Computational Geometry Lab:
MONTE CARLO ON TETRAHEDRONS

John Burkardt
Information Technology Department
Virginia Tech
http://people.sc.fsu.edu/~jburkardt/presentations/cg_lab_monte_carlo_tetrahedrons.pdf

August 28, 2018

1 Introduction

This lab continues the study of Computational Geometry. We are now concerned with the case when the *Monte Carlo method* is to be applied to a single tetrahedron. The Monte Carlo method is a general procedure for understanding a system by selecting inputs at random and analyzing the resulting outputs. For the tetrahedron, we will concentrate on the application in which we are attempting to estimate the integral of a function.

In a separate lab, we study ways of estimating an integral by using a quadrature rule, that is, a table of abscissas and weights. A quadrature rule usually has the advantage that the points and weights have been selected in a way that guarantees that the integral estimate is exact for low order polynomials. While a Monte Carlo rule has very limited exactness properties, we will see by contrast that we can get as much accuracy as we demand, simply by increasing the number of points used.

Because of the element of randomness in the Monte Carlo method, we will need to investigate the somewhat puzzling question of how to pick a point uniformly at random from a tetrahedron. Here "uniformly at random" is a stronger requirement than simply "arbitrary" or "scrambled". If points are not properly selected, then the algorithm will lose a significant portion of its accuracy.

The main reason for investigating the approximation of integrals over a single tetrahedron, of course, is to prepare for the "real" problem, which is to estimate an integral over a 3D region which has been broken down into a mesh of tetrahedrons. That is the subject for a later lab.

2 Uniform Sampling by the Rejection Method

Recall the definition of the reference tetrahedron, **Tref**={{1,0,0},{0,1,0},{0,0,1},{0,0,0}}. Suppose we were asked to choose a point *uniformly at random* point from this tetrahedron. What are we being asked to do, and how can we ensured that we do what is required?

These questions are easier to pose and answer for the unit cube, so let's back up and consider that question. If we are asked to choose a point uniformly at random from the unit cube, then it should be the case that each point has the same chance of being picked. Since there are an infinite number of points, we can't sensibly compute the probability of any one point being picked. Instead, let's suppose that we subdivide the cube into an NxNxN array of subcubes. Then it should be the case that each subcube has an equal chance of being the source of our point. Of course, only one subcube can be chosen, so the best way to see if our selection process is uniform is to select many points by the same process. If the number of points chosen from each subcube is roughly equal, we have evidence that our selection process is uniform. Another

way of judging is to show that the number of points picked from any subregion corresponds roughly to the volume of that subregion.

Uniform sampling is surprisingly easy on the unit cube, since our programming language usually provides some kind of function, perhaps called **rand()**, which returns values selected uniformly at random from the unit interval [0,1]. It is easy to see that in this case, we can simply pick let **rand()** produce three values which we can use to form the point **{r,s,t}**, selected uniformly at random from the unit cube.

However, when sampling from a tetrahedron, we must modify our selection process in some way to account for the geometry. Perhaps the simplest method of doing so is called the *rejection method*. Since the reference triangle lies inside the unit cube, we simply pick a point uniformly at random from the unit cube. If that point happens to lie inside the tetrahedron as well, we accept it. Otherwise, we reject the point, and try again.

Rejection method:
Select uniformly at random a point (R,S,T) in the reference tetrahedron.

```
do
  Select a random point in the unit cube.
  r = rand();
  s = rand();
  t = rand();
  Success if the point lies in the tetrahedron.
  If r+s+t <= 1.0 then
    return
  end if
  Otherwise, try another point.
end do
```

The wonderful feature of the rejection method is how little thinking we have to do in order to employ it. And you can convince yourself that if the original method uniformly samples the cube, that means the sample points are evenly distributed throughout the cube. If we then "slice out" the tetrahedron and examine it, the sample points included in this tetrahedron must also be evenly distributed, just like raisins in a slice of cake.

On the other hand, the rejection method is *inefficient*. Since the unit cube has volume 1, and the reference tetrahedron has volume $\frac{1}{6}$, we tend to reject about 5 points for every 1 we accept.

Try to have some perspective on this inefficiency, though. Maybe it's not all that bad. What we're really saying is that, roughly speaking, we compute the typical sample point using 18 lines of code (computations) rather than 3. So if another method is "better" because it never rejects a point, we can ask how many extra lines of code are involved in this other method. It may be the case that 30 or 50 computations are required, in which case the rejection method doesn't look so bad!

3 Program #1: Uniform Sampling by the Rejection Method

Write a program which approximates the integral **I** of $f(x,y,z)$ over the reference tetrahedron by a quadrature estimate **Q** using the Monte Carlo algorithm, and with **N** random sample points generated by the rejection method.

For your function, use $f(x,y,z) = 2520 * xy^2z$; compute a series of estimates, with **N** equal to successive powers of 2 from 1 to 65536 points.

Make a plot of log(**N**) versus the error |**I-Q**|.

The expected behavior is

$$|\mathrm{I} - \mathrm{Q}| \quad \propto \quad \frac{1}{\sqrt{N}} = N^{-\frac{1}{2}}$$

so your graph of log(**N**) versus error should have a slope of roughly $-\frac{1}{2}$.

4 Efficient Sampling (but is it Uniform?)

By now, you have probably run across the barycentric coordinate system that can be used in any tetrahedron. This is a method by which a set of four values $(\xi_a, \xi_b, \xi_c, \xi_d)$ can be used to identify a point in the reference tetrahedron. Given the vertices of a general triangle, these same coordinates can be used to identify a point there as well.

It is natural to try to find a simple way of choosing these coordinates at random, as a means of sampling any tetrahedron. If we simply choose 4 values of the **rand()** function, then the result is certainly nonnegative, but the coordinates will not correspond to a point inside the tetrahedron unless it is also true that:

$$\xi_a + \xi_b + \xi_c + \xi_d = 1$$

However, it's easy to make our data satisfy this extra constraint: we simply divide each of our original values by the sum.

We now have a way of selecting points in the reference tetrahedron. It's not hard to see that every value $(\xi_a, \xi_b, \xi_c, \xi_d)$ we compute will now correspond to some point in the tetrahedron, and a little thought will convince you that, conversely, every point in the tetrahedron can be chosen by this process. So the process seems "random", *but is it actually uniform?*.

$$\begin{aligned}(r, s, t, u) &= (\text{rand}(), \text{rand}(), \text{rand}(), \text{rand}())\\ (\xi_a, \xi_b, \xi_c, \xi_d) &= \frac{(r, s, t, u)}{r + s + t + u}\\ P &= \xi_a * a + \xi_b * b + \xi_c * c + \xi_d * d\end{aligned}$$

In fact, this method will not sample the points in the unit tetrahedron in a uniform way. This, in turn, will mean cause the error behavior of our quadrature procedure to deteriorate. So even if we don't work out the deficiencies of this selection method, we might suspect that there is a problem if we pay attention to the error behavior of our algorithm on some test cases.

5 Program #2: Efficient Sampling (but is it Uniform?)

Repeat the previous program, but now generate the **N** random sample points by the "efficient but nonuniform" method.

For your function, use $f(x, y, z) = 2520 * xy^2z$; compute a series of estimates, with **N** equal to successive powers of 2 from 1 to 65536 points.

Since this method is **not** a uniform sampling of the tetrahedron, the expected value of the error is not governed by the $N^{-\frac{1}{2}}$ behavior; hence we may expect it to be worse than the results we saw in Program #1.

Make a plot of log(**N**) versus the error |**I-Q**|. Does the evidence suggest that there is a problem?

6 Efficient Uniform Sampling

Here is a third scheme for selecting a point from a tetrahdron. Like the previous scheme, there is no rejection. But like the first scheme, the method is a uniform selection over the unit tetrahedron.

$$
\begin{aligned}
(r,s,t,u) &= (\text{rand}(), \text{rand}(), \text{rand}(), \text{rand}()) \\
(r,s,t,u) &= (-\log(r), -\log(s), -\log(t), -\log(u)) \\
(\xi_a, \xi_b, \xi_c, \xi_d) &= \frac{(r,s,t,u)}{r+s+t+u} \\
P &= \xi_a * a + \xi_b * b + \xi_c * c + \xi_d * d
\end{aligned}
$$

7 Program #3: Efficient Uniform Sampling

Repeat the previous program, but now use the "efficient uniform" scheme to generate sample points.

Use the same **f(x,y,z)** and sequence of values of **N** as in the previous programs.

Make a plot of log(**N**) versus the error |**I-Q**|. Does the error behavior look similar to program #1 or #2?

The "efficient, uniform" scheme is a correct way to sample points from a tetrahedron in a uniform manner. For the remainder of this lab, we will assume that this is the sampling method being used.

8 Monte Carlo Quadrature on the Reference Tetrahedron

Now that we have some confidence in sampling points uniformly from a tetrahedron, we are ready to try to apply the Monte Carlo method to estimate an integral.

The Monte Carlo estimate for the integral of a function $f(x,y,z)$ over the reference triangle **Tref**, using **n** points, has the form:

$$
I(f(x,y,z), \text{Tref}) = \int_0^1 \int_0^x \int_0^{1-x-y} f(x,y,z)\, dz\, dy\, dx \approx MC(f(x,y,z), \text{Tref}, n) = \frac{1}{6} \frac{1}{n} \sum_{i=1}^{n} f(x_i, y_i, z_i)
$$

where each (x_i, y_i, z_i) is generated by uniform sampling of the reference tetrahedron. The factor of $\frac{1}{n}$ averages the function values, and the factor of $\frac{1}{6}$ accounts for the volume of the unit tetrahedron. In particular, this guarantees that if $f(x,y,z) = 1$, our estimate will be equal to $\frac{1}{6}$, the volume of the reference tetrahedron.

Over the reference tetrahedron, there is an exact formula for the integral of any monomial $f(x,y,z) = x^p y^q z^r$, with $0 \le p$, $0 \le q$, $0 \le r$:

$$
I(x^p y^q z^r, \text{Tref}) = \int_0^1 \int_0^x \int_0^{1-x-y} x^p\, y^q\, z^r\, dz\, dy\, dx = \frac{p!\, q!\, r!}{(p+q+r+3)!}
$$

This formula will allow us to compare our estimated results against exact values.

9 Program #4: Monte Carlo on the Reference Tetrahedron

Write a program which approximates the integral of $f(x,y,z) = x^p y^q z^r$ over the reference tetrahedron using the Monte Carlo algorithm.

The program might be outlined as:

```
Read the powers p, q and r;
Read the random number seed and initialize the random number generator.
Read the number of sample points N;
Compute I, the exact integral of x^p y^q z^r;
Compute Q, the Monte Carlo estimate of the integral;
Print (p,q,r), n, I, Q, and abs (I-Q).
```

Start your testing with a "sanity check" using **N**=10 and **(p,q,r)**=(0,0,0). You should get the result **Q**=1 exactly.

Now fix **(p,q,r)**=(4,1,2), and run the program for a sequence of values of **N** that start at 100 and increase by doubling to 200, 400, up to **N**=204,800. Plot log(**N**) versus the error. Do you see a behavior in which multiplying **N** by 4 tends to divide the error by 2?

10 Monte Carlo on a General Tetrahedron

How can we take what we have learned about the reference tetrahedron and apply it to situations where we want to estimate an integral over a general tetrahedron? Our formula for computing a random sample point

$$P = \xi_a * a + \xi_b * b + \xi_c * c + \xi_d * d$$

was already written assuming that the tetrahedron is given by a general list of vertices {a,b,c,d}, so the sampling formula is ready for the general case.

The other issue is that if we are working with a general tetrahedron, we must multiply our quadrature sum by the volume of this tetrahedron:

$$I(f(x,y,z),\Delta) = \int_\Delta f(x,y,z)\,dz\,dy\,dx \approx MC(f(x,y,z),\Delta,n) = \frac{Volume(\Delta)}{n}\sum_{i=1}^{n} f(x_i,y_i,z_i)$$

This is why, when we were working in the reference tetrahedron, we multiplied the quadrature sum by $\frac{1}{6}$. Thus, to use the Monte Carlo procedure to estimate an integral on a general tetrahedron only requires using the vertex information in the sampling formula, and computing the volume to use as a factor when the estimate is computed. We have already encountered a formula for computing the volume of a tetrahedron in the lab on tetrahedrons.

Thus, it seems that handling the quadrature problem for a general tetrahedron involves only a few small changes from the case of the reference tetrahedron!

11 Program #5: Monte Carlo on the General Tetrahedron

Write a program which approximates the integral of $f(x,y,z) = x^p\,y^q\,z^r$ over an arbitrary tetrahedron **T**, using the Monte Carlo algorithm.

The program might be outlined as:

```
Read the tetrahedron T;
Read the powers p, q and r;
Read the random number seed and initialize the random number generator.
Read the number of sample points N;
Estimate the integral using the Monte Carlo algorithm (you know this part by now!);
```

Consider our example tetrahedron #1 or **"Tet1"**, whose definition is

```
{{ 1.0,  2.0,  3.0},
 { 4.0,  1.0,  2.0},
 { 2.0,  4.0,  4.0},
 { 3.0,  2.0,  5.0}}
```

Do a "sanity check" using **N**=10 and **(p,q,r)**=(0,0,0). You should get the result **Q**=$\frac{8}{3}$ exactly, the volume of the tetrahedron.

Since we are working in a general tetrahedron, we don't have a formula for the exact integrals. Here are several values:

Try to approximate some of these values with your program.

Table 1: Values of some test integrals over tetrahedron **Tet1**

(p,q,r)	$\int x^p y^q z^r \, dx \, dy \, dz$
(0,0,0)	2.6666666667
(1,0,0)	6.6666666667
(0,2,0)	14.1333333333
(0,0,3)	121.3333333333
(1,1,2)	187.6952380952