A Certified Lower Bounds of Roundoff Errors using Semidefinite Programming

Victor Magron, Mountassir Farid

Certified Lower Bounds of Roundoff Errors using Semidefinite Programming

VICTOR MAGRON, CNRS Verimag
MOUNTASSIR FARID, Ensimag

A longstanding problem related to floating-point implementation of numerical programs is to provide efficient yet precise analysis of output errors.

We present a framework to compute lower bounds of absolute roundoff errors for numerical programs implementing polynomial functions with box constrained input variables. Our study relies on semidefinite programming (SDP) relaxations and is complementary of over-approximation frameworks, consisting of obtaining upper bounds for the absolute roundoff error.

Our method is based on a new hierarchy of convergent robust SDP approximations for certain classes of polynomial optimization problems. Each problem in this hierarchy can be exactly solved via SDP. By using this hierarchy, one can provide a monotone non-decreasing sequence of lower bounds converging to the absolute roundoff error of a program implementing a polynomial function.

We investigate the efficiency and precision of our method on non-trivial polynomial programs coming from space control, optimization and computational biology.

CCS Concepts: •**Design and analysis of algorithms** → **Approximation algorithms analysis;** *Numeric approximation algorithms;* **Mathematical optimization;** *Continuous optimization;* Semidefinite programming; Convex optimization; •**Logic** → **Automated reasoning;**

Additional Key Words and Phrases: roundoff error, polynomial optimization, semidefinite programming, floating-point arithmetic, Generalized Eigenvalue, robust optimization

ACM Reference Format:
Victor Magron and Mountassir Farid, 2016. Certified Lower Roundoff Error Bounds using Semidefinite Programming. *ACM Trans. Math. Softw.* V, N, Article A (January YYYY), 15 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Over the last four decades, numerical programs have extensively been written and executed with finite precision implementations [Dekker 1971], often relying on single or double floating-point numbers to perform fast computation. A ubiquitous related issue, especially in the context of critical system modeling, is to precisely analyze the gap between the real and floating-point output of such programs. The existence of a possibly high roundoff error gap is a consequence of multiple rounding occurrences, happening most likely while performing operations with finite precision systems, such as IEEE 754 standard arithmetic [IEEE 2008].

The present study focuses on computing a *certified* lower bound of the absolute roundoff error while executing a program implementing a multivariate polynomial function, when each input variable takes a value within a given closed interval. Exact resolution of this

This work has been partially supported by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01) funded by the French program "Investissement d'avenir" and by the European Research Council (ERC) "STATOR" Grant Agreement nr. 306595.
Author's addresses: V. Magron, CNRS Verimag, 700 av Centrale 38401 Saint-Martin d'Hères FRANCE; M. Farid, 681 Rue de la Passerelle 38400 Saint-Martin-d'Hères, France.

DOI: http://dx.doi.org/10.1145/0000000.0000000

problem is nontrivial as it requires to compute the maximum of a polynomial, which is know to be NP-hard [Laurent 2009] in general.

Several existing methods allow to obtain *lower bounds* of roundoff errors. The easiest way to obtain such a bound for the maximum of a given function is to evaluate this function at several points within the function input domain before taking the minimum over all evaluations. Testing approaches aim at finding the inputs causing the worst error. Such techniques often rely on meta-heuristic search as in CORAL [Borges et al. 2012] or guided random testing as in `s3fp` [Chiang et al. 2014].

Lower bound analyses are complementary with tools computing validated *upper bounds*. These tools are mainly based on interval arithmetic (e.g. GAPPA [Daumas and Melquiond 2010], FLUCTUAT [Delmas et al. 2009], `Rosa` [Darulova and Kuncak 2014]) or methods coming from global optimization such as Taylor approximation in `FPTaylor` by [Solovyev et al. 2015], Bernstein expansion in `FPBern` by [Rocca et al. 2016]. The recent framework by [Magron et al. 2016], related to the `Real2Float` software package, employs semidefinite programming (SDP) to obtain a hierarchy of upper bounds converging to the absolute roundoff error. This hierarchy is derived from the general moment-sum-of-squares hierarchy (also called Lasserre's hierarchy) initially provided by [Lasserre 2001] in the context of polynomial optimization. While this first SDP hierarchy allows to approximate from above the maximum of a polynomial, [Lasserre 2011] provides a complementary SDP hierarchy, yielding this time a sequence of converging lower bounds.

Contributions. In a similar way, we provide an SDP hierarchy inspired from [Lasserre 2011] to obtain a sequence of converging lower bounds for the absolute roundoff error. This hierarchy and the one developed in [Magron et al. 2016] complement each other as the combination of both now allows to enclose the roundoff error in smaller and smaller intervals.
We release a software package called `FPSDP`[1] implementing this SDP hierarchy.

The rest of the article is structured as follows: in Section 2 we provide preliminary background about floating-point arithmetic and SDP, allowing to state the considered problem of roundoff error. This problem is then addressed in Section 3 with our SDP hierarchy of converging lower bounds. Section 4 is devoted to numerical experiments in order to compare the performance of our `FPSDP` software with existing tools.

2. FLOATING-POINT ARITHMETIC AND SEMIDEFINITE PROGRAMMING

2.1. Floating-Point Arithmetic and Problem Statement

Let us denote by ε the machine precision, $\mathbb{R}$ the field of real numbers and $\mathbb{F}$ the set of binary floating-point numbers. Both overflow and denormal range values are neglected. Under this assumption, any real number $x \in \mathbb{R}$ is approximated with its closest floating-point representation $\hat{x} = x(1+e)$, with $|e| \leq \varepsilon$ and $\hat{\cdot}$ being the rounding operator (selected among either rounding toward zero, rounding toward $\pm\infty$ or rounding to nearest). We refer to [Higham 2002] for related background.

The number $\varepsilon := 2^{-\text{prec}}$ bounds from above the relative floating-point error, with prec being called the *precision*. For single (resp. double) precision floating-point, the value of the machine precision is $\varepsilon = 2^{-24}$ (resp. $\varepsilon = 2^{-53}$).

To comply with IEEE 754 standard arithmetic [IEEE 2008], for each real-valued operation $\texttt{bop}_{\mathbb{R}} \in \{+, -, \times, /\}$, the result of the corresponding floating-point operation $\texttt{bop}_{\mathbb{F}} \in \{\oplus, \ominus, \otimes, \oslash\}$ satisfies:

$$\texttt{bop}_{\mathbb{F}}(\hat{x}, \hat{y}) = \texttt{bop}_{\mathbb{R}}(\hat{x}, \hat{y})(1+e) \ , \quad |\, e \,| \leq \varepsilon = 2^{-\text{prec}} \ . \tag{1}$$

[1] https://github.com/magronv/FPSDP

Semantics. Our program semantics is based on the encoding of polynomial expressions in the `Real2Float` software [Magron et al. 2016]. The input variables of the program are constrained within interval floating-point bounds.

We denote by `C` the type for numerical constants, being chosen between double precision floating-point and arbitrary-size rational numbers. This type `C` is used for the interval bounds and for the polynomial coefficients.

As in [Magron et al. 2016, Section 2.1], the type `pexprC` of polynomial expressions is the following inductive type:

```
type pexprC = Pc of C | Px of positive | − pexprC
| pexprC − pexprC | pexprC + pexprC | pexprC × pexprC
```

The constructor `Px` allows to represent any input variable x_i with the positive integer i.

Lower bounds of roundoff errors. Let us consider a program implementing a polynomial function $f(\mathbf{x})$ of type `pexprC` (with the above semantics), which depends on input variables $\mathbf{x} := (x_1, \ldots, x_n)$ constrained in a box, i.e. a product of closed intervals $\mathbf{X} := [\underline{x_1}, \overline{x_1}] \times \ldots [\underline{x_n}, \overline{x_n}]$. After rounding each coefficient and elementary operation involved in f, we obtain a polynomial rounded expression denoted by $\hat{f}(\mathbf{x}, \mathbf{e})$, which depends on the input variables $\mathbf{x}$ as well as additional roundoff error variables $\mathbf{e} := (e_1, \ldots, e_m)$. Following (1), each variable e_i belongs to the interval $[-\varepsilon, \varepsilon]$, thus $\mathbf{e}$ belongs to $\mathbf{E} := [\varepsilon, \varepsilon]^m$.

Here, we are interested in bounding from below the absolute roundoff error $|r(\mathbf{x}, \mathbf{e})| := |\hat{f}(\mathbf{x}, \mathbf{e}) - f(\mathbf{x})|$ over all possible input variables $\mathbf{x} \in \mathbf{X}$ and roundoff error variables $\mathbf{e} \in \mathbf{E}$. Let us define $\mathbf{K} := \mathbf{X} \times \mathbf{E}$ and let $r^\star$ stands for the maximum of $|r(\mathbf{x}, \mathbf{e})|$ over $\mathbf{K}$, that is $r^\star := \max_{(\mathbf{x},\mathbf{e})\in\mathbf{K}} |r(\mathbf{x}, \mathbf{e})|$.

Following the same idea used in [Solovyev et al. 2015; Magron et al. 2016], we first decompose the error term r as the sum of a term $l(\mathbf{x}, \mathbf{e})$, which is linear w.r.t. $\mathbf{e}$, and a nonlinear term $h(\mathbf{x}, \mathbf{e}) := r(\mathbf{x}, \mathbf{e}) - l(\mathbf{x}, \mathbf{e})$. Then a valid lower bound of $r^\star$ can be derived by using the reverse triangular inequality:

$$r^\star \geq \max_{(\mathbf{x},\mathbf{e})\in\mathbf{K}} |l(\mathbf{x}, \mathbf{e})| - \max_{(\mathbf{x},\mathbf{e})\in\mathbf{K}} |h(\mathbf{x}, \mathbf{e})| =: l^\star - h^\star \ . \tag{2}$$

We emphasize the fact that $h^\star$ is *a priori* negligible compared to $l^\star$ since h contains products of error terms with degree at least 2 (such as $e_i e_j$), thus can be bounded by $O(\epsilon^2)$. This bound is likely much smaller than the roundoff error induced by the linear term l. To compute a bound of $h^\star$, it is enough in practice to compute second-order derivatives of r w.r.t. $\mathbf{e}$ then use Taylor-Lagrange inequality to get an interval enclosure of h as in [Solovyev et al. 2015]. Doing so, one obtains an upper bound of $h^\star$.

Then, subtracting this upper bound to any lower bound of $l^\star$ yields a valid lower bound of $r^\star$. Hence, from now on, we focus on approximating the bound $l^\star$ of the linear term. The framework [Magron et al. 2016] allows to obtain a hierarchy of converging upper bounds of $l^\star$ using SDP relaxations. By contrast with [Magron et al. 2016], our goal is to compute a hierarchy of converging lower bounds for $l^\star$. For the sake of clarity, we define $\underline{l} := \min_{(\mathbf{x},\mathbf{e})\in\mathbf{K}} l(\mathbf{x}, \mathbf{e})$ and $\overline{l} := \max_{(\mathbf{x},\mathbf{e})\in\mathbf{K}} l(\mathbf{x}, \mathbf{e})$. Computing $l^\star$ can then be cast as follows:

$$l^\star := \max_{(\mathbf{x},\mathbf{e})\in\mathbf{K}} |l(\mathbf{x}, \mathbf{e})| = \max\{|\underline{l}|, |\overline{l}|\} \, . \tag{3}$$

Note that the computation of $\underline{l}$ can be formulated as a maximization problem since $\underline{l} := \min_{(\mathbf{x},\mathbf{e})\in\mathbf{K}} l(\mathbf{x}, \mathbf{e}) = -\max_{(\mathbf{x},\mathbf{e})\in\mathbf{K}} -l(\mathbf{x}, \mathbf{e})$. Thus, any method providing lower bounds for $\overline{l}$ can also provide upper bounds for $\underline{l}$, eventually yielding lower bounds for $l^\star$.

We now present our main `fpsdp` algorithm, given in Figure 1. This procedure is similar to the algorithm implemented in the upper bound tool `Real2Float` [Magron et al. 2016], except that we obtain lower bounds for absolute roundoff errors. Given a program implementing a

Input: input variables $\mathbf{x}$, input box $\mathbf{X}$, polynomial f, rounded polynomial $\hat{f}$, error variables $\mathbf{e}$, error box $\mathbf{E}$, relaxation procedure `sdp_bound`, relaxation order k

Output: lower bound of the absolute roundoff error $\mid \hat{f} - f \mid$ over $\mathbf{K} := \mathbf{X} \times \mathbf{E}$

1: Define the absolute error $r(\mathbf{x}, \mathbf{e}) := \hat{f}(\mathbf{x}, \mathbf{e}) - f(\mathbf{x})$
2: Compute $l(\mathbf{x}, \mathbf{e}) := \sum_{j=1}^{m} \frac{\partial r(\mathbf{x},\mathbf{e})}{\partial e_j}(\mathbf{x}, 0)\, e_j$ and $h := r - l$
3: Compute an upper bound for $h^\star$: $\overline{h} :=$ `ia_bound`$(h, \mathbf{K})$
4: Compute a lower bound of $\overline{l} : \overline{l}_k :=$ `sdp_bound`$(l, \mathbf{K}, k)$
5: Compute an upper bound of $\underline{l}$: $\underline{l}_k := -$`sdp_bound`$(-l, \mathbf{K}, k)$
6: Compute a lower bound of $l^\star : l_k := \max\{\mid \underline{l}_k \mid, \mid \overline{l}_k \mid\}$
7: **return** $l_k - \overline{h}$

Fig. 1. `fpsdp`: our algorithm to compute lower bounds of absolute roundoff errors for polynomial programs.

polynomial f with input variables $\mathbf{x}$ being constrained in the box $\mathbf{X}$, the `fpsdp` algorithm takes as input $\mathbf{x}$, $\mathbf{X}$, f, the rounded expression $\hat{f}$ of f, the error variables $\mathbf{e}$ as well as the set $\mathbf{E}$ of bound constraints over $\mathbf{e}$. The roundoff error $r := \hat{f} - f$ (Line 1) is decomposed as the sum of a polynomial l which is linear w.r.t. the error variables $\mathbf{e}$ and a remainder h. As in [Magron et al. 2016; Solovyev et al. 2015], we obtain l by computing the partial derivatives of r w.r.t. $\mathbf{e}$ (Line 2). The computation of the upper bound of $h^\star$ (Line 3) is performed as explained earlier on, with the so-called procedure `ia_bound` relying on basic interval arithmetic. Our algorithm also takes as input a `sdp_bound` procedure, which computes lower bounds of the maximum of polynomials. In our case, we use `sdp_bound` in Line 4 (resp. Line 5) to compute a lower (resp. upper) bound of $\overline{l}$ (resp. $\underline{l}$). In the sequel, we describe three possible instances of `sdp_bound`, all relying on a hierarchy of semidefinite programming (SDP) relaxations, respectively in Section 2.2.1, Section 2.2.2 and Section 3. Each step of these SDP hierarchies is indexed by an integer k, called *relaxation order* and given as input to `fpsdp`.

2.2. Existing Hierarchies of Lower Bounds for Polynomial Maximization

Here, we recall mandatory background explaining how to obtain hierarchies of lower bounds for a given polynomial maximization problem using SDP relaxations [Lasserre 2011]. Given $p \in \mathbb{R}[\mathbf{y}]$ a multivariate polynomial in N variables $y_1, \ldots, y_N$ and a box $\mathbf{K} := [\underline{y_1}, \overline{y_1}] \times [\underline{y_N}, \overline{y_N}]$, one considers the following polynomial maximization problem:

$$p^* := \max_{\mathbf{y} \in \mathbf{K}} p(\mathbf{y})\,. \tag{4}$$

The set of box constraints $\mathbf{K} \subseteq \mathbb{R}^N$ is encoded by

$$\mathbf{K} := \{\mathbf{y} \in \mathbb{R}^N : g_1(\mathbf{y}) \geq 0, \ldots, g_N(\mathbf{y}) \geq 0\}\ ,$$

for polynomials $g_1 := (y_1 - \underline{y_1})(\overline{y_1} - y_1), \ldots, g_N := (y_N - \underline{y_N})(\overline{y_N} - y_N)$.

For a given vector of N nonnegative integers $\alpha \in \mathbb{N}^N$, we use the notation $\mathbf{y}^\alpha := y_1^{\alpha_1} \cdots y_N^{\alpha_N}$ and $|\alpha| := \sum_{i=0}^{N} \alpha_i$. Any polynomial $p \in \mathbb{R}[\mathbf{y}]$ of degree k can then be written as $p(\mathbf{y}) = \sum_{|\alpha| \leq k} p_\alpha \mathbf{y}^\alpha$. We write $\mathbb{N}^N_k := \{\alpha \in \mathbb{N}^N : |\alpha| \leq k\}$. The cardinal of this set is equal to $\binom{N+k}{k} = \frac{(N+k)!\,k!}{N!}$.

We recall that a finite Borel measure μ on $\mathbb{R}^N$ is a nonnegative set function such that $\mu(\emptyset) = 0$, $\mu(\mathbb{R}^N)$ is finite and μ is countably sub-additive. The support of μ is the smallest closed set $\mathbf{K} \subseteq \mathbb{R}^N$ such that $\mu(\mathbb{R}^N \backslash \mathbf{K}) = 0$ (see [Royden 1988] for more details).

Let μ be a given finite Borel measure supported on $\mathbf{K}$ and $\mathbf{z}$ be the sequence of moments of μ, given by $z_\alpha := \int_{\mathbf{K}} \mathbf{y}^\alpha d\mu(\mathbf{y})$ for all $\alpha \in \mathbb{N}^N$. In some cases, one can explicitly compute

z_α for each $\alpha \in \mathbb{N}^N$. This includes the case when μ is the uniform measure with density 1, i.e. $d\mu(\mathbf{y}) = d\mathbf{y}$, as $\mathbf{K}$ is a product of closed intervals. For instance with $N = 2$, $\mathbf{K} = [0,1]^2$ and $\alpha = (1,0)$, one has $z_{1,0} = \int_{\mathbf{K}} y_1 \, d\mathbf{y} = \frac{1}{2}$. With $\alpha = (2,1)$, one has $z_{2,1} = \int_{\mathbf{K}} y_1^2 \, y_2 \, d\mathbf{y} = \frac{1}{3} \times \frac{1}{2} = \frac{1}{6}$.
Given a real sequence $\mathbf{z} = (z_\alpha)$, we define the multivariate linear functional $L_{\mathbf{z}} : \mathbb{R}[\mathbf{y}] \to \mathbb{R}$ by $L_{\mathbf{z}}(p) := \sum_\alpha p_\alpha z_\alpha$, for all $p \in \mathbb{R}[\mathbf{y}]$. For instance if $p(\mathbf{y}) := y_1^2 y_2 + 3y_1 - \frac{2}{3}$, $\mathbf{K} = [0,1]^2$ then $L_{\mathbf{z}}(p) = z_{2,1} + 3z_{1,0} - \frac{2}{3} z_{0,0} = \frac{1}{6} + \frac{3}{2} - \frac{2}{3} = 1$.

Moment matrix. The *moment* matrix $\mathbf{M}_k(\mathbf{z})$ is the real symmetric matrix with rows and columns indexed by $\mathbb{N}^N_k$ associated with a sequence $\mathbf{z} = (z_\alpha)$, whose entries are defined by:

$$\mathbf{M}_k(\mathbf{z})(\beta, \gamma) := L_{\mathbf{z}}(\mathbf{y}^{\beta+\gamma}), \quad \forall \beta, \gamma \in \mathbb{N}^N_k .$$

Localizing matrix. The *localizing* matrix associated with a sequence $\mathbf{z} = (z_\alpha)$ and a polynomial $p \in \mathbb{R}[\mathbf{y}]$ (with $p(\mathbf{y}) = \sum_\alpha p_\alpha \mathbf{y}^\alpha$) is the real symmetric matrix $\mathbf{M}_k(p\,\mathbf{z})$ with rows and columns indexed by $\mathbb{N}^N_k$, and whose entries are defined by:

$$\mathbf{M}_k(p\,\mathbf{z})(\beta, \gamma) := L_{\mathbf{z}}(p(\mathbf{y})), \quad \forall \beta, \gamma \in \mathbb{N}^N_k .$$

The size of $\mathbf{M}_k(p\,\mathbf{z})$ is equal to the cardinal of $\mathbb{N}^N_k$, i.e. $\binom{N+k}{k}$. Note that when $p = 1$, one retrieves the moment matrix as special case of localizing matrix.

Example 2.1. With $p(\mathbf{y}) := y_1^2 y_2 + 3y_1 - \frac{2}{3}$, $\mathbf{K} = [0,1]^2$ and $k = 1$, one has $\mathbf{M}_k(\mathbf{z}) = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{3} \end{pmatrix}$ and $\mathbf{M}_k(\mathbf{z}) = \begin{pmatrix} 1 & \frac{19}{24} & \frac{19}{36} \\ \frac{19}{24} & \frac{113}{180} & \frac{5}{12} \\ \frac{19}{36} & \frac{5}{12} & \frac{13}{36} \end{pmatrix}$. For instance, the bottom-right corner of the localizing matrix $\mathbf{M}_1(\mathbf{z})$ is obtained by computing $L_{\mathbf{z}}(p(\mathbf{y})\, y_2^2) = z_{2,3} + 3z_{1,2} - \frac{2}{3} z_{0,2} = \frac{1}{12} + \frac{1}{2} - \frac{2}{3} \times \frac{1}{3} = \frac{13}{36}$.

Next, we briefly recall two existing methods to compute lower bounds of $p^\star$ as defined in (4).

2.2.1. Hierarchies of generalized eigenvalue problems. Let us note $\mathbb{R}^{n \times n}$ the vector space of $n \times n$ real matrices. For a symmetric matrix $\mathbf{M} \in \mathcal{M}_n(\mathbb{R})$, the notation $\mathbf{M} \succeq 0$ stands for $\mathbf{M}$ is semidefinite positive (SDP), i.e. has only nonnegative eigenvalues. The notation $\mathbf{A} \succeq \mathbf{B}$ stands for $\mathbf{A} - \mathbf{B} \succeq 0$. A semidefinite optimization problem is an optimization problem where the cost is a linear function and the constraints state that some given matrices are semidefinite positive (see [Vandenberghe and Boyd 1994] for more details about SDP).

The following sequence of SDP programs can be derived from [Lasserre 2011], for each $k \in \mathbb{N}$:

$$\begin{aligned} \lambda_k(p) := \min_{\lambda} \quad & \lambda \\ \text{s.t.} \quad & \lambda\, \mathbf{M}_k(\mathbf{z}) \succeq \mathbf{M}_k(p\,\mathbf{z}), \\ & \lambda \in \mathbb{R}. \end{aligned} \tag{5}$$

The only variable of Problem (5) is λ together with a single SDP constraint of size $\binom{N+k}{N}$. This constraint can be rewritten as $\mathbf{M}_k((\lambda - p)\mathbf{z}) \succeq 0$ by linearity of the localizing matrices. Solving Problem (5) allows to obtain a non-decreasing sequence of lower bounds which converges to the global minimum $p^\star$ of the polynomial p. Problem (5) is a *generalized eigenvalue* problem. As mentioned in [de Klerk et al. 2015, Section 2.3], the computation of the number $\lambda_k(p)$ requires at most $O\big(\binom{N+k}{k}^3\big)$ floating-point operations (flops).

THEOREM 2.2. *([Lasserre 2011, Theorem 4.1]) For each $k \in \mathbb{N}$, Problem* (5) *admits an optimal solution $\lambda_k(p)$. Furthermore, the sequence $(\lambda_k(p))$ is monotone non-decreasing and $\lambda_k(p) \uparrow p^\star$ as $k \to \infty$.*

The convergence rate have been studied later on in [de Klerk et al. 2016], which states that $p^\star - \lambda_k(p) = O(\frac{1}{\sqrt{k}})$.

Example 2.3. With $p(\mathbf{y}) := y_1^2 y_2 + 3y_1 - \frac{2}{3}$, $\mathbf{K} = [0,1]^2$, we obtain the following sequence of lower bounds: $\lambda_1(p) = 0.82 \leq \lambda_2(p) = 1.43 \leq \lambda_3(p) = 1.83 \leq \cdots \leq \lambda_{20}(p) = 2.72 \leq p^\star = \frac{10}{3}$. The computation takes $16.2s$ on an Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz. Here, we notice that the convergence to the maximal value $p^\star$ is slow in practice, confirming what the theory suggests.

2.2.2. Hierarchies of bounds using elementary computations. By contrast with the above method, further work by [de Klerk et al. 2015] provides a second method only requiring elementary computations. This method also yields a monotone non-decreasing sequence of lower bounds converging to the global maximum of a polynomial p while considering for each $k \in \mathbb{N}$:

$$p_k^H := \min_{(\eta,\beta)\in\mathbb{N}^{2N}_{2k}} \sum_{|\alpha|\leq d} p_\alpha \frac{\gamma_{\eta+\alpha,\beta}}{\gamma_{\eta,\beta}}\,, \tag{6}$$

where, for each $(\eta,\beta) \in \mathbb{N}^{2N}_{2k}$ the scalar $\gamma_{\eta,\beta}$ is the corresponding moment of the measure whose density is the multivariate beta distribution:

$$\gamma_{\eta,\beta} := \int_{\mathbf{K}} \mathbf{y}^\eta\,(\mathbf{1}-\mathbf{y})^\beta d\mathbf{y} = \int_{\mathbf{K}} y_1^{\eta_1}\cdots y_N^{\eta_N}\,(1-y_1)^{\beta_1}\cdots(1-y_N)^{\beta_N} d\mathbf{y}\,. \tag{7}$$

As mentioned in [de Klerk et al. 2015, Section 2.3], the computation of the number p_k^H requires at most $O\big(\binom{2N+2k-1}{2k}\big)$ floating-point operations (flops).

THEOREM 2.4. *([de Klerk et al. 2015, Lemma 2.4, Theorem 3.1]) The sequence (p_k^H) is monotone non-decreasing and $p_k^H \uparrow p^\star$ as $k \to \infty$.*

As for the sequence $(\lambda_k(p))$, the convergence rate is also in $O(\frac{1}{\sqrt{k}})$ (see [de Klerk et al. 2015, Theorem 4.9]).

Example 2.5. With $p(\mathbf{y}) := y_1^2 y_2 + 3y_1 - \frac{2}{3}$, $\mathbf{K} = [0,1]^2$, we obtain the following sequence of 20 lower bounds: $p_1^H = 0.52 \leq p_2^H = 0.95 \leq p_3^H = 1.25 \leq \cdots \leq p_{20}^H = 2.42 \leq p^\star = \frac{10}{3}$. The computation takes $17.1s$ on an Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz. For small order values, this method happens to be more efficient than the one previously used in Example 2.3 but yields coarser bounds. At high order, both methods happen to yield similar accuracy and performance with a slow rate of convergence.

3. A NEW SDP HIERARCHY FOR LOWER BOUNDS OF ROUNDOFF ERRORS

This section is dedicated to our main theoretical contribution, that is a new SDP hierarchy of converging lower bounds for the absolute roundoff error of polynomial programs.
The two existing SDP hierarchies presented in Section 2.2 can be directly applied to solve Problem (3), that is the computation of lower bounds for $l^\star := \max_{(\mathbf{x},\mathbf{e})\in\mathbf{K}} |l(\mathbf{x},\mathbf{e})|$. In our case, $N = n + m$ is the sum of the number of input and error variables, $p = l$ and $\mathbf{y} = (\mathbf{x},\mathbf{e}) \in \mathbf{K} = \mathbf{X} \times \mathbf{E}$. At order k, a first relaxation procedure, denoted by `geneig`, returns the number $\lambda_k(l)$ by solving Problem (5). A second relaxation procedure, denoted by `mvbeta`, returns the number l_k^H by solving Problem (6). In other words, this already gives two implementations `geneig` and `mvbeta` for the relaxation procedure `sdp_bound` in the algorithm `fpsdp` presented in Figure 1.

However, these two procedures can be computationally demanding to get precise bounds for programs with large number of variables, i.e. for high values of k and $N = n + m$. Experimental comparisons performed in Section 4 will support this claim. The design of a third implementation is also motivated by the fact that both `geneig` and `mvbeta` do not

take directly into account the special structure of the polynomial l, that is the linearity w.r.t. $\mathbf{e}$.

We first note that $l(\mathbf{x},\mathbf{e}) = \sum_{j=1}^m e_j s_j(\mathbf{x})$, for polynomials $s_1,\ldots,s_m \in \mathbb{R}[\mathbf{x}]$. The maximization problem $\bar{l} := \max_{(\mathbf{x},\mathbf{e})\in\mathbf{K}} l(\mathbf{x},\mathbf{e})$ can then be written as follows:

$$\begin{aligned} \bar{l} := \min_{\lambda} \quad & \lambda \\ \text{s.t.} \quad & \lambda \geq \sum_{j=1}^m e_j\, s_j(\mathbf{x})\,, \quad \forall \mathbf{x}\in\mathbf{X}\,, \forall \mathbf{e}\in\mathbf{E}\,, \\ & \lambda \in \mathbb{R}\,. \end{aligned}$$

From now on, we denote by $(\mathbf{z}^{\mathbf{X}})$ the moment sequence associated with the uniform measure on $\mathbf{X}$. We first recall the following useful property of the localizing matrices associated to $\mathbf{z}^{\mathbf{X}}$:

PROPERTY 3.1. *Let $f \in \mathbb{R}[\mathbf{x}]$ be a polynomial. Then f is nonnegative over $\mathbf{X}$ if and only if $\mathbf{M}_k(f\, z^{\mathbf{X}}) \succeq 0$, for all $k \in \mathbb{N}$.*

PROOF. This is a special case of [Lasserre 2011, Theorem 3.2 (a)] applied to the uniform measure supported on $\mathbf{X}$ with moment sequence $z^{\mathbf{X}}$. □

In particular for $f = 1$, Property 3.1 states that the moment matrix $\mathbf{M}_k(z^{\mathbf{X}})$ is semidefinite positive, for all $k \in \mathbb{N}$. Let us now consider the following hierarchy of *robust* SDP programs, indexed by $k \in \mathbb{N}$:

$$\begin{aligned} \lambda'_k(l) := \min_{\lambda} \quad & \lambda \\ \text{s.t.} \quad & \lambda\, \mathbf{M}_k(\mathbf{z}^{\mathbf{X}}) \succeq \sum_{j=1}^m e_j\, \mathbf{M}_k(s_j\, \mathbf{z}^{\mathbf{X}})\,, \quad \forall \mathbf{e}\in\mathbf{E}\,, \\ & \lambda \in \mathbb{R}\,. \end{aligned} \tag{8}$$

Problem (8) is called robust SDP as it consists of minimizing the (worst-case) cost while satisfying SDP constraints for each possible value of the parameters $\mathbf{e}$ within the given box $\mathbf{E}$.

LEMMA 3.2. *For each $k \in \mathbb{N}$, Problem (8) admits a finite optimal solution $\lambda'_k(l)$. Furthermore, the sequence $(\lambda'_k(l))$ is monotone non-decreasing and $\lambda'_k(l) \uparrow \bar{l}$ as $k \to \infty$.*

PROOF. The proof is inspired from the one of [Lasserre 2011, Theorem 4.1] since Problem (8) is a robust variant of Problem (5).

First, let us define for all $\mathbf{e} \in \mathbf{E}$ the polynomial $l_{\mathbf{e}}(\mathbf{x}) := l(\mathbf{x},\mathbf{e})$ in $\mathbb{R}[\mathbf{x}]$. The polynomial $\bar{l} - l$ is nonnegative over $\mathbf{X}\times\mathbf{E}$, thus for all $\mathbf{e}\in\mathbf{E}$, the polynomial $\bar{l} - l_{\mathbf{e}}$ is nonnegative over $\mathbf{X}$. By using Property 3.1, all localizing matrices of $\bar{l} - l_{\mathbf{e}}$ are semidefinite positive. This yields $\mathbf{M}_k((\bar{l} - l_{\mathbf{e}})\mathbf{z}^{\mathbf{X}}) \succeq 0$, for all $\mathbf{e}\in\mathbf{E}$. By linearity of the localizing matrix, we get $\bar{l}\,\mathbf{M}_k(\mathbf{z}^{\mathbf{X}}) \succeq \sum_{j=1}^m e_j\, \mathbf{M}_k(s_j\, \mathbf{z}^{\mathbf{X}})$, for all $\mathbf{e}\in\mathbf{E}$. For all $k\in\mathbb{N}$, this proves that $\bar{l}$ is feasible for Problem (8) and $\lambda'_k(l) \leq \bar{l}$. Next, let us fix $k\in\mathbb{N}$ and an arbitrary feasible solution λ for Problem (8). Since for all $\mathbf{e}\in\mathbf{E}$, one has $\mathbf{M}_k((\lambda - l_{\mathbf{e}})\mathbf{z}^{\mathbf{X}}) \succeq 0$, this is in particular the case for $\mathbf{e} = 0$, which yields $\lambda\mathbf{M}_k(\mathbf{z}^{\mathbf{X}}) \succeq 0$. Since the moment matrix $\mathbf{M}_k(\mathbf{z}^{\mathbf{X}})$ is semidefinite positive, one has $\lambda \geq 0$. Thus the feasible set of Problem (8) is nonempty and bounded, which proves the existence of a finite optimal solution $\lambda'_k(l)$.

Next, let us fix $k\in\mathbb{N}$. For all $\mathbf{e}\in\mathbf{E}$, $\mathbf{M}_k((\lambda - l_{\mathbf{e}})\mathbf{z}^{\mathbf{X}})$ is a sub-matrix of $\mathbf{M}_{k+1}((\lambda - l_{\mathbf{e}})\mathbf{z}^{\mathbf{X}})$, thus $\mathbf{M}_{k+1}((\lambda - l_{\mathbf{e}})\mathbf{z}^{\mathbf{X}}) \succeq 0$ implies that $\mathbf{M}_k((\lambda - l_{\mathbf{e}})\mathbf{z}^{\mathbf{X}}) \succeq 0$, yielding $\lambda'_k(l) \leq \lambda'_{k+1}(l)$. Hence, the sequence $(\lambda'_k(l))$ is monotone non-decreasing. Since for all $k\in\mathbb{N}$, $\lambda'_k(l) \leq \bar{l}$,

one has $(\lambda'_k(l))$ converges to $\lambda'(l) \leq \bar{l}$ as $k \to \infty$. For all $\mathbf{e} \in \mathbf{E}$, for all $k \in \mathbb{N}$, one has $\mathbf{M}_k((\lambda'(l) - l_{\mathbf{e}})\mathbf{z}^{\mathbf{X}}) \succeq \mathbf{M}_k((\lambda'_k(l) - l_{\mathbf{e}})\mathbf{z}^{\mathbf{X}}) \succeq 0$. By using again Property 3.1, this shows that for all $\mathbf{e} \in \mathbf{E}$, the polynomial $\lambda'(l) - l_{\mathbf{e}}$ is nonnegative over $\mathbf{X}$, yielding $\lambda'(l) \geq \bar{l}$ and the desired result $\lambda'(l) = \bar{l}$. □

Next, we use the framework developed in [Ghaoui et al. 1998] to prove that for all $k \in \mathbb{N}$, Problem (8) is equivalent to the following SDP which involves the additional real variable τ:

$$\begin{aligned} \lambda''_k(l) := \min_{\lambda,\tau} \quad & \lambda \\ \text{s.t.} \quad & \begin{pmatrix} \lambda\,\mathbf{M}_k(\mathbf{z}^{\mathbf{X}}) - \tau\,\mathbf{L}_k\,\mathbf{L}_k^{\mathrm{T}} & \mathbf{R}_k^{\mathrm{T}} \\ \mathbf{R}_k & \tau\mathbf{I} \end{pmatrix} \succeq 0\,, \\ & \lambda, \tau \in \mathbb{R}\,. \end{aligned} \tag{9}$$

Both matrices $\mathbf{L}_k = [\mathbf{L}_k^1 \cdots \mathbf{L}_k^m]$ and $\mathbf{R}_k = [\mathbf{R}_k^1 \ldots \mathbf{R}_k^m]^{\mathrm{T}}$ are obtained by performing a full rank factorization of the localizing matrix $\mathbf{M}_k(s_j\,\mathbf{z}^{\mathbf{X}})$ for each $j = 1, \ldots, m$. This can be done e.g. with the $\mathrm{PLDL^TP^T}$ decomposition [Golub and Van Loan 1996, Section 4.2.9] and is equivalent to find two matrices $\mathbf{L}_k^j$ and $\mathbf{R}_k^j$ such that $\mathbf{M}_k(s_j\,\mathbf{z}^{\mathbf{X}}) = 2\,\mathbf{L}_k^j\,\mathbf{R}_k^j$. For the sake of clarity we use the notations $\mathbf{L}_k$ and $\mathbf{R}_k$ while omitting the dependency of both matrices w.r.t. $\mathbf{z}^{\mathbf{X}}$.
For each $j = 1, \ldots, m$, the matrix $\mathbf{L}_k^j$ (resp. $\mathbf{R}_k^j$) has the same number of lines (resp. columns) as $\mathbf{M}_k(s_j\,\mathbf{z}^{\mathbf{X}})$, i.e. $\binom{n+k}{k}$, and the same number of columns (resp. lines) as the rank r_j of $\mathbf{M}_k(s_j\,\mathbf{z}^{\mathbf{X}})$. The size of $\mathbf{I}$ is $m\binom{n+k}{k}$.

THEOREM 3.3. *For each $k \in \mathbb{N}$, Problem (9) admits a finite optimal solution $\lambda''_k(l) = \lambda'_k(l)$. Furthermore, the sequence $\lambda''_k(l)$ is monotone non-decreasing and $\lambda''_k(l) \uparrow \bar{l}$ as $k \to \infty$.*

PROOF. It is enough to prove the equivalence between Problem (9) and Problem (8) since then the result follows directly from Lemma 3.2. Let us note $\mathbf{0} := (0, \ldots, 0) \in \mathbb{R}^n$. Problem (8) can be cast as Problem (4) in [Ghaoui et al. 1998] with $x = (\lambda, \mathbf{0})$, $F(x) = \lambda\,\mathbf{M}_k(\mathbf{z}^{\mathbf{X}})$, $\Delta = \mathrm{diag}(0, \mathbf{e})$, $c = (1, \mathbf{0})$, $\mathcal{D} = \mathbb{R}^{(m+1)\times(m+1)}$, $\rho = 1$.
In addition, the robust SDP constraint of Problem(8) can be rewritten as (8) in [Ghaoui et al. 1998], i.e. $F + L\,\Delta\,(I - D\,\Delta)^{-1}\,R + R^{\mathrm{T}}\,\Delta\,(I - D\,\Delta)^{-\mathrm{T}}\,L^{\mathrm{T}} \succeq 0$, with $L = \mathbf{L}_k$, $R = \mathbf{R}_k$ and $D = 0$. Then, the desired equivalence result follows from [Ghaoui et al. 1998, Theorem 3.1]. □

This procedure provides a third choice, called `robsdp`, for the relaxation procedure `sdp_bound` in the algorithm `fpsdp` presented in Figure 1.

Computational considerations. As for the `geneig` procedure, one also obtains the convergence rate $\bar{l} - \lambda''_k(l) = O(\frac{1}{\sqrt{k}})$. However, the resolution cost of Problem (9) can be less expensive.
Indeed, from [Golub and Van Loan 1996, Section 4.2.9], the cost of each full rank factorization is cubic in each localizing matrix size, yielding a total factorization cost of $O\big(m\,\binom{n+k}{k}^3\big)$ flops.
From [Nesterov and Nemirovskii 1994, Section 11.3] the SDP solving cost is proportional to the cube of the matrix size, yielding $O\big(m^3\,\binom{n+k}{k}^3\big)$ flops for Problem (9).
Hence, the overall cost of the `robsdp` procedure is bounded by $O\big(m^3\,\binom{n+k}{k}^3\big)$ flops. This is in contrast with the cost of $O\big(\binom{n+m+k}{k}^3\big)$ flops for `geneig` as well as the cost of $O\big(\binom{2n+2m+2k-1}{2k}$

flops for `mvbeta`. In the sequel, we compare these expected costs for several values of n, m and k.

4. RESULTS AND DISCUSSION

Now, we present experimental results obtained by applying our algorithm `fpsdp` (see Figure 1) with the three relaxation procedures `geneig`, `mvbeta` and `robsdp` to various examples coming from physics, biology, space control and optimization. The procedures `geneig`, `mvbeta` and `robsdp` provide lower bounds of a polynomial maximum by solving respectively Problem (5), Problem (6) and Problem (9). The `fpsdp` algorithm is implemented as a software package written in MATLAB, called `FPSDP`. Setup and usage of `FPSDP` are described on the dedicated web-page[2] with specific instructions[3]. The three procedures are implemented using YALMIP [Löfberg 2004] which is a toolbox for advanced modeling and solution of convex/non-convex optimization problems in MATLAB. For solving SDP problems, we rely on MOSEK 7.0 [Andersen and Andersen 2000]. For more details about the installation and usage of YALMIP (resp. MOSEK), we refer to the dedicated web-page [4] (resp. [5]) and the setup instructions [6] (resp. [7]). Full rank factorization within the `robsdp` procedure is performed with the function `rref` from MATLAB.

4.1. Benchmark Presentation

All examples are displayed in 5 and all results have been obtained on an Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz. For the sake of further presentation, we associate an alphabet character (from `a` to `i`) to identify each of the 9 polynomial nonlinear programs which implement polynomial functions: `a-b` come from physics, `c-e` are derived from expressions involved in the proof of Kepler Conjecture [Hales 2006] and `f-h` implement polynomial approximations of the sine and square root functions. All programs are used for similar upper bound comparison in [Magron et al. 2016, Section 4.1]. Each program implements a polynomial f with n input variables $\mathbf{x} \in \mathbf{X}$ and yields after rounding m error variables $\mathbf{e} \in \mathbf{E} = [-\varepsilon, \varepsilon]^m$.

Example 4.1. The program `c` (see 5) implements the polynomial expression

$$f(\mathbf{x}) := x_2 \times x_5 + x_3 \times x_6 - x_2 \times x_3 - x_5 \times x_6 \\ + x_1 \times (-x_1 + x_2 + x_3 - x_4 + x_5 + x_6),$$

and the program input is the six-variable vector $\mathbf{x} := (x_1, x_2, x_3, x_4, x_5, x_6)$. The set $\mathbf{X}$ of possible input values is a product of closed intervals: $\mathbf{X} = [4.00, 6.36]^6$. The polynomial f is obtained by performing 15 basic operations (1 negation, 3 subtractions, 6 additions and 5 multiplications). When executing this program with a set $\hat{\mathbf{x}}$ of floating-point numbers defined by $\hat{\mathbf{x}} := (\hat{x}_1, \hat{x}_2, \hat{x}_3, \hat{x}_4, \hat{x}_5, \hat{x}_6) \in \mathbf{X}$, one obtains the floating-point result $\hat{f}$. The error variables are $e_1, \ldots, e_{21} \in [-\varepsilon, \varepsilon]$ and $\mathbf{E} := [-\varepsilon, \varepsilon]^{21}$.

For the sake of conciseness, we only considered to compare the performance of `FPSDP` on programs implemented in double ($\varepsilon = 2^{-53}$) precision floating point. To compute lower bounds of the roundoff error $|f(\mathbf{x}) - \hat{f}(\mathbf{x}, \mathbf{e})| = |l(\mathbf{x}, \mathbf{e}) + h(\mathbf{x}, \mathbf{e})|$, we use `Real2Float` to obtain l and to bound h. We refer to [Magron et al. 2016, Section 3.1] for more details.

Table I compares expected magnitudes of computational costs for `geneig`, `mvbeta` and `robsdp`, following from the study at the end of Section 3. For each program, we show the

[2] https://github.com/magronv/FPSDP
[3] see the `README.md` file in the top level directory
[4] http://users.isy.liu.se/johanl/yalmip/
[5] https://www.mosek.com/
[6] http://users.isy.liu.se/johanl/yalmip/pmwiki.php?n=Main.WhatIsYALMIP
[7] http://docs.mosek.com/7.0/toolbox/Installation.html

Table I. Expected magnitudes of computational costs (in flops) for the three procedures `geneig`, `mvbeta` and `robsdp`.

Benchmark	id	n	m	k	`geneig`	`mvbeta`	`robsdp`
`rigidBody1`	`a`	3	10	1	2.75e+03	3.50e+03	6.40e+04
				8	8.43e+15	1.04e+12	4.50e+09
`rigidBody2`	`b`	3	15	1	6.86e+03	9.99e+03	2.16e+05
				7	1.12e+17	1.02e+13	5.84e+09
`kepler0`	`c`	6	21	1	2.20e+04	3.12e+04	3.18e+06
				4	3.12e+13	6.19e+10	8.58e+10
`kepler1`	`d`	4	28	1	3.60e+04	5.83e+04	2.75e+06
				4	2.05e+14	2.98e+11	7.53e+09
`kepler2`	`e`	6	42	1	1.18e+05	1.96e+05	2.55e+07
				4	1.99e+16	9.99e+12	6.87e+11
`sineTaylor`	`f`	1	13	1	3.38e+03	5.28e+03	1.76e+04
				8	3.27e+16	3.45e+12	1.61e+06
`sineOrder3`	`g`	1	6	1	5.12e+02	6.30e+02	1.73e+03
				8	2.67e+11	4.08e+08	1.58e+05
`sqroot`	`h`	1	15	1	4.92e+03	7.92e+03	2.70e+04
				7	1.48e+16	2.51e+12	1.73e+06
`himmilbeau`	`i`	2	11	1	2.75e+03	3.87e+03	3.60e+04
				8	8.43e+15	1.14e+12	1.22e+08

cost for the initial relaxation order $k = 1$ as well as for the highest one used for error computation in Table II. The results indicate that we can expect the procedure `geneig` to be more efficient at low relaxation orders while being outperformed by `robsdp` at higher orders. Besides, the `mvbeta` procedure is likely to have performance lying in between the two others. We mention that the interested reader can find more detailed experimental comparisons between the two SDP relaxation procedures `geneig` and `mvbeta` in [de Klerk et al. 2015].

4.2. Numerical Evaluation

For each benchmark, Table II displays the quality of the roundoff error bounds with corresponding execution times. We compared the three SDP relaxation procedures and `s3fp` [Chiang et al. 2014], relying on several possible heuristic search algorithms. We emphasize that our `FPSDP` tool relies on the simple rounding model described in Section 2.1 while `s3fp` measures output errors after executing programs written in `C++` with certain input values. The rounding occurring while executing such programs is more likely to fit with an improved model, based for instance on a piecewise constant absolute error bound (see e.g. [Magron et al. 2016, Section 1.2] for more explanation about such models). We indicated the performance obtained with `s3fp` for the sake of completeness even if a head-to-head comparison is more difficult.

For the sake of homogeneous presentation, we also associated an order k to `s3fp`, corresponding to a timeout parameter of the tool. For each k, we ran `s3fp` with a timeout of $2 \times 10^{2+k}$ (the default parameter being 2×10^5) in sequential mode with the heuristic called Binary Guided Random Testing (BGRT). These settings were selected among other to obtain the best performance as well as the most accurate lower bounds. We also provide best known upper bounds from [Magron et al. 2016, Table II] for comparison purpose.

As shown in Table II, the `s3fp` tool provides the tightest bounds for programs `b-c` and `e`. For all other benchmarks, the `robsdp` procedure is the most accurate. For relaxations order greater than 2, `s3fp` is the most efficient for programs `c`, while `robsdp` is faster for all other programs. Except for program `c`, either `geneig` or `mvbeta` yields better performance at the first relaxation order. The symbol "$-$" in a column entry means that we aborted the execution of the corresponding procedure after running more than 1e6 seconds. Note that

Table II. Comparison results of lower bounds and execution times (in seconds) for absolute roundoff errors. The winner results among `s3fp`, `geneig`, `mvbeta` and `robsdp` are emphasized using **bold fonts**.

id	k	s3fp		geneig		mvbeta		robsdp		upper
		bound	time	bound	time	bound	time	bound	time	bound
a	1	1.69e–13	1.71	1.05e–15	0.63	1.85e–16	0.31	3.30e–14	0.75	3.87e–13
	2	2.04e–13	17.7	2.84e–14	1.69	4.07e–15	20.9	7.52e–14	0.79	
	3	2.37e–13	42.3	5.83e–14	29.7	8.88e–15	645.	1.10e–13	1.06	
	4	2.47**e**–13	>2e4	8.72e–14	>1e4	1.73e–14	>1e5	1.62e–13	2.09	
	8	–	–	–	–	–	–	**3.55e–13**	164.	
b	1	1.42e–11	1.71	8.40e–14	1.10	4.99e–14	1.83	3.56e–12	0.31	5.24e–11
	2	1.92e–11	19.8	1.31e–12	2.75	2.41e–13	226.	5.31e–12	0.42	
	3	2.52e–11	173.	2.89e–12	288.	5.08e–13	>1e5	8.04e–12	1.04	
	4	**2.88e–11**	>1e4	–	–	–	–	1.13e–11	4.11	
	7	–	–	–	–	–	–	2.60e–11	152.	
c	1	3.48e–14	1.89	9.45e–15	1.25	3.95e–15	4.34	9.68e–15	0.73	1.05e–13
	2	3.71e–14	18.9	1.64e–14	37.3	7.89e–15	>1e4	1.48e–14	6.69	
	3	4.21e–14	42.9	–	–	–	–	2.62e–14	172.	
	4	**4.38e–14**	>1e4	–	–	–	–	–	–	
d	1	8.88e–14	1.79	3.01e–14	1.63	1.41e–14	13.6	1.49e–13	1.67	4.47e–13
	2	1.07e–13	16.2	5.38e–14	163.	2.45e–14	>4e4	2.22e–13	3.73	
	3	1.36e–13	53.1	–	–	–	–	3.04e–13	33.3	
	4	1.44e–13	>1e4	–	–	–	–	**4.06e–13**	275.	
e	1	4.87e–13	1.93	9.72e–28	5.74	5.55e–14	86.1	2.88e–13	2.53	2.09e–12
	2	5.97e–13	17.2	–	–	–	–	4.48e–13	55.2	
	3	5.97e–13	56.5	–	–	–	–	–	–	
	4	**6.97e–13**	>1e4	–	–	–	–	–	–	
f	1	2.12e–16	1.82	8.34e–17	0.89	1.50e–17	0.59	1.98e–16	1.28	6.03e–16
	2	2.66e–16	16.2	1.52e–16	2.24	4.50e–17	45.9	2.31e–16	1.29	
	3	2.78e–16	171.	2.07e–16	65.5	7.95e–17	>1e4	2.72e–16	1.30	
	4	2.85e–16	>2e4	–	–	–	–	3.04e–16	1.30	
	8	–	–	–	–	–	–	**4.43e–16**	1.40	
g	1	2.61e–16	1.75	1.09e–16	0.33	4.93e–17	0.04	3.99e–16	1.22	9.97e–16
	2	3.06e–16	16.7	2.43e–16	0.97	1.18e–16	0.94	4.83e–16	1.22	
	3	3.82e–16	40.1	3.68e–16	1.71	1.78e–16	10.6	5.62e–16	1.23	
	4	3.84e–16	>1e4	4.72e–16	7.21	2.33e–16	79.9	6.37e–16	1.24	
	6	–	–	6.28e–16	646.	2.87e–16	>1e4	7.85e–15	1.25	
	8	–	–	–	–	–	–	**9.30e–16**	1.28	
h	1	4.07e–16	1.94	2.30e–16	1.52	1.29e–16	0.28	4.83e–16	1.34	7.13e–16
	2	4.07e–16	16.5	4.00e–16	2.87	2.55e–16	28.4	5.40e–16	1.35	
	3	4.36e–16	41.7	5.36e–16	161.	3.28e–16	>1e4	5.78e–16	1.39	
	4	4.57e–16	>1e4	–	–	–	–	6.13e–16	1.40	
	7	–	–	–	–	–	–	**7.00e–16**	1.50	
i	1	4.36e–13	2.12	3.07e–14	0.58	1.60e–14	0.61	1.56e–13	1.32	1.32e–12
	2	5.69e–13	16.5	6.71e–14	2.48	2.68e–14	42.5	2.13e–13	1.39	
	3	6.74e–13	186.	1.25e–13	47.1	3.72e–14	>1e4	2.84e–13	1.43	
	4	6.74e–13	>1e4	1.92e–13	>1e4	5.35e–14	>2e5	3.63e–13	1.62	
	8	–	–	–	–	–	–	**7.67e–13**	7.34	

such behavior systematically occurs when analyzing programs `f-i` with `s3fp`, `geneig` and `mvbeta` at maximal relaxation orders. This confirms the expectation results from Table I, as `robsdp` yields more tractable SDP relaxations. Note that for these benchmarks, we performed experiments for each intermediate order k between 4 and the maximal indicated one. For conciseness, we have not displayed all intermediate results in the table but use them later on in Figure 2. One way to increase the performance of the `FPSDP` tool would be

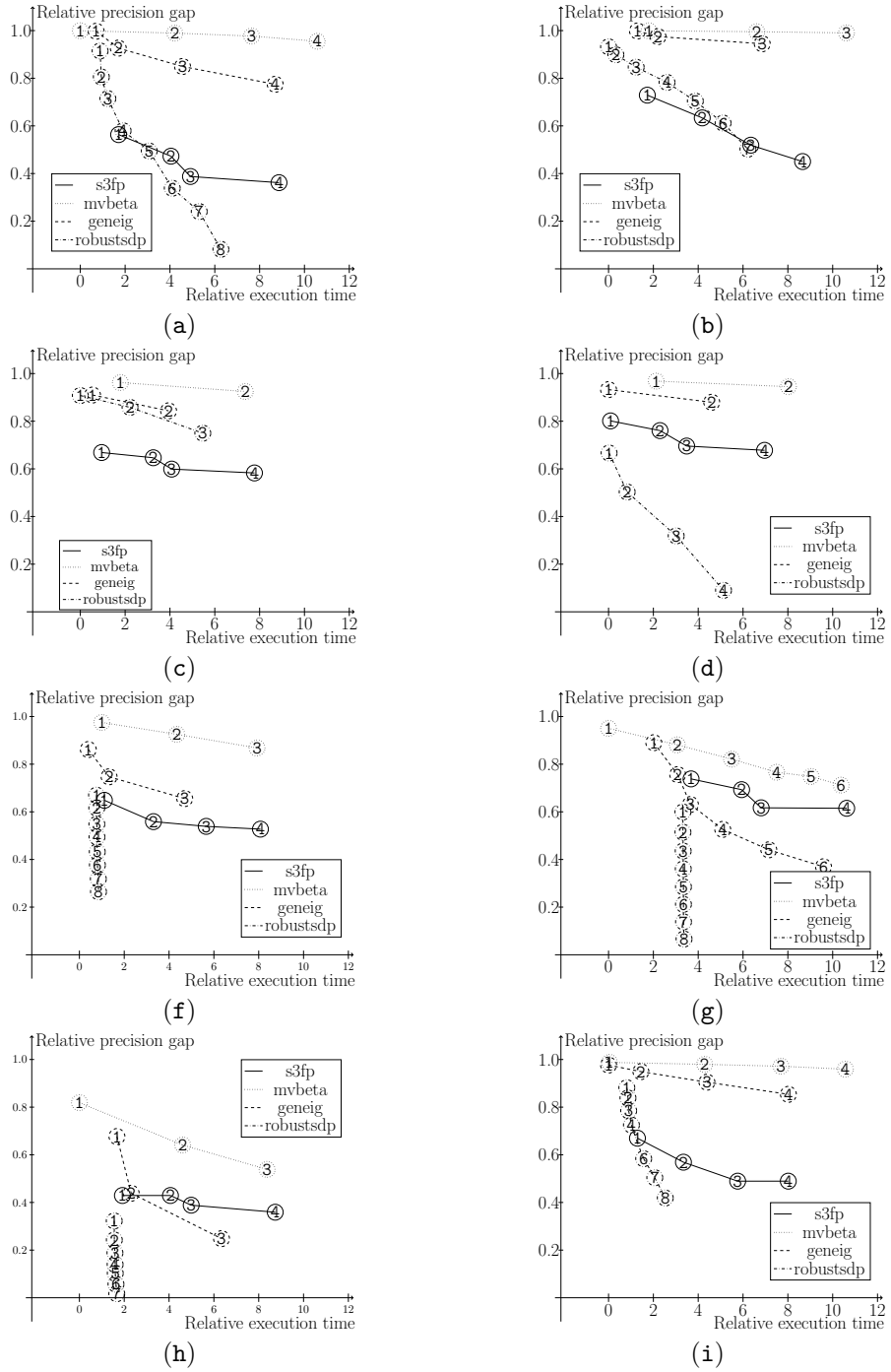

Fig. 2. Relative gap and execution time results for benchmarks.

to vectorize the current code which creates moment/localizing matrices, instead of writing loop-based code.

The purpose of Figure 2 is to emphasize the ability of FPSDP to make a compromise between accuracy and precision. All program results (except e due to the lack of experimental data) are reported in Figure 2. Each value of k corresponds to a circled integer point. For each experiment, we define the four execution times $t_{\texttt{s3fp}}$, $t_{\texttt{geneig}}$, $t_{\texttt{mvbeta}}$ and $t_{\texttt{robsdp}}$ and the minimum t among the four values. The x-axis coordinate of the circled point is $\ln(\frac{t_{\texttt{s3fp}}}{t})$ for the s3fp procedure (and similarly for the other procedures). The corresponding lower bounds are denoted by $\varepsilon_{\texttt{s3fp}}$, $\varepsilon_{\texttt{geneig}}$, $\varepsilon_{\texttt{mvbeta}}$ and $\varepsilon_{\texttt{robsdp}}$. With $\overline{\varepsilon}$ being the reference upper bound, the y-axis coordinate of the circled point is the relative error gap for s3fp, i.e. $r_{\texttt{s3fp}} := 1 - \frac{\varepsilon_{\texttt{s3fp}}}{\overline{\varepsilon}}$ and similarly for the other procedures.

For each k, the relative location of the corresponding circled integers indicate which procedure either performs better or provides more accurate bounds. When comparing the two procedures s3fp and robsdp, the former is more accurate for program a when the relative execution time is less than 2 then becomes less precise for higher relaxation orders. The curve of s3fp is always below the three other curves for programs b-c, which confirms previous observation that it is worth relying on this procedure for these two programs. For programs d and f-i, the curve of robsdp is always below. In particular for i, the two curves of s3fp and robsdp are superposed for low relative execution times (less than 1.5) then robsdp outperforms the other procedures. We also observe that mvbeta is more efficient at low relaxation orders for programs a and g-i as well as geneig for programs c-d, f and i. The SDP procedure geneig is always more accurate than mvbeta. When comparing with s3fp, this happens only when relative execution time is less than 4 for program g and less than 2.5 for program h.

5. CONCLUSION AND PERSPECTIVES

We present three procedures based on semidefinite programming (SDP) relaxations to compute lower bounds of roundoff errors for programs implementing polynomials with input variables being box constrained. While the two first procedures are direct applications of existing methods in the context of polynomial optimization, the third one relies on a new hierarchy of robust SDP relaxations, allowing to tackle specifically the roundoff error problem. Experimental results obtained with our FPSDP tool, implementing these three procedures, prove that SDP relaxations are able to provide accurate lower bounds in an efficient way.
A first direction of further research is the extension of the SDP relaxation framework to programs implementing either finite or infinite loops. This requires to derive a hierarchy of inner converging SDP approximations for reachable sets of discrete-time polynomial systems in either finite or infinite horizon. Another topic of interest is the formal verification of lower bounds with a proof assistant such as Coq [Coq 2016]. To achieve this goal, we could benefit from recent formal libraries [Dénès et al. 2012] in computational algebra.

POLYNOMIAL PROGRAM BENCHMARKS

- a rigibody1 : $(x_1, x_2, x_3) \mapsto -x_1x_2 - 2x_2x_3 - x_1 - x_3$ defined on $[-15, 15]^3$.
- b rigibody2 : $(x_1, x_2, x_3) \mapsto 2x_1x_2x_3 + 6x_3^2 - x_2^2x_1x_3 - x_2$ defined on $[-15, 15]^3$.
- c kepler0 : $(x_1, x_2, x_3, x_4, x_5, x_6) \mapsto x_2x_5 + x_3x_6 - x_2x_3 - x_5x_6 + x_1(-x_1 + x_2 + x_3 - x_4 + x_5 + x_6)$ defined on $[4, 6.36]^6$.
- d kepler1 : $(x_1, x_2, x_3, x_4) \mapsto x_1x_4(-x_1 + x_2 + x_3 - x_4) + x_2(x_1 - x_2 + x_3 + x_4) + x_3(x_1 + x_2 - x_3 + x_4) - x_2x_3x_4 - x_1x_3 - x_1x_2 - x_4$ defined on $[4, 6.36]^4$.
- e kepler2 : $(x_1, x_2, x_3, x_4, x_5, x_6) \mapsto x_1x_4(-x_1 + x_2 + x_3 - x_4 + x_5 + x_6) + x_2x_5(x_1 - x_2 + x_3 + x_4 - x_5 + x_6) + x_3x_6(x_1 + x_2 - x_3 + x_4 + x_5 - x_6) - x_2x_3x_4 - x_1x_3x_5 - x_1x_2x_6 - x_4x_5x_6$ defined on $[4, 6.36]^6$.
- f sineTaylor : $x \mapsto x - \frac{x^3}{6.0} + \frac{x^5}{120.0} - \frac{x^7}{5040.0}$ defined on $[-\frac{\tilde{\pi}}{2}, \frac{\tilde{\pi}}{2}]$, with $\frac{\tilde{\pi}}{2} := 1.57079632679$.
- g sineOrder3 : $x \mapsto 0.954929658551372x - 0.12900613773279798x^3$ defined on $[-2, 2]$.
- h sqroot : $x \mapsto 1.0 + 0.5x - 0.125x^2 + 0.0625x^3 - 0.0390625x^4$ defined on $[0, 1]$.

`i` himmilbeau : $(x_1, x_2) \mapsto (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$ defined on $[-5, 5]^2$.

REFERENCES

ErlingD. Andersen and KnudD. Andersen. 2000. The Mosek Interior Point Optimizer for Linear Programming: An Implementation of the Homogeneous Algorithm. In *High Performance Optimization*, Hans Frenk, Kees Roos, Tamás Terlaky, and Shuzhong Zhang (Eds.). Applied Optimization, Vol. 33. Springer US, 197–232.

Mateus Borges, Marcelo d'Amorim, Saswat Anand, David Bushnell, and Corina S. Pasareanu. 2012. Symbolic Execution with Interval Solving and Meta-heuristic Search. In *Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST '12)*. IEEE Computer Society, Washington, DC, USA, 111–120.

Wei-Fan Chiang, Ganesh Gopalakrishnan, Zvonimir Rakamaric, and Alexey Solovyev. 2014. Efficient Search for Inputs Causing High Floating-point Errors. In *Proceedings of the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '14)*. ACM, New York, NY, USA, 43–52.

Coq 1984–2016. The Coq Proof Assistant. (1984–2016). http://coq.inria.fr/.

Eva Darulova and Viktor Kuncak. 2014. Sound Compilation of Reals. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '14)*. New York, NY, USA, 235–248.

Marc Daumas and Guillaume Melquiond. 2010. Certification of Bounds on Expressions Involving Rounded Operators. *ACM Trans. Math. Softw.* 37, 1, Article 2 (Jan. 2010), 20 pages.

Etienne de Klerk, Jean B. Lasserre, Monique Laurent, and Zhao Sun. 2015. Bound-constrained polynomial optimization using only elementary calculations. (2015). To appear in *Mathematics of Operations Research*.

Etienne de Klerk, Monique Laurent, and Zhao Sun. 2016. Convergence analysis for Lasserre's measure-based hierarchy of upper bounds for polynomial optimization. *Mathematical Programming A* (2016), 1–30.

Theodorus J. Dekker. 1971. A Floating-point Technique for Extending the Available Precision. *Numer. Math.* 18, 3 (June 1971), 224–242.

David Delmas, Eric Goubault, Sylvie Putot, Jean Souyris, Karim Tekkal, and Franck Védrine. 2009. Towards an Industrial Use of FLUCTUAT on Safety-Critical Avionics Software. In *Formal Methods for Industrial Critical Systems*, María Alpuente, Byron Cook, and Christophe Joubert (Eds.). Lecture Notes in Computer Science, Vol. 5825. Springer Berlin Heidelberg, 53–69.

Maxime Dénès, Anders Mörtberg, and Vincent Siles. 2012. *A Refinement-Based Approach to Computational Algebra in Coq*. Springer Berlin Heidelberg, Berlin, Heidelberg, 83–98.

Laurent El Ghaoui, François Oustry, Hervé Lebert, and Jean B. Lasserre. 1998. Robust Solutions to uncertain semidefinite programs. *SIAM J. Opt* 9, 1 (1998), 33–52.

Gene H. Golub and Charles F. Van Loan. 1996. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA.

Thomas C. Hales. 2006. Introduction to the Flyspeck Project. In *Mathematics, Algorithms, Proofs (Dagstuhl Seminar Proceedings)*, Thierry Coquand, Henri Lombardi, and Marie-Françoise Roy (Eds.). Dagstuhl, Germany.

Nick J. Higham. 2002. *Accuracy and Stability of Numerical Algorithms: Second Edition*. SIAM.

IEEE. 2008. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008* (Aug 2008), 1–70.

Jean B. Lasserre. 2001. Global Optimization with Polynomials and the Problem of Moments. *SIAM Journal on Optimization* 11, 3 (2001), 796–817.

Jean B. Lasserre. 2011. A New Look at Nonnegativity on Closed Sets and Polynomial Optimization. *SIAM J. Opt* 21, 3 (2011), 864–885.

Monique Laurent. 2009. Sums of squares, moment matrices and optimization over polynomials. In *Emerging applications of algebraic geometry*. Springer, 157–270.

Johan Löfberg. 2004. YALMIP : A Toolbox for Modeling and Optimization in MATLAB. In *Proceedings of the CACSD Conference*. Taipei, Taiwan. http://users.isy.liu.se/johanl/yalmip

Victor Magron, George Constantinides, and Alastair Donaldson. 2016. Certified Roundoff Error Bounds Using Semidefinite Programming. (2016). Accepted for Publication in *ACM Transactions on Mathematical Software*.

Yuri Nesterov and Arkadii Nemirovskii. 1994. *Interior-Point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics.

Alexandre Rocca, Victor Magron, and Thao Dang. 2016. Certified Roundoff Error Bounds using Bernstein Expansions and Sparse Krivine-Stengle Representations. (2016). Submitted.

Halsey L. Royden. 1988. *Real Analysis*. Macmillan.

Alexey Solovyev, Charles Jacobsen, Zvonimir Rakamarić, and Ganesh Gopalakrishnan. 2015. Rigorous Estimation of Floating-Point Round-off Errors with Symbolic Taylor Expansions. In *Proceedings of the 20th International Symposium on Formal Methods (FM) (Lecture Notes in Computer Science)*, Nikolaj Bjørner and Frank de Boer (Eds.), Vol. 9109. Springer, 532–550.

Lieven Vandenberghe and Stephen Boyd. 1994. Semidefinite Programming. *SIAM Rev.* 38 (1994), 49–95.