Efficient Sampling from Feasible Sets of SDPs and Volume Approximation

Apostolos Chalkis[a,b], Ioannis Z. Emiris[a,b], Vissarion Fisikopoulos[b], Panagiotis Repouskos[b], Elias Tsigaridas[c]

[a]*ATHENA Research Center, Greece*
[b]*Department of Informatics & Telecommunications*
National & Kapodistrian University of Athens, Greece
[c]*Inria Paris, IMJ-PRG,*
Sorbonne Université and Paris Université

Abstract

We present algorithmic, complexity, and implementation results on the problem of sampling points from a spectrahedron, that is the feasible region of a semidefinite program.

Our main tool is geometric random walks. We analyze the arithmetic and bit complexity of certain primitive geometric operations that are based on the algebraic properties of spectrahedra and the polynomial eigenvalue problem. This study leads to the implementation of a broad collection of random walks for sampling from spectrahedra that experimentally show faster mixing times than methods currently employed either in theoretical studies or in applications, including the popular family of Hit-and-Run walks. The different random walks offer a variety of advantages, thus allowing us to efficiently sample from general probability distributions, for example the family of log-concave distributions which arise in numerous applications. We focus on two major applications of independent interest: (i) approximate the volume of a spectrahedron, and (ii) compute the expectation of functions coming from robust optimal control.

We exploit efficient linear algebra algorithms and implementations to address the aforementioned computations in very high dimension. In particular, we provide a C++ open source implementation of our methods that scales efficiently, for the first time, up to dimension 200. We illustrate its efficiency on various data sets.

Keywords: spectahedra, semidefinite-programming, sampling, random walk, polynomial eigenvalue problem, volume approximation, optimization

1. Introduction

Spectrahedra are probably the most well studied shapes after polyhedra. We can represent polyhedra as the intersection of the positive orthant with an affine subspace. Spectrahedra generalize polyhedra, in the sense that they are the intersection of the cone of positive semidefinite matrices —*i.e.,* symmetric matrices with non-negative eigenvalues— with an affine space. In other words, a spectrahedron $S \subset \mathbb{R}^n$ is the feasible set of a linear matrix inequality. That is, let

$$\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{A}_0 + x_1\boldsymbol{A}_1 + \cdots + x_n\boldsymbol{A}_n, \tag{1}$$

where $\boldsymbol{A}_i$ are symmetric matrices in $\mathbb{R}^{m\times m}$, then $S = \{\boldsymbol{x} \in \mathbb{R}^n \,|\, \boldsymbol{F}(\boldsymbol{x}) \succeq 0\}$, where $\succeq$ denotes positive semidefiniteness. We assume throughout that S is bounded of dimension n. Spectrahedra

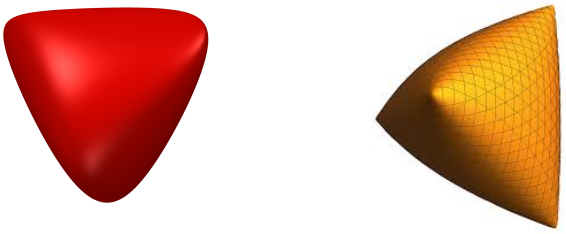

Figure 1: Left, a 3D elliptope (image from `https://en.wikipedia.org/wiki/Spectrahedron`); an elliptope –a special case of spectrahedron– is the set of all real, square symmetric matrices whose diagonal entries are all equal to one and whose eigenvalues are all non-negative. Right, a spectrahedron from [8]; it is called cubic spectrahedron.

are convex sets (Figures 2 and 4) and every polytope is a spectrahedron, but not the opposite. They are the feasible regions of semidefinite programs [47] in the way that polyhedra are feasible regions of linear programs.

Efficient methods for sampling points in spectrahedra are crucial for many applications, such as volume approximation [16], integration [39], semidefinite optimization [39, 27], and applications in robust control analysis [11, 10, 53]. To sample in the interior or on the boundary of S, we employ geometric random walks [56]. A geometric random walk on S starts at some interior point and at each step moves to a "neighboring" point that we choose according to some distribution, depending only on the current point; thus it is a special category of Markov chains. For example, in the so-called ball walk, we move to a point p that we choose uniformly at random in a ball of fixed radius δ, if $p \in S$. The complexity of a random walk depends on its mixing time —the number of steps required to bound the distance between the current and the stationary distribution— and the complexity of the basic geometric operations performed at each step of the walk; we call the latter per-step complexity.

The majority of geometric random walks are defined for general convex bodies and are based on an oracle; usually the membership oracle. There are also a few walks, e.g., Vaidya walk [14] and the sub-linear ball walk [41], specialized for polytopes. Most results on their analysis focus on convergence and mixing time, while they define abstractly the operations they perform at each step and they enclose them in the corresponding oracle. That is why the complexity bounds involve the number of oracle calls.

To specialize a random walk for a family or representation of convex bodies one has to come up with efficient algorithms for the basic geometric operations to realize the (various) oracles. These operations should exploit the underlying geometric and algebraic properties and are of independent interest. They depend on efficient (numerical) linear algebra computations. More importantly, they dominate the per-step complexity and are hence crucial both for the overall complexity to sample a point from the target distribution, as well as for an efficient implementation.

The study of basic geometric operations to sample from non-linear convex objects finds its roots in non-linear computational geometry. During the last two decades, there are combined efforts [7, 22] to develop efficient algorithms for the basic operations (predicates) that are behind classical geometric algorithms, like convex hull, arrangements, Voronoi diagrams, to go beyond points and lines and handle curved objects. For this, one exploits efficiently structure and

symmetry in linear algebra computations, and develops novel algebraic tools.

To our knowledge, only the *Random Directions Hit and Run* (W-HnR) random walk [50] has been studied for spectrahedra [10]. To exploit the various other walks, like *Ball walk* [56], *Billiard walk* (W-Billard) [24], and *Hamiltonian Monte Carlo with boundary reflections* (W-HMC-r) [1], one needs to provide geometric operations. For example, we need to compute the reflection of a curve at the boundary, and the intersection point of a curve with the boundary (of a spectrahedron).

We should mention that there is a gap [17, 5] between the theoretical worst case bounds for the mixing times and the practical performance of the random walk algorithms. Furthermore, there are random walks without known theoretical mixing times, such as Coordinate Directions Hit and Run (W-CHnR), W-Billard and W-HMC-r. To study them experimentally, it is imperative to provide an efficient realization of the corresponding oracles.

Previous Work. Sampling convex sets via random walks has attracted a lot of interest in the last decades. Most of the works assume that the convex sets are polytopes; [41] provides an overview of the state-of-the-art. Random walks on spectrahedra are studied in [46, 18], where they exploit the W-HnR and the computation of the intersection of the walk with the boundary reduces to a generalized eigenvalue problem.

The W-Billard [24] is a general way of sampling in convex or non-convex shapes from the uniform distribution. A mathematical billiard consists of a domain $\mathcal{D}$ and a point-mass, moving freely in $\mathcal{D}$ [52]. When this point-mass hits the boundary, it performs a specular reflection, albeit without losing velocity. An application of billiards is the study of optical properties of conics [52, Section 4].

If the trajectory is not a line, but rather a parametric curve, then the intersection operation reduces to solving a polynomial eigenvalue problem (PEP); W-HMC-r requires this operation. PEP is a well-studied problem in computational mathematics, e.g., [54], and it appears in many applications. There are important results both for the perturbation analysis of PEP [54, 4, 19], as well as for the condition-based analysis of algorithms for the real and complex versions of PEP, if we assume random inputs [2, 3].

For the closely related problem of volume computation, there is also an extensive bibliography [9]. The bulk of the theoretical studies are either for general convex bodies [20, 16] or polytopes [35]. Practical algorithms and implementations exist only for polytopes [21, 17]. Nevertheless, there are notable exceptions that consider algorithms for computing the volume of compact (basic) semi-algebraic sets. For example, in [31] they exploit the periods of rational integrals. In the same setting, [30, 26] introduce numerical approximation schemes for volume computations, which rely on the moment-based algorithms and semi-definite programming.

Finally, sampling from a multivariate distribution is a central problem in numerous applications. For example, it is useful in robust control analysis [10, 11, 53] to overcome the worst case hardness as well as in integration [39] and convex optimization [18, 27].

Our contribution. We present a framework of basic geometric operations for computations with spectrahedra. In particular, we analyze the arithmetic and bit complexity of the three fundamental operations, membership, intersection, reflection, by reducing them to linear algebra computations. Based on this framework, we support a rich class of geometric random walks, which in turn we employ in order to build efficient methods for sampling points from spectahedra, under various probability distributions. We apply these tools to approximate the volume of spectrahedra, as well as to integrate over spectrahedral domains. This extends the limits of the state-of-the-art

methods that sample from spectrahedra, which actually involve only the W-HnR [18, 10]. We offer an efficient C++ implementation of our algorithms as a development branch of the package `volesti`[1], an open source library for high dimensional sampling and volume computation. While the implementation is in C++, there is also an `R` interface, for easier access to its functionality. Our implementation is based on state-of-the-art algorithms in numerical linear algebra to address computation in high dimension. Our software makes use of powerful C++ libraries such as `Eigen` [25] and `Spectra` [51].

We demonstrate the efficiency of our approach and implementation on problems from robust control and optimization. First, as a special case of integration, we approximate the volume of spectrahedra up to dimension 100; this is, to the best of our knowledge, the first time such computations for non-linear objects are performed in high dimensions. Then, we approximate the expected value of a function $f : \mathbb{R}^n \to [0, 1]$, whose argument is a random variable having uniform distribution over a spectrahedron of dimension 200.

Finally, we sample from the Boltzmann distribution using W-HMC-R; this exploits a random walk in a spectrahedron that employs a polynomial trajectory of degree two. Sampling using W-HMC-R from truncated distributions is a classical problem in computational statistics [15, 1]; alas, existing approaches handle either special distributions or special cases of constraints [42, 32]. We equip W-HMC-R with geometric operations to handle log-concave densities truncated by linear matrix inequalities (LMI) constraints. A combination of Boltzmann distribution with a simulated annealing technique [27] could lead to a practical solver for semidefinite programs (SDP).

A very short version of this paper has appeared as a poster in [13].

Paper organization. First, we introduce our notation. In Section 2 we introduce the basic geometric operations used to efficiently implement membership and boundary oracles. In Section 3, we develop the different types of random walks. Finally, Section 4 presents our implementation, our applications, and various experiments.

Notation. We denote by O, respectively O_B, the arithmetic, respectively bit, complexity and we use $\widetilde{O}$, respectively $\widetilde{O}_B$, to ignore (poly-)logarithmic factors. The bitsize of a univariate polynomial $A \in \mathbb{Z}[x]$ is the maximum bitsize of its coefficients. We use bold letters for matrices, $\boldsymbol{A}$, and vectors, $\boldsymbol{v}$; we denote by $A_{i,j}$, respectively v_i, their elements; $\boldsymbol{A}^\top$ is the transpose and $\boldsymbol{A}^*$ the adjoint of $\boldsymbol{A}$. If $\boldsymbol{x} = (x_1, \ldots, x_n)$, then $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{A}_0 + \sum_{i=1}^n x_i \boldsymbol{A}_i$, see (1). For two points $\boldsymbol{x}$ and $\boldsymbol{y}$, we denote the line through them by $\ell(\boldsymbol{x}, \boldsymbol{y})$ and their segment as $[\boldsymbol{x}, \boldsymbol{y}]$. For a spectrahedron S, let its interior be S° and its boundary ∂S. We represent a probability distribution π with a probability density function $\pi(x)$. When π is truncated to S the support of $\pi(x)$ is S. If π is log-concave, then $\pi(\boldsymbol{x}) \propto e^{-\alpha f(\boldsymbol{x})}$, where $f : \mathbb{R}^d \to \mathbb{R}$ a convex function. Finally, let $\mathcal{B}_n$ be the n-dimensional unit ball and denote by $\partial\mathcal{B}_n$ its boundary.

2. Basic geometric operations

Our toolbox for computations with spectrahedra and implementing random walks, consists of three basic geometric operations: MEMBERSHIP, INTERSECTION, and REFLECTION.

For a spectrahedron S, MEMBERSHIP decides whether a point lies inside S, INTERSECTION computes the intersection of an algebraic curved trajectory C with the boundary ∂S, and REFLECTION

[1] `https://github.com/GeomScale/volume_approximation/tree/sample_spectrahedra`

computes the reflection of an algebraic curved trajectory when it hits ∂S. The last two operations are required because random walks can move along non-linear trajectories inside convex bodies. For those studied in this paper, the trajectories are parametric polynomial curves, of various degrees. Computation with these curves reduces to solving the polynomial eigenvalue problem (PEP).

2.1. Analysis of PEP

To estimate the complexity of INTERSECTION we need to bound the complexity of PEP. The Polynomial Eigenvalue Problem (PEP) consists in computing $\lambda \in \mathbb{R}$ and $\boldsymbol{x} \in \mathbb{R}^m$ that satisfy the (matrix) equation

$$P(\lambda)\,\boldsymbol{x} = 0 \Leftrightarrow (\boldsymbol{B}_d\lambda^d + \cdots + \boldsymbol{B}_1\lambda + \boldsymbol{B}_0)\boldsymbol{x} = 0\ , \tag{2}$$

where $P(\lambda)$ is a univariate matrix polynomial whose coefficients are matrices $\boldsymbol{B}_i \in \mathbb{R}^{m\times m}$. We further assume that $\boldsymbol{B}_d$ and $\boldsymbol{B}_0$ are invertible. In general, there are $\delta = m\,d$ values of λ. We refer the reader to [54] for a thorough exposition of PEP.

One approach for solving PEP is to linearalize the problem and to express the λ's as the eigenvalues of a larger matrix. For this, we transform Equation (2) into a linear pencil of dimension δ. Following [4], the *Companion Linearization* consists in solving the generalized eigenvalue problem $\boldsymbol{C}_0 - \lambda\boldsymbol{C}_1$, where

$$\boldsymbol{C}_0 = \begin{bmatrix} \boldsymbol{B}_d & 0 & \cdots & 0 \\ 0 & \boldsymbol{I}_m & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \boldsymbol{I}_m \end{bmatrix} \text{ and } \boldsymbol{C}_1 = \begin{bmatrix} \boldsymbol{B}_{d-1} & \boldsymbol{B}_{d-2} & \cdots & \boldsymbol{B}_0 \\ -\boldsymbol{I}_m & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & -\boldsymbol{I}_m & 0 \end{bmatrix},$$

and $\boldsymbol{I}_m$ denotes the $m \times m$ identity matrix. The eigenvectors $\boldsymbol{x}$ and $\boldsymbol{z}$ are related as follows: $\boldsymbol{z} = [1, \lambda, \ldots, \lambda^{d-2}, \lambda^{d-1}]^\top \otimes \boldsymbol{x}$.

To obtain an exact algorithm for PEP we exploit the assumption that $\boldsymbol{B}_d$ is invertible so as to transform the problem to the following classical eigenvalue problem $(\lambda\boldsymbol{I}_d - \boldsymbol{C}_2)\boldsymbol{z} = 0$, where

$$\boldsymbol{C}_2 = \begin{bmatrix} \boldsymbol{B}_{d-1}\boldsymbol{B}_d^{-1} & \boldsymbol{B}_{d-2}\boldsymbol{B}_d^{-1} & \cdots & \boldsymbol{B}_0\boldsymbol{B}_d^{-1} \\ -\boldsymbol{I}_m & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & -\boldsymbol{I}_m & 0 \end{bmatrix}.$$

The eigenvectors are roots of the characteristic polynomial of $\boldsymbol{C}_2$. Now the problem is to compute the eigenvalues of $\boldsymbol{C} \in \mathbb{R}^{\delta\times\delta}$. From a complexity point of view, the best algorithm relies on computing the roots of the characteristic polynomial [49], which is the method we follow in order to derive complexity bounds. Of course, in practice other methods are faster or more stable.

Lemma 2.1. *Consider a PEP of degree d, involving matrices of dimension $m \times m$, with integer elements of bitsize at most τ, see Equation (2). There is a randomized algorithm for computing the eigenvalues and the eigenvectors of PEP up to precision $\epsilon = 2^{-L}$, in $\widetilde{O}_B(\delta^{\omega+3}L)$, where $\delta = md$ and $L = \Omega(\delta^3\tau)$. The arithmetic complexity is $\widetilde{O}(\delta^{2.697} + \delta\lg(1/\epsilon))$.*

Proof. We can compute the characteristic polynomial of an $N \times N$ matrix M in $\widetilde{O}_B(N^{2.697+1} \lg\|M\|)$ using a randomized algorithm, see [28] and references therein. Here $\|M\|$ denotes the largest entry in absolute value. In our case, the elements of $\boldsymbol{C}_2$ have bitsize $\widetilde{O}(\delta\tau)$ and its characteristic polynomial is of degree d and coefficient bitsize $\widetilde{O}_B(\delta^2\tau)$. We compute it in $\widetilde{O}_B(\delta^{2.697+1}\delta\tau) = \widetilde{O}_B(\delta^{4.697}\tau)$ and isolate all its real roots in $\widetilde{O}_B(\delta^5 + \delta^4\tau)$ [43]; they correspond to the real eigenvalues of PEP. We can decrease the width of the isolating interval by a factor of $\epsilon = 2^{-L}$ for *all* the roots in $\widetilde{O}_B(\delta^3\tau + \delta L)$ [44]. Thus, the overall complexity is $\widetilde{O}_B(\delta^5 + \delta^{4.697}\tau + \delta L)$.

It remains to compute the corresponding eigenvectors. For each eigenvalue λ we may compute the corresponding eigenvector $\boldsymbol{z}$ by Gaussian elimination and back substitution to the (augmented) matrix $[\lambda \boldsymbol{I}_\delta - \boldsymbol{C}_2 \,|\, \boldsymbol{0}]$. This requires $\widetilde{O}(\delta^\omega)$ arithmetic operations. However, as λ is a root of the characteristic polynomial, one has to operate on algebraic numbers, which is highly nontrivial, and one needs to bound the number of bits needed to compute the elements of $\boldsymbol{z}$ correctly and to recover $\boldsymbol{x}$.

Hence, we employ separation bounds for polynomial system adapted to eigenvector computation [23]. One needs, as in the case of eigenvalues, $\widetilde{O}_B(\delta^4 + \delta^3\tau)$ bits to isolate the coordinates of the eigenvectors. To decrease the width of the corresponding isolating intervals by a factor of $\epsilon = 2^{-L}$, the number of bits becomes $\widetilde{O}_B(\delta^4 + \delta^3\tau + L)$. Thus, we compute the eigenvectors in $\widetilde{O}_B(\delta^\omega(\delta^4 + \delta^3\tau + L)) = \widetilde{O}_B(\delta^{\omega+4} + \delta^{\omega+3}\tau + \delta^\omega L)$.

For the arithmetic complexity we proceed as follows: We compute the characteristic polynomial in $\widetilde{O}(\delta^{2.697})$, we approximate its roots up to ϵ in $\widetilde{O}(\delta \lg(1/\epsilon))$. Finally, we compute the eigenvectors with $\widetilde{O}(\delta^\omega)$ arithmetic operations. So the overall cost is $\widetilde{O}(\delta^{2.697} + \delta \lg(1/\epsilon))$. □

2.2. MEMBERSHIP

The operation MEMBERSHIP$(\boldsymbol{F}, \boldsymbol{p})$ decides whether a point $\boldsymbol{p}$ lies in the interior of a spectrahedron $S = \{\boldsymbol{x} \in \mathbb{R}^n \,|\, \boldsymbol{F}(\boldsymbol{x}) \succeq 0\}$. For this, first, we construct the matrix $\boldsymbol{F}(\boldsymbol{p})$. If it is positive definite, then $\boldsymbol{p} \in S^\circ$. If it is positive semidefinite, then $\boldsymbol{p} \in \partial S$, otherwise $\boldsymbol{p} \in \mathbb{R}^n \setminus S$. The pseudo-code appears in Algorithm 1.

Algorithm 1: MEMBERSHIP$(\boldsymbol{F}, \boldsymbol{p})$

Input : An LMI $\boldsymbol{F}(\boldsymbol{x}) \succeq 0$ representing a spectrahedron S and a point $\boldsymbol{p} \in \mathbb{R}^n$.
Output: TRUE if $\boldsymbol{p} \in S$, FALSE otherwise.

```
λ_min ← smallest eigenvalue of F(p);
if λ_min ≥ 0 then return TRUE ;
return FALSE ;
```

Lemma 2.2. *Algorithm 1,* MEMBERSHIP$(\boldsymbol{F}, \boldsymbol{p})$*, requires* $\widetilde{O}(nm^2 + m^{2.697})$ *arithmetic operations. If* $\boldsymbol{F}$ *and* $\boldsymbol{p}$ *have integer elements of bitsize at most* τ*, respectively* σ*, then the bit complexity is* $\widetilde{O}_B((nm^2 + m^{3.697})(\tau + \sigma))$.

Proof. We construct $\boldsymbol{F}(\boldsymbol{p})$ in $O(nm^2)$. Then, with $O(m^{2.697})$ operations we compute its characteristic polynomial [28] and in $\widetilde{O}(m)$ we decide whether it has negative roots, for example by solving [43] or using fast subresultant algorithms [33, 36]. For the bit complexity, the construction costs $\widetilde{O}_B(nm^2(\tau + \sigma))$ and the computation of the characteristic polynomial costs $\widetilde{O}_B(m^{2.697+1}(\tau + \sigma))$, using a randomized algorithm [28]. One may test for negative roots, and thus eigenvalues, in $\widetilde{O}_B(m^2 n(\tau + \sigma))$ [36]. □

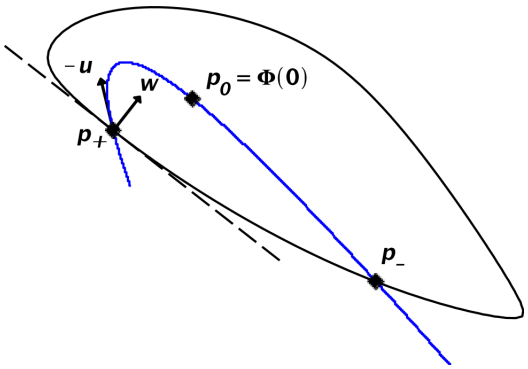

Figure 2: *A spectrahedron S described by $\boldsymbol{F}(\boldsymbol{x})$ and a parameterized curve Φ. The point $\boldsymbol{p}_0 = \Phi(0)$ lies in the interior of S, and the points $\boldsymbol{p}_+ = \Phi(t_+)$ and $\boldsymbol{p}_- = \Phi(t_-)$ on the boundary. Vector $\boldsymbol{w}$ is the surface normal of ∂S at $\boldsymbol{p}_+$ and $\boldsymbol{u}$ is the direction of Φ at time $t = t_+$.*

2.3. INTERSECTION

Consider a parametric polynomial curve C such that it has a non-empty intersection with a spectrahedron S. Throughout, we consider only the real trace of C. Assume that the value of the parameter $t = 0$ corresponds to a point $\boldsymbol{p}_0$, that lies in $C \cap S^\circ$. Furthermore, assume that the segment of C on which $\boldsymbol{p}_0$ lies, intersects the boundary of S transversally at two points, say $\boldsymbol{p}_-$ and $\boldsymbol{p}_+$. The operation INTERSECTION computes the parameters, t_- and t_+, corresponding to these two points. Figure 2 illustrates this discussion and the pseudo-code of INTERSECTION appears in Algorithm 2.

To prove correctness and estimate the complexity we proceed as follows: As before (see also Equation 1), S is the feasible region of linear matrix inequalities (LMI) $\boldsymbol{F}(\boldsymbol{x}) \succeq 0$. Consider the real trace of a polynomial curve C, with parameterization

$$\begin{array}{rccl} \Phi & : & \mathbb{R} & \to & \mathbb{R}^n \\ & & t & \mapsto & \Phi(t) := (p_1(t), \ldots, p_n(t)), \end{array} \tag{3}$$

where $p_i(t) = \sum_{j=0}^{d_i} p_{i,j} t^j$ are univariate polynomials in t of degree d_i, for $i \in [m]$. Also let $d = \max_{i\in[m]}\{d_i\}$. If the coefficients of the polynomials are integers, then we further assume that the maximum coefficient's bitsize is bounded by τ.

As t varies over the real line, there may be several disjoint intervals, for which the corresponding segment of C lies in S°. We aim to compute the endpoints, t_- and t_+, of a maximal such interval containing $t = 0$. Let $\boldsymbol{p}_0 = \Phi(0)$; by assumption it holds $\boldsymbol{F}(\Phi(0)) = \boldsymbol{F}(\boldsymbol{p}_0) \succ 0$.

The input of INTERSECTION (Algorithm 2) is $\boldsymbol{F}$, the LMI representation of S, and $\Phi(t)$, the polynomial parameterization of C. Its crux is a routine, PEP, that solves a polynomial eigenvalue problem. The following lemma exploits this relation.

Lemma 2.3 (PEP and $S \cap C$)**.** *Consider the spectrahedron $S = \{\boldsymbol{x} \in \mathbb{R}^n \,|\, \boldsymbol{F}(\boldsymbol{x}) \succeq 0\}$. Let $\Phi : \mathbb{R} \to \mathbb{R}^n$ be a parameterization of a polynomial curve C, such that $\Phi(0) \in S^\circ$. Let $[t_-, t_+]$ be the maximal interval containing 0, such that the corresponding part of C lies in S. Then, t_-, respectively t_+, is the maximum negative, respectively minimum positive, polynomial eigenvalue of $\boldsymbol{F}(\Phi(t))\boldsymbol{x} = 0$, where $\boldsymbol{F}(\Phi(t)) = \boldsymbol{B}_0 + t\boldsymbol{B}_1 + \cdots + t^d \boldsymbol{B}_d$.*

Algorithm 2: INTERSECTION($\boldsymbol{F}, \Phi(t)$)

Input : An LMI $\boldsymbol{F}(\boldsymbol{x}) \succeq 0$ for a spectrahedron S and a parameterization $\Phi(t)$ of a polynomial curve C

Require: $\Phi(0) \in S^\circ$

Output : t_-, t_+ s.t. $\Phi(t_-), \Phi(t_+) \in \partial S$

1 $T := \{t_1 \leq t_2 \leq \cdots \leq t_\ell\} \leftarrow$ PEP($\boldsymbol{F}(\Phi(t))$);
2 $t_- \leftarrow \max\{t \in T \mid t < 0\}$; // *max negative polynomial eigenvalue*
3 $t_+ \leftarrow \min\{t \in T \mid t > 0\}$; // *min positive polynomial eigenvalue*
4 **return** t_-, t_+;

Proof. The composition of $\boldsymbol{F}(\boldsymbol{x})$ and $\Phi(t)$ gives

$$\boldsymbol{F}(\Phi(t)) = \boldsymbol{A}_0 + p_1(t)\boldsymbol{A}_1 + \cdots + p_n(t)\boldsymbol{A}_n. \tag{4}$$

We rewrite Expression (4) by grouping the coefficients for each t^i, $i \in [d]$, then

$$\boldsymbol{F}(\Phi(t)) = \boldsymbol{B}_0 + t\boldsymbol{B}_1 + \cdots + t^d\boldsymbol{B}_d, \tag{5}$$

where $\boldsymbol{B}_k = \sum_{j=0}^{n} p_{j,k}\,\boldsymbol{A}_j$, for $0 \leq k \leq d$. We use the convention that $p_{j,k} = 0$, when $k > d_j$.

For $t = 0$, it holds, by assumption, that $\boldsymbol{F}(\Phi(0)) = \boldsymbol{B}_0 \succ 0$: point $\Phi(0)$ lies in the interior of S. Actually, for any $\boldsymbol{x} \in S^\circ$ it holds $\boldsymbol{F}(\boldsymbol{x}) \succ 0$. On the other hand, if $\boldsymbol{x} \in \partial S$, then $\boldsymbol{F}(\boldsymbol{x}) \succeq 0$. Our goal is to compute the maximal interval $[t_-, t_+]$ that contains 0 and for every t in it, we have $\boldsymbol{F}(\Phi(t)) \succeq 0$.

Starting from point $\Phi(0)$, by varying t, we move on the trajectory that C defines (in both directions) until we hit the boundary of S. When we hit ∂S, matrix $\boldsymbol{F}(\Phi(t))$ is not strictly definite anymore. Thus, its determinant vanishes.

Consider the function $\theta : \mathbb{R} \to \mathbb{R}$, where $\theta(t) = \det \boldsymbol{F}(\Phi(t))$ is a univariate polynomial in t. If a point $\Phi(t)$ is on the boundary of the spectrahedron, then $\theta(t) = 0$. We opt to compute t_- and t_+, such that $t_- \leq 0 \leq t_+$ and $\theta(t_-) = \theta(t_+) = 0$. At $t = 0$, $\theta(0) > 0$ and the graph of θ is above the t-axis. So C intersects the boundary when the graph of θ touches the t-axis for the first time at $t_1 \leq 0 \leq t_2$. It follows that $t_- = t_1$ and $t_+ = t_2$ are the maximum negative and minimum positive roots of θ, or equivalently the corresponding polynomial eigenvalues of $\boldsymbol{F}(\Phi(t))$. □

Lemma 2.4. *Algorithm 2,* INTERSECTION($\boldsymbol{F}, \Phi(t)$)*, uses* $\widetilde{O}((md)^{2.697} + md \lg L)$ *arithmetic operations in order to compute the intersection, up to precision* $\epsilon = 2^{-L}$*, of an LMI,* $\boldsymbol{F}$*, consisting of* n *matrices of dimension* $m \times m$ *with a parametric curve,* $\Phi(t)$*, of degree* d*.*

Proof. We have to construct PEP and solve it. Since $\Phi(t)$ has degree d, then $\boldsymbol{F}(\Phi(t)) = \boldsymbol{B}_0 + t\boldsymbol{B}_1 + \cdots + t^d\boldsymbol{B}_d$ is a PEP of degree d. This construction costs $O(dnm^2)$ operations. The solving phase, using Lemma 2.1, requires $\widetilde{O}((md)^{2.697} + md \lg L)$ arithmetic operations and dominates the complexity bound of the algorithm. □

2.4. REFLECTION

The REFLECTION operation (Algorithm 3) takes as input an LMI representation, $\boldsymbol{F}$, of a spectrahedron S and a polynomial curve C, given by a parameterization Φ. Assume that $t = 0$ corresponds to a point $\Phi(0) \in S^\circ \cap C$. Starting from $t = 0$, we increase t along the positive real

Algorithm 3: REFLECTION $(\boldsymbol{F}, \Phi(t))$

Input : An LMI $\boldsymbol{F}(\boldsymbol{x}) \succeq 0$ for a spectrahedron S and a parameterization $\Phi(t)$ of a polynomial curve C.

Require: (i) $\Phi(0) \in S^\circ$
(ii) C intersects ∂S transversally at a smooth point.

Output : t_+ such that $\Phi(t_+) \in \partial S$ and the direction of the reflection, $\boldsymbol{s}_+$, at this point.

$t_-, t_+ \leftarrow$ INTERSECTION $(\boldsymbol{F}, \Phi(t))$;
$\boldsymbol{w} \leftarrow \nabla \det \boldsymbol{F}(\Phi(t_+))$;
$\boldsymbol{w} \leftarrow \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$; // *Normalize* $\boldsymbol{w}$
$\boldsymbol{s}_+ \leftarrow \frac{d\Phi}{dt}(t_+) - 2\langle \nabla \frac{d\Phi}{dt}(t_+), \boldsymbol{w} \rangle \boldsymbol{w}$;
return $t_+, \boldsymbol{s}_+$;

semi-axis. As t changes, we move along the curve C through $\Phi(t)$, until we hit the boundary of S at the point $\boldsymbol{p}_+ := \Phi(t_+) \in \partial S$, for some $t_+ > 0$. Then, a specular reflection occurs at this point with direction $\boldsymbol{s}_+$; this is the reflected direction. We output t_+ and $\boldsymbol{s}_+$. Figure 2 depicts the procedure.

The boundary of S, ∂S, with respect to the Euclidean topology, is a subset of the real algebraic set $\{\boldsymbol{x} \in \mathbb{R}^n \mid \det(\boldsymbol{F}(\boldsymbol{x})) = 0\}$. The latter is a real hypersurface defined by one (determinantal) equation. For any $\boldsymbol{x} \in \partial S$ we have $\texttt{rank}(\boldsymbol{F}(\boldsymbol{x})) \leq m - 1$. We assume that $\boldsymbol{p}_+ = \Phi(t_+)$ is such that $\texttt{rank}(\boldsymbol{F}(\boldsymbol{p}_+)) = m - 1$. The normal direction at a point $\boldsymbol{p} \in \partial S$, is the gradient of $\det \boldsymbol{F}(\boldsymbol{p})$.

We compute the reflected direction using the following formula

$$\boldsymbol{s}_+ = \boldsymbol{u} - \tfrac{2}{|\boldsymbol{w}|^2}\langle \boldsymbol{u}, \boldsymbol{w} \rangle \boldsymbol{w}, \tag{6}$$

where $\boldsymbol{w}$ is the normalized gradient vector at the point $\Phi(t_+)$ and $\boldsymbol{u} = \frac{d\Phi}{dt}(t_+)$ is the direction of the trajectory at this point. We illustrate the various vectors in Figure 2.

Lemma 2.5 (Gradient of $\det \boldsymbol{F}(\boldsymbol{x})$)**.** *Assume that $\boldsymbol{x} \in \partial S$ and the rank of the $m \times m$ matrix $\boldsymbol{F}(\boldsymbol{x})$ is $m - 1$. Then*

$$\nabla \det(\boldsymbol{F}(\boldsymbol{x})) = c \cdot (\boldsymbol{v}^\top \boldsymbol{A}_1 \boldsymbol{v}, \cdots, \boldsymbol{v}^\top \boldsymbol{A}_n \boldsymbol{v}), \tag{7}$$

where $c = \frac{\mu(\boldsymbol{F}(\boldsymbol{x}))}{|\boldsymbol{v}|^2}$, $\mu(\boldsymbol{F}(\boldsymbol{x}))$ is the product of the nonzero eigenvalues of $\boldsymbol{F}(\boldsymbol{x})$, and $\boldsymbol{v}$ is a non-trivial vector in the kernel of $\boldsymbol{F}(\boldsymbol{x})$. If $\texttt{rank}(\boldsymbol{F}(\boldsymbol{x})) \leq m - 2$, then the gradient is the zero.

Proof. From Lemma Appendix A.2:

$$\frac{\partial \det \boldsymbol{F}(\boldsymbol{x})}{\partial x_k} = \texttt{Trace}\left(\boldsymbol{F}(\boldsymbol{x})^* \boldsymbol{A}_k\right). \tag{8}$$

If $\texttt{rank}(\boldsymbol{F}(\boldsymbol{x})) \leq -2$, then $\boldsymbol{F}(\boldsymbol{x})^*$ is the zero matrix. If we assume that $\texttt{rank}(\boldsymbol{F}(\boldsymbol{x})) = m - 1$, then using Lemma Appendix A.3:

$$\begin{aligned}\texttt{Trace}\left(\boldsymbol{F}(\boldsymbol{x})^* \boldsymbol{A}_k\right) &= \texttt{Trace}\left(\mu(\boldsymbol{F}(\boldsymbol{x}))\frac{\boldsymbol{v}\boldsymbol{u}^\top}{\boldsymbol{u}^\top \boldsymbol{v}}\boldsymbol{A}_k\right) \\ &= \frac{\mu(\boldsymbol{F}(\boldsymbol{x}))}{\boldsymbol{u}^\top \boldsymbol{v}} \cdot \texttt{Trace}\left(\boldsymbol{v}\boldsymbol{u}^\top \boldsymbol{A}_k\right) = \frac{\mu(\boldsymbol{F}(\boldsymbol{x}))}{\boldsymbol{u}^\top \boldsymbol{v}} \cdot \boldsymbol{u}^\top \boldsymbol{A}_k \boldsymbol{v}.\end{aligned}$$

However, since $\boldsymbol{F}(\boldsymbol{x})$ is symmetric, we can choose $\boldsymbol{v} = \boldsymbol{u}$, so:

$$\frac{\mu(\boldsymbol{F}(\boldsymbol{x}))}{\boldsymbol{u}^\top \boldsymbol{v}} \cdot \boldsymbol{u}^\top \boldsymbol{A}_k \boldsymbol{v} = \frac{\mu(\boldsymbol{F}(\boldsymbol{x}))}{|\boldsymbol{v}|^2} \cdot \boldsymbol{v}^\top \boldsymbol{A}_k \boldsymbol{v}.$$

□

The algorithm REFLECTION exploits Lemma 2.5. Nevertheless, it is not necessary to perform all then computations that the lemma indicates. For example, because we will normalize the resulting vector and we do not need its actual direction (internal or external), we can omit the computation of c. Moreover, the nonzero vector $\boldsymbol{v}$, which satisfies $\boldsymbol{F}(\boldsymbol{p})\boldsymbol{v} = 0$, corresponds to the eigenvector w.r.t. the eigenvalue t_+ from the PEP (Lemma 2.3). This is true because $\boldsymbol{p} = \Phi(t_+) \in \partial S$ and thus $\det \boldsymbol{F}(\Phi(t_+)) = 0$.

At this point we should note that we compute the eigenvalues of PEP up to some precision. Since the matrix-vector multiplication is backward stable, a small perturbation on $\boldsymbol{v}$ does not affect the computation of each coordinate of $\nabla \det(\boldsymbol{F}(\boldsymbol{x}))$ [55, p. 104]. We quantify the accuracy of the computed $\nabla \det(\boldsymbol{F}(\boldsymbol{x}))$ using floating point arithmetic as follows:

Lemma 2.6. *The relative error of each coordinate of the gradient given in Lemma 2.5 when we compute it using floating point arithmetic with machine epsilon ϵ_M is $O(\frac{\epsilon_M}{\sigma_{\max}(A_i)})$, for $i \in [n]$, where $\sigma_{\max}$ is the largest singular value of A_i.*

Proof. Let $A \in \mathbb{R}^{m \times m}$ be a symmetric matrix and consider the map $f : \boldsymbol{v} \mapsto \boldsymbol{v}^T A \boldsymbol{v}$. The relative condition number of f as defined in [55, p. 90] is

$$k(\boldsymbol{v}) = \frac{\|J(\boldsymbol{v})\|}{\|f(\boldsymbol{v})\|/\|\boldsymbol{v}\|} = 2\frac{\|A\boldsymbol{v}\|}{\boldsymbol{v}^T A_i \boldsymbol{v}} = 2\frac{\sigma_{\max}(A)}{\sigma^2_{\max}(A)} = \frac{2}{\sigma_{\max}(A)},$$

where $J(\cdot)$ is the Jacobian of f. According to Theorem 15.1 in [55, p. 111], since the matrix-vector multiplication is backward stable, the relative error of each coordinate in the gradient computation of Lemma 2.5 is $O(\frac{\epsilon_M}{\sigma_{\max}(A_i)})$, for $1 \leq i \leq n$. □

Lemma 2.7. *Let S be a spectrahedron represented by an LMI, $\boldsymbol{F}(\boldsymbol{x})$, consisting of n matrices of dimension $m \times m$. Also let C be a parametric curve with parameterization $\Phi(t)$, involving polynomials of degree at most d. Algorithm 3, REFLECTION$(\boldsymbol{F}, \Phi(t))$, computes the intersection, up to precision $\epsilon = 2^{-L}$, of S with C, and the reflection of C at ∂S, by performing $\widetilde{O}((md)^{2.697} + md \lg L + dnm^2)$ arithmetic operations.*

Proof. By inspecting Algorithm 3 we notice that the complexity of the algorithm depends on the construction of $\nabla \det(\boldsymbol{F}(\boldsymbol{x}))$ and the call to INTERSECTION.

To compute $\nabla \det(\boldsymbol{F}(\boldsymbol{x}))$ we just need to compute $(\boldsymbol{v}^\top \boldsymbol{A}_1 \boldsymbol{v}, \cdots, \boldsymbol{v}^\top \boldsymbol{A}_n \boldsymbol{v})$. If we have already computed $\boldsymbol{v}$, then this computation requires $O(nm^2)$ operations. The computation of the derivative of $\Phi(t)$ is straightforward, as Φ is a univariate polynomial. Taking into account the complexity of INTERSECTION, the total complexity for REFLECTION is $\widetilde{O}((md)^{2.697} + md \lg L + dnm^2 + nm^2) = \widetilde{O}((md)^{2.697} + md \lg L + dnm^2)$. □

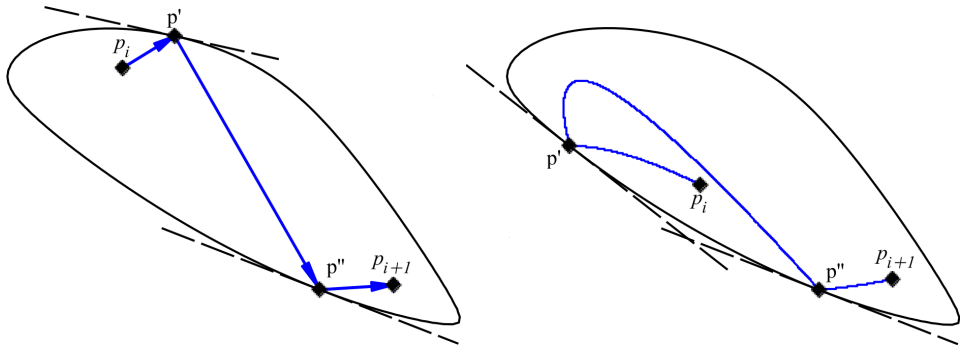

Figure 3: *The i-th step of the* W-Billard *[Algorithm 5] (left) and of the* W-HMC-r *[Algorithm 6] (right) random walks.*

	per-step Complexity
W-HnR	$O(m^{2.697} + m \lg L + nm^2)$
W-CHnR	$O(m^{2.697} + m \lg L + m^2)$
W-Billard	$\widetilde{O}(\rho(m^{2.697} + m \lg L + nm^2))$
W-HMC-r	$\widetilde{O}(\rho((dm)^{2.697} + md \lg L + dnm^2))$

Table 1: *The per-step complexity of the random walks in Section 3.*

2.5. An example in 2D

Consider a spectrahedron S in the plane (Figure 3), given by an LMI $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{A}_0 + x_1\boldsymbol{A}_1 + x_2\boldsymbol{A}_2$. The matrices $\boldsymbol{A}_i$, $0 \leq i \leq 2$, are in the appendix.

Starting from the point $\boldsymbol{p}_0 = (-1, 1)^\top$, we walk along the line L with parameterization: $\Phi(t) = \boldsymbol{p}_0 + t\boldsymbol{u}$, where $\boldsymbol{u} = (1.3, 0.8)^\top$. Then, intersection finds the intersection of S with L, by solving the degree one PEP, $(\boldsymbol{B}_0 + t\boldsymbol{B}_1)\boldsymbol{x} = 0$, where $\boldsymbol{B}_0 = \boldsymbol{F}(\boldsymbol{p}_0)$ and $\boldsymbol{B}_1 = u_1\boldsymbol{A}_1 + u_2\boldsymbol{A}_2$. Acquiring $t_- = -0.8$ and $t_+ = 0.5$, we obtain the intersection point $\boldsymbol{p_1}$, which corresponds to $\boldsymbol{p}_0 + t_+\boldsymbol{u} = (-0.3, 1.4)^\top$.

To compute the direction of the trajectory, immediately after we reflect on the boundary of S at $\boldsymbol{p_1}$, reflection computes

$$\boldsymbol{w} = \frac{\nabla \det \boldsymbol{F}(\Phi(t_+))}{|\nabla \det \boldsymbol{F}(\Phi(t_+))|} = (\boldsymbol{v}^\top \boldsymbol{A}_1 \boldsymbol{v}, \boldsymbol{v}^\top \boldsymbol{A}_2 \boldsymbol{v})^\top = (-0.2, -1)^\top, \tag{9}$$

where $\boldsymbol{v}$ is the eigenvector of $(\boldsymbol{B}_0 + t\boldsymbol{B}_1)\boldsymbol{x} = 0$, with eigenvalue t_+. The reflected direction is $\boldsymbol{u'} = \boldsymbol{u} - 2\langle \boldsymbol{u}, \boldsymbol{w} \rangle \boldsymbol{w} = (0.8, -1.3)^\top$.

3. Random walks

Using the basic geometric operations of Section 2, we implement and analyze three random walks for spectrahedra: Hit and Run (W-HnR), its variant Coordinate Directions Hit and Run (W-CHnR), Billiard Walk (W-Billard), and Hamiltonian Monte Carlo with reflections (W-HMC-r). In Table 1, we present the per-step arithmetic complexity for each random walk.

3.1. Hit and Run

W-HnR (Algorithm 4) is a random walk that samples from any probability distribution π truncated to a convex body K; in our case a spectrahedron S. We should mention that there exist bounds for its mixing time only when π is log-concave distribution, for example the uniform distribution, which is $\widetilde{O}(n^3)$.

At the i-th step, W-HnR chooses uniformly at random a (direction of a) line ℓ, passing from its current position $\boldsymbol{p}_i$. Let $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ be the intersection points of ℓ with S. Let π_ℓ be the restriction of π on the segment $[\boldsymbol{p_2}, \boldsymbol{p_2}]$. Then, we choose $\boldsymbol{p}_{i+1}$ from $[\boldsymbol{p_1}, \boldsymbol{p_2}]$ w.r.t. the distribution π_ℓ.

Algorithm 4: Hit-and-Run_Walk (W-HnR)

Input : LMI $\boldsymbol{F}(\boldsymbol{x}) \succeq 0$ for a spectrahedron S & a point $\boldsymbol{p}_i$.
Require: $\boldsymbol{p}_i \in S$
Output : The point $\boldsymbol{p}_{i+1}$ of the $(i+1)$-th step of the walk.

1 BO ($\boldsymbol{F}$, interior point $\boldsymbol{p_i}$) $\boldsymbol{v} \leftarrow_R \mathcal{U}(\partial\mathcal{B}_n)$; // *choose direction*
2 $\Phi(t) := \boldsymbol{p}_i + t\boldsymbol{v}$; // *define trajectory*
3 $t_-, t_+ \leftarrow$ intersection $(\boldsymbol{F}, \Phi(t))$;
4 $\boldsymbol{p}_{i+1} \leftarrow_R [\boldsymbol{p}_i + t_-\boldsymbol{v},\ \boldsymbol{p}_i + t_+\boldsymbol{v}]$ w.r.t. π_ℓ;
5 **return** $\boldsymbol{p}_{i+1}$;

Lemma 3.1. *The per-step complexity of* W-HnR *is* $\widetilde{O}(m^{2.697} + m \lg 1/\epsilon + nm^2)$.

Proof. The per-step complexity of W-HnR is dominated by the intersection, which requires $O(nm^2)$ operations for the construction of the pep and $\widetilde{O}(m^{2.697} + m \lg 1/\epsilon)$ for solving it; in the case where we want to approximate the intersection point up to a factor or $\epsilon = 2^{-L}$ (Lemma 2.4). □

There is also a variation of W-HnR, the *coordinate directions Hit and Run* (W-CHnR) [50]. This walk chooses the direction vector randomly and uniformly among the vector basis $\{\boldsymbol{e}_i,\ i \in [n]\}$. In W-CHnR, for every step aside the first, the construction of the pep takes $O(m^2)$ operations, and the complexity does not depend on the dimension n. The reason for this improvement is that to build the pep, $\boldsymbol{F}(\boldsymbol{p}_i + t\boldsymbol{e}_j) = \boldsymbol{F}(\boldsymbol{p}_i) + t\boldsymbol{A}_j$, we can obtain the value of $\boldsymbol{F}(\boldsymbol{p}_i)$ from $\boldsymbol{F}(\boldsymbol{p}_i) = \boldsymbol{F}(\boldsymbol{p}_{i-1}) + \hat{t}\boldsymbol{A}_k$, assuming that at the previous step we have chosed $\boldsymbol{e}_k$ as direction. There is no theoretical mixing time for W-CHnR.

3.2. Billiard walk

W-Billard [45], Algorithm 5, samples a convex body K under the uniform distribution; no theoretical results for its mixing time exist. At i-th step, being at position $\boldsymbol{p}_i$, it chooses uniformly a direction vector $\boldsymbol{v}$ and a number ℓ, where $\ell = -\tau \ln \eta$ and $\eta \sim U(0,1)$. Then, it moves at the direction of $\boldsymbol{v}$ for distance ℓ. If during the movement, it hits the boundary without having covered the required distance ℓ, then it continues on a reflected trajectory. If the number of reflections exceeds a bound ρ, it stays at $\boldsymbol{p}_i$. In [45] they experimentally conclude that W-Billard mixes faster when $\tau \approx diam(K)$, where $diam(K)$ is the diameter of K.

Lemma 3.2. *The per-step complexity of* W-Billard *is* $\widetilde{O}(\rho(m^{2.697} + m \lg L + nm^2))$*, where* ρ *is the number of reflections.*

Algorithm 5: BILLIARD_WALK (W-BILLARD)

Input : An LMI $\boldsymbol{F}(\boldsymbol{x}) \succeq 0$ for a spectrahedron S, a point $\boldsymbol{p_i}$, the diameter τ of S and a bound ρ on the number of reflections.
Require: $\boldsymbol{p_i} \in S$
Output : The point $\boldsymbol{p}_{i+1}$ of the $(i+1)$-th step of the walk.

$\ell \leftarrow -\tau \ln \eta$; $\eta \leftarrow_R \mathcal{U}((0,1))$; // *choose length*
$\boldsymbol{v} \leftarrow_R \mathcal{U}(\partial\mathcal{B}_n)$; // *choose direction*
$\boldsymbol{p} \leftarrow \boldsymbol{p_i}$;
do
  $\Phi(t) := \boldsymbol{p} + t\boldsymbol{v}$; // *define trajectory*
  $t_+, \boldsymbol{s}_+ \leftarrow$ REFLECTION $(\boldsymbol{F}, \Phi(t))$;
  $\hat{t} \leftarrow \min\{t_+, \ell\}$; $\boldsymbol{p} \leftarrow \Phi(\hat{t})$;
  if $\bar{t} < \ell$ **then** $\boldsymbol{v} \leftarrow \boldsymbol{s}_+$;
  $\ell \leftarrow \ell - \hat{t}$;
while $\ell > 0$;
if #{*reflections*} $> \rho$ **then return** $\boldsymbol{p_{i+1}} = \boldsymbol{p_i}$;
return $\boldsymbol{p}_{i+1} = \boldsymbol{p}$

Proof. The per-step complexity of W-BILLARD is dominated by the REFLECTION, which requires $\widetilde{O}(m^{2.697} + m \lg L + nm^2)$ arithmetic operations (Lemma 2.7), when we want to approximate the intersection point up to a factor of 2^{-L}. Since we allow at most ρ reflections per step, the total complexity becomes $\widetilde{O}(\rho(m^{2.697} + m \lg L + nm^2))$. □

3.3. Hamiltonian Monte Carlo with Reflections

Algorithm 6: HMC_W_REFLECTION (W-HMC-R)

Input : An LMI $\boldsymbol{F}(\boldsymbol{x}) \succeq 0$ representing a spectrahedron S, a point $\boldsymbol{p_i}$, the diameter τ of S and a bound ρ to the number of reflections.
Require: $\boldsymbol{p_i} \in S$
Output : The point $\boldsymbol{p}_{i+1}$ of the $(i+1)$-th step of the walk.

$\ell \leftarrow \tau\eta$; $\eta \leftarrow_R \mathcal{U}((0,1))$; // *choose length*
$\boldsymbol{v} \leftarrow_R \mathcal{N}(0, I_n)$; // *choose direction*
do
  Compute trajectory $\Phi(t)$ from ODE (10);
  $t_+, \boldsymbol{s}_+ \leftarrow$ REFLECTION $(\boldsymbol{F}, \Phi(t))$;
  $\hat{t} \leftarrow \min\{t_+, \ell\}$; $\boldsymbol{p} \leftarrow \Phi(\hat{t})$; $\boldsymbol{v} \leftarrow \boldsymbol{s}$; $\ell \leftarrow \ell - \hat{t}$;
while $\ell > 0$;
if # {*reflections*} $> \rho$ **then return** $\boldsymbol{p}_{i+1} = \boldsymbol{p_i}$;
return $\boldsymbol{p}_{i+1} = \boldsymbol{p}$;

Hamiltonian Monte Carlo (HMC), can be used to sample from any probability distribution π. Our focus lies again on the log-concave distributions, that is $\pi(\boldsymbol{x}) \propto e^{-\alpha f(\boldsymbol{x})}$. We exploit the setting in [34] as they approximate the Hamiltonian trajectory with a polynomial curve. In this setting, if we assume that f is a strongly convex function, then the mixing time of HMC is $O(k^{1.5} \log(n/\epsilon))$,

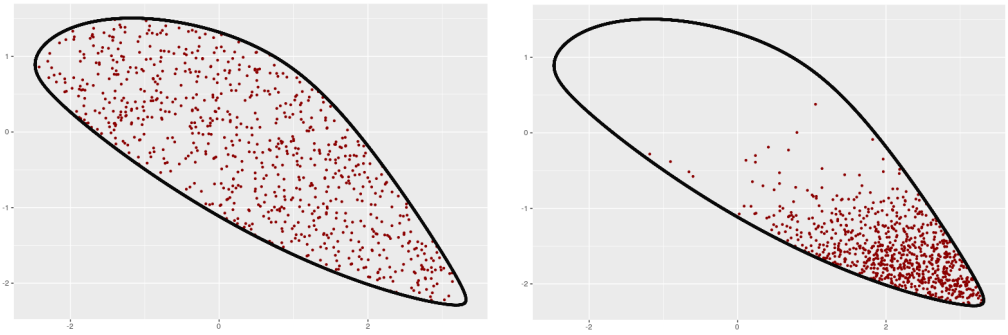

Figure 4: *Samples from the uniform distribution with* W-Billard *(left) and from the Boltzmann distribution* $\pi(x) \propto e^{-cx/T}$, *where* $T = 1$, $c = [-0.09, 1]^T$, *with* W-HMC-r *(right). The volume of this spectrahedron is* 10.23.

where κ is the condition number of $\nabla^2 f$ [34]. If we truncate π by considering its restriction in a convex body, then we can use boundary reflections (W-HMC-r), as in Algorithm 6, to ensure that the random walk converges to the target distribution [15]; however, in this case the mixing time is unclear.

The Hamiltonian dynamics behind HMC operate on a n-dimensional position vector $\boldsymbol{p}$ and a n-dimensional momentum $\boldsymbol{v}$. So the full state space has $2n$ dimensions. The system is described by a function of $\boldsymbol{p}$ and $\boldsymbol{v}$ known as the Hamiltonian,

$$H(\boldsymbol{p}, \boldsymbol{v}) = U(\boldsymbol{p}) + K(\boldsymbol{v}) = f(\boldsymbol{p}) + \frac{1}{2}|\boldsymbol{v}|^2.$$

To sample from π, one has to solve the following system of Ordinary Differential Equations (ODE):

$$\begin{aligned} \frac{d\boldsymbol{p}}{dt} &= \frac{\partial H(\boldsymbol{p}, \boldsymbol{v})}{\partial \boldsymbol{v}} \\ \frac{d\boldsymbol{v}}{dt} &= -\frac{\partial H(\boldsymbol{p}, \boldsymbol{v})}{\partial \boldsymbol{p}} \end{aligned} \quad \Rightarrow \quad \begin{cases} \frac{d\boldsymbol{p}(t)}{dt} = \boldsymbol{v}(t) \\ \frac{d\boldsymbol{v}(t)}{dt} = -\alpha \nabla f(\boldsymbol{p}) \end{cases} . \tag{10}$$

If $\pi(\boldsymbol{x})$ is a log-concave density, then we can approximate the solution of the ODE with a low degree polynomial trajectory [34], using the collocation method. A degree $d = O(1/\log(\epsilon))$ suffices to obtain a polynomial trajectory with error $O(\epsilon)$, for a fixed time interval, while we perform just $\widetilde{O}(1)$ evaluations of $\nabla f(\boldsymbol{x})$.

HMC at the i-th step uniformly samples a step ℓ from a proper interval to move on the trajectory implied by ODE (10), choses $\boldsymbol{v}$ randomly from $\mathcal{N}(0, I_n)$, and updates $\boldsymbol{p}$ using the ODE in (10), for $t \in [0, \ell]$. When π is truncated in a convex body, then W-HMC-r fixes an upper bound ρ on the number of reflections and reflects a polynomial trajectory as we describe in Section 2.4.

Each step of W-HMC-r, when $\pi(x)$ is a log-concave density truncated by S, costs $\widetilde{O}(\rho((dm)^{2.697} + md \lg L + dnm^2))$, if we approximate the intersection points up to a factor 2^{-L}, where d is the degree of the polynomial that approximates the solution of the ODE (10).

Lemma 3.3. *The per-step complexity of* W-HMC-r *is* $\widetilde{O}(\rho((dm)^{2.697} + md \lg L + dnm^2))$, *where* ρ *is the number of reflections.*

4. Applications and experiments

This section demonstrates and compares the algorithms of Section 3 and the efficiency of our software on three applications that rely on sampling from spectrahedra.

We call *walk length* the number of the intermediate points that a random walk visits before producing a single sample. The longer the walk length of a random walk is, the smaller the distance of the current distribution to the stationary (target) distribution becomes. Typically we choose a sufficiently large length for the first sample, this procedure is often called "burning".

Our code is parameterized by the floating point precision of the computations. We use `Eigen` [25] for basic linear algebra operations, such as Cholesky decomposition and matrix multiplication. For eigenvalue computations, we employ `Spectra` [51], which is based on `Eigen` and offers crucial optimizations. First, it solves generalized eigenvalue problems of special structure; that is $(B_0 - \lambda B_1)v = 0$, when B_0 is positive semidefinite and B_1 symmetric. This operation is encountered when W-Billard or W-HnR call intersection. Second, it offers directly the computation of the largest eigenvalue which corresponds to t_+ after a simple transformation. Finally, `Spectra` provides $\sim$ 20x speedup over the default eigenvalue computation by `Eigen`. To the best of our knoledge, our software is the first that can sample efficiently from spectrahedra and estimates volumes up to a few hundred dimensions. It is accessible on github.[2]

For our experiments, we generate random spectrahedra following [18]. In particular, to construct the LMI of Equation (1) we set A_0 to be positive semidefinite, i.e., $A_0 = ZZ^T + I_m$, where we pick the elements of $Z \in \mathbb{R}^{m\times m}$ uniformly at random from $[0, 1]$. Then, for $A_i,\ i = 1, \ldots, n$ we set,

$$A_i = \begin{bmatrix} \tilde{Q} & 0 \\ 0 & -\tilde{Q} \end{bmatrix},\ \tilde{Q} = Q + Q^T,$$

where we pick the elements of $Q \in \mathbb{R}^{(m/2)\times(m/2)}$ uniformly at random from $[-1, 1]$. We performed all the experiments on PC with `Intel Core i7-6700 3.40GHz × 8 CPU` and `32GB RAM`.

4.1. Volume computation

We use the geometric operations (Section 2) and the random walks (Section 3) to compute the volume $\mathrm{vol}(S)$ of spectrahedron S. Our implementation approximates $\mathrm{vol}(S)$ within relative error 0.1 with high probability in a few minutes, for dimension $n = 100$.

A typical randomized algorithm for volume approximation exploits a Multiphase Monte Carlo (MMC) technique, which reduces volume approximation of convex body S to computing a telescoping product of ratios of integrals over S. In particular, for any sequence of functions $\{f_0, \ldots, f_k\}$, where $f_i : \mathbb{R}^n \to \mathbb{R}$, we have:

$$\mathrm{vol}(S) = \int_S 1dx = \int_S f_k(x)dx \frac{\int_S f_{k-1}(x)dx}{\int_S f_k(x)dx} \cdots \frac{\int_S 1dx}{\int_S f_0(x)dx}. \tag{11}$$

Notice that $\frac{\int_P f_{i-1}(x)dx}{\int_P f_i(x)dx} = \int_P \frac{f_{i-1}(x)}{f_i(x)} \frac{f_i(x)}{\int_P f_i(x)dx}dx$. To estimate each ratio of integrals, we sample N points from a distribution proportional to f_i and, we use the unbiased estimator $\frac{1}{N}\sum_{j=1}^{N} \frac{f_{i-1}(x_j)}{f_i(x_j)}$. To exploit Equation (11) we have to (i) fix the sequence such that k is as small as possible, (ii) select f_i's such that we can compute efficiently each integral ratio, and (iii) compute $\int_P f_k(x)dx$. The

[2] `https://github.com/GeomScale/volume_approximation/tree/sample_spectrahedra`

S-n-m	$\mu \pm t_{\alpha,\nu-1}\frac{s}{\sqrt{\nu}}$	*Points*	*Time* (*sec*)
S-40-40	(1.34 ± 0.12)e-06	9975.2	6.7
S-60-60	(1.23 ± 0.11)e-20	20370.9	28.5
S-80-80	(4.24 ± 0.26)e-33	31539.1	124.4
S-100-100	(1.21 ± 0.10)e-51	52962.7	362.3

Table 2: *(Volume of spectrahedra) For each S-n-m we run $\nu = 10$ experiments; m is the matrix dimension in LMI and n the ambient dimension. μ stands for the average volume, s for the standard deviation; we give a confidence interval with level of confidence $\alpha = 0.05$; $t_{\alpha,\nu-1}$ is the critical value of student's distribution with $\nu - 1$ degrees of freedom. Points denotes the average number of points generated and Time the average runtime in seconds. For all the above we set the error parameter $e = 0.1$.*

best theoretical result of [16] fixes a sequence of spherical Gaussians $\{f_0, \ldots, f_k\}$ with the mode being in S, parameterized by the variance. The overall complexity is $\widetilde{O}(n^3)$ MEMBERSHIP calls. The implementation in [17] is based on this algorithm but handles only convex polytopes in H-representation as it requires the facets of the polytope and an inscribed ball to fix the sequence of Gaussians. Both the radius of the inscribed ball and the number of facets strongly influence the performance of the algorithm. So, it cannot handle efficiently the case of convex bodies without a facet description, e.g., zonotopes [17], as it results a big sequence of ratios that spoil practical efficiency.

Our approach is to consider the f_i's as a sequence of indicator functions of concentric balls centered in S, as in [21]. In particular, let f_k and f_0 be the indicator functions of $r\mathcal{B}_n$ and $R\mathcal{B}_n$ respectively, while $r\mathcal{B}_n \subseteq S \subseteq R\mathcal{B}_n$ and $S_i = (2^{(k-i)/n} r\mathcal{B}_n) \cap S$ for $i = 0, \ldots, k$. Thus, it suffices to compute $\mathrm{vol}(r\mathcal{B}_n)$ and apply the following:

$$\mathrm{vol}(S) = \mathrm{vol}(S_k)\frac{\mathrm{vol}(S_{k-1})}{\mathrm{vol}(S_k)} \cdots \frac{\mathrm{vol}(S_0)}{\mathrm{vol}(S_1)}, \quad k = \lceil n \lg(R/r) \rceil. \tag{12}$$

Furthermore, we employ the annealing schedule from [12] to minimize k, without computing neither an enclosed ball $r\mathcal{B}_n$ nor an enclosing ball $R\mathcal{B}_n$ of S. We do so by probabilistically bounding each ratio of Equation (12) in an interval $[r, r + \delta]$, which is given as input. To approximate each ratio of volumes, we sample uniformly distributed points from S_i and count points in S_{i-1}. We follow the experimental results of Section 4.2 and use W-BILLARD which mixes faster than W-HNR.

Table 2 reports the average volume, runtime, number of points generated for each S-n-m over 10 trials. We also compute a 95% confidence interval for the volume. Notice that for all cases the extreme values of each interval imply an error ≤ 0.1, which was the requested error. For $n = 40$ just a few seconds suffice to approximate the volume and for $n = 100$ our implementation takes a few minutes.

4.2. Expected value of a function

Randomized algorithms are commonly used for problems in robust control analysis to overcome the (worst case) hardness, especially in probabilistic robustness [6, 29, 53, 11]. A central problem is to approximate the integral of a function over a spectrahedron, e.g. [10, 48] and thus uniform sampling is of particular interest. To put our experiments into perspective, we present experiment up to $n = 200$, while in [10] and [11] they use only W-HNR for experiments in $n \leq 10$.

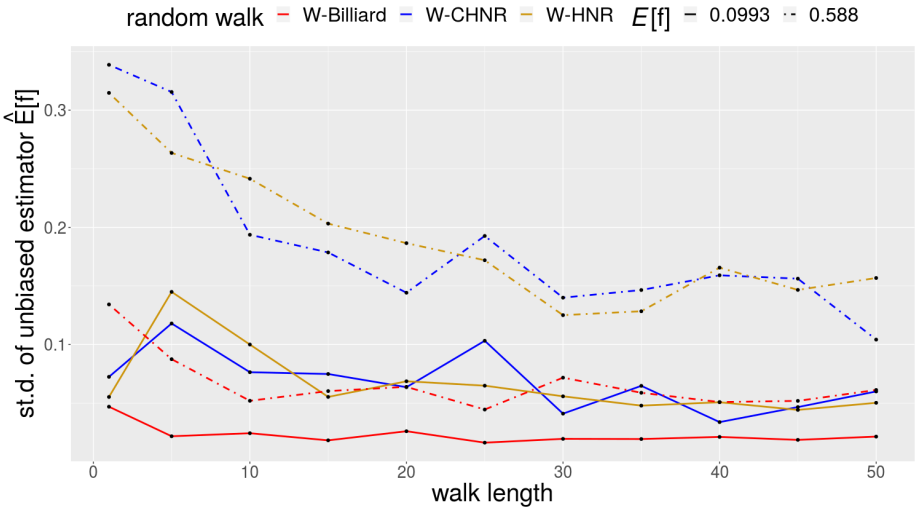

Figure 5: *The standard deviation of* $\hat{\mathbb{E}}_N[f]$ *over* $M = 20$ *trials, estimating 2 functions with* $\mathbb{E}[f_1] = 0.0993$ *and* $\mathbb{E}[f_2] = 0.588$. *For each walk length we sample* $N = 200$ *points and we repeat* $M = 20$ *times.*

Our goal is to compute the expected value of a function $f : \mathbb{R}^n \to [0, 1]$, with respect to the measure given by the uniform distribution π over S, i.e., $I = \int_S f(x)\pi(x)dx$. A standard approach is the Monte Carlo method, which suggests to sample N independent samples from π. Then,

$$\hat{\mathbb{E}}_N[f] = \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

is an unbiased estimator for I. We employ the random walks of Section 3 to sample uniformly distributed points from S (i.e., W-HnR, W-CHnR, and W-Billard) and we experimentally compare their efficiency. It turns out that W-Billard mixes much faster and results to better accuracy (see Figures 5 & 6). This observation agrees with the experiments on the rate of convergence for W-HnR and W-Billard in [45]. To come to a decisive conclusion we need to perform a more detailed practical study on the mixing time of these random walks; we leave this study as future work.

The variance of an estimator is a crucial as it bounds the approximation error. Using Chebyshev's inequality and [37], we have

$$Prob[|\hat{\mathbb{E}}_N[f] - \mathbb{E}[f]| \leq \epsilon] \leq \frac{var(\hat{\mathbb{E}}_N[f])}{\epsilon^2} \leq \frac{4M_\epsilon}{N\epsilon^2}, \tag{13}$$

where M_ϵ is the mixing time of the random walk one uses to sample "ϵ close" to the uniform distribution from S. Thus, for fixed N and ϵ, the smaller the mixing time of the random walk is, the smaller the variance of estimator $\hat{\mathbb{E}}_N[f]$ and hence, the better the approximation. We estimate I when $f := 1(S \cap H)$, where H is the union of two half-spaces $H := \{x \mid cx \leq b_1 \text{ or } cx \geq b_2\}$, where $b_2 > b_1$ and $1(\cdot)$ is the indicator function. Note that $I = \text{vol}(S \cap H)/\text{vol}(S)$. To estimate it we sample approximate uniformly distributed points from S and we count the number of points that lie in $S \cap H$.

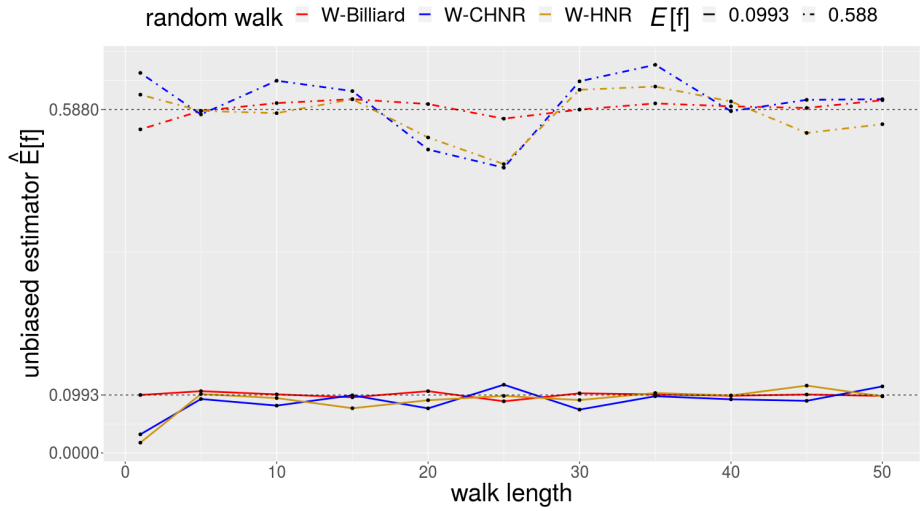

Figure 6: *The mean value of the estimator* $\hat{\mathbb{E}}_N[f]$ *over* $M = 20$ *trials, estimating two functions with* $\mathbb{E}[f_1] = 0.0993$ *and* $\mathbb{E}[f_2] = 0.588$. *For each walk length we sample* $N = 200$ *points and we repeat* $M = 20$ *times.*

walk length	1	5	10	20	30	40	50
S-100-100	1.4	3.2	7.7	9.5	16.1	21.4	28.2
S-200-200	16.2	75.7	148	303	443	584	722

Table 3: *Average time in sec to sample* 200 *points with* W-Billard *from* 10 *random spectrahedra* S*-n-m; n for the dimension that* S*-n-m lies; m for the dimension of the matrix in LMI.*

We estimate two functions f_1, f_2 with $\mathbb{E}[f_1] = 0.0993$ and $\mathbb{E}[f_2] = 0.5880$ in dimension $n = 50$ and for various walk lengths. For each walk length we sample $N = 200$ points and we repeat $M = 20$ times. Then, for each N-set we compute $\frac{1}{N}\sum_{i=1}^{N} f(x_i)$ and we take the average and the standard deviation (st.d.) over M. Figures 5, & 6 illustrates these values, while the walk length increases. Notice that the st.d. is much smaller and the approximation more stable when W-Billard is used compared to both W-HnR and W-CHnR. As W-Billard mixes faster, we report in Table 3 the average time our software needs to sample $N = 200$ points for various walk lengths for W-Billard in $n = 100,\ 200$. The average time to generate a point is ≈ 0.3 and ≈ 7.2 milliseconds respectively.

4.3. Sampling from non-uniform distributions

The random walks of Section 3 open a promising avenue for approximating the optimal solution of a semidefinite program, that is

$$\min\langle \boldsymbol{c}, \boldsymbol{x}\rangle, \text{ subject to } \boldsymbol{x} \in S. \tag{14}$$

We parameterize the optimization algorithm in [27] with the choice of random walk and demonstrate that its efficiency relies heavily on the sampling method. We perform experiments with W-HMC-r and W-HnR, as both can sample from the distribution the algorithm requires. Deterministic approximations to the optimal solutions of these tests, were acquired via the `SDPA` library [57].

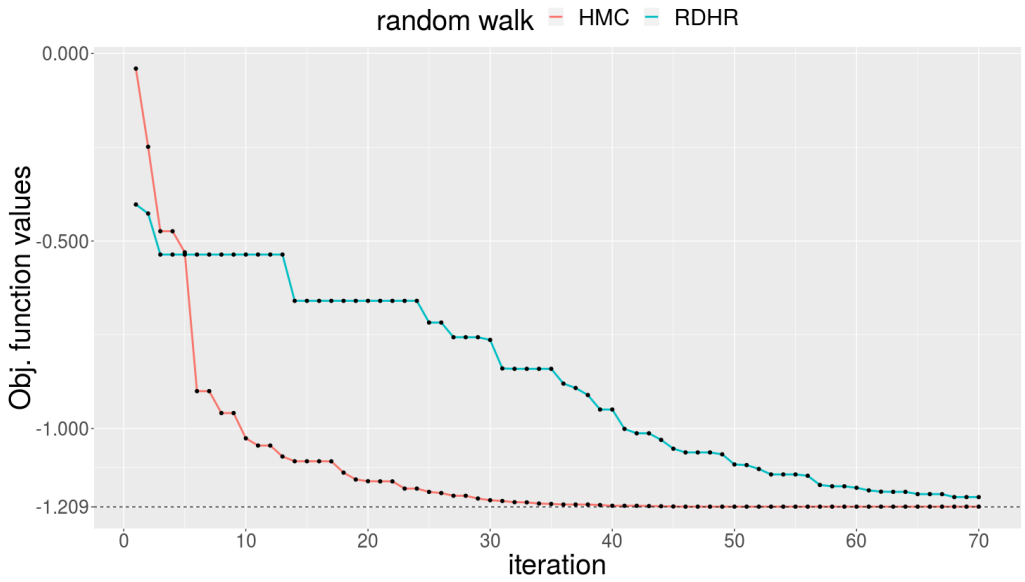

Figure 7: *Sample a point from* $\pi(x) \propto e^{-cx/T_i}$ *and update the objective current best in each iteration, with* $T_0 \approx diam(S)$ *and* $T_i = T_{i-1}(1 - 1/\sqrt{n})$, $i = 1, \ldots 70$. *The walk length equals to one for* W-HMC-R *and* $500 + 4n^2 = 10\,500$ *for* W-HNR.

The strategy to approximate the optimal solution $\boldsymbol{x}^*$ of Equation (14), is based on sampling from the Boltzmann distribution, i.e., $\pi(\boldsymbol{x}) \propto e^{-\boldsymbol{c}\boldsymbol{x}/T}$, truncated to S. The scalar T, is called *temperature*. As the temperature T diminishes, the mass of π tends to concentrate around its mode, which is $\boldsymbol{x}^*$. Thus, one could obtain a uniform point using the algorithm in [38], and then use it as a starting point to sample from $\pi_0 \propto e^{-cx/T_0}$, where $T_0 = R$ and $S \subseteq R\mathcal{B}_n$. Then, the cooling schedule $T_{i+1} = T_i(1 - 1/\sqrt{n})$ guarantees that a sample from π_i yields a good starting point for π_{i+1}. After $\widetilde{O}(\sqrt{n})$ steps the temperature will be low enough, to sample a point within distance ϵ from $\boldsymbol{x}^*$ with high probability.

In [27], they use only W-HNR. We also employ W-HMC-R. To sample from Boltzmann distributions with W-HMC-R, at each step, starting from $\boldsymbol{p_i}$ and with momenta $\boldsymbol{v_i}$, the ODE of Equation (10) becomes

$$\frac{d^2}{dt^2}\boldsymbol{p}(t) = -\frac{\boldsymbol{c}}{T}, \ \frac{d}{dt}\boldsymbol{p}(0) = \boldsymbol{v_i}, \ \boldsymbol{p}(0) = \boldsymbol{p_i}. \tag{15}$$

Its solution is the polynomial $\boldsymbol{p}(t) = -\frac{\boldsymbol{c}}{2T}t^2 + \boldsymbol{v_i}t + \boldsymbol{p_i}$, which is a parametric representation of a polynomial curve, see Equation (3).

In Table 4 we follow the cooling schedule described, after setting $T_0 \approx R$ and sampling the first uniform point with W-BILLARD. We give the optimal solution as input and we stop dropping T when an error $\epsilon \leq 0.05$ is achieved. Even in the case when the walk length is set equal to one, W-HMC-R still converges to the optimal solution. To the best of our knowledge, this is the first time that a randomized algorithm, which is based on random walks, is functional even when the walk length is set to one. On the other hand, we set the walk length of W-HNR $O(\sqrt{n})$ or $O(n)$ in our experiments. Notice that for the smaller walk length, its runtime decreases, but the method becomes unstable, as it sometimes fails to converge. For both cases its runtime is worse than that of W-HMC-R.

S-n-m	W-HMC-R	W-HNR W_1	W-HNR W_2
S-30-30	20.1 / 2.9/ 0	184.3 / 3.4 / 1	52.1 / 5.2 / 0
S-40-40	24.6 / 7.9 / 0	223.3 / 9.9 / 2	61.9 / 17.1 / 0
S-50-50	29.2 / 12.7 / 0	251.2 / 22.3 / 3	72.3 / 44.6 / 0
S-60-60	32.8 / 24.32 / 0	272.7 / 41.1 / 3	81.5 / 98.9 / 0

Table 4: *The average #iteration / runtime / failures over* 10 *generated S-n-m, to achieve relative error $\epsilon \leq 0.05$. The walk length is one for* W-HMC-R *and $W_1 = 4\sqrt{n}$ and $W_2 = 4n$ for* W-HNR. *With "failures" we count the number of times the method fails to converge. Also m is the dimension of the matrix in LMI and n is the dimension that S-n-m lies.*

Acknowledgements

ET is partially supported by ANR JCJC GALOP (ANR-17-CE40-0009), the PGMO grant ALMA, and the PHC GRAPE.

Appendix A. Additional proofs

To prove lemma 2.5 we will need the following lemmas.

Lemma Appendix A.1 (Partial Derivative of Determinant)**.** *Let $\boldsymbol{A}$ be a symmetric $m \times m$ matrix. Then*

$$\frac{\partial \det \boldsymbol{A}}{\partial A_{ij}} = c_{ij}$$

where c_{ij} the cofactor of A_{ij}.

Proof. From Laplace expansion:

$$\det \boldsymbol{A} = \sum_{j=1}^{m} A_{ij} c_{ij}$$

Notice that $c_{1j}, \cdots, c_{mj}$ are independent of A_{ij}, so we have

$$\frac{\partial \det \boldsymbol{A}}{\partial A_{ij}} = c_{ij}$$

□

Lemma Appendix A.2. *Let $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{A}_0 + x_1 \boldsymbol{A}_1 + \cdots + x_n \boldsymbol{A}_n$. Then*

$$\frac{\partial \det \boldsymbol{F}(\boldsymbol{x})}{\partial x_k} = \texttt{Trace}\left(\boldsymbol{F}(\boldsymbol{x})^* \boldsymbol{A}_k\right))$$

Proof. The function $\det \boldsymbol{F}(\boldsymbol{x})$ is the composition of $\det \boldsymbol{A}$ and $\boldsymbol{A} = \boldsymbol{F}(\boldsymbol{x})$, so from Lemma Appendix A.1 and the chain rule:

$$\frac{\partial \det \boldsymbol{F}(\boldsymbol{x})}{\partial x_k} = \sum_{i=1}^{m} \sum_{j=1}^{m} \frac{\partial \det \boldsymbol{F}}{\partial \boldsymbol{F}_{ij}} \cdot \frac{\partial \boldsymbol{F}_{ij}}{\partial x_k} = \sum_{i=1}^{m} \sum_{j=1}^{m} c_{ij} A_{ij}^k = \texttt{Trace}\left(\boldsymbol{F}(\boldsymbol{x})^* \boldsymbol{A}_k\right)$$

where A_{ij}^k the ij-th element of matrix $\boldsymbol{A}_k$ □

Lemma Appendix A.3 (Adjoint Matrix of $\boldsymbol{A}$)**.** *Let $\boldsymbol{A}$ be a $m \times m$ matrix of rank $r(\boldsymbol{A}) = m - 1$. Then*

$$\boldsymbol{A}^* = \mu(\boldsymbol{A}) \frac{\boldsymbol{v}\boldsymbol{u}^\top}{\boldsymbol{u}^\top \boldsymbol{v}}$$

where $\mu(\boldsymbol{A})$ is the product of the $m - 1$ non-zero eigenvalues of $\boldsymbol{A}$, and $\boldsymbol{x}$ and $\boldsymbol{y}$ satisfy $\boldsymbol{A}\boldsymbol{v} = \boldsymbol{A}^\top \boldsymbol{u} = \boldsymbol{0}$ (see chapter 3.2 in [40]).

Appendix B. Matrices of the Example

The spectrahedron was randomly generated as in [18]. Due to space considerations, the entries of the matrices are rounded to the first decimal.

$$\boldsymbol{A_0} = \begin{bmatrix} 16.7 & 3.7 & 12.3 & 8.7 & 5.1 & 10.4 \\ 3.7 & 9.4 & 2.3 & 4 & -2.3 & -1 \\ 12.3 & 2.3 & 26.8 & 18.7 & 7.1 & 16.7 \\ 8.7 & 4 & 18.7 & 20 & 3.7 & 12.3 \\ 5.1 & -2.3 & 7.1 & 3.7 & 6.1 & 5.4 \\ 10.4 & -1 & 16.7 & 12.3 & 5.4 & 18.7 \end{bmatrix} \tag{B.1}$$

$$\boldsymbol{A_1} = \begin{bmatrix} 0.5 & -0.4 & 2.7 & 0 & 0 & \\ -0.4 & 1.4 & -0.2 & 0 & 0 & 0 \\ 2.7 & -0.2 & 1.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & -0.4 & 2.7 \\ 0 & 0 & 0 & -0.4 & 1.4 & -0.2 \\ 0 & 0 & 0 & 2.7 & -0.2 & 1.7 \end{bmatrix} \tag{B.2}$$

$$\boldsymbol{A_2} = \begin{bmatrix} 2.6 & -0.1 & 3 & 0 & 0 & 0 \\ -0.1 & 1 & -0.1 & 0 & 0 & 0 \\ 3 & -0.1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2.6 & -0.1 & 3 \\ 0 & 0 & 0 & -0.1 & 1 & -0.1 \\ 0 & 0 & 0 & 3 & -0.1 & -1 \end{bmatrix} \tag{B.3}$$

[1] H. M. Afshar and J. Domke. Reflection, Refraction, and Hamiltonian Monte Carlo. In *Proc. 28th NeurIPS*, pages 3007–3015, Cambridge, MA, USA, 2015. MIT Press.

[2] D. Armentano and C. Beltrán. The polynomial eigenvalue problem is well conditioned for random inputs. *SIMAX*, 40(1):175–193, 2019.

[3] C. Beltrán and K. Kozhasov. The real polynomial eigenvalue problem is well conditioned on the average. *J. Foundations of Computational Math.*, pages 1–19, 2019.

[4] M. Berhanu. *The polynomial eigenvalue problem*. PhD thesis, University of Manchester, 2005.

[5] M. Betancourt. A conceptual introduction to Hamiltonian Monte Carlo. *arXiv preprint arXiv:1701.02434*, 2017.

[6] V. D. Blondel and J. N. Tsitsiklis. A survey of computational complexity results in systems and control. *Automatica*, 36(9):1249–1274, 2000.

[7] J.-D. Boissonnat and M. Teillaud. *Effective computational geometry for curves and surfaces*. Springer, 2006.

[8] P. Breiding, K. Kozhasov, and A. Lerario. Random spectrahedra. *SIAM J. Optimization*, 29(4):2608–2624, 2019.

[9] B. Beler, A. Enge, and K. Fukuda. Exact Volume Computation for Polytopes: A Practical Study. In G. Kalai and G. M. Ziegler, editors, *Polytopes Combinatorics and Computation*, DMV Seminar, pages 131–154. Birkhuser, Basel, 2000.

[10] G. Calafiore. Random walks for probabilistic robustness. In *Proc. CDC*, volume 5, pages 5316–5321. IEEE, 2004.

[11] G. Calafiore and M. Campi. Robust convex programs: Randomized solutions and applications in control. In *Proc. CDC*, volume 3, pages 2423–2428. IEEE, 2003.

[12] A. Chalkis, I. Z. Emiris, and V. Fisikopoulos. Practical volume estimation by a new annealing schedule for cooling convex bodies. *arXiv preprint arXiv:1905.05494*, 2019.

[13] A. Chalkis, V. Fisikopoulos, P. Repouskos, and E. Tsigaridas. Sampling the feasible sets of SDPs and volume approximation. *poster version in ISSAC 2020, to appear in ACM Communications in Computer Algebra*, 2020.

[14] Y. Chen, R. Dwivedi, M. J. Wainwright, and B. Yu. Vaidya walk: A sampling algorithm based on the volumetric barrier. In *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1220–1227, Oct 2017.

[15] A. Chevallier, S. Pion, and F. Cazals. Hamiltonian Monte Carlo with boundary reflections, and application to polytope volume calculations. Research Report RR-9222, INRIA, Sophia-Antipolis, France, Nov. 2018.

[16] B. Cousins and S. Vempala. Bypassing KLS: Gaussian cooling and an $O^*(n^3)$ volume algorithm. In *Proc. ACM Symposium Theory of computation (STOC)*, pages 539–548, 2015.

[17] B. Cousins and S. Vempala. A practical volume algorithm. *Mathematical Programming Computation*, 8, June 2016.

[18] F. Dabbene, P. Shcherbakov, and B. T. Polyak. A randomized cutting plane method with probabilistic geometric convergence. *SIAM J. Optimization*, 20:3185–3207, 2010.

[19] J.-P. Dedieu and F. Tisseur. Perturbation theory for homogeneous polynomial eigenvalue problems. *Linear algebra & Appl.*, 358(1-3):71–94, 2003.

[20] M. Dyer, A. Frieze, and R. Kannan. A random polynomial-time algorithm for approximating the volume of convex bodies. *J. ACM*, 38(1):1–17, 1991.

[21] I. Emiris and V. Fisikopoulos. Practical polytope volume approximation. *ACM Trans. Math. Soft.*, 44(4):38:1–38:21, 2018. Prelim. version: Proc. SoCG, 2014.

[22] I. Emiris, F. Sottile, and T. Theobald, editors. *Nonlinear computational geometry*, volume 151 of *I.M.A. volumes in Math & its Applications*. Springer, Berlin, Oct. 2010.

[23] I. Z. Emiris, B. Mourrain, and E. Tsigaridas. Separation bounds for polynomial systems. *J. Symbolic Computation*, 2019.

[24] E. Gryazina and B. Polyak. Random sampling: Billiard walk algorithm. *European J. Operational Research*, 238, 11 2012.

[25] G. Guennebaud, B. Jacob, et al. Eigen v3, 2010. `http://eigen.tuxfamily.org`.

[26] D. Henrion, J. B. Lasserre, and C. Savorgnan. Approximate volume and integration for basic semialgebraic sets. *SIAM review*, 51(4):722–743, 2009.

[27] A. T. Kalai and S. Vempala. Simulated annealing for convex optimization. *Math. Oper. Res.*, 31(2):253–266, Feb. 2006.

[28] E. Kaltofen and G. Villard. On the complexity of computing determinants. *Computational complexity*, 13(3-4):91–130, 2005.

[29] P. Khargonekar and A. Tikku. Randomized algorithms for robust control analysis and synthesis have polynomial complexity. In *Proceedings of 35th IEEE Conference on Decision and Control*, volume 3, pages 3470–3475, Dec 1996.

[30] M. Korda and D. Henrion. Convergence rates of moment-sum-of-squares hierarchies for volume approximation of semialgebraic sets. *Optimization Letters*, 12(3):435–442, 2018.

[31] P. Lairez, M. Mezzarobba, and M. Safey El Din. Computing the volume of compact semi-algebraic sets. In *Proc. ISSAC*. ACM Press, 2019.

[32] S. Lan, B. Zhou, and B. Shahbaba. Spherical Hamiltonian Monte Carlo for Constrained Target Distributions. *JMLR workshop and Conf procs.*, 32:629–637, 2014.

[33] G. Lecerf. On the complexity of the Lickteig–Roy subresultant algorithm. *J. Symbolic Computation*, 92:243–268, May 2019.

[34] Y. T. Lee, Z. Song, and S. S. Vempala. Algorithmic theory of odes and sampling from well-conditioned logconcave densities, 2018.

[35] Y. T. Lee and S. Vempala. Convergence Rate of Riemannian Hamiltonian Monte Carlo and Faster Polytope Volume Computation. In *Proc. ACM Symp. Theory of Computation (STOC)*, 2018.

[36] T. Lickteig and M.-F. Roy. Sylvester–habicht sequences and fast cauchy index computation. *J. Symbolic Computation*, 31(3):315–341, Mar. 2001.

[37] L. Lovász, R. Kannan, and M. Simonovits. Random walks and an $O^*(n^5)$ volume algorithm for convex bodies. *Random Structures & Algorithms*, 11:1–50, 1997.

[38] L. Lovász and S. Vempala. Simulated annealing in convex bodies and an $O^*(n^4)$ volume algorithm. In *Proc. IEEE Symp. Foundation of Comp. Science (FOCS)*, volume 2003, pages 650–659, 2003.

[39] L. Lovasz and S. Vempala. Fast algorithms for logconcave functions: Sampling, rounding, integration and optimization. In *Proc. IEEE Symp. Foundation of Comp. Science (FOCS)*, pages 57–68, 2006.

[40] J. Magnus and H. Neudecker. *Matrix Differential Calculus with Applications in Statistics and Econometrics (Revised Edition)*. John Wiley & Sons Ltd, 1999.

[41] O. Mangoubi and N. K. Vishnoi. Faster polytope rounding, sampling, and volume computation via a sub-linear ball walk. In *Proc. IEEE Symp. Foundation of Comp. Science (FOCS)*, pages 1338–1357, 2019.

[42] A. Pakman and L. Paninski. Exact Hamiltonian Monte Carlo for Truncated Multivariate Gaussians. *J. Computational & Graphical Statistics*, 23(2):518–542, Apr. 2014.

[43] V. Y. Pan. Univariate polynomials: nearly optimal algorithms for numerical factorization and root-finding. *J. Symbolic Computation*, 33(5):701–733, 2002.

[44] V. Y. Pan and E. P. Tsigaridas. Nearly optimal refinement of real roots of a univariate polynomial. *J. Symbolic Computation*, 74:181–204, 2016.

[45] B. Polyak and E. Gryazina. Billiard walk - a new sampling algorithm for control and optimization. *IFAC Proceedings Volumes*, 47(3):6123–6128, 2014.

[46] B. Polyak and P. Shcherbakov. The D-decomposition technique for linear matrix inequalities. *Automation and Remote Control*, 67:1847–1861, 11 2006.

[47] M. Ramana and A. Goldman. Some geometric results in semidefinite programming. *J. Global Optimization*, 7, 02 1999.

[48] L. R. Ray and R. F. Stengel. A monte carlo approach to the analysis of control system robustness. *Automatica*, 29(1):229–236, 1993.

[49] A. Schönhage. The fundamental theorem of algebra in terms of computational complexity. *Manuscript. Univ. of Tübingen, Germany*, 1982.

[50] R. L. Smith. Efficient monte carlo procedures for generating points uniformly distributed over bounded regions. *Operations Research*, 32(6):1296–1308, 1984.

[51] Spectra. A header-only C++ library for large scale eigenvalue problems. `https://spectralib.org`, 2020. v0.9.0.

[52] S. Tabachnikov. *Geometry and billiards*. Student mathematical library. American Mathematical Society, Providence, RI, 2005.

[53] R. Tempo, G. Calafiore, and F. Dabbene. *Randomized algorithms for analysis and control of uncertain systems: with applications*. Springer, 2012.

[54] F. Tisseur. Backward error and condition of polynomial eigenvalue problems. *Linear Algebra & Appl.*, 309(1):339–361, 2000.

[55] L. N. Trefethen and D. Bau. *Numerical linear algebra.* SIAM, 1997.

[56] S. Vempala. Geometric random walks: A survey. *Combinatorial & Computational Geometry*, 52, 01 2005.

[57] M. Yamashita, K. Fujisawa, and M. Kojima. Implementation and evaluation of sdpa 6.0. *Optimization Methods & Software*, 18(4):491–505, 2003.