Computational Geometry Lab:
MONTE CARLO ON TRIANGLES

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

August 28, 2018

1 Introduction

This lab continues the study of Computational Geometry. We are now concerned with the *Monte Carlo method*, which is a general procedure for understanding a system by selecting inputs at random and analyzing the resulting outputs. Depending on the application, the Monte Carlo method can be used for sampling, for simulation, or even, as we shall see, for numerical quadrature, that is, the estimating of integrals.

We have already considered numerical quadrature using quadrature rules, in which we are given a table of points and weights. When we apply the Monte Carlo method to the same problem, it will be natural to make comparisons with what we learned in that work. In particular, we will wonder what advantages the Monte Carlo method might offer, and whether we can do things such as improve an estimated integral, or make an educated guess as to the amount of error remaining in our estimate. Because of the element of randomness in the Monte Carlo method, we will also need to investigate some entirely new issues, including the somewhat puzzling question of how to pick a point at random from a triangle. This turns out to be a little trickier than one might think!

This lab, which looks at approximating integrals over a single triangle, is of course a stepping stone towards the next lab, which will look at approximating integrals over a full triangulation.

2 Monte Carlo Quadrature over an Interval

Before getting to the triangles and triangulations that are our main interest, we will introduce the Monte Carlo method on a problem so simple that we can concentrate on the method's peculiar features. So we suppose that we wish to estimate the integral of a function $f(x)$ over an interval $[a, b]$:

$$I(f(x), [a, b]) = \int_a^b f(x)\, dx$$

The mean value theorem of calculus says that for a continuous function, there is always a point ξ within the integration integral so that

$$I(f(x), [a, b]) = f(\xi) * (b - a)$$

The value $f(\xi)$ is called the mean value of the function over the interval $[a, b]$, and it really is the average value of the function. So one message of the mean value theorem is that you can estimate an integral by estimating the mean value and multiplying by the integration region's length (or area or volume).

There are many ways of estimating the average value of a function. Our choice will be to make a random selection of argument values and average them. One advantage of a randomized approach is that the same

selection process can be used over and over. Another advantage is that, at any point in the selection process, we can estimate the integral with the current data; if we wish to continue the selection process, it is not necessary to start over; we simply add more data to our current list.

So now it is time to write a simple program for estimating an integral with the Monte Carlo method. Since the random number generator returns a value in $[0, 1]$, let's keep things simple and use that as our integration interval as well.

3 Program #1: Monte Carlo Quadrature on the Unit Interval

Write a program which approximates the integral of $f(x) = 3x^2$ over the interval [0,1] using the Monte Carlo algorithm.

Notice that we are going to do this calculation in stages. N will count the total number of points, and S the sum of all function values. At each stage, we improve the current estimate by evaluating the function at an additional $NMORE$ points and adding these results to our sum.

Program #1 might be outlined as:

```
Read the random number seed and initialize the random number generator.
Compute I(f(x),[0,1], the exact integral of 3x^2.
Set N=0,  S=0.0, KMAX=10.
For K = 0 to KMAX
  NMORE = 2^ K
  For J = 1 to NMORE
    X = rand;
    S =  S + f(X)
  End J loop
  N = N + NMORE
  Q = S / N
  Print N, I, Q, abs(I-Q).
End K Loop
```

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 $-\frac{1}{2}$. If you can't see the trend, get more data by increasing **KMAX**.

4 Picking a Random Point From a Triangle

If we want to apply the Monte Carlo quadrature method to a triangle instead of an interval, surely the first thing we need to consider is the problem of picking a "random" point from a triangle. The selection of points must be done in a uniform way. That is, we wish to choose points in such a way that the number of points taken from any particular subregion of the triangle will tend to be proportional to the relative area of that subregion. This will correspond to our visual perception that the number of points are evenly scattered over the triangle.

If we can assume we have a function called **rand** which returns real numbers uniformly at random in the unit interval [0,1], then here is one possible procedure for selecting points from a triangle whose vertices are the points {a, b, c}. We will call this procedure "Scheme #1". As described in the lab on triangle mapping, any point in a given triangle can be described by its barycentric coordinates, a trio of numbers $\{\xi_a, \xi_b, \xi_c\}$ each of which is between 0 and 1, which sum up to 1. So if every point in the triangle has such an address,

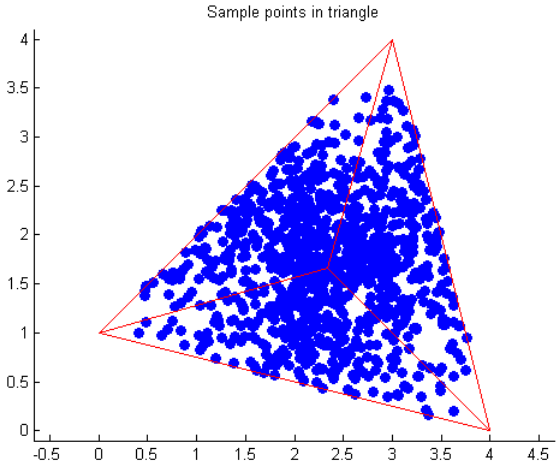

Figure 1: Sample Points in Triangle Tex4 using Scheme #1

why don't we simply pick such an address "at random"? Without thinking too hard about what that means, let's just pick three random numbers, and then normalize them so the add up to one. That's the address of a point in the triangle, and we seem to have gone about the selection process in a random way.

$$\begin{aligned} r &= rand; \\ s &= rand; \\ t &= rand; \\ \xi_a &= r/(r+s+t); \\ \xi_b &= s/(r+s+t); \\ \xi_c &= t/(r+s+t); \\ P &= \xi_a * a + \xi_b * b + \xi_c * c; \end{aligned}$$

However, just because we used random numbers in the selection process doesn't mean that the result is random, or more importantly, that the points are uniformly scattered throughout the triangle.

There are two simple tests we can make. The first is to divide the triangle into three subtriangles, using the centroid and pairs of vertices. We know, from the lab on triangles, how to tell whether a point is contained in a triangle,

5 Program #2: Triangle Sampling With Scheme #1

Write a program which reads a triangle **t**, a random number seed **seed**, and a number **N** of points to generate, and carries out Scheme #1.

Let **m** indicate the centroid of the triangle. Denote by **mbc** the subtriangle formed by vertices **a**, **b** and the centroid, with similar definitions for **amc** and **abm**. As a simple test of uniformity, we hope that roughly equal numbers of the **N** sample points fall into each subtriangle.

Program #2 might be outlined as:

```
Read the triangle T.
Read the random number seed and initialize the random number generator.
Read N, the number of sample points.
```

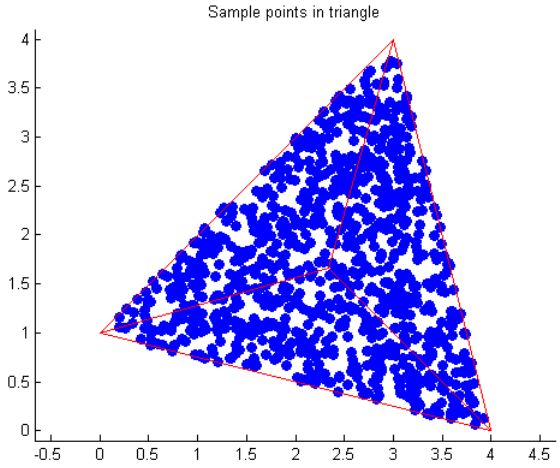

Figure 2: Sample Points in Triangle Tex4 using Scheme 2

```
Generate N sample points P in T using scheme #1;
Count and print N1, N2, N3, the number of sample points
contained in subtriangles mbc, amc and abm;
```

For **T** use example triangle Tex4 defined by {{4,0},{3,4},{0,1}}. Use **N** = 1000. Are the values **N1, N2, N3** roughly equal?

Now plot the sample points. The easiest way to do this is to store the coordinates of the sample points in arrays **X** and **Y** and then call

```
scatter ( x, y )
axis equal
```

Now does the sampling seem roughly uniform throughout the triangle?

6 Program #3: Triangle Sampling With Scheme 2

Here is a second scheme for selecting a point from a triangle. Again, we will select a point by constructing a set of barycentric coordinates. You should check that the coordinates are each between 0 and 1, and sum to 1, as required. We will not explain the formulas, except to suggest that they are chosen in a way to exactly correct the undesirable clustering and nonuniformity seen in the first scheme.

$$
\begin{aligned}
r &= rand; \\
s &= rand; \\
\xi_a &= 1.0 - \sqrt{s}; \\
\xi_b &= (1.0 - r) * \sqrt{s}; \\
\xi_c &= r * \sqrt{s}; \\
P &= \xi_a * a + \xi_b * b + \xi_c * c;
\end{aligned}
$$

Program #3 might be outlined as:

```
Repeat the previous program, but now use scheme #2 to generate sample points.
```

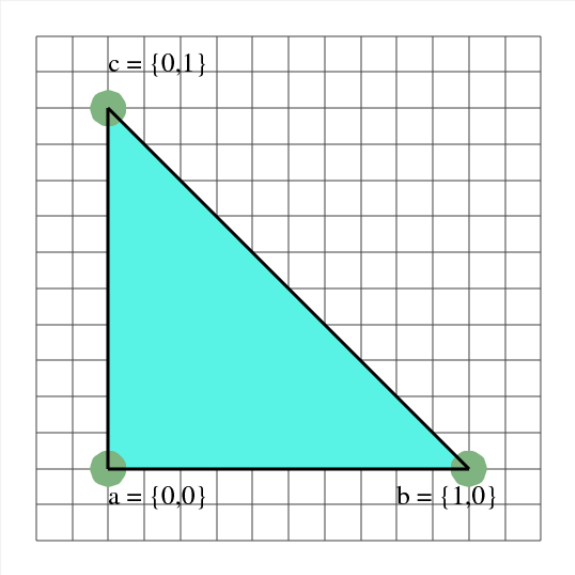

Figure 3: The unit triangle or reference triangle **Tref**.

Use the same **T** and **N** as in the previous run.

Are the values **N1, N2, N3** roughly equal? When you plot the sample points this time, does the sampling seem roughly uniform throughout the triangle?

Scheme #2 is the correct way to sample points from a triangle in a uniform manner. From now on, we will assume that this is the sampling method being used.

7 Monte Carlo Quadrature on the Unit Triangle

Now that we have some confidence in sampling points uniformly from a triangle, 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 $f(x,y)$ over the unit triangle **Tref**, using **n** points, has the form:

$$I(f(x,y),\text{Tref}) = \int_0^1 \int_0^x f(x,y)\,dy\,dx \approx MC(f(x,y),\text{Tref},n) = \frac{1}{2}\frac{1}{n}\sum_{i=1}^{n} f(x_i,y_i)$$

where each (x_i,y_i) is generated by uniform sampling of the unit triangle. The factor of $\frac{1}{n}$ averages the function values, and the factor of $\frac{1}{2}$ accounts for the area of the unit triangle. In particular, this guarantees that if $f(x,y)=1$, our estimate will be equal to $\frac{1}{2}$, the area of the triangle.

Over the unit triangle {{1,0}, {0,1}, {0,0}}, there is an exact formula for the integral of any monomial $f(x,y)=x^p y^q$, with $0 \le p$, and $0 \le q$:

$$I(x^p y^q,\text{Tref}) = \int_0^1 \int_0^x x^p\, y^q\,dy\,dx = \frac{p!\,q!}{(p+q+2)!}$$

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

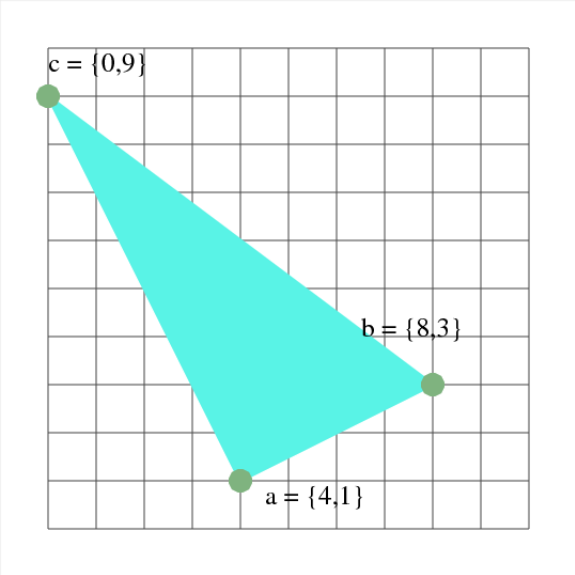

Figure 4: Example triangle #1, "Tex1".

8 Program #4: Monte Carlo on the Unit Triangle

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

Program #4 might be outlined as:

```
Read the powers p and q;
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;
Compute Q, the Monte Carlo estimate of the integral;
Print p, q, n, I, Q, and abs (I-Q).
```

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

Now fix **(p,q)**=(4,2), 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?

9 Monte Carlo on a General Triangle

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

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

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

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

$$I(f(x,y),\Delta) = \int_\Delta f(x,y)\,dydx \approx MC(f(x,y),\Delta,n) = \frac{Area(\Delta)}{n} \sum_{i=1}^{n} f(x_i,y_i)$$

Table 1: Values of some test integrals over triangle **Tex1**

(p,q)	$\int x^p y^q dxdy$
(0,0)	20.00
(1,0)	80.00
(0,1)	86.66
(2,0)	373.33
(1.1)	306.66
(0,2)	433.33
(2,1)	1333.33
(2,2)	5294.22

This is why, when we were working in the unit triangle, we multiplied the quadrature sum by $\frac{1}{2}$. Thus, to use the Monte Carlo procedure to estimate an integral on a general triangle only requires using the vertex information in the sampling formula, and computing the area to use as a factor when the estimate is computed. We have already encountered a formula for computing the area of a triangle in the lab on triangles.

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

10 Program #5: Monte Carlo on the General Triangle

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

Program #5 might be outlined as:

```
Read the triangle T;
Read the powers p and q;
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 triangle #1 or **"Tex1"**, whose definition is

```
{ { 4, 1 },
  { 8, 3 },
  { 0, 9 } }
```

Do a "sanity check" using **N**=10 and **(p,q)**=(0,0). You should get the result **Q**=20 exactly.

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

Try to approximate some of these values with your program.