Guide for Volume Algorithm

Ben Cousins* Santosh Vempala†

June 13, 2015

Table of Contents

Contents

1 Introduction

The main function is **Volume.m**, which can be called as **volume = Volume(P, E, ε, flags)**, where

- **P** is a structure defining a polyhedron, with the properties $\mathbf{P.A}, \mathbf{P.b}, \mathbf{P.A\_eq}, \mathbf{P.b\_eq}, \mathbf{P.p}$ such that
  - the polyhedron is $\left\{x \middle| \mathbf{P.A} \cdot x \leq \mathbf{P.b} \text{ and } \mathbf{P.A\_eq} \cdot x = \mathbf{P.b\_eq}\right\}$.
  - **P.p** is a point inside the polyhedron (preferably far away from the boundary)

*Georgia Tech. Email: bcousins3@gatech.edu
†Georgia Tech. Email: vempala@gatech.edu

- **E** is a structure defining an ellipsoid, with the properties $\mathbf{E}.\mathbf{E}, \mathbf{E}.\mathbf{v}$ such that the ellipsoid is $\{x|(x-\mathbf{E}.\mathbf{v})^T\mathbf{E}.\mathbf{E}(x-\mathbf{E}.\mathbf{v}) \leq 1\}$.
- ε is the target relative error parameter. Our goal is to estimate the volume of $\mathbf{P} \cap \mathbf{E}$ within an ε-fraction.
- **flags** is a string encoding the flags that modify the behavior of **Volume** (Section 2).
- **volume** is the estimated volume of $\mathbf{P} \cap \mathbf{E}$.

The arguments **P** and **E** are optional, but one of them must be provided. For instance, if you are computing the volume of a cube, then only **P** should be provided, and set $\mathbf{E} = [\quad]$.

1.1 Running the Test Bodies

Provided with the code are a set of test bodies. It may be helpful to try these out before computing the volume of your own convex body. The wrapper functions **Cube.m** and **Simplex.m** are provided to demonstrate the algorithm on simple examples. The first argument is the dimension of the body. The second argument is the target error. The third argument is the type of the body. For example,

- **Cube(10,0.2,1)** will compute the volume of a 10-dimensional $[-1,1]^n$ cube with error parameter 0.2.
- **Cube(5,0.2,2)** will compute the volume of a 5-dimensional randomly transformed $[-1,1]^n$ cube with error parameter 0.2. Rounding will occur here.
- **Simplex(10,0.2,1)** will compute the volume of a 10-dimensional isotropic simplex with error parameter 0.2.
- **Simplex(15,0.2,2)** will compute the volume of a 15-dimensional standard simplex with error parameter 0.2. Rounding will occur here.

There are more example bodies provided by the function **makeBody.m**. Here is a complete list:

- **Polytopes**:
  - **Cube**: a standard $[-1,1]^n$ cube, which has volume 2^n.
  - **Isotropic Simplex**: a regular n-simplex, which has volume $\sqrt{n+1}/(n!\sqrt{2^n})$.
  - **Standard Simplex**: a standard n-simplex, which has volume $1/n!$.
  - **Long box**: the cube, with one axis stretched out: a $[-1,1]^{n-1} \times [-100,100]$ box, with volume $100 \cdot 2^n$.
  - **Birkhoff**: the Birkhoff polytope, which is an $(n-1)^2$ dimensional polytope of all perfect matchings on the complete bipartite graph $K_{n,n}$. There is no known closed form for its volume, but exact values have been computed for $n \leq 10$.
  - **Triply Stochastic**: A convex body of all triply stochastic matrices, i.e. $k \times k \times k$ matrices where all 1-dimensional slices sum to 1.
- **Ellipsoids**

- **Ball**: An n-dimensional unit ball, with volume $\pi^{n/2}/\Gamma(n/2+1)$.
- **Ellipsoid**: An axis-aligned ellipsoid with radius 1 along $n-1$ axes and radius 100 along 1 axis. The volume of the shape is then $100\pi^{n/2}/\Gamma(n/2+1)$.

To create a polytope P as a 10-dimensional cube, you can run:

[P, actual_vol] = makeBody('cube',10);

The definition of **P** is given in Section 1. You can then run the volume algorithm with $\varepsilon = 0.2$ on this body by typing:

[volume] = Volume(P, [ ], 0.2,);

To create an ellipsoid E as a 10-dimensional ellipsoid, you can run:

[E, actual_vol] = makeBody('ellipsoid',10);

The definition of **E** is given in Section 1. To compute the volume of this ellipsoid, you can run:

[volume] = Volume([ ], E, 0.2);

Then, if you wanted to compute the volume of $\mathbf{P} \cap \mathbf{E}$, you can provided both arguments to the function: (note that in this case, you want $\mathbf{P}.\mathbf{p}$ to be a point inside $\mathbf{P} \cap \mathbf{E}$, and preferably far away from the boundary)

[volume] = Volume(P, E, 0.2);

1.2 Running your own Convex Body

You can run any convex bodies that you can describe as $\mathbf{P} \cap \mathbf{E} = \{x|Ax \leq b\} \cap \{x|(x-v)^T Q^{-1}(x-v) \leq 1\}$. The structure of **P** and **E** is given in Section 1. You then need to provide a point $\mathbf{p} \in \mathbf{P} \cap \mathbf{E}$. Rounding is turned off by default, but your body may require it; please refer to Section 2.1.

It is important that the convex body you provide be full-dimensional. That is, if the polytope is in R^n, then the polytope needs to have an n-dimensional volume. If this is not true, then the **Volume** algorithm will fail. However, we can restrict to a lower dimensional subspace and compute the volume there. The function **preprocess** will do this for polytopes. It will take in a structure **P**, and return a equivalent, full-dimensional version of this polytope. For equality constraints $\mathbf{P}.\mathbf{A\_eq} \cdot \mathbf{x} = \mathbf{P}.\mathbf{b\_eq}$, we can simply restrict to this space. We then determine degeneracies in the system of constraints $\mathbf{P}.\mathbf{A} \cdot \mathbf{x} \leq \mathbf{P}.\mathbf{b}$, and further restrict as necessary until we are left with a full-dimensional system of linear constraints.

If you do not know a point inside your convex body and your body is a polytope, the function **preprocess** will attempt to find one. Once you have a point inside the polytope, you can call the **Volume** function.

2 Flags

In the call to **Volume.m**, there is an optional last argument "flags" which is a string with encoded arguments, which are extracted in **parseFlags.m**. For instance,

flags='-round 1000 -num_t 3 -verb 0'

will turn rounding on with parameter 1000, set the number of threads at 3, and set the verbosity level to 0.

2.1 Rounding

Rounding is turned off by default!

Rounding is the most expensive part of the volume algorithm. It is turned off by default because if your body is already sufficiently round, you do not need this step. However, if your body is quite "skew", for instance a very long, thin cylinder, our volume algorithm has no hope to accurately compute this volume. Therefore, you need a rounding preprocessing step. Rounding is currently implemented by taking a large amount of sample points from our body K, determine the linear transformation that rounds the points, and then apply that transformation to the body K. Multiple rounding iterations may be required to accurately round K.

When you turn on rounding, there is an optional argument R which is an upper bound on the ratio of the smallest enclosing ball the largest inscribed ball in K. That is, if $r_1 B_n \subseteq K \subseteq r_2 B_n$, then $R \geq r_2/r_1$. This quantity lets us bound the number of rounding iterations which may be required. If the argument is not provided, we assume an upper bound.

Note that if you only need to round your convex body and do not need to compute volume, you can call **round** externally. The function is located in the object file **ConvexBody.m**. You first need to create a **ConvexBody** object, which can be created using the same arguments as **Volume**. Then,

[T] = round(K,5);

will approximately round **K** with a linear transformation **T**. Note that **K** will be modified inside **round** by the transformation **T**. The number 5 is the number of independent hit-and-run threads that are run to obtain samples to compute the rounding matrix **T**.

2.2 List of Flags

Here we give a list of all flags which can be provided, with a brief description of each:

- **-round X**: turn on rounding with parameter X (see Section 2.1).
- **-a_0 X**: set the starting function to be a spherical Gaussian with variance $1/(2X)$.
- **-verb X**: set the verbosity level to be X. Possible values of X are $0, 1, 2$. Default level is 1.
- **-ratio X**: set the cooling ratio to be a fixed quantity X (i.e. $a_i = a_{i-1} \cdot X$). By default, the cooling is done adaptively.
- **-C X**: bound the quantity $E(Y^2)/E(Y)^2 \leq X$ that is used in the adaptive cooling. Default value of X is 2. A higher value will allow for faster cooling, but at the price of how long each phase takes.
- **-num_t X**: Set the number of independent threads of hit-and-run to be X. Default is 5.
- **-a_stop X**: Set the cooling to stop at a spherical Gaussian with variance $1/(2X)$. This will integrate a spherical Gaussian over the convex body. Default value is 0.
- **-c_test X**: Change the convergence test used for deciding when a phase is mixed. You can refer to the bottom of **Volume.m** or **Sample.m** for possible types of convergence tests.

- **-walk X**: Change the random walk that the algorithm uses for generating samples. Possible walks are coordinate hit-and-run, hit-and-run, and ball walk. The default walk is coordinate hit-and-run.
- **-plot**: Turn on plotting of the distribution of points taken in each phase. The points are n-dimensional, but a 2-dimensional projection of the sample points will be shown.
- **-plot_c**: Turn on plotting of the convergence test, i.e. plot the value that the convergence tests monitors to determine convergence.

3 Sampling

3.1 Obtaining a random sample point

The function **Sample.m** is provided, and using it is very similar to using **Volume.m**. It can be called as **x = Sample(P, E, ε, flags, y)**, where x is the returned sample point and y is the starting point for the walk. All other arguments are analogous to **Volume**. The discussion below is for obtaining a uniform random point from a convex body K, but you can extend them to obtain a point from any spherical Gaussian by setting the **-a_stop** parameter in **flags**. To obtain a single approximately uniformly random point x from a convex body $K = \mathbf{P} \cap \mathbf{E}$ with error parameter $\varepsilon = 0.20$, type the following:

[x] = Sample(P, E, 0.2);.

The error parameter ε does not have a strict meaning, but smaller values of ε will give a point x closer to the target distribution. Further discussion is given in Section 4.

If you wish to compute multiple random points, it could be more efficient to use each point as the starting point for the next walk, e.g.

- **x = Sample(P,E,.2);**
- **y = Sample(P,E,.2,",x);**

Note that calling **Sample** multiple times will give approximately independent points. For some applications, it may be preferable to use a large number of dependent points, which is what we do to estimate the volume. If you find you need this, you can directly call the function **getNextPt** in the object file **ConvexBody.m** to take one step of the Markov chain (by default, coordinate hit-and-run). Note you need to first create a **ConvexBody** object **K**. The next point obtained is highly dependent on the current point. For instance,

[x] = getNextPt(K, y, a);

will return the point $\mathbf{x} \in \mathbf{K}$ after taking one step of hit-and-run from $\mathbf{y} \in \mathbf{K}$ with respect to the distribution $f(z) = e^{-a\|z\|^2}$.

4 Convergence Tests

A big question for both computing volume and generating sample is when to stop. At the core of each of these algorithms is a Markov chain, so by nature they are iterative. But, say for volume computation, we want a target relative error ε. When is the current estimate within the target error ε?

To attempt to determine when we have reached our target error, we use statistical tests. The current approach utilizes a sliding window, which keeps track of the last W estimates, for some value of W. When all of the last values are within ε of each other, we declare convergence. The parameters were optimized to give a 75% success rate for bodies of dimension at most 100 and $\varepsilon \in [0.10, 0.20]$. If you are noticing that this error is off, first trying turning on rounding (Section 2.1). If that does not help, you could try changing the size W of the sliding window in Volume.m. Increasing the size of the sliding window will decrease the error.

You can also add your own convergence tests. At the bottom of **Volume.m**, there are functions **newVCT** and **updateVCT** where you create and update, respectively, your convergence test. You can then select this convergence test to be used by the argument '-c_test' (see Section 2). There is analagous functionality in **Sample.m**.

5 Other Comments

5.1 Integrate/sample general log-concave functions

Currently, there is only functionality to compute the volume of, or sample from, of a convex body with respect to a spherical Gaussian or the uniform distribution. However, similar approaches can be used to sample from a convex body according to a logconcave distribution f (i.e. the logarithm of the function is concave), or to integrate a logconcave function f over a convex body. We hope to add this functionality in the near future (please contact us and let us know if you would like this functionality!).

5.2 Contact

If you find anything that could be a bug, or just something doesn't seem quite right, please email us at **bcousins3@gatech.edu** and **vempala@gatech.edu**. There are likely some cases that won't work well that we did not discover in our testing. Also, contact us if you have any questions, suggestions, praise, or ridicule. We would love to hear about any applications you use this code for, and any way that we could improve the program to make it better for your application.