Google Summer of Code 2019

CODING PROJECT PROPOSAL

State-of-the-art geometric random walks in R

Name: Apostolos Chalkis
Affiliation: Department of Informatics & Telecommunications
National & Kapodistrian University of Athens, Greece
Program: PhD student
Second participation in GSoC
Mentors: Vissarion Fisikopoulos, Elias Tsigaridas and Zafeirakis Zafeirakopoulos
Email: tolis.chal@gmail.com
Github: https://github.com/TolisChal
IRC nick: Chalkis8128
Address: Athens, Gyzi, Ragkavi 57-59, 11474
Phone: +30 693 883 5919, +30 694 023 0078

Contents

1. Synopsis

Sampling algorithms and volume computation of convex polytopes are very useful in many scientific fields and applications. The package `VolEsti` at [15] is a `C++` software with an `R` interface which provides three geometric random walks for sampling and two state-of-the-art methods for volume estimation for convex polytopes being the first package providing such a variety of options in geometric statistics. `VolEsti` currently scales to a few hundreds dimensions in contrast to other available `R` packages, as `geometry`, or `C++` package `VINCI` that both only scale typically up to 15-20 dimensions. Thus it could be an essential tool for a quite large number of scientific applications in convex analysis, economics, biology or statistics, that need fast volume approximation or sampling in high dimensions. However, the possibility to scale from a few hundred to a few thousand dimensions was considered as a very far-reaching goal for many years. The goal of this project is to provide the first ever implementations for sampling and volume computation for convex polytopes in a few thousand dimensions. We exploit some very recent theoretical results that guarantee fast convergence and numerical stability in order to propose an efficient implementation of the current state-of-the-art geometrical random walks, i.e. Hamiltonian Monte Carlo and Vaidya walk. The proposed implementations will be a decisive contribution to other scientific fields as computational geometry, finance and optimization (see section 3). In section 4 we give a week time schedule for the coding project. We hope this project will be a decisive contribution towards the first complete and efficient tool for sampling, volume estimation and geometrical statistics in general and thus, help educational programs, research or even serve as a building block towards an international, interdisciplinary community in geometrical statistics.

2. The Project

The goal of this project is to provide efficient implementations for various geometric random walk algorithms to sample points from convex polytopes and volume computation in high dimensions. In particular, I propose the implementation of a) two random walk algorithms that belong to the family of Hamiltonian Monte Carlo (HMC), b) a new method for volume estimation of convex polytopes, and c) the addition of two more random walks by modifying the `C++` implementation of [22]. HMC has been proven to be a remarkable empirical success. However, only recently we became aware of a rigorous understanding of why it performs so well on difficult problems in practice and how it is best applied. HMC is the current state-of-the-art random walk for convex polytopes and the corresponding implementations I propose, would make possible to develop, for the first time, the theoretically fastest randomized algorithm for volume estimation [16]. Moreover in [9] they show that Vaidya and John walk algorithms are faster than HMC methods for specific convex polytopes. Hence the implementation of these methods will result the most complete package for sampling and volume estimation in a few thousand dimensions for the first time!

During my first participation at Google Summer Summer of Code 2018, last year, I also worked on `VolEsti` package[1], becoming one of the authors. I extended `VolEsti` with new efficient sampling and volume estimation methods and created a `CRAN` version of `VolEsti` to introduce easy accessible geometric random walk algorithms and computations to scientific and business communities.

The main part of the implementation will be in `C++` given as a natural extension of the current software of `VolEsti` and will be easily accessible through the `Rcpp` wrapping. I am going to use some of the geometrical concepts that are already implemented, e.g., points and convex polytopes. The package in [15] is currently under submission to `CRAN` repository so we expect until the end of GSoC 2019 to have a `CRAN` version of the package and, moreover, the proposed implementations would be the first updated version of the package.

[1] https://tolischal.github.io/GSoC2018/

Proposal structure. The rest of the section presents the structure of the package, an overview of the methods that I will implemented and all the technical details and the methodology of the implementation. The section 3 presents the benefits for the scientific and business communities. The section 4 presents a detailed time schedule starting from the bonding period until the end of GSoC 2019. The section 5 presents some related work I have done for the proposed project. The section 6 presents the implemented solutions of the tree tests that are requested from the mentors. The section 7 contains biographical information and personal motivation.

2.1 Understanding the code structure and design of VolEsti.

The `R` package `VolEsti` combines the efficiency of a `C++` implementation and the friendly interface of `R`. The package consists of more than 3 000 lines of `C++` code. To call `C++` code there is one `Rcpp` function for each procedure (i.e., sampling, volume estimation etc.) and it is exported as an `R` function. To export `C++` classes that represent convex polytopes we use `Rcpp` modules [13] to avoid useless `R` code and to maintain the code easily.

Name	Input	Description	Parameterizations
`sample_points()`	A convex Polytope	Samples from the input polytope using random walks or perfect uniform sampling from some specific bodies	i) Random walk method ii) Walk step iii) Target distribution iv) Parameters for the exact uniform sampling
`volume()`	A convex Polytope	Estimates the volume of a convex polytope	i) Random walk method ii) Walk step iii) Requested error iv) Other parameters for the two methods
`rounding()`	A convex Polytope	Brings the polytope to an approximated well-rounded position	i) Random walk method ii) Walk step
`rotating()`	A convex Polytope	Rotates randomly the input polytope	--

Table 1: Overview of the most important functions of package `VolEsti`.

We use continuous integration practice to maintain and test the `C++` part of `VolEsti` [2]. Additionally, there is a test suite for the functions of the `R` package. The package provides 8 functions, 4 exposed classes for convex polytopes and 8 polytope generators. Table 1 presents the most important functions of the package for our project. `VolEsti` provides 4 different representations for a convex polytope with a corresponding exposed `C++` class for each representation. The HMC walk family and the implemented random walks in [22] can be applied only for the H-representation, i.e., when the polytope is given by a set of linear inequalities.

The package provides the following three random walk methods for sampling from the input polytope: a) Coordinate Directions Hit-and-Run (CDHR), b) Random Directions Hit-ad-Run (RDHR), and c) Ball walk. Moreover the user could choose between uniform and multidimensional spherical Gaussian target distribution. The CDHR it is shown to be the most efficient random walk in practice for many applications among many random walk algorithms. However, it does not scale beyond a few hundred dimensions, regardless the application. The plots in Figure 1 demonstrate two samples, one for each target distribution that `VolEsti` provides. Both methods (`SeqOfBalls` [14] and `CoolingGaussian`[12]) for volume estimation are based on sampling using a random walk while CDHR is the default choice. `SeqOfBalls` uses uniform sampling and `CoolingGaussian` uses Gaussian, while the first is statistically more accurate and `CoolingGaussian` is faster.

[2] https://circleci.com/gh/GeomScale/volume_approximation

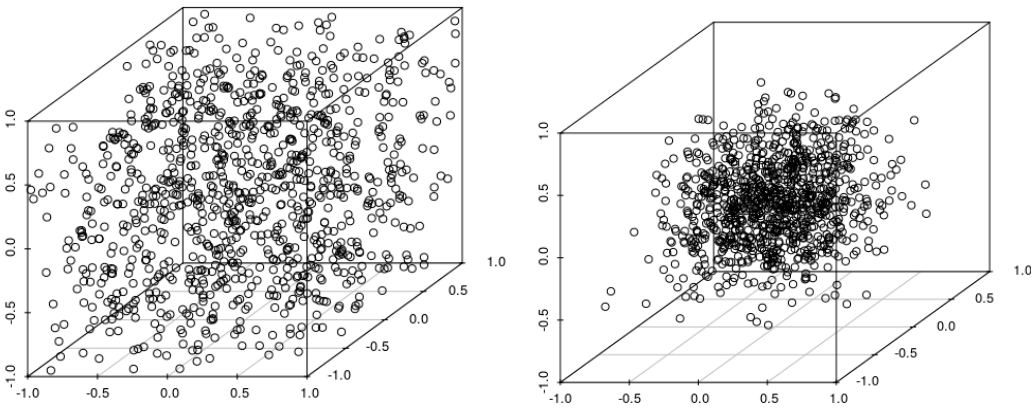

Figure 1: Sampling from 3D unit cube using `VolEsti`. Uniform target distribution (Left) and spherical Gaussian with $\sigma^2 = 0.1$ centered at the origin (Right).

The main technical characteristics of the implementations of `VolEsti` that will be useful to our project are as follows:

- There is no dependency on any external library for the geometrical objects that appear in every method. The random walk algorithms require a variety of random number generators for which `R` package `BH` is used, which provide `R` with the most useful subset of `Boost` libraries among `CRAN`. The `R` package `RcppEigen` is used for the matrices which represent polytopes and the numerical linear algebra computations. The package `RcppEigen` gives to `R` access to the `C++` library `Eigen`, which is an open source header-only library for numerical analysis and linear algebra computations.

- The computation of an H-polytope's largest inscribed ball reduces to a linear program. Moreover sampling from V-polytopes or zonotopes require solving linear programs as well. All these linear programs are solved by the `lpSolve` package [1]. The rounding of the polytope requires the computation of the smallest enclosing ellispoid of a point cloud which is computed by an implementation of Khachiyan's algorithm from package `BNMin1` [18].

- In folder `./include/convex_bodies` the header file `polytopes.h` contains `C++` classes for all the possible representations of a convex polytope. The class `Hpolytope` represents an H-polytope and it is the `C++` code that we are going to use and modify for our project as all the random walk algorithms we propose can be applied only for H-polytopes.

- The folder `./include/samplers` contains the header files with the implementations of the random walks for both uniform and Gaussian target distribution. There are different implementations for all the random walks depending the target distribution: file `samplers.h` contains implementations for uniform sampling and `gaussian_samplers.h` for Gaussian sampling. Moreover, there are implementations for uniform sampling from the surface or the interior of a hypersphere and from the interior of an arbitrary simplex.

2.2 Equip R with the state-of-the-art geometric random walks

Table 2 presents the methods I propose for implementation. For the last two random walks there is an open `C++` implementation in [22], so during the project I will make the necessary adjustments to add them to `VolEsti`.

HMC walk family. Sampling from convex bodies in high dimensions [21] is a classical problem in statistics and computational geometry. Here we give an intuition behind HMC success in difficult problems and the challenges towards an efficient practical implementation of the method.

Given a convex polytope P and a target distribution with probability density function $\pi(x)$, the problem that motivates us is to compute the expectation of a function f, say $\mathbb{E}_\pi[f]$, which

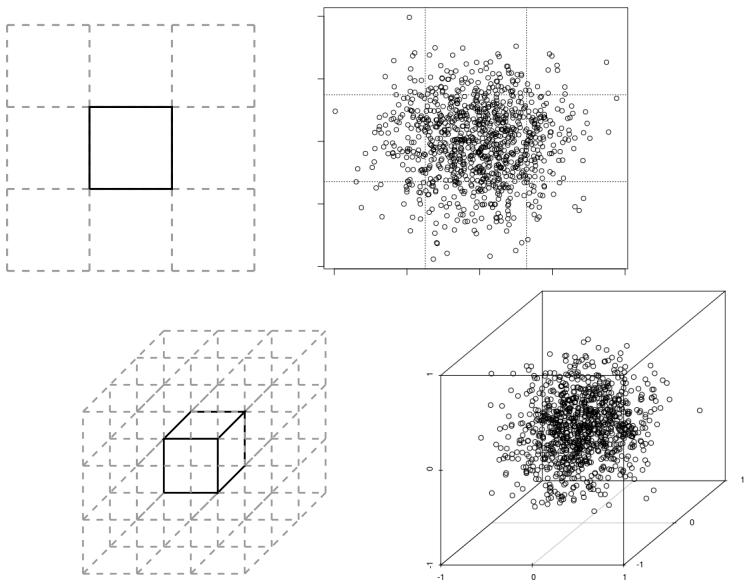

Figure 2: Consider the spherical Gaussian centered at the origin with $\sigma^2 = 0.1$ and a sample of size 1000. The mode of the PDF in two dimensions it is 1/9 and in three dimensions it is only 1/27 of the volume of the cube. In two dimensions 509 points lie in the mode, in three dimensions 326 points lie in the mode. The samples generated with CDHR with a large walk step equal to 100.

Method	Type	References
Riemannian HMC	random walk	Y.T. Lee, S. Vempala [16]
HMC with reflections	random walk	A. Chevallier, S. Pion, F. Cazals, F. [10]
`CoolingGibbs`	volume estimation	Y.T. Lee, S. Vempala [16]
Vaidya walk	random walk	Y. Chen, R. Dwivedi M.J. Wainwright B. Yu [9]
John walk	random walk	Y. Chen, R. Dwivedi M.J. Wainwright B. Yu [9]

Table 2: The proposed for implementation methods during GSoC 2019.

reduces to the integral,

$$\mathbb{E}_\pi[f] = \int_P f(x)\pi(x)dx \tag{1}$$

Of course we cannot compute the exact value of (1). So the goal is to approximate it by drawing samples $X_1, \ldots, X_k$ from distribution $\pi(x)$ in P. Then, as k goes to infinity

$$\int_P f(x)\pi(x)dx \approx \frac{1}{k}\sum_i f(X_i)$$

So the problem reduces to sample from P following a given target distribution $\pi(x)$. In high dimensions a probability density, $\pi(x)$, will concentrate around its mode (local maximum of $\pi(x)$). Thus an idea to get a good estimation of the expectation (1) would be to sample around the mode to get points with dominant contribution. Unfortunately this approach is useless as

Figure 3: We can interpret the mode of the target density as a massive planet and the gradient of the target density as that planet' s gravitational field. The typical set becomes the space around the planet through which we want a satellite to orbit [3].

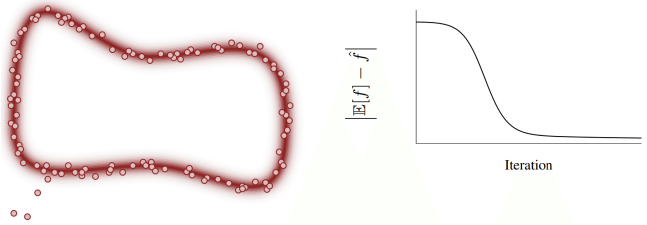

Figure 4: HMC walks towards the typical set and when it locates it walks on this set (Left). Usually the error of 1 reduces significantly when HMC starts the walk on the typical set (Right) [3].

contributions to the expectation are determined by the product of density and volume, $\pi(x)dx$, in addition the volume dx over which we integrate the density $\pi(x)$ is much larger away from the mode. The plots in Figure 2 show this phenomenon for the unit cube in two and tree dimensions. If we sample 1 000 points from the same distribution, as in Figure 2, in the 20 dimensional unit cube only 2 points would lie in the mode[3]! Actually the points with dominant contribution to (1) concentrate in a neighborhood called the typical set.

The idea of HMC is to locate the typical set and to draw samples from it to obtain a good estimation of (1) very fast (see Figure 4). So starting from a point p_0 in the interior of P we move towards the typical set and when we locate it we start walking (defining a Markov Chain) on it. To move from a point q_i to the next q_{i+1} we need a direction. The choice of the derivative of $\pi(x)$ would be wrong as it points directly towards its mode. So HMC defines trajectories that guide the walk inside the typical set. In Figure 3 you can see a physical analogue of the task of HMC. The problem of computing a trajectory is similar to that of introducing exactly the right amount of momentum to a satellite in order to be captured on a stable orbit. HMC is inspired from the Hamiltonian dynamics and uses the Hamilton's equations that is

$$\begin{aligned} \frac{dq}{dt} &= \frac{dH}{dp} = \frac{dK}{dp} \\ \frac{dp}{dt} &= -\frac{dH}{dq} = -\frac{dK}{dq} - \frac{dV}{dq} \end{aligned} \tag{2}$$

The system of Ordinary Differential Equations (ODE) in (2) has to be solved arithmetically to compute a trajectory in every step of HMC. That was a challenging problem for many years due to numerical instability issues for the most of the versions of HMC walks. The first HMC walk we are going to implement is called Riemannian HMC [16] and is the fastest random walk algorithm in theory. To solve (2) we are going to use the new result from [17] which provides a robust and very fast method (collocation method) to solve it.

Another new version of HMC walk, that we will implement as well, is given in [10]. In that paper they define reflections of the trajectories on the boundary of the polytope and introduce a new walk on the reflected trajectories, see Figure 6. Moreover, they solve a different system

[3]The result of a single run, as well as in Figure 2

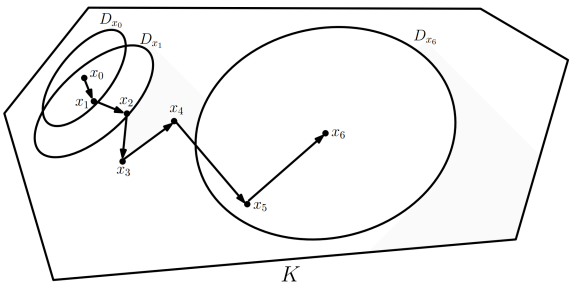

Figure 5: A realization of the main idea of the ellipsoidal family of random walks. The D_i is the ellipsoids that these methods construct in every step, depending the current point of the walk [9].

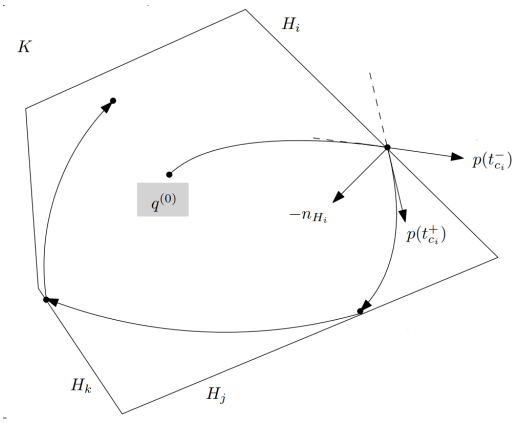

Figure 6: A realization of HMC with reflection in a 2-dimensional polytope [10].

than (2) and they give a closed formula for the solution which provides robust computations. However, boundary reflections require interval arithmetics to guarantee that the walk stays in the polytope, when a point of the random walk is close to the boundary. In particular, a point is represented by a small cube so that the number of total operations doubled per step of the random walk. All the details for the implementation are given in [10].

The algorithm for volume estimation of convex polytopes in [16] is similar to `CoolingGaussian` method that is implemented in `VolEsti`, but it uses a sequence of Gibbs distributions instead of spherical Gaussians and HMC to generate samples from each distribution.

Ellipsoidal walk family. This family of random walks is based on the interior point methods for solving convex optimization problems. I am going to add two of them in `VolEsti`: Vaidya walk and John walk. Both of them are similar to Dikin walk which I implemented as preliminary work (see section 6). In [9] they show that Vaidya walk is faster than classical HMC for some families of convex polytopes. The methods of this family define a Markov Chain using inscribed ellipsoids in the convex polytope they sample from.

At the i-th step being at point q_i these methods follow the steps below:

1. Create an ellipsoid centered at q_i and generate a random point y in that ellipsoid.
2. Create a second ellipsoid, E_y, centered at y.
3. Move to y with some probability if q_i lies in E_y.

The Figure 5 shows a simple example in a two dimensional polytope for the Dikin walk. The main difference in the other two methods is the construction of the ellipsoid in each step.

2.3 Implementation

I now give all the details about the implementations and how I plan to update `VolEsti`'s software. In the time schedule in section 4 I propose to spend the 12th week for some general code optimizations. All the details for these optimizations are in section 4.

`C++` coding

- Both HMC methods use the log barrier function of the convex polytope they sample from. In order to solve (2) I need to evaluate this function many times in order to estimate the solution. Thus I am going to add a member function `eval_log_barrier()` in the `C++` class `Hpolytope` in the header file `polytopes.h`. This function would take as input a point in the polytope and return the value of the log barrier function and will be useful for the ellipsoidal family as well.

- In the header file `samplers.h` I am going to create the function `collocation_solver()` in order to implement the collocation method to solve (2). In [17] the steps and the analysis of the method are given with all the necessary details for our implementation.

- In the header file `samplers.h` I am going to create the function `refl_traj_solver()` to implement the solution of the system of ODEs introduced in the HMC with reflections which is given in a closed formula [10].

- In the header file `samplers.h` I am going to create the functions: `Riem_HMC()` and `Refl_HMC()` to implement the random walks of HMC family accordingly. These functions would take as input the current point of the walk and the polytope and they will return the next point of the walk. Both of them will use the `eval_log_barrier()` and the corresponding methods (implemented in previous steps) in order to solve (2).

- In the folder `./include/annealing` I am going to create a new header file `gibbs_annealing.h`. In this header file I will implement all the functions needed for the annealing schedule of Gibbs distribution, as follows,

    1. The function `get_first_gibbs()` that computes the first Gibbs distribution of the sequence.
    2. The function `get_next_gibbs()` that takes as input a Gibbs distribution and computes the next one.
    3. The function `get_gibbs_sequence()` that takes as input the polytope and calls the two previous functions to compute the sequence of Gibbs distributions.

- In the header file `volume.h` I am going to create the function `cooling_gibbs()` to implement the method for volume estimation in [16]. For this implementation I will follow the structure of function `volume_gaussian_annealing()`. I am going to use function `get_gibbs_sequence()` to compute the sequence of Gibbs distributions and then I am going to estimate the ratio between the PDFs for all the successive pairs in the sequence using functions `Riem_HMC()` or `Refl_HMC()` to sample from Riemannian HMC or HMC with reflections respectively. The choice of the random walk would be an option of the user.

- I am going to make adjustments to the code in [22] in order to add implementations of Vaidya and John walk to `VolEsti`. That implementations are based on `Eigen` library which is used in `VolEsti` as well, which is quite convinient. In [22] they have implemented a superclass `walker` and all the samplers based in each random walk are implemented as subclasses of `walker` superclass. I will use that code in order to implement the following steps, so that we will be consistent with `VolEsti` structure,

    1. Create functions `get_vaidya_ellipsoid()`, `get_john_ellipsoid()` in the header file `polytopes.h` in order to create the corresponding ellipsoids of Vaidya and John walk, centered at a point inside the polytope.
    2. In the header file `samplers.h` create functions `vaidya_walk()` and `john_walk()` which would take as input a polytope and an interior point and will return the next point of the random walk.

`Rcpp` wrapping

- I am going to modify the `Rcpp` function `sample_points()` in oder to provide `R` with the new sampling methods implemented in previous steps. The input string `WalkType` of function `sample_points()` can be used by the user to choose a random walk. Thus, in order to expose in `R` the new `C++` functions, I will implement the following steps:

    1. I will let the input string `WalkType` to take values that would declare the new random walks (e.g. `"RiemHMC", "Vaidya"`).

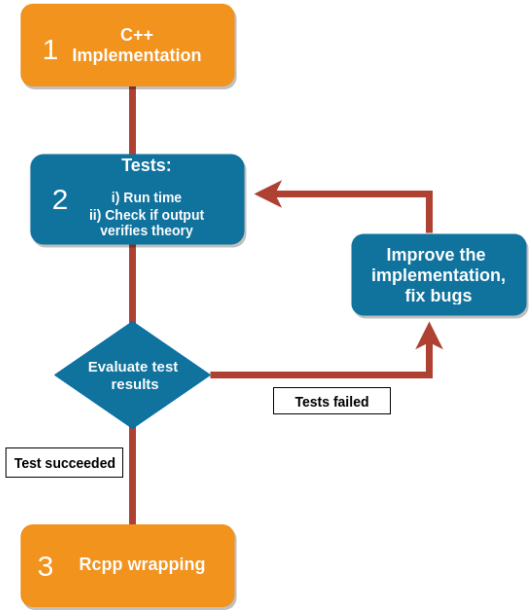

Figure 7: The steps I will follow for the implementation of each method.

2. I will add checks for the new possible values of `WalkType` to call the new sampling functions. If the given polytope is not in H-representation then an error exception would be thrown.
3. For Riemannian HMC and HMC with reflections I will add an extra check: if, in addition, uniform sampling is requested set the parameters of the target distribution accordingly.

- I am going to modify the `Rcpp` function `volume()` in oder to provide `R` with the new volume estimation method implemented in previous steps. The input string `Algo` can be used to declare the method to use for the volume estimation of the given convex polytope. In particular, I am going to implement the following steps:

  1. I will let the input string `Algo` to take values that would declare the new method (e.g. `"cgibbs"`).
  2. I will add checks for the new possible values of `Algo` in order to call the new method. If the given polytope is not in H-representation then an error exception would be thrown.

2.4 Methodology

The implementations discussed in the previous subsection will have to be tested very carefully before I add them to the `R` package `VolEsti`. So every implementation will have to pass successfully some tests before I continue with the next one. In particular, I will follow the steps I describe in the sequel in order to complete the implementations and to guarantee efficiency and stability (see also Figure 7).

1. Implement the method in `C++` following the corresponding papers and the steps we described in subsection 2.3.
2. Create tests for the run time and the correctness of the output. Check if both of them verify the theoretical results.
3. If tests failed then improve the implementations and fix bugs until tests succeed.
4. If tests succeed then implement the `Rcpp` wrappers and modify the `Rcpp` functions of `VolEsti`.

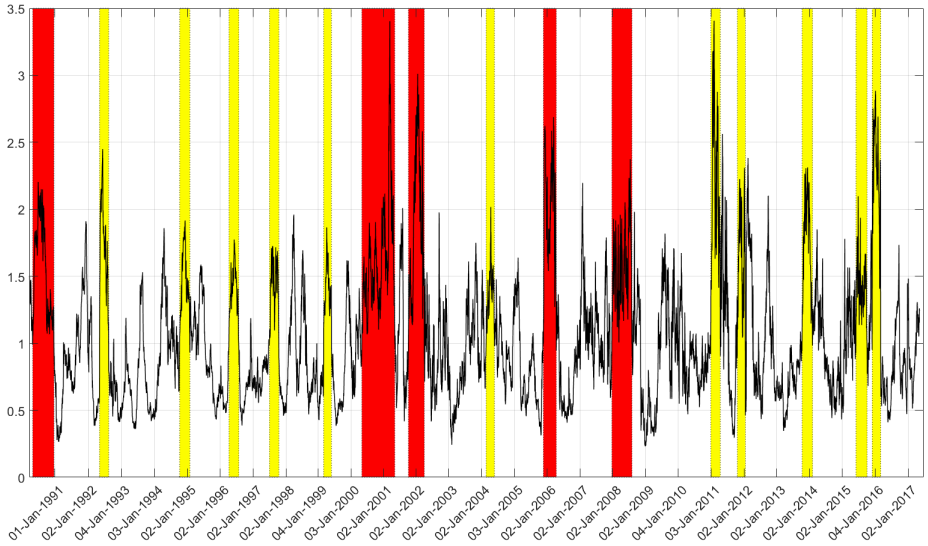

Figure 8: Representation of the periods over which the Crises Warning indicator is greater than one for 61-100 days (yellow) and over 100 days (red). The computation of the indicator is based on convex and non-convex bodies volume approximations. More details can be found in [4].

Tests for random walk algorithms. I am going to test the mixing time and the quality of the samples. For each random walk algorithm I am going to start with walk step equals to one and increase it gradually. For each walk step I will create plots of the sampled points from well known 2 and 3 dimensional convex polytopes, e.g. simplices, cubes. Moreover I could sample from high dimensional bodies and plot the projections in 2 or 3 dimensions. To test the run times I am going to keep the walk step constant and increase the dimension for well known convex polytopes. Then I will check if the run times verify the theoretical time complexity.

Tests for the volume estimation method. I am going to test accuracy and run time of the method for well known convex polytopes. I will use the `VolEsti` polytope generators and the exact computations that `VolEsti` provides to check the accuracy and if the run time verifies the theoretical time complexity as the dimension increases.

3. Benefits to the Community

The project proposal will significantly update `VolEsti` package to the first all-in-one package for sampling and volume estimation including all the state-of-the-art methods for both problems. Hence it is very important for researchers and users working on applications like statistics, optimization and economics that need sampling or volume computations in high dimensions. Moreover our project's implementations will give the potential for some very interesting future extensions. We hope this project will be a decisive contribution for the first complete tool for geometrical statistics which could be used in many scientific and business applications.

In finance they need to compute the volume of the intersection of a simplex with a set of halfspaces or ellipsoids [20] in order to predict financial crisis. From these computations we can directly derive 2-dimensional copulas, which is an estimation of the joint distribution between returns and volatility of the assets' returns in a financial market. Both of them are computed using 100 assets. In Figure 8 you can see a Crises Warning indicator (which is built using copulas) that successfully detects past crises. The new random walks and the new method for volume estimation would make possible to handle financial markets with a lot of hundreds of assets for the first time. Thus, we could more accurately predict financial crisis or efficiently optimize portfolios' returns.

Efficient sampling in practice from convex bodies can directly result to efficient optimization

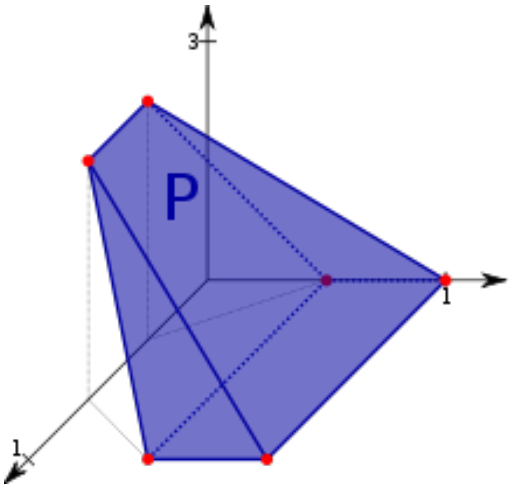

Figure 9: The non-redundant detection is used to determine the feasible region of linear optimization problems.

algorithms [2] with convex constraints (e.g. linear programming). The updated `VolEsti` will provide the most efficient random walks implemented in `C++` with an `R` interface. So it will be possible to implement new competitive optimization randomized methods specialized according the optimization problem. Moreover we would have for the first time the opportunity to compare all the state-of-the-art theoretical results in practice and to build new practical statistical tests such as uniformity tests or tests to compare samples.

Moreover the boundary sampling I have implemented as a preliminary work is connected to the non-redundant constraint detection problem which consists to detect the useless linear constraints when a set of such constraints is given. In Figure 9 we give a visual representation of non-redundant constraint detection feature. That problem arises as a subproblem in many applications and is considered as a very basic problem, in numerical linear algebra problems, in LP optimization problems, in business, industry or scheduling [19], [11] and in statistics. Some times the volume computation requires this step or becomes faster when we apply non-redundant detection.

The project is expected to be beneficial to both (a) educational and (b) research open source communities. For (a) university students studying computational geometry, statistics or convex analysis will have access to an efficient library that will help them build intuition as well as it can serve as a platform for student projects. About (b) researchers could use the library in their research projects and also extend it according to their needs. Ultimately, I hope that this project will help building an international and interdisciplinary community in the intersection of the scientific fields of computational convex and discrete geometry, statistics, optimization, economics and other scientific applications.

4. Deliverables

The total number of weeks is thirteen. Thus I give an exact time schedule for the coding project starting at 6th May. At the end of the time schedule I will have completed all the implementations described in section 2. I divide the schedule to bonding and coding periods. In bonding period we are going to prepare all the technical details and environments in order to be ready to begin the coding right away at 27th May.

However, `VolEsti` provides us a perfect basis for our implementation. Moreover, considering that I have worked a lot on this package in the near past, I will need just a few days to set up all the necessary coding environments. Thus, ideally I could start coding before bonding period is over.

4.1 Bonding period (6th May - 26th May)

We will have three weeks to prepare everything for the coding period. I give a week time schedule in order to start coding at 27th May in the worst case.

- **1st week** (6th May - 11th May)
  1. I have to create a branch of the the current `VolEsti` project at [15].
  2. I am going to create a blog in order to report our progress in a weekly basis, as I did for last year's GSoC project [4].
  3. Work with mentors on studding additional papers on geometric random walks in order to optimize the coding plan.

- **2nd week** (12th May - 18th May)
  1. Work with mentors in the current code structure of `VolEsti` in order to set all the necessary environments and minor implementations. In particular, I will replace the `std::vector` with `Eigen` vector in `./include/cartesian_geom/point.h` for the representation of points and modify the rest of the implementations in the package accordingly. This improvement will help us to apply linear transformation to samples more easily and will result to a more consistent and efficient implementation.

- **3rd week** (19th May - 26th May)
  1. Probably start coding. (Of course I do not count this week as a coding week, but ideally I will be ready to start).

4.2 Coding period (27th May - 26th August)

- **1st & 2nd week.** *Implement Riemannian Hamiltonian Monte Carlo.* (27th May - 8th June)

  I am going to implement the steps I described in subsections 2.3 and 2.4.

- **3rd & 4th week.** *Implement Hamiltonian Monte Carlo with reflections.* (9th - 22nd June)

  I am going to implement the steps I described in subsections 2.3 and 2.4.

- **5th week.** *Documentation, tests, code development for HMC walks.* (23rd - 29th June)
  1. Write `roxygen` comments in order to create the corresponding `.Rd` files with `roxygen2` library for the documentation.
  2. Add comments to both HMC walks `C++` code.
  3. Modify `Rcpp` function `sample_points()` such that both random walks can be called by a user as we describe in subsection 2.3.
  4. Write `R` tests in the script `./R-proj/test/testthat/test_sampling.R` for both HMC random walks.
  5. Run experiments to compare runtimes between the two new random walks.

  Comment: The implementations of this week might be merged into the two previous weeks depending the random walk we implement. I plan to spend 5 weeks in total for both HMC random walks including tests, documentation, comments and `Rcpp` wrapping. At the end of this week we would be able to sample from convex polytopes in a few thousand dimensions for the first time!

[4] https://tolischal.github.io/GSoC2018/

- **6th & 7th week.** *Implement annealing schedule of volume estimation method.* (30th June - 13th July)

  Create the new header file `gibbs_annealing.h` and implement all the functions described in subsection 2.3 for the annealing schedule that constructs the sequence of Gibbs distributions truncated in the input polytope.

- **8th & 9th week.** *Complete implementation of volume estimation method.* (14th - 27th July)

  Implement the rest of the functions we described in subsection 2.3. At the end of this week we would be able to estimate volumes in a few thousand dimensions for the first time!

- **10th week.** *Documentation, tests, code development.* (28th July - 3rd August)

  1. Write `roxygen` comments in order to create the corresponding `.Rd` files with `roxygen2` library for the documentation.
  2. Add comments to the new `C++` code.
  3. Modify `Rcpp` function `volume()`, as we describe in subsection 2.3, such that the new method can be called by a user.
  4. Write `R` tests in the script `./R-proj/test/testthat/test_volume.R` for the new method.
  5. Run experiments to compare runtimes between the methods already included in `VolEsti` and the new one.

- **11th week.** *Implementations of ellipsoidal family random walks.* (4th - 10th August)

  Implement the steps I described in subsection 2.3 to modify and add the `C++` implementations of [22] to `VolEsti`.

- **12th week.** `C++` *code optimizations.* (11th - 17th August)

  1. Use `Eigen` library to represent points. In the header file `point.h` in the folder `./include/cartesian_geom` I am going to replace the `std::vector` with `Eigen` vector. I will have to modify the rest of `VolEsti` implementations accordingly.
  2. Use `Eigen` matrices that stored in row-major order[5]. This change, considering the previous one as well, would accelerate the matrix-vector multiplication in our code and hence would improve a lot the run time of sampling and volume computations.
  3. Improve the implementation of Hit-and-Run random walk for V-polytopes and zonotopes. In each step of the walk we have to solve two linear programs which have the same basic feasible solution. I am going to modify the implementations in the header file `solve_lp.h` in folder `./include`, to use the same feasible solution for both linear programs.

- **13th week.** *Code development, final submission and* `CRAN` *new version* (18th - 26th August)

  1. Run `CRAN` tests and eliminate warnings and NOTES.
  2. Improve the documentation and the code's comments.
  3. Submit the new version of `VolEsti` package to `CRAN`.
  4. Submit the final evaluation of GSoC project.

[5] https://eigen.tuxfamily.org/dox/group__TopicStorageOrders.html

Comment: The above time schedule is a worst-case schedule. However, if I stuck in some step that I do not expect at the moment, we will scrimp 12th week as it is about C++ optimizations so we could prefer to focus on providing R with new algorithmic features. Of course if the coding exceeds my present expectations I will discuss with mentors new additions and implementations.

4.3 Schedule Conflicts

There are not schedule conflicts during the summer. This GCoC project will be a full time job in the summer for me and I am going to commit in a daily basis.

5. Related work

I participated in last year Google Summer of Code 2018. We worked with mentors F. Vissarion and Z. Zafeiraki on package `VolEsti`. In particular, we algorithmically extended `VolEsti` with new random walks and volume computation methods and provide an `R` interface and a `CRAN` version which is currently under submission.

I am a PhD student working on sampling, volume computation and clustering in high dimensions. Thus I have a very good understanding and solid theoretical background on the topic of the current project proposal as I have been doing research and contribute with some new results. In [4] we examine volume computation of high dimensional polytopes and more general convex bodies and develop an application for financial crisis prediction and portfolios' optimization. We worked on `VolEsti` improvement and we experimentally extended `VolEsti` to non-convex bodies with very encouraging results. More precisely we extend `VolEsti` to convex bodies defined by the intersection of an ellipsoid with other convex bodies as polytopes or parallel half-spaces and to non-convex bodies defined by the intersection of two concentric ellipsoids with the simplex and parallel half-spaces. My master thesis is based on this project and further experimental and theoretical results are obtained.

In [8] we design and implement new algorithms for the volume computations of convex bodies by providing the most efficient implementation for volume estimation for convex polytopes and specially for V-polytopes and zonotopes, contributing to some scientific applications in engineering and bio-geography.

6. Tests

The mentors of the project request to complete one of the three tests they provide. I completed all the tests and, in addition, I added an extra implementation. The results are in [7]. I present some of the technical and algorithmic details in the sequel.

- **Easy**

I added a check for the requested error. If the given value is non-positive then an exception is thrown. We added checks for both R and C++ implementations. For example if you install the R package and run,

```
> P=GenCube(2,'H')
> p=volume(P, error=0)
```

the output is:
`Error in volume(P, error = 0) : The requested error must be positive!`

- **Medium**

i) I have implemented Dikin walk. In file `samplers.h` in folder `./include/samplers` we added functions `Dikin_walk()`, `get_point_in_ellipsoid()` and `is_in_ell()` which implement the

walk. In header file `polytopes.h` in folder `./include/convex_bodies` we added member functions `get_Dikin_ell()` and `set_dikin_rep()` to `Hpolytope` C++ class in order to construct Dikin ellipsoids and get the right representation of the polytope respectively. We add the option to give string `"Dikin"` to the flag `WalkType` of the `Rcpp` function `sample_points()` in order to request uniform sampling from a convex polytope in H-representation with Dikin walk. For example,

```
> P=GenCube(2,'H')
> points=sample_points(P, WalkType = 'Dikin', N=1000)
> plot(points[1,],points[2,])
```

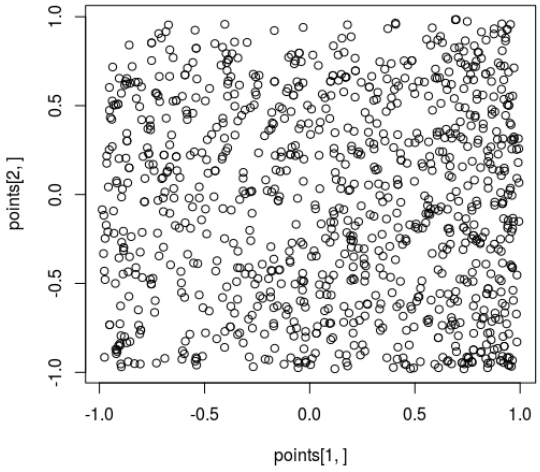

Figure 10: Uniform sampling with Dikin walk from a 2-dimensional unit cube.

you get the Figure 10.

ii) I have modified both Random and Coordinate Directions Hit-and-Run in order to perform boundary sampling from a convex polytope. In file `samplers.h` in folder `./include/samplers` we implemented function `boundary_rand_point_generator()` which takes as input a convex polytope in any representation and samples the boundary. In the `Rcpp`function `sample_points()` we added the boolean flag boundary which has to be TRUE in order to request boundary sampling. For example,

```
> P=GenCross(2,'H')
> b_points = sample_points(P, boundary = TRUE, N=2000)
> plot(b_points[1,], b_points[2,])
```

you get the Figure 11.

- **Hard**

I have added an implementation for the computation of a low dimensional polytope given by a set of linear inequalities and a set of linear equalities. In the folder `./include/volume` I created header file `low_dimensional.h` to implement function `get_low_dimensional_poly()` which takes as input the matrices that define the linear inequalities and equalities and returns the matrices that define the linear inequalities of the corresponding low dimensional polytope. In addition, we modified `Rcpp` function `sample_points()` to add flags `A, b, Aeq, beq` for the input matrices. For example,

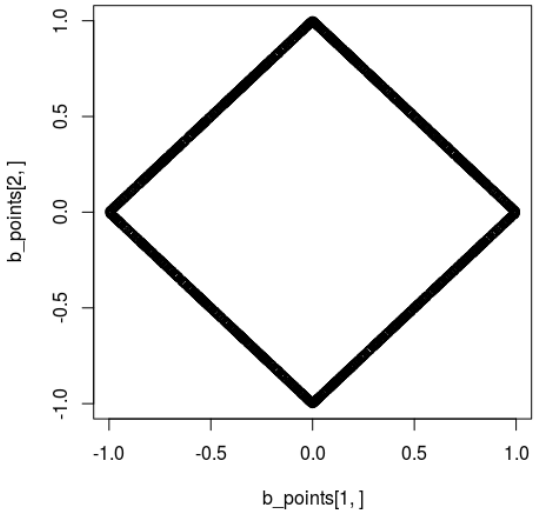

Figure 11: Boundary sampling from a 2-dimensional cross polytope.

```
P=GenCube(3,'H')
A=P$A
b=P$b
Aeq = matrix(c(0,0,1),byrow = TRUE, nrow = 1)
beq = c(1)
volume(A=A, b=b, Aeq=Aeq, beq=beq)
> [1] 3.973613
```

while the exact value is 4 because the low dimensional polytope is a slice of the 3-dimensional cube.

Extra. I have also modified the function `sample_points()` to sample points from a low dimensional polytope. For example,

```
P=GenCube(3,'H')
A=P$A
b=P$b
Aeq = matrix(c(rnorm(3)),byrow = TRUE, nrow = 1)
beq = c(rnorm(1))
points = sample_points(A=A, b=b, Aeq=Aeq, beq=beq, N=1000)
scatterplot3d(points[1,], points[2,], points[3,])
```

we get the Figure 12.

7. Biographical Information

7.1 Academic

- BSc degree from the (5 years curriculum) School of Applied Mathematics and Physics of National and Technical University of Athens. Area of concentration: Mathematics.
- Bachelor's thesis in Numerical Linear Algebra (title: "Numerical algorithms for eigenvalue and singular value decomposition", advisor: Konstantinos Chrysafinos).

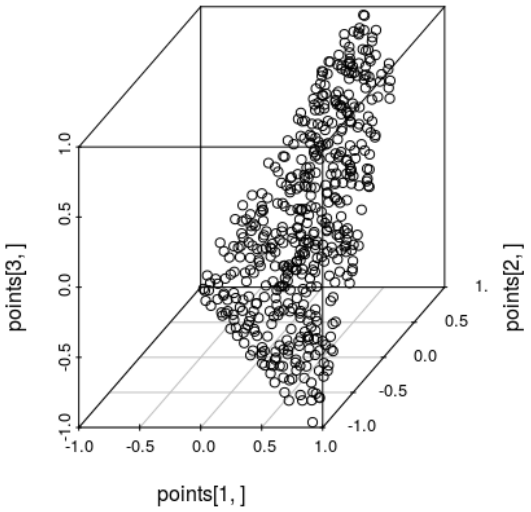

Figure 12: Uniform points from a low dimensional polytope in $\mathbb{R}^3$.

- Graduate student at the Department of Informatics & Telecommunications of National & Kapodistrian University of Athens at the M.Sc. specialization of Theoretical Computer Science (THE)
- Master's thesis in Computation Geometry (title: "Practical Volume Computation of Structured Convex Bodies for Modeling Financial Crises", advisor: Ioannis Z. Emiris).
- PhD student where thesis focus on sampling, volume computation and clustering in high dimensions.

7.2 Programming

- Author of `R` package `VolEsti`.
- Very Experienced in algorithm implementation and general programming.
- Very experienced in programming with `R` and `C/C++` (`CGAL`, `Eigen`, `MPI`, `OpenMP`).
- Completed two Python projects in 2017. *SudoPy* at [6] and *PageRank* at [5].

7.3 Personal motivation

During my bachelor studies I was specially interested in algorithms and programming. That is the reason I chose my master to be in Theoretical Computer Science while I stayed active as a programmer by developing my programming skills and learning new programming languages. My PhD studies are focused on random walks and volume computation in high dimensions giving great deal of value to efficient implementations as I am a very active programmer especially in `C++` and `R` communities.

Working on package `VolEsti` during last year's Google Summer of Code was a great experience. Working with mentors was superb as we met all of our goals and we developed a new `CRAN` package. This year's Google Summer of Code is a great opportunity to continue and significantly upgrade this work by providing `R` with new geometrical and statistical tools for the first time.

Coding was always one of my best activities. I enjoy to integrate algorithms that I study, to chose the proper programming language for each project and to learn new programming skills

and languages. I think that the combination of C++ with R is very effective and practical as we could obtain speed and to provide nice interfaces to scientific or business communities that are not familiar with low level programming. So I think that the integration with open-source communities, that GSoC provides, is a great opportunity for me to participate and to contribute more.

I have been working on the project I propose more than a year. Therefore I think this project is a great opportunity to open some new theoretical and practical questions for my PhD studies and to build a complete tool for geometrical statistics and to show up its importance for a large variety of scientific and business applications.

References

[1] lpSolve. https://cran.r-project.org/web/packages/lpSolve/index.html, 2015.

[2] Dimitris Bertsimas and Santosh S. Vempala. Solving convex programs by random walks. J. ACM, 51:540–556, 2004.

[3] M. Betancourt. A conceptual introduction to hamiltonian monte carlo. CoRR, arXiv:1701.02434, Jan 2018.

[4] Ludovic Calès, Apostolos Chalkis, Ioannis Z. Emiris, and Vissarion Fisikopoulos. Practical volume computation of structured convex bodies, and an application to modeling portfolio dependencies and financial crises. CoRR, arXiv:1803.05861, March 2018. 34th Int. Sympos. on Comput. Geometry, 2018.

[5] Apostolos Chalkis. PageRank. https://github.com/BurnYourPc/PageRank, 2017.

[6] Apostolos Chalkis. Sudopy. https://github.com/BurnYourPc/Sudoku, 2017.

[7] Apostolos Chalkis. Preliminary work (tests) of the proposed project. https://github.com/TolisChal/volume_approximation/tree/gsoc19, 2019.

[8] Apostolos Chalkis, Ioannis Z. Emiris, and Vissarion Fisikopoulos. Practical volume estimation by a new annealing schedule for cooling convex bodies. 35th European Workshop on Computational Geometry, Utrecht, The Netherlands, March 2019.

[9] 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.

[10] 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, 2018.

[11] Jiwoong Choi and In-Chan Choi. Identifying redundancy in multi-dimensional knapsack constraints based on surrogate constraints. International Journal of Computer Mathematics, doi: 10.1080/00207160.2014.885020:2470–2482, 2014.

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

[13] D. Eddelbuettel and R. François. Exposing C++ functions and classes with Rcpp modules. http://dirk.eddelbuettel.com/code/rcpp/Rcpp-modules.pdf, 2018.

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

[15] Vissarion Fisikopoulos and Apostolos Chalkis. GeomScale organization, Practical volume computation and sampling in high dimensions . https://github.com/GeomScale/volume_approximation, 2018-2019.

[16] Y.T. Lee and S. Vempala. Convergence rate of Riemannian Hamiltonian Monte Carlo and faster polytope volume computation. In Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, pages 1115–1121, 2018.

[17] Y.T. Lee and S. Vempala. Geodesic walks in polytopes. CoRR, arXiv:1606.04696, Jun 2016.

[18] Bojan Nikolic. BNMin1: a minimisation library. https://www.mrao.cam.ac.uk/~bn204/oof/bnmin1.html, 2015.

[19] Paulraj S. and Sumathi P. A comparative study of redundant constraints identification methods in linear programming problems. Mathematical Problems in Engineering, doi:10.1155/2010/723402, 2010.

[20] Mamta Sharma, G.N. Srinivasa Prasanna, and Abhilasha Aswal. N-Dimensional Volume Estimation of Convex Bodies: Algorithms and Applications. http://slideplayer.com/slide/4550776/, 2016.

[21] Santosh Vempala. Geometric random walks: A survey. 2005.

[22] M.J. Wainwright Y. Chen, R. Dwivedi and B. Yu. Implementation of Vaidya and Dikin walks and experiments. https://github.com/rzrsk/vaidya-walk, 2018.