Guide for Volume Algorithm

Ben Cousins* Santosh Vempala†

October 2, 2013

Table of Contents

Contents

1 Introduction

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

- $\mathbf{P} = [\mathbf{A} \quad \mathbf{b}]$ is a polytope $\{x|Ax \leq b\}$.
- $\mathbf{E} = [\mathbf{Q} \quad \mathbf{v}]$ is an ellipsoid $\{x|(x-v)^T Q^{-1}(x-v) \leq 1\}$.

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

- ε is the target relative error parameter. Our goal is to estimate to volume of $\mathbf{P} \cap \mathbf{E}$ within an ε-fraction.
- $\mathbf{p}$ is a point inside the convex body, i.e. $\mathbf{p} \in \mathbf{P} \cap \mathbf{E}$.
- **flags** is a string encoding the flags that change the behavior of **Volume** (Section 2).
- **volume** is the estimated volume of $\mathbf{P} \cap \mathbf{E}$.

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

1.1 Running the Test Bodies

Provided with the code are a set of test bodies, you should try these first before trying to run your own convex body. First, try out the wrapper functions **Cube.m** and **Simplex.m** that are provided.

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

So, now you have a feel for how the volume algorithm works. There are more 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 $n \leq 10$.
- **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, p, actual_vol] = makeBody('cube',10);

The definitions of **P** and **p** are 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, p);

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

[E, p, 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, p);

Then, if you wanted to compute the volume of $\mathbf{P} \cap \mathbf{E}$, you can provided both arguments to the function: ($\mathbf{p} = \mathbf{0}$ for both cases, so $\mathbf{p} \in \mathbf{P} \cap \mathbf{E}$)

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

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.

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 (roughly) 0.
- **-c_test X**: (only for **Sample.m**) Change the convergence test used for deciding when a phase is mixed. Possible values of X are $1, 2$.

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, p, 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, p);.

As with **Volume.m**, **p** is any point in K close to the center. 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 3.2.

As with **Volume**, you can set the starting Gaussians used in the annealing schedule. For **Volume**, setting this values could, for instance, compute the ratio of two spherical Gaussians over a convex set. With respect to **Sample**, this has a different benefit: if you already have a random point $y \in K$ from the target distribution, there is no need to use a sequence of Gaussians; you can simply use y as a starting point for the walk and immediately use hit-and-run with respect to the target distribution. For example,

[y] = Sample(P, E, 0.2, p);
[x] = Sample(P, E, 0.2, p,'-a_0 0', y);

will give two random points **x, y** from $\mathbf{P} \cap \mathbf{E}$. This will run faster than simply calling **Sample** twice.

As a note, the above will give two points $x, y \in K$ that are approximately independent. 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 **hitAndRun** in the object file **ConvexBody.m** to take one step of 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] = hitAndRun(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 $f(z) = e^{-a\|z\|^2}$.

3.2 Convergence Tests

The error parameter ε provided to **Sample** will be used similar to the sliding window discussed in Section 4.1, but monitoring a different quantity. We provide two different convergence tests, one of which could perform better for your application. The first convergence test ('-c_test 1', used by default) monitors the proportion of points that lie in a random halfspace through the current mean. The sliding window then monitors this proportion, and announces convergence once the values stop changing significantly. The second convergence test ('-c_test 2') monitors the quantity $\mathsf{E}(\|X - \mu\|^2)$, the expected squared distance from the mean. We average this value over all sample points seen so far. As with the first test, once the averages are within some ε over a sliding window, we announce convergence and return the current point.

4 Other Comments

4.1 Controlling Error

The biggest issue with the current implementation is that the volume estimate guaranteed to be within the relative error ε. The current approach was optimized to get approximately 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. If you are achieving "too little" error, then the program could be more efficient by decreasing the size W of the window, and you would be closer to your target.

4.2 Ball Walk

In ConvexBody.m, there is a commented out function for sampling according to the ball walk with radius δ. That is, from a current point x, sample y uniformly at random from $x + \delta B_n$, and move to y if it is in K (for arbitrary densities, also apply a Metropolis filter). We use hit-and-run throughout the implementation, but feel free to comment this function out and use it. You need to provide a radius δ, and in brief tests, we found that this walk performed best if δ was chosen so that the average local conductance (the probability of a proper step, where the walk does not stay at the current point x) was around 1/2. Too low of a δ, and it takes too long to mix. Too high of a δ, and the probability that a proper step is made is too low. One could potentially search on this value.

4.3 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.

4.4 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.

4.5 Acknowledgements

We acknowledge that the function **min_elp_dist.m** that computes the minimum distance from a point to the surface of an ellipsoid was written by Stanley Chan (http://videoprocessing.ucsd.edu/∼stanleychan/publication/unpublished/Ellipse.pdf).