^{m} J_j = \{1, \ldots, p\}$,
- (Running Intersection Property) for all $j = 1, \ldots, m - 1$, there exists $s \leq j$ s.t. $I_{j+1} \cap \bigcup_{i=1}^{j} I_i \subseteq I_s$.

As an example, the four conditions stated in Definition 1 are satisfied while considering $f(x) = x_1 x_2 + x_1^2 x_3$ on the hypercube $K = [0,1]^3$. Indeed, one has $f_1(x) = x_1 x_2 \in \mathbb{R}[x, I_1]$, $f_2(x) = x_1^2 x_3 \in \mathbb{R}[x, I_2]$ with $I_1 = \{1,2\}$, $I_2 = \{1,3\}$.

Let us consider a given sparsity pattern as stated above. By noting $n_j = |I_j|$, $p_j = |J_j|$, then the set $K = \{ x \in \mathbb{R}^n : 0 \leq g_j(x) \leq 1, i = 1, \ldots, p \}$ yields subsets $K_j = \{ x \in \mathbb{R}^n : 0 \leq g_j(x) \leq 1, i \in J_j \}$, with $j = 1, \ldots, m$. If $K$ is a compact subset of $\mathbb{R}^n$ then each $K_j$ is a compact subset of $\mathbb{R}^n$. In the dense case, let us note $h_{\alpha_j, \beta_j} := g^{\alpha_j}/(1-g)^{\beta_j}$, for given $\alpha_j, \beta_j \in \mathbb{N}^n$.

The following result, a sparse variant of Theorem 2, can be retrieved from [27, Theorem 1] but we also provide here a shorter alternative proof by using [11].

Theorem 4 (Sparse Krivine-Stengle representations). Let $f, g_1, \ldots, g_p \in \mathbb{R}[x]$ be given and assume that there exist $I_j$ and $J_j$, $j = 1, \ldots, m$, which satisfy the four conditions stated in Definition 1. If $f$ is positive over $K$, then there exist $\phi_j \in \mathbb{R}[x, I_j]$, $j = 1, \ldots, m$ such that $f = \sum_{j=1}^{m} \phi_j$ and $\phi_j > 0$ over $K_j$.

In addition, there exist $k \in \mathbb{N}$ and finitely many nonnegative weights $\lambda_{\alpha_j, \beta_j}$, $j = 1, \ldots, m$, such that:

$$\phi_j = \sum_{|\alpha_j + \beta_j| \leq k} \lambda_{\alpha_j, \beta_j} h_{\alpha_j, \beta_j}, \quad j = 1, \ldots, m.$$  

(8)

Proof. From [11, Lemma 3], there exist $\phi_j \in \mathbb{R}[x, I_j]$ such that $f = \sum_{j=1}^{m} \phi_j$ and $\phi_j > 0$ on $K_j$. Applying Theorem 2 on each $\phi_j$, there exist $k_j \in \mathbb{N}$ and finitely many nonnegative weights $\lambda_{\alpha_j, \beta_j}$ such that $\phi_j = \sum_{|\alpha_j + \beta_j| \leq k_j} \lambda_{\alpha_j, \beta_j} h_{\alpha_j, \beta_j}$.

With $k = \max_{1 \leq j \leq m} \{k_j\}$, we complete the representations with as many zero as necessary, we obtain the desired result.

In Theorem 4 one assumes that $f$ can be written as the sum $f = \sum_{j=1}^{m} f_j$, where each $f_j$ is not necessarily positive. The first result of the theorem states that that $f$ can be written as another sum $f = \sum_{j=1}^{m} \phi_j$, where each $\phi_j$ is now positive. As in the dense case, the $\lambda_{\alpha_j, \beta_j}$ can be computed by equalizing the coefficients in the monomial basis. We also obtain a hierarchy of LP relaxations to approximate the solution of polynomial optimization problems. For the sake of conciseness, we only provide these relaxations as well as their computational costs in the particular context of roundoff error bounds in Section 3.3.
3 Two new methods to compute roundoff errors bounds

This section is dedicated to our main contributions. We provide two new methods to compute absolute roundoff error bounds using either Bernstein expansions or sparse Krivine-Stengle representations. Here we consider a given program which implements a polynomial expression $f$ with input variables $x$ satisfying a set of input constraints encoded by $X$. We restrict ourselves to the case where $X$ is the unit box $[0, 1]^n$.

Following the simple rounding model described in Section 2.1, we note $\hat{f}(x, e)$ the rounded expression of $f$ after introduction of the rounding variables $e$ (one additional variable is introduced for each real variable $x_i$ or constant as well as for each arithmetic operation $+, \times$ or $-$). For a given machine epsilon $\varepsilon$, these error variables also satisfy a set of constraints encoded by the box $[-\varepsilon, \varepsilon]^m$. As explained in [19, Section 3.1], we can decompose the roundoff error as follows: $r(x, e) := \hat{f}(x, e) - f(x) = l(x,e) + h(x,e)$, where $l(x,e) := \sum_{j=1}^{m} \frac{\partial f}{\partial x_j}(x,0) e_j = \sum_{j=1}^{m} s_j(x) e_j$. One obtains an enclosure of $h$ using interval arithmetic to bound second-order error terms in the Taylor expansion of $r$ w.r.t. $e$ (as in [25,19]).

We note $d$ the degree of $l$. After dividing each error variable $e_j$ by $\varepsilon$, we now consider the optimization of the (scaled) linear part $l' := l/\varepsilon$ of the roundoff error. In other words, we focus on computing upper bounds of the maximal absolute value $l'^* := \max_{(x,e) \in X \times E} |l'(x, e)|$ where $E = [-1, 1]^m$.

3.1 Bernstein expansions of roundoff errors

The first method is the approximation of $l'^*$ with the Bernstein expansions. Let $d$ be the multi-degree of $l'$. From the above definition of $l$, note that $d$ is also the multi-degree of $f$. For each $k \geq d$, let us note $l'_k := \max_{\alpha \leq k} \sum_{j=1}^{m} |b^{(s_j)}_{\alpha}|$ and $l'^*_k := -l'_k$. Our procedure is based on the following lemma:

**Lemma 1.** For each $k \geq d$, the polynomial $l'(x, e)$ can be bounded as follows:

$$l'_k \leq l'(x, e) \leq l'^*_k, \quad \forall (x, e) \in X \times E.$$  \hfill (9)

**Proof.** Write $l'_k \in \mathbb{R}[x]$ the polynomial $l'(x, e)$ for a given $e \in E$. Property 3 provides the enclosure of $l'_k(x)$ w.r.t. $x$ for a given $e \in E$:

$$\min_{\alpha \leq k} b^{(s_j)}_{\alpha} \leq l'_k(x) \leq \max_{\alpha \leq k} b^{(s_j)}_{\alpha}, \quad \forall x \in [0, 1]^n,$$  \hfill (10)

where each Bernstein coefficient satisfies $b^{(s_j)}_{\alpha} = \sum_{j=1}^{m} e_j h^{(s_j)}_{\alpha}$ by Property 2 (each $e_j$ being a scalar in $[-1, 1]$). The proof of the left inequality comes from:

$$\min_{e \in [-1,1]^m} \left( \min_{\alpha \leq k} \sum_{j=1}^{m} e_j h^{(s_j)}_{\alpha} \right) = \min_{\alpha \leq k} \left( \min_{e \in [-1,1]^m} \left( \sum_{j=1}^{m} e_j h^{(s_j)}_{\alpha} \right) \right) = \min_{\alpha \leq k} \sum_{j=1}^{m} -|h^{(s_j)}_{\alpha}| = -\max_{\alpha \leq k} \sum_{j=1}^{m} |h^{(s_j)}_{\alpha}|.$$


The proof of the right inequality is similar. □

Remark 1. The computational cost of $l'_*^k$ is $m(k + 1)^1$ since we need to compute the Bernstein coefficients for each $s_j(x)$. This cost is polynomial in the degree and exponential in $n$ but is linear in $m$. In the implementation described in Section 4 we first compute each $b'_j(e)$ as a function of $e$ and then optimize afterwards over $[-1, 1]^m$.

Example 1. For the polynomial $l$ defined in (11) (Section 1.1), one has $l(x, e) = (2x^2 - x)e_1 + x^2 e_2 + (x^2 - x)e_3$. Applying the above method with $k = d = 2$, one considers the following Bernstein coefficients:

$$b'_0(e) = 0, \quad b'_1(e) = -\frac{e_1}{2} - \frac{e_3}{2}, \quad b'_2(e) = e_1 + e_2.$$ 

The number of Bernstein coefficients w.r.t. $x$ is 3, which is much lower than the one w.r.t. $(x, e)$, which is equal to 24. One can obtain an upper bound (resp. lower bound) by taking the maximum (resp. minimum) of the Bernstein coefficients. In this case, $\max_{e \in [-1, 1]} b'_j(e) = 0$, $\max_{e \in [-1, 1]} b'_2(e) = 1$ and $\max_{e \in [-1, 1]} b'_3(e) = 2$. Thus, one obtains $\overline{l} = 2$ as an upper bound of $l^*$ yielding $l^* \leq 2\varepsilon$.

### 3.2 Sparse Krivine-Stengle representations of roundoff errors

Here we explain how to compute lower bounds of $l'_* := \min_{(x, e) \in X \times E} l'(x, e)$ by using sparse Krivine-Stengle representations. We obtain upper bounds of $\overline{l} := \max_{(x, e) \in X \times E} l'(x, e)$ in a similar way.

For the sake of consistency with Section 2.3, we introduce the variable $y \in \mathbb{R}^{n+m}$ defined by $y_j := x_j$, $j = 1, \ldots, n$ and $y_j := e_{j-n}$, $j = n + 1, \ldots, n + m$. Then, one can write the set $K = X \times E$ as follows:

$$K = \{ y \in \mathbb{R}^{n+m} : 0 \leq g_j(y) \leq 1, \quad j = 1, \ldots, n + m \}, \quad (11)$$

with $g_j(y) := x_j$, for each $j = 1, \ldots, n$ and $g_j(y) := \frac{1}{2} + \frac{e_j}{2}$, for each $j = n + 1, \ldots, n + m$.

Lemma 2. For each $j = 1, \ldots, m$, let us define $I_j := \{1, \ldots, n + j\}$ and $J_j := I_j$. Then the sets $I_j$ and $J_j$ satisfy the four conditions stated in Definition 7.

Proof. The first condition holds as $l'(y) = l'(x, e) = \sum_{j=1}^m s_j(x, e)e_j = \sum_{j=1}^m s_j(y)e_j$, with $s_j(y) \in \mathbb{R}[y, I_j]$. The second and third condition are obvious. The running intersection property comes from $I_{j+1} \cap I_j = \{1, \ldots, n\} \subseteq I_j$. □

Given $\alpha, \beta \in \mathbb{N}^{n+1}$, one can write $\alpha = (\alpha', \gamma)$ and $\beta = (\beta', \delta)$, for $\alpha', \beta' \in \mathbb{N}^n$, $\gamma, \delta \in \mathbb{N}$. In our case, this gives the following formula for the polynomial $h_{\alpha, \beta}(y) = g^\alpha(1 - g)^\beta$:

$$h_{\alpha, \beta}(y) = h_{\alpha', \beta', \gamma, \delta}(x, e) = x^{\alpha'}(1 - x)^{\beta'}(\frac{1}{2} + \frac{e_j}{2})^{\gamma_j}(\frac{1}{2} - \frac{e_j}{2})^{\delta_j}.$$
For instance, with the polynomial \( l' \) considered in Section 11 and depending on \( x, e_1, e_2, e_3 \), one can consider the multi-indices \( \alpha_1 = (1, 2), \beta_1 = (2, 3) \) associated to the roundoff variable \( e_1 \). Then \( h_{\alpha_1, \beta_1}(y) = x(1-x)^2(\frac{1}{2} + \frac{e_1}{2})^2(\frac{1}{2} - \frac{e_1}{2})^3 \).

Now, we consider the following hierarchy of LP relaxations, for each \( k \geq d \):

\[
\begin{align*}
\ell_k' & := \max_{t, \lambda_{\alpha_j, \beta_j}} t, \\
\text{s.t} \quad & l' - t = \sum_{j=1}^{m} \phi_j, \\
& \phi_j = \sum_{|\alpha_j + \beta_j| \leq k} \lambda_{\alpha_j, \beta_j} h_{\alpha_j, \beta_j}, \quad j = 1, \ldots, m, \\
& \lambda_{\alpha_j, \beta_j} \geq 0, \quad j = 1, \ldots, m.
\end{align*}
\]

Similarly, we obtain \( \ell_k' \) while replacing max by min and \( l' - t \) by \( t - l' \) in LP (12).

**Lemma 3.** The sequence of optimal values \( \ell_k' \) (resp. \( \ell_k' \)) satisfies \( \ell_k' \uparrow \ell' \) (resp. \( \ell_k' \downarrow \ell' \)) as \( k \to +\infty \). In addition, \( \ell_k' := \max\{||\ell_k'||, ||\ell_k'||\} \to l'^* \) as \( k \to +\infty \).

**Proof.** By construction \( \ell_k' \) is monotone nondecreasing. For a given arbitrary \( \varepsilon' > 0 \), the polynomial \( l' - \ell' + \varepsilon' \) is positive over \( K \). By Lemma 2, the subsets \( I_j \) and \( J_j \) satisfy the four conditions stated in Definition 1, so we can apply Theorem 1 to \( l' - \ell' + \varepsilon' \). This yields the existence of \( \phi_j, j = 1, \ldots, m \), such that \( l' - \ell' + \varepsilon' = \sum_{j=1}^{m} \phi_j \) and \( \phi_j = \sum_{|\alpha_j + \beta_j| \leq k} \lambda_{\alpha_j, \beta_j} h_{\alpha_j, \beta_j}, j = 1, \ldots, m \).

Hence, \( (\ell_k', \varepsilon', \phi_j, \lambda_{\alpha_j, \beta_j}) \) is feasible for LP (12). It follows that there exists \( k \) such that \( \ell_k' \geq l' - \varepsilon' \). Since \( \ell_k' \leq l' \) and \( \varepsilon' \) has been arbitrary chosen, we obtain the convergence result for the sequence \( \ell_k' \). The proof is analogous for \( \ell_k' \) and yields \( \max\{||\ell_k'||, ||\ell_k'||\} \to \max\{||\ell'||, ||\ell'||\} = l'^* \) as \( k \to +\infty \), the desired result. \( \square \)

**Remark 2.** In the special case of roundoff error computation, one can prove that the number of variables of LP (12) is \( m^{2(n+1)+k} \) with a number of constraints equal to \( m^{(n+k)} + 1 \). This is in contrast with the dense case where the number of LP variables is \( (2(n+m)+k) \) with a number of constraints equal to \( (n+m+k) \).

**Proof of Remark 2.** As we replace a function \( \phi \) of dimension \( (n+m) \) by a sum of \( m \) functions \( \phi_j \) of dimension \( n+1 \), the number of coefficients \( \lambda_{\alpha_j, \beta_j} \) is \( m^{2(n+1)+k} \). This leads to a total of \( m^{2(n+1)+k} + 1 \) variables when adding \( t \).

The number of equality constraints is the number of monomials involved in \( \sum_{j=1}^{m} \phi_j \). Each \( \phi_j \) has \( (n+k) \) monomials. However there are redundant monomials between all the \( \phi_j \); the ones depending of only \( x \), and not \( e \). These \( (n+k) \) monomials should appear only once. This leads to a final number of \( m^{(n+1)+k} - (m-1)(n+k) \) monomials which is equal to \( m^k + 1 \). \( \square \)
Example 2. Continuing Example 1 for the polynomial \( l \) defined in (1) (Section 1.1), we consider LP (12) at the relaxation order \( k = d = 3 \). This problem involves \( 3(2^{(1+1)}+3) + 1 = 106 \) variables and \( \left(\frac{3^3}{3} + 1\right)\left(\frac{4}{3}\right) = 22 \) constraints. This is in contrast with a dense Krivine-Stengle representation, where the corresponding LP involves 35 linear equalities and 166 variables. Computing the values of \( l_k' \) and \( l_k'' \) provides an upper bound of 2 for \( l^* \), yielding \( l^* \leq 2\varepsilon \).

4 Implementation & Results

The \texttt{FPBern} and \texttt{FPKriSten} software packages. We provide two distinct software packages to compute certified error bounds of roundoff errors for programs implementing polynomial functions with floating point precision. The first tool \texttt{FPBern} relies on the method from Section 3.1 and the second tool \texttt{FPKriSten} on the method from Section 3.2.

\texttt{FPBern} is built on top of the software presented in [8] to manipulate Bernstein expansions, which includes a C++ module \texttt{FPBern(a)} and a Matlab module \texttt{FPBern(b)}. Their main difference is that Bernstein coefficients are computed with double precision floating point arithmetic in \texttt{FPBern(a)} and with rational arithmetic in \texttt{FPBern(b)}. Polynomial operations are handled with 
\texttt{GiNaC} in \texttt{FPBern(a)} and with Matlab Symbolic Toolbox in \texttt{FPBern(b)}. Note that the Bernstein coefficient computations are not fully certified with \texttt{FPBern(a)} yet.

\texttt{FPKriSten} is built on top of the SBSOS software related to [27] which handles sparse polynomial optimization problems by solving a hierarchy of convex relaxations. This hierarchy is obtained by mixing Krivine-Stengle and Putinar representations of positive polynomials. To improve the overall performance in our particular case, we only consider the former representation yielding the hierarchy of LP relaxations (12). Among several LP solvers, CPLEX [13] yields the best performance in our case (see also [1] for more comparisons). Polynomials are handled with the YALMIP [18] toolbox available for Matlab. Even though the semantics of programs considered in this paper is actually much simpler than that considered by other tools such as \texttt{Rosa} [5] or \texttt{Fluctuat} [7], we emphasize that those tools may be combined with external non-linear solvers to solve specific sub-problems, a task that either \texttt{FPBern} or \texttt{FPKriSten} can fulfill.

Experimental results. We tested our two software packages with 20 programs (see Appendix A) where 12 are existing benchmarks coming from biology, space control and optimization fields, and 8 are generated as follows, with \( x = (x_1, \ldots, x_n) \in [-1, 1]^n \).

\[
ex-n-n\text{Sum-deg}(x) := \sum_{j=0}^{n-1} \prod_{k=1}^{\text{deg} (\sum_{i=1}^{n} x_i)}. \tag{13}
\]

The first 9 programs are used for similar comparison in [19, Section 4.1], the following 3 come from [24]. Eventually the 8 generated benchmarks allow to
evaluate independently the performance of the tools w.r.t. either the number of input variables (through the variable $n$), the degree (through $deg$) or the number of error variables (through $nSum$). Taking $x \in [-1,1]^n$ allows avoiding monotonicity of the polynomial (which could be exploited by the Bernstein techniques).

We recall that each program implements a polynomial function $f(x)$ with box constrained input variables. To provide an upper bound of the absolute roundoff error $|f(x) - f(x,e)| = |l(x,e) + h(x,e)|$, we rely on Real2Float to generate $l$ and to bound $h$ (see [19, Section 3.1]). Then the optimization methods of Section 3 are applied to bound a function $l'$, obtained after linear transformation of $l$, over the unit box.

At a given multi-degree $k$, FPBern computes the bound $\overline{l}_k'$ (see Lemma 1). Similarly, at a given relaxation order $k$, FPKriSten computes the bound $\underline{l}_k'$ (see Lemma 3). To achieve fast computations, the default value of $k$ is the multi-degree $d$ of $l'_e$ (equal to the multi-degree of the input polynomial $f$) and the default value of $k$ is the degree $d$ of $l'$ (equal to the successor of the degree of $f$).

The experiments were carried out on an Intel Core i7-5600U (2.60Ghz, 16GB) with Ubuntu 14.04LTS, Matlab 2015a, GiNaC 1.7.1, and Cplex 12.63. Our benchmark settings are similar to [19, Section 4] as we compare the accuracy and execution times of our two tools with Rosa real compiler [5] (version from May 2014), Real2Float [19] (version from July 2016) and FPTaylor [25] (version from May 2016) on programs implemented in double precision while considering input variables as real variables. All these tools use a simple rounding model (see Section 2.1) and have been executed with their default parameters.

Table 1 shows the result of the absolute roundoff error while Table 2 displays execution times obtained through averaging over 5 runs. For each benchmark, we indicate the number $n$ (resp. $m$) of input (resp. error) variables as well as the degree $d$ of $l'$. For FPKriSten the CPLEX solving time in Table 2 is given between parentheses. Note that the overall efficiency of the tool could be improved by constructing the hierarchy of LP (12) with a C++ implementation.

Our two methods yield more accurate bounds for the 3 benchmarks kepler1, sineTaylor and kepler2, which is the program involving the largest number of error variables.

For kepler1, FPBern(a) and FPKriSten are less precise than FPBern(b) but are still 6% more precise than Real2Float and FPTaylor and 53% more precise than Rosa. For kepler2, our two tools are 3% (resp. 42%) more precise than FPTaylor and Real2Float (resp. Rosa). In addition, Property 4 holds for these three programs with FPBern(b), which ensures bound optimality. For all other benchmarks FPTaylor provides the most accurate upper bounds. Our tools are more accurate than Real2Float except for sineOrder3 and himmilbeau. In particular, for himmilbeau, FPBern and FPKriSten are 40% (resp. 50%) less precise than Real2Float (resp. FPTaylor). One way to obtain better bounds would be to increase the degree $k$ (resp. relaxation order $k$) within FPBern.
Table 1. Comparison results of upper bounds for absolute roundoff errors. The best results are emphasized using bold fonts.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>n</th>
<th>m</th>
<th>d</th>
<th>FFPBern(a)</th>
<th>FFPBern(b)</th>
<th>FPKriSten</th>
<th>Real2Float</th>
<th>Rosa</th>
<th>FPTaylor</th>
</tr>
</thead>
<tbody>
<tr>
<td>rigidBody1</td>
<td>3</td>
<td>10</td>
<td>3</td>
<td>5.33e−13</td>
<td>5.33e−13</td>
<td>5.33e−13</td>
<td>5.33e−13</td>
<td>5.05e−13</td>
<td>3.87e−13</td>
</tr>
<tr>
<td>rigidBody2</td>
<td>3</td>
<td>15</td>
<td>5</td>
<td>6.48e−11</td>
<td>6.48e−11</td>
<td>6.48e−11</td>
<td>6.48e−11</td>
<td>6.48e−11</td>
<td>5.24e−11</td>
</tr>
<tr>
<td>kepler0</td>
<td>6</td>
<td>21</td>
<td>3</td>
<td>1.08e−13</td>
<td>1.08e−13</td>
<td>1.08e−13</td>
<td>1.18e−13</td>
<td>1.16e−13</td>
<td>1.05e−13</td>
</tr>
<tr>
<td>kepler1</td>
<td>4</td>
<td>28</td>
<td>4</td>
<td>4.23e−13</td>
<td>4.04e−13</td>
<td>4.23e−13</td>
<td>4.47e−13</td>
<td>6.49e−13</td>
<td>4.49e−13</td>
</tr>
<tr>
<td>kepler2</td>
<td>6</td>
<td>42</td>
<td>2</td>
<td>2.03e−12</td>
<td>2.03e−12</td>
<td>2.03e−12</td>
<td>2.09e−12</td>
<td>2.89e−12</td>
<td>2.10e−12</td>
</tr>
<tr>
<td>sineTaylor</td>
<td>1</td>
<td>13</td>
<td>8</td>
<td>5.51e−16</td>
<td>5.48e−16</td>
<td>5.51e−16</td>
<td>6.03e−16</td>
<td>9.56e−16</td>
<td>6.75e−16</td>
</tr>
<tr>
<td>sineOrder3</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>1.35e−15</td>
<td>1.35e−15</td>
<td>1.25e−15</td>
<td>1.19e−15</td>
<td>1.11e−15</td>
<td>9.97e−16</td>
</tr>
<tr>
<td>sqrt</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1.29e−15</td>
<td>1.29e−15</td>
<td>1.29e−15</td>
<td>1.29e−15</td>
<td>8.41e−16</td>
<td>7.13e−16</td>
</tr>
<tr>
<td>himmilbeau</td>
<td>2</td>
<td>11</td>
<td>5</td>
<td>2.00e−12</td>
<td>2.00e−12</td>
<td>1.97e−12</td>
<td>1.43e−12</td>
<td>1.43e−12</td>
<td>1.32e−12</td>
</tr>
<tr>
<td>schwefel</td>
<td>3</td>
<td>15</td>
<td>5</td>
<td>1.48e−11</td>
<td>1.48e−11</td>
<td>1.48e−11</td>
<td>1.49e−11</td>
<td>1.49e−11</td>
<td>1.03e−11</td>
</tr>
<tr>
<td>magnetism</td>
<td>7</td>
<td>27</td>
<td>3</td>
<td>1.27e−14</td>
<td>1.27e−14</td>
<td>1.27e−14</td>
<td>1.27e−14</td>
<td>1.27e−14</td>
<td>7.61e−15</td>
</tr>
<tr>
<td>caprasse</td>
<td>4</td>
<td>34</td>
<td>5</td>
<td>4.49e−15</td>
<td>4.49e−15</td>
<td>4.49e−15</td>
<td>5.63e−15</td>
<td>5.96e−15</td>
<td>3.04e−15</td>
</tr>
</tbody>
</table>

(ex-2-2-5) 2 9 3 2.23e−14 2.24e−14 2.23e−14 2.23e−14 2.23e−14 1.96e−14
(ex-2-2-10) 2 14 3 5.33e−14 5.33e−14 5.33e−14 5.33e−14 5.33e−14 4.85e−14
(ex-2-2-20) 2 24 3 1.49e−13 1.49e−13 1.49e−13 TIMEOUT 1.49e−13 1.40e−13
(ex-2-5-2) 2 9 6 1.67e−13 1.67e−13 1.67e−13 1.67e−13 1.67e−13 1.41e−13
(ex-2-10-2) 2 14 11 1.05e−11 1.05e−11 1.34e−11 1.05e−11 1.05e−11 8.76e−12
(ex-5-2-2) 5 12 3 8.55e−14 8.55e−14 8.55e−14 8.55e−14 8.55e−14 8.72e−14
(ex-10-2-2) 10 22 3 5.16e−13 TIMEOUT 5.16e−13 5.16e−13 4.82e−13

(resp. FPKriSten). Preliminary experiments indicate modest accuracy improvement at the expense of performance.

FFBern(a) is the fastest for almost all benchmarks (except program ex-10-2-2 where Rosa yields best performance). FFPBern(b) is much slower due to its Matlab implementation, and the use of certified rational arithmetic. We plan to implement a similar certification scheme within FFPBern(a).

On the first 12 benchmarks, FFPBern(a) is always the fastest while having a similar precision to Real2Float or Rosa.

The results obtained with the 8 generated benchmarks emphasize the limitations of each method. The Bernstein method performs very well when the number of input variables is low, even if the degree increases, as shown in the results for the 6 programs from ex-2-2-5 to ex-2-10-2. This is related to the polynomial dependency w.r.t. the degree when fixing the number of input variables. However, for the last 2 programs ex-5-2-2 and ex-10-2-2 where the dimension increases, the computation time increases exponentially. This confirms the theoretical result stated in Remark 1 as the number of Bernstein coefficients is exponential w.r.t. the dimension at fixed degree.

On the same programs, the method based on Krivine-Stengle representations performs better when the dimension increases, at fixed degree. This confirms the constraint dependency w.r.t. $\left\lfloor \frac{mk}{n+k} + 1 \right\rfloor^{(n+k)}$ stated in Remark 2.
Table 2. Comparison of execution times (in seconds) for absolute roundoff error bounds. For each model, the best results are emphasized using bold fonts.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>n m d</th>
<th>FPPern(a)</th>
<th>FPPern(b)</th>
<th>FPKrSten</th>
<th>Real2Float</th>
<th>Rosa</th>
<th>FPTaylor</th>
</tr>
</thead>
<tbody>
<tr>
<td>rigidBody1</td>
<td>3 10 3</td>
<td>5e-4</td>
<td>0.88</td>
<td>0.22(0.02)</td>
<td>0.58</td>
<td>0.13</td>
<td>1.84</td>
</tr>
<tr>
<td>rigidBody2</td>
<td>3 15 5</td>
<td>2e-3</td>
<td>1.87</td>
<td>2.78(0.47)</td>
<td>0.26</td>
<td>2.17</td>
<td>3.01</td>
</tr>
<tr>
<td>kepler0</td>
<td>6 21 3</td>
<td>4e-3</td>
<td>9.62</td>
<td>1.93(0.18)</td>
<td>0.22</td>
<td>3.78</td>
<td>4.93</td>
</tr>
<tr>
<td>kepler1</td>
<td>4 28 4</td>
<td>6e-3</td>
<td>6.91</td>
<td>3.93(0.53)</td>
<td>17.6</td>
<td>63.1</td>
<td>9.33</td>
</tr>
<tr>
<td>kepler2</td>
<td>6 42 4</td>
<td>5e-2</td>
<td>64.9</td>
<td>20.5(3.75)</td>
<td>16.5</td>
<td>106</td>
<td>19.1</td>
</tr>
<tr>
<td>sineTaylor</td>
<td>1 13 8</td>
<td>6e-4</td>
<td>0.50</td>
<td>0.92(0.27)</td>
<td>1.05</td>
<td>3.50</td>
<td>2.91</td>
</tr>
<tr>
<td>sineOrder3</td>
<td>1 6 4</td>
<td>2e-4</td>
<td>0.27</td>
<td>0.08(0.01)</td>
<td>0.40</td>
<td>0.48</td>
<td>1.90</td>
</tr>
<tr>
<td>sqroot</td>
<td>1 15 5</td>
<td>2e-4</td>
<td>0.34</td>
<td>0.24(0.02)</td>
<td>0.14</td>
<td>0.77</td>
<td>2.70</td>
</tr>
<tr>
<td>himmilbeau</td>
<td>2 11 5</td>
<td>1e-3</td>
<td>1.72</td>
<td>0.77(0.22)</td>
<td>0.20</td>
<td>2.51</td>
<td>3.28</td>
</tr>
<tr>
<td>schwefel</td>
<td>3 15 5</td>
<td>2e-3</td>
<td>3.04</td>
<td>2.90(0.56)</td>
<td>0.23</td>
<td>3.91</td>
<td>0.53</td>
</tr>
<tr>
<td>magnetism</td>
<td>7 27 3</td>
<td>9e-2</td>
<td>176</td>
<td>3.07(0.26)</td>
<td>0.29</td>
<td>1.95</td>
<td>5.91</td>
</tr>
<tr>
<td>caprasse</td>
<td>4 34 5</td>
<td>6e-3</td>
<td>6.03</td>
<td>18.8(4.89)</td>
<td>3.63</td>
<td>17.6</td>
<td>12.2</td>
</tr>
</tbody>
</table>

Results for the 4 programs from ex-2-2-5 to ex-2-2-20 also indicate that our methods are the least sensible to an increase of error variables. We note that FPKriSten is often the second fastest tool.

Let us now provide an overall evaluation of our tools. Our tools are comparable with Real2Float (resp. Rosa) in terms of accuracy and faster than them. In comparison with FPTaylor, our tools are in general less precise but still very competitive in accuracy, and they outperform FPTaylor in computation time. A salient advantage of our tools, in particular FPKriSten, over FPTaylor is a good trade-off between computation time and accuracy for large polynomials. As we can see from the experimental results, for ex-10-2-2, FPKriSten took only 6.11s while FPTaylor took 34.6s for comparable precisions. Note that the experiments were done with FPPern(b) and FPKriSten implemented in Matlab; their C++ implementations would allow a significant speed-up.

The good time performances of our tools come from the exploitation of sparsity. Indeed, a direct Bernstein expansion of the polynomial $l$ associated to kepler2 leads to compute $3^5 \times 2^{12}$ coefficients against $42 \times 3^6$ with FPPern. Similarly, dense Krivine-Stengle representations yield an LP with $\binom{100}{4} + 1 = 3\ 921\ 226$ variables while LP (12) involves $42\binom{18}{4} + 1 = 128\ 521$ variables.
5 Conclusion and Future Works

We propose two new methods to compute upper bounds of absolute roundoff errors occurring while executing polynomials programs with floating point precision. The first method uses symbolic Bernstein expansions of polynomials while the second one relies on a hierarchy of LP relaxations derived from sparse Krivine-Stengle representations. The overall computational cost is drastically reduced compared to the dense problem, thanks to a specific exploitation of the sparsity pattern between input and error variables, yielding promising experimental results.

Our approach is currently limited to programs implementing polynomials with box constrained variables. First, a direction of further research investigation is an extension to handle more complicated input sets. Extending to semialgebraic sets is theoretically possible with the hierarchy of LP relaxations based on sparse Krivine-Stengle representations but requires careful implementation in order not to compromise efficiency. For our method based on Bernstein expansions, it would be worth adapting the techniques described in [20] to obtain polygonal approximations of semialgebraic sets. Second, we intend to aim at formal verification of bounds by interfacing either FPPBern with the PVS libraries [22] related to Bernstein expansions, or FPKirSten with the Coq libraries available in Real2Float [19]. Finally, a delicate but important open problem is to apply such optimization techniques in order to handle roundoff errors of programs implementing finite or infinite loops as well as conditional statements.
References


A Polynomial Program Benchmarks

- rigibody1 : \((x_1, x_2, x_3) \mapsto -x_1 x_2 - 2 x_2 x_3 - x_1 - x_3\) defined on \([-15, 15]^3\).
- rigibody2 : \((x_1, x_2, x_3) \mapsto 2 x_1 x_2 x_3 + 6 x_3^2 - x_2^3 x_1 x_3 - x_2\) defined on \([-15, 15]^3\).
- kepler0 : \((x_1, x_2, x_3, x_4, x_5, x_6) \mapsto x_2 x_3 x_5 - x_2 x_3 x_6 - x_5 x_6 + x_1 (-x_1 + x_2 + x_3 - x_4 + x_5 + x_6)\) defined on \([4.6.36]^6\).
- kepler1 : \((x_1, x_2, x_3, x_4) \mapsto x_1 x_4 (-x_1 + x_2 + x_3 - x_4) + x_2 (x_1 - x_2 + x_3 + x_4) + x_3 (x_1 + x_2 - x_3 + x_4) - x_2 x_3 x_4 - x_1 x_3 - x_1 x_2 - x_4\) defined on \([4.6.36]^4\).
- kepler2 : \((x_1, x_2, x_3, x_4, x_5, x_6) \mapsto x_1 x_4 (-x_1 + x_2 + x_3 - x_4 + x_5 + x_6) + x_2 x_5 (x_1 - x_2 + x_3 + x_4 - x_5 + x_6) + x_3 x_6 (x_1 + x_2 - x_3 + x_4 + x_5 - x_6) - x_2 x_3 x_4 - x_1 x_3 x_5 - x_1 x_2 x_6 - x_4 x_5 x_6\) defined on \([4.6.36]^6\).
- sineTaylor : \(x \mapsto x - \frac{x^3}{6.0} + \frac{x^5}{120.0} - \frac{x^7}{5040.0}\) defined on \([-1.57079632679, 1.57079632679]\).
- sineOrder3 : \(x \mapsto 0.954929658551372 x - 0.12900613773279798 x^3\) defined on \([-2, 2]\).
- sroott : \(x \mapsto 1.0 + 0.5 x - 0.125 x^2 + 0.0625 x^3 - 0.0390625 x^4\) defined on \([0, 1]\).
- himmilbeau : \((x_1, x_2) \mapsto (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2\) defined on \([-5.5]^2\).
- schwefel : \((x_1, x_2, x_3) \mapsto (x_1 - x_2)^2 + (x_2 - 1)^2 + (x_1 - x_2^2)^2 + (x_3 - 1)^2\) defined on \([-10, 10]^3\).
- magnetism : \((x_1, x_2, x_3, x_4, x_5, x_6, x_7) \mapsto x_1^2 + 2 x_2^2 + 2 x_3^2 + 2 x_4^2 + 2 x_5^2 + 2 x_6^2 + 2 x_7^2 - x_1\) defined on \([-1, 1]^7\).
- caprass : \((x_1, x_2, x_3, x_4) \mapsto x_1 x_3 + 4 x_2 x_3^2 x_4 + 4 x_1 x_3 x_4^3 + 2 x_2 x_4^3 + 4 x_1 x_3 + 4 x_3^2 - 10 x_2 x_4 - 10 x_2^2 + 2\) defined on \([-0.5, 0.5]^4\).