A Certified Lower Bounds of Roundoff Errors using Semidefinite Programming
Victor Magron, Mountassir Farid

To cite this version:
Victor Magron, Mountassir Farid. A Certified Lower Bounds of Roundoff Errors using Semidefinite Programming. 2017. hal-01448160

HAL Id: hal-01448160
https://hal.science/hal-01448160
Preprint 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.
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:

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
problem is nontrivial as it requires to compute the maximum of a polynomial, which is
known 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, and 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 [https://github.com/magronvFPSDP]
implementing this SDP hierarchy.

The rest of the article is structured as follows: in Section 2 we provide preliminary back-
ground 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 \( \varepsilon \) 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
\( \text{bop}_\mathbb{R} \in \{+,-,\times,\div\} \), the result of the corresponding floating-point operation
\( \text{bop}_\mathbb{F} \in \{\oplus,\ominus,\otimes,\oslash\} \) satisfies:

\[
\text{bop}_\mathbb{F}(\hat{x}, \hat{y}) = \text{bop}_\mathbb{R}(\hat{x}, \hat{y})(1 + e) \ , \quad |e| \leq \varepsilon = 2^{-\text{prec}} .
\]
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 $\mathbb{C}$ the type for numerical constants, being chosen between double precision floating-point and arbitrary-size rational numbers. This type $\mathbb{C}$ is used for the interval bounds and for the polynomial coefficients.

As in [Magron et al. 2016] Section 2.1, the type $\text{pexpr}\mathbb{C}$ of polynomial expressions is the following inductive type:

\[
\text{type } \text{pexpr}\mathbb{C} = \text{Pc} \mid \text{Px} \mid - \text{pexpr}\mathbb{C} \mid \text{pexpr}\mathbb{C} - \text{pexpr}\mathbb{C} \mid \text{pexpr}\mathbb{C} + \text{pexpr}\mathbb{C} \mid \text{pexpr}\mathbb{C} \times \text{pexpr}\mathbb{C}
\]

The constructor $\text{Px}$ 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 $f(x)$ of type $\text{pexpr}\mathbb{C}$ (with the above semantics), which depends on input variables $x := (x_1, \ldots, x_n)$ constrained in a box, i.e. a product of closed intervals $X := [x_1, \bar{x}_1] \times \ldots \times [x_n, \bar{x}_n]$. After rounding each coefficient and elementary operation involved in $f$, we obtain a polynomial rounded expression denoted by $\hat{f}(x, e)$, which depends on the input variables $x$ as well as additional roundoff error variables $e := (e_1, \ldots, e_m)$.

Following [1], each variable $e_i$ belongs to the interval $[-\varepsilon, \varepsilon]$, thus $e$ belongs to $\mathbb{E} := \{\varepsilon, -\varepsilon\}^m$.

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

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 $l(x, e)$, which is linear w.r.t. $e$, and a nonlinear term $h(x, e) := r(x, e) - l(x, e)$. Then a valid lower bound of $r^*$ can be derived by using the reverse triangular inequality:

\[
r^* \geq \max_{(x, e) \in K} |l(x, e)| - \max_{(x, e) \in K} |h(x, e)| =: l^* - h^* . \tag{2}
\]

We emphasize the fact that $h^*$ is a priori negligible compared to $l^*$ since $h$ contains products of error terms with degree at least 2 (such as $e_ie_j$), thus can be bounded by $O(\varepsilon^2)$. This bound is likely much smaller than the roundoff error induced by the linear term $l$. To compute a bound of $h^*$, it is enough in practice to compute second-order derivatives of $r$ w.r.t. $e$ 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 $h^*$.

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

\[
l^* := \max_{(x,e)\in K} |l(x, e)| = \max\{|\bar{l}|, |\bar{l}|\} . \tag{3}
\]

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

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
Given as input to Line 4 ∈ \( p \) for a given polynomial maximization problem using SDP relaxations [Lasserre 2011].

2.2. Existing Hierarchies of Lower Bounds for Polynomial Maximization

Each step of these SDP hierarchies is indexed by an integer programming (SDP) relaxations, respectively in Section 2.2.1, Section 2.2.2 and Section 3.

We describe three possible instances of \( sdp\_bound \), 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 \( p \in \mathbb{R}[y] \) a multivariate polynomial in \( N \) variables \( y_1, \ldots, y_N \) and a box \( K := [y_1, y_1] \times [y_N, y_N] \), one considers the following polynomial maximization problem:

\[
p^* := \max_{y \in K} p(y). \tag{4}
\]

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

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

for polynomials \( g_1 := (y_1 - y_1)(y_1 - y_1), \ldots, g_N := (y_N - y_N)(y_N - y_N) \).

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

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

Let \( \mu \) be a given finite Borel measure supported on \( K \) and \( z \) be the sequence of moments of \( \mu \), given by \( z_{\alpha} := \int_K y^\alpha d\mu(y) \) for all \( \alpha \in \mathbb{N}^N \). In some cases, one can explicitly compute...
\[ z_\alpha \text{ for each } \alpha \in \mathbb{N}^N. \] This includes the case when \( \mu \) is the uniform measure with density 1, i.e. \( d\mu(y) = dy \), as \( K \) is a product of closed intervals. For instance with \( N = 2 \), \( K = [0, 1]^2 \) and \( \alpha = (1, 0) \), one has \( z_{1,0} = \int_K y_1 \, dy = \frac{1}{2} \). With \( \alpha = (2, 1) \), one has \( z_{2,1} = \int_K y_1^2 y_2 \, dy = \frac{1}{4} \times \frac{1}{4} = \frac{1}{16} \).

Given a real sequence \( z = (z_\alpha) \), we define the multivariate linear functional \( L_z : \mathbb{R}[y] \to \mathbb{R} \) by \( L_z(p) := \sum_\alpha p_\alpha z_\alpha \), for all \( p \in \mathbb{R}[y] \). For instance if \( p(y) := y_1^2 y_2 + 3y_1 - \frac{2}{3} \), \( K = [0, 1]^2 \) then \( L_z(p) = z_{2,1} + 3z_{1,0} - \frac{2}{3} z_{0,0} = \frac{1}{6} + \frac{2}{3} - \frac{2}{3} = 1 \).

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

\[ M_k(z)(\beta, \gamma) := L_z(y^{\beta+\gamma}), \quad \forall \beta, \gamma \in \mathbb{N}^k. \]

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

\[ M_k(p \, z)(\beta, \gamma) := L_z(p(y)), \quad \forall \beta, \gamma \in \mathbb{N}^k. \]

The size of \( M_k(p \, z) \) is equal to the cardinal of \( \mathbb{N}^k \), 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 \( p(y) := y_1^2 y_2 + 3y_1 - \frac{2}{3} \), \( K = [0, 1]^2 \) and \( k = 1 \), one has \( M_1(z) = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{pmatrix} \) and \( M_k(z) = \begin{pmatrix} \frac{13}{36} & \frac{10}{36} & \frac{10}{36} \\ \frac{10}{36} & \frac{3}{12} & \frac{3}{12} \\ \frac{10}{36} & \frac{3}{12} & \frac{3}{12} \end{pmatrix} \). For instance, the bottom-right corner of the localizing matrix \( M_1(z) \) is obtained by computing \( L_z(p(y) \, y_2^2) = z_{2,3} + 3z_{1,2} - \frac{2}{3} z_{0,2} = \frac{1}{12} + \frac{2}{3} - \frac{2}{3} = \frac{13}{36} \).

Next, we briefly recall two existing methods to compute lower bounds of \( p^* \) 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 \( M \in \mathcal{M}_n(\mathbb{R}) \), the notation \( M \succeq 0 \) stands for \( M \) is semidefinite positive (SDP), i.e. has only nonnegative eigenvalues. The notation \( A \succeq B \) stands for \( A - B \succeq 0 \). 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 \( k \in \mathbb{N} \):

\[ \lambda_k(p) := \min_{\lambda} \lambda \]

\[ \text{s.t. } \lambda \, M_k(z) \succeq M_k(p \, z), \quad \lambda \in \mathbb{R}. \tag{5} \]

The only variable of Problem (5) is \( \lambda \) together with a single SDP constraint of size \( \binom{N+k}{k} \). This constraint can be rewritten as \( M_k((\lambda - p)z) \succeq 0 \) 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 \( p^* \) 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 \( O\left(\binom{N+k}{k}^3\right) \) 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 \( \lambda_k(p) \uparrow p^* \) as \( k \to \infty \).
The convergence rate have been studied later on in [de Klerk et al. 2016], which states that $p^* - \lambda_k(p) = O\left(\frac{1}{\sqrt{k}}\right)$.

**Example 2.3.** With $p(y) := y_1^2y_2 + 3y_1 - \frac{2}{3}$, $K = [0,1]^2$, we obtain the following sequence of lower bounds: $\lambda_1(p) = 0.82 \leq \lambda_2(p) = 1.43 \leq \lambda_3(p) = 1.83 \leq \cdots \leq \lambda_{20}(p) = 2.72 \leq p^* = \frac{10}{3}$. 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 $p^*$ 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 $k \in \mathbb{N}$:

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

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

$$\gamma_{n, \beta} := \int_K y^n (1 - y)^\beta dy = \int_K y_1^n \cdots y_N^n (1 - y_1)^{\beta_1} \cdots (1 - y_N)^{\beta_N} dy. \tag{7}$$

As mentioned in [de Klerk et al. 2015, Section 2.3], the computation of the number $p_k^H$ requires at most $O\left(\frac{n^N}{2k} + 1\right)$ 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 $p_k^H \uparrow p^*$ as $k \to \infty$.

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

**Example 2.5.** With $p(y) := y_1^2y_2 + 3y_1 - \frac{2}{3}$, $K = [0,1]^2$, we obtain the following sequence of 20 lower bounds: $p_1^H = 0.52 \leq p_2^H = 0.95 \leq p_3^H = 1.25 \leq \cdots \leq p_{20}^H = 2.42 \leq p^* = \frac{10}{3}$. 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 4, that is the computation of lower bounds for $l^* := \max_{(x,e) \in K} ||(x,e)||$. In our case, $N = n + m$ is the sum of the number of input and error variables, $p = l$ and $y = (x,e) \in K = X \times E$. At order $k$, a first relaxation procedure, denoted by geneig, returns the number $\lambda_k(l)$ by solving Problem 6. A second relaxation procedure, denoted by mvbeta, returns the number $1/l_k^H$ 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. \( e \).

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

\[
\overline{\mathcal{P}} := \min_{\lambda} \quad \lambda \\
\text{s.t.} \quad \lambda \geq \sum_{j=1}^{m} e_j s_j(x), \quad \forall x \in X, \forall e \in E, \\
\lambda \in \mathbb{R}.
\]

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

**Property 3.1.** Let \( f \in \mathbb{R}[x] \) be a polynomial. Then \( f \) is nonnegative over \( X \) if and only if \( M_k(f z^X) \geq 0 \), 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 \( X \) with moment sequence \( z^X \).

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

\[
\lambda'_{k}(l) := \min_{\lambda} \quad \lambda \\
\text{s.t.} \quad \lambda M_k(z^X) \geq \sum_{j=1}^{m} e_j M_k(s_j z^X), \quad \forall e \in E, \\
\lambda \in \mathbb{R}.
\]

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 \( e \) within the given box \( E \).

**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 \( \lambda'_{k}(l) \uparrow \overline{l} \) 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 \( e \in E \) the polynomial \( l_e(x) := l(x,e) \) in \( \mathbb{R}[x] \). The polynomial \( \overline{l} - l \) is nonnegative over \( X \times E \), thus for all \( e \in E \), the polynomial \( \overline{l} - l_e \) is nonnegative over \( X \). By using Property 3.1, all localizing matrices of \( \overline{l} - l_e \) are semidefinite positive. This yields \( M_k((\overline{l} - l_e) z^X) \geq 0 \), for all \( e \in E \). By linearity of the localizing matrix, we get \( \overline{l} M_k(z^X) \geq \sum_{j=1}^{m} e_j M_k(s_j z^X) \), for all \( e \in E \). For all \( k \in \mathbb{N} \), this proves that \( \overline{l} \) is feasible for Problem (8) and \( \lambda'_{k}(l) \leq \overline{l} \). Next, let us fix \( k \in \mathbb{N} \) and an arbitrary feasible solution \( \lambda \) for Problem (8). Since for all \( e \in E \), one has \( M_k((\lambda - l_e) z^X) \geq 0 \), this is in particular the case for \( e = 0 \), which yields \( \lambda M_k(z^X) \geq 0 \). Since the moment matrix \( M_k(z^X) \) 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 \( e \in E \), \( M_k((\lambda - l_e) z^X) \) is a sub-matrix of \( M_{k+1}((\lambda - l_e) z^X) \), thus \( M_{k+1}((\lambda - l_e) z^X) \geq 0 \) implies that \( M_k((\lambda - l_e) z^X) \geq 0 \), yielding \( \lambda'_{k}(l) \leq \lambda'_{k+1}(l) \).

Hence, the sequence \( (\lambda'_{k}(l)) \) is monotone non-decreasing. Since for all \( k \in \mathbb{N} \), \( \lambda'_{k}(l) \leq \overline{l} \),
one has $(\lambda'(l))$ converges to $\lambda'(l) \leq \overline{l}$ as $k \to \infty$. For all $e \in E$, for all $k \in \mathbb{N}$, one has $M_k((\lambda'(l) - \lambda_e)z^X) \geq M_k((\lambda'(l) - \lambda_e)z^X) \geq \mathbf{0}$. By using again Property [3.1], this shows that for all $e \in E$, the polynomial $(\lambda'(l) - \lambda_e)$ is nonnegative over $X$, yielding $\lambda'(l) \geq \overline{l}$ and the desired result $\lambda'(l) = \overline{l}$. \hfill $\square$

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 $\tau$:

$$
\begin{align*}
\lambda_k''(l) := & \min \lambda, \\
\text{s.t.} \quad & (\lambda M_k(z^X) - \tau L_k L_k^T R_k^T \tau I_{1}) \succeq 0, \\
& \lambda, \tau \in \mathbb{R}.
\end{align*}
$$

(9)

Both matrices $L_k = [L_k^1 \cdots L_k^m]$ and $R_k = [R_k^1 \cdots R_k^m]^T$ are obtained by performing a full rank factorization of the localizing matrix $M_k(s_j, z^X)$ for each $j = 1, \ldots, m$. This can be done e.g. with the PLDLT decomposition [Golub and Van Loan 1996] Section 4.2.9 and is equivalent to find two matrices $L_j^T$ and $R_j^T$ such that $M_k(s_j, z^X) = 2 L_j^T R_j$. For the sake of clarity we use the notations $L_k$ and $R_k$ while omitting the dependency of both matrices w.r.t. $z^X$.

For each $j = 1, \ldots, m$, the matrix $L_k^j$ (resp. $R_k^j$) has the same number of lines (resp. columns) as $M_k(s_j, z^X)$, i.e. $\binom{n+k}{k}$, and the same number of columns (resp. lines) as the rank $r_j$ of $M_k(s_j, z^X)$. The size of $I$ 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 $\lambda_k''(l) \uparrow \overline{l}$ 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 $0 := (0, \ldots, 0) \in \mathbb{R}^n$. Problem (8) can be cast as Problem (4) in [Ghaoui et al. 1998] with $x = (\lambda_0, 0)$, $F(x) = \lambda M_k(z^X)$, $\Delta = \text{diag}(0, e)$, and $\rho = 1$. In addition, the robust SDP constraint of Problem (8) can be rewritten as (8) in [Ghaoui et al. 1998], i.e. $F + L \Delta (I - D \Delta)^{-1} R + R^T \Delta (I - D \Delta)^{-T} L^T \succeq 0$, with $L = L_k$, $R = R_k$ and $D = 0$. Then, the desired equivalence result follows from [Ghaoui et al. 1998, Theorem 3.1]. \hfill $\square$

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 $\overline{l} - \lambda_k''(l) = O\left(\frac{1}{\sqrt{k}}\right)$. 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 $O(m \binom{n+k}{k}^3)$ flops.

From [Nesterov and Nemirovskii 1994] Section 11.3 the SDP solving cost is proportional to the cube of the matrix size, yielding $O(m^3 \binom{n+k}{k}^3)$ flops for Problem (9).

Hence, the overall cost of the robsdp procedure is bounded by $O(m \binom{n+k}{k}^3)$ flops. This is in contrast with the cost of $O\left(\binom{n+m+k}{k}^3\right)$ flops for geneig as well as the cost of $O\left(2n^2+2n^2k-1\right)$

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 robdsdp to various examples coming from physics, biology, space control and optimization. The procedures geneig, mvbeta and robdsdp 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 with specific instructions. 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 (resp. []) and the setup instructions (resp. []). Full rank factorization within the robdsdp 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 \( x \in X \) and yields after rounding \( m \) error variables \( e \in E = [-\varepsilon, \varepsilon]^m \).

Example 4.1. The program \( c \) (see [5]) implements the polynomial expression

\[
f(x) := x_2 \times x_5 + x_3 \times x_6 - x_2 \times x_3 - x_5 \times x_6 + x_1 \times (-x_1 + x_2 + x_3 - x_4 + x_5 + x_6),
\]

and the program input is the six-variable vector \( x := (x_1, x_2, x_3, x_4, x_5, x_6) \). The set \( X \) of possible input values is a product of closed intervals: \( X = [4.00, 6.36]^6 \). 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 \( \hat{x} \) of floating-point numbers defined by \( \hat{x} := (\hat{x}_1, \hat{x}_2, \hat{x}_3, \hat{x}_4, \hat{x}_5, \hat{x}_6) \in X \), one obtains the floating-point result \( \hat{f} \). The error variables are \( e_1, \ldots, e_{21} \in [-\varepsilon, \varepsilon] \) and \( E := [-\varepsilon, \varepsilon]^{21} \).

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 \( |f(x) - \hat{f}(x, e)| = |f(x, e) + h(x, e)| \), we use Real2Float to obtain \( l \) and to bound \( h \). We refer to [Magron et al. 2016] Section 3.1] for more details.

Table 1 compares expected magnitudes of computational costs for geneig, mvbeta and robdsdp, following from the study at the end of Section 3. For each program, we show the
cost for the initial relaxation order \( k = 1 \) as well as for the highest one used for error computation in Table I. The results indicate that we can expect the procedure \texttt{geneig} to be more efficient at low relaxation orders while being outperformed by \texttt{robsdp} at higher orders. Besides, the \texttt{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 \texttt{geneig} and \texttt{mvbeta} in de Klerk et al. 2015.

### 4.2. Numerical Evaluation

For each benchmark, Table I displays the quality of the roundoff error bounds with corresponding execution times. We compared the three SDP relaxation procedures and \texttt{s3fp} [Chiang et al. 2014], relying on several possible heuristic search algorithms. We emphasize that our \texttt{FPSDP} tool relies on the simple rounding model described in Section 2.1, while \texttt{s3fp} measures output errors after executing programs written in \text{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 \texttt{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 \texttt{s3fp}, corresponding to a timeout parameter of the tool. For each \( k \), we ran \texttt{s3fp} with a timeout of \( 2 \times 10^{k+5} \) (the default parameter being \( 2 \times 10^5 \)) 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 I, the \texttt{s3fp} tool provides the tightest bounds for programs \texttt{b}–\texttt{c} and \texttt{e}. For all other benchmarks, the \texttt{robsdp} procedure is the most accurate. For relaxations order greater than 2, \texttt{s3fp} is the most efficient for programs \texttt{c}, while \texttt{robsdp} is faster for all other programs. Except for program \texttt{c}, either \texttt{geneig} or \texttt{mvbeta} yields better performance at the first relaxation order. The symbol “\( \ast \)” in a column entry means that we aborted the execution of the corresponding procedure after running more than 16 seconds. Note that (Table I. Expected magnitudes of computational costs (in flops) for the three procedures \texttt{geneig}, \texttt{mvbeta} and \texttt{robsdp}.)
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 robsdps 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 FPSDPS tool would be

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 \( t_{s3fp}, t_{geneig}, t_{mvbeta} \) and \( t_{robsdp} \) and the minimum \( t \) among the four values. The x-axis coordinate of the circled point is \( \ln \left( \frac{4 \epsilon}{7} \right) \) for the \( s3fp \) procedure (and similarly for the other procedures). The corresponding lower bounds are denoted by \( \varepsilon_{s3fp}, \varepsilon_{geneig}, \varepsilon_{mvbeta} \) and \( \varepsilon_{robsdp} \). With \( \pi \) being the reference upper bound, the y-axis coordinate of the circled point is the relative error gap for \( s3fp \), i.e., \( r_{s3fp} := 1 - \frac{\varepsilon_{s3fp}}{\pi} \) 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 \( b-c \). 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 [Dénes et al. 2012] in computational algebra.

POLYNOMIAL PROGRAM BENCHMARKS

- a rigibody1 : \( (x_1,x_2,x_3) \mapsto -x_1 x_2 - 2 x_2 x_3 - x_1 - x_3 \) defined on \([-15,15]^3\).
- b rigibody2 : \( (x_1,x_2,x_3) \mapsto 2 x_1 x_2 x_3 + 6 x_3^2 - x_3 x_1 x_3 - x_2 \) defined on \([-15,15]^3\).
- c kepler0 : \( (x_1,x_2,x_3,x_4,x_5) \mapsto x_2 x_5 + x_3 x_6 - x_2 x_4 - x_3 x_6 + x_1 (-x_1 + x_2 + x_3 - x_4 + x_5 + x_6) \) defined on \([4,6.36]^6\).
- d kepler1 : \( (x_1,x_2,x_3,x_4) \mapsto x_1 x_4 (-x_1 + x_2 + x_3 - x_4) + x_2 (x_1 - x_2 + x_3 + x_4) + x_3 (x_1 + x_2 - x_3 + x_4) - x_2 x_3 x_4 - x_3 x_2 - x_3 - x_2 - x_4 \) defined on \([4,6.36]^4\).
- e kepler2 : \( (x_1,x_2,x_3,x_4,x_5,x_6) \mapsto x_1 x_4 (-x_1 + x_2 + x_3 - x_4 + x_5 + x_6) + x_2 x_5 (x_1 - x_2 + x_3 + x_4 - x_5 + x_6) + x_3 x_6 (x_1 + x_2 - x_3 + x_4 + x_5 - x_6) - x_2 x_3 x_4 - x_3 x_2 x_5 + x_1 x_2 x_5 - x_4 x_5 x_6 \) defined on \([4,6.36]^6\).
- f sineTaylor : \( x \mapsto x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} \) defined on \([-\frac{5}{7}, \frac{5}{7}]\), with \( \hat{x} := 1.57079632679 \).
- g sineOrder3 : \( x \mapsto 0.954929658551372 x - 0.1290061377279789 x^3 \) defined on \([-2,2]\).
- h sqroot : \( x \mapsto 1.0 + 0.5 x - 0.125 x^2 + 0.0625 x^3 - 0.0390625 x^4 \) 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 \([-5, 5]^2\).

REFERENCES


