Computational Geometry Lab:
MONTE CARLO OVER A TRIANGULATION

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

August 28, 2018

1 Introduction

This lab continues the study of Computational Geometry. We have just considered the Monte Carlo method for estimating an integral over a single triangle. We now take the next step, to estimate an integral over a triangulation.

From our previous work, we can expect that if we wish to estimate an integral over a region $\mathcal{R}$, we must first produce a triangulation τ. In our previous work with quadrature rules, we immediately moved to the problem of estimating integrals over individual triangles T_i and then summing the result. We could essentially mimic this process with the Monte Carlo method, that is, for each triangle, use a sampling method to estimate the integral there, and then add up the results.

However, especially if the triangulation includes triangles of a wide range of areas, this will not be the most efficient method. A more suitable approach essentially estimates the integral over the full triangulation. Each time we pick a random number r, its value will tell us which triangle T_i we should go to, and where to find the selected point within that triangle.

2 Monte Carlo Quadrature over each Triangle of a Triangulation

Now we are finally ready to look at the big picture, which is: how can we estimate the integral of a function $f(x, y)$ over a region $\mathcal{R}$? We know that the first step is to triangulate the region. And since the integral over the sum of the triangles is the sum of the integrals over the triangles, we could simply apply the Monte Carlo procedure to each triangle separately, and add the results.

While this seems straightforward, we should also consider variations to the problem or our desires that might suggest alternative procedures.

To begin with, it may be the case that we have a limited budget of point evaluations that we can carry out. In that case, we might find it advantageous to select some triangles in the triangulation where we will carry out more evaluations, based on our knowledge of the function's behavior.

It may also be the case that the triangulation we are working with is itself nonuniform - that is, some triangles may be much smaller than others. Then, what might seem a sensible uniform policy of using the same number of evaluation points in each triangle amounts to an overall *nonuniform* policy when we look at how the full set of sample points was used over the entire triangulation. Nonuniform triangulations are often used in response to known singularities in the data, sharp or reentrant corners in the boundary, or in cases where it is possible to estimate the approximation error being made over each triangle.

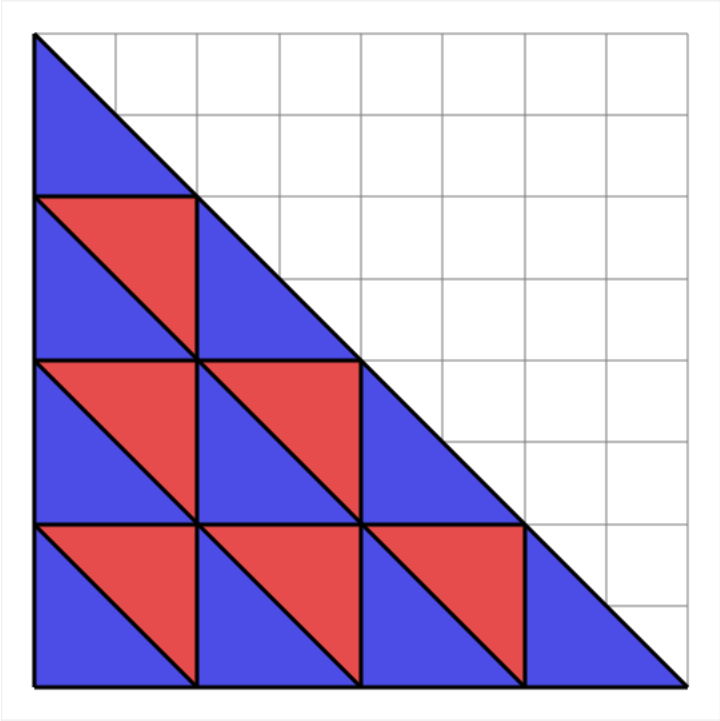

Figure 1: tau_1, A Uniform Triangulation.

3 Examples of Uniform and Nonuniform Triangulations

To illustrate the treatment of nonuniformity, we will consider two triangulations of the unit triangle.

Our uniform triangulation, tau_1, is described by:

```
nodes = {
  {0.00, 0.00}, {0.25, 0.00}, {0.50, 0.00}, {0.75, 0.00}, {1.00, 0.00}
  {0.00, 0.25}, {0.25, 0.25}, {0.50, 0.25}, {0.75, 0.25},
  {0.00, 0.50}, {0.25, 0.50}, {0.50, 0.50},
  {0.00, 0.75}, {0.25, 0.75},
  {0.00, 1.00}
}
triangles = {
  {1,2,6}, (2,3,7}, {3,4,8}, {4,5,9}, {2,7,6}, {3,8,7}, {4,9,8},
  {6,7.10}, {7,8,11}, {8,9,12}, {7,11,10}, {8,12,11},
  {10,11,13}, {11,12,14}, {13,14,15}
}
```

The nonuniform triangulation, tau_2, has the description:

```
nodes = {
  {0.00000, 0.00000}, {1.00000, 0.00000}, {0.00000, 0.50000}, {0.50000, 0.50000},
  {0.00000, 0.75000}, {0.25000, 0.75000}, {0.00000, 0.87500}, {0.12500, 0.87500},
  {0.00000, 0.93750}, {0.62500, 0.93750}, {0.00000, 0.96875}, {0.03125, 0.96875},
  {0.00000, 1.00000}
}
```

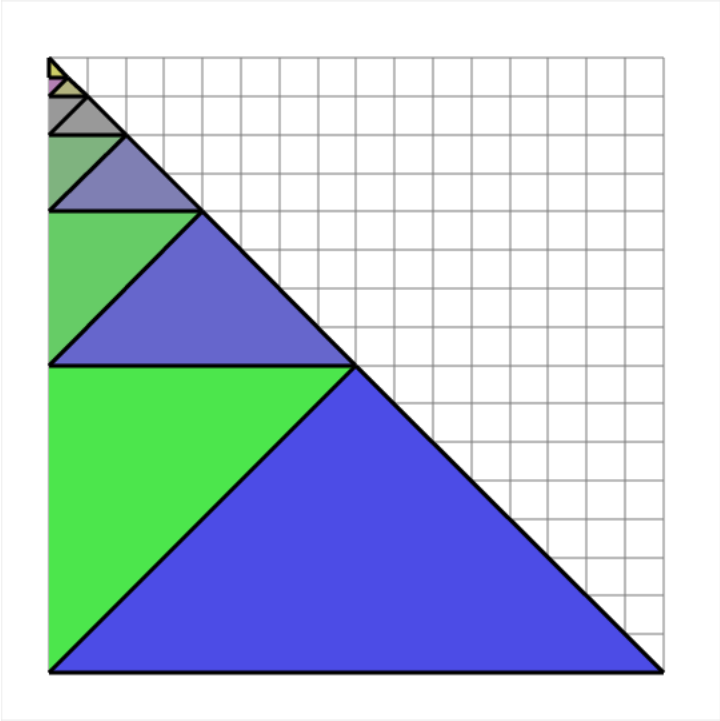

Figure 2: tau_2, A Nonuniform Triangulation

```
triangles = {
  {1,2,4}, (1,4,3}, {3,4,6}, {3,6,5}, {5,6,8}, {5,8,7},
  {7,8,10}, {7,10,9}, {9,10,12}, {9,12,11}, {11,12,13}
}
```

We will use these triangulations in the program exercises.

4 Program #1: Monte Carlo Quadrature over each Triangle of a Triangulation

Write a program which approximates the integral of $f(x) = x^p\, y^q$ over a triangulation of the unit triangle **Tref**, by applying the Monte Carlo algorithm with N points to each triangle and summing the result.

Since you've already written a program that applies the Monte Carlo method to a single triangle, your new program can simply call that program repeatedly, each time passing it a new triangle, and receiving in return the integral estimate over that triangle. By summing the returned results, your new program produces the integral estimate for the triangulation.

Your program should:

- read the powers **p** and **q**;
- compute $I(f(x), Tref)$, the exact integral;
- read the random number seed;
- read the number of sample points N;

- read the number of triangles in the triangulation
- initialize Q to zero
- read each triangle, integrating over that triangle using N points, add the result to Q
- print the values of Q, I, and abs(Q-I).

Choose $(p, q) = (2, 3)$, and let $N = 1000$. Run your program using triangulation τ_1 and τ_2. Does one triangulation give a better estimate?

Repeat the exercise with $N = 100,000$. Is the same triangulation still better?

5 Monte Carlo Quadrature over each Triangle of a Triangulation

The approach of applying the MC method to each triangle individually is suitable for situations in which the triangulation is relatively uniform, that is, if the individual triangles are all roughly the same size. However, when the triangles vary significantly in size, we will be expending more work in some parts of the region than others. This is obvious if you plot the sample points.

As a matter of efficiency, we would prefer that the sample points be spread out evenly over the entire triangulated region. The accuracy of our estimate is governed by the point density over the triangles; when this density varies, the accuracy is governed by the **worst** value, that is, the relatively low density in the largest trinagle. In effect, the higher density in the other triangles is wasted.

So in such a situation, it is best to step back from the individual triangles and work at the level of the full triangulation. All we want to do is to uniformaly sample the area of the triangulation. This is easy to do in a systematic way. Let Area(T_i) be the area of triangle T_i, and Area(τ) the area of the triangulation. Now define

$$\text{CumArea}(T_i) = \frac{\sum_{j=1}^{i} \text{Area}(T_i)}{\text{Area}(\tau)}$$

and for convenience define CumArea(T_0) = 0. Then the values in CumArea are an increasing sequence whose first entry is 0 and whose last entry is 1.

Now suppose we pick a random number r and find the triangle index i so that

$$\text{CumArea}(T_{i-1}) \leq r \leq \text{CumArea}(T_i)$$

The CumArea vector gives us a way to select a triangle from the triangulation in a way that is uniform with respect to area. If we then choose a point at random from triangle T_i, we now have a procedure for selecting points in a way that is uniform over the entire triangulation. In other words, a plot of the selected points should show no unusual concentration of values, whether or not the individual triangles of the triangulation are uniform in size or vastly different.

6 Program #2: Monte Carlo Quadrature over a Triangulation

Write a program which carries out the scheme for uniform sampling over a nonuniform triangulation, in order to approximate the integral of $f(x) = x^p\, y^q$ over a triangulation of the unit triangle **Tref**,

Your program should:

- read the powers **p** and **q**;
- read the random number seed;
- read the number of sample points N;

- read the number of triangles in the triangulation
- read all the triangles
- compute the cumulative area vector
- initialize Q to zero
- apply the Monte Carlo method to the entire triangulation, using the cumulative area vector to select a triangle
- print the final value of Q as the integral estimate

Notice in this case that the variable N now counts the total number of sample points. So to make a fair comparison with the previous program, this program should use a value of N that is about 16 times the value used for that program, since the uniform triangulation τ_1 used 16 triangles.

Choose $(p, q) = (2, 3)$, and let $N = 16000$. Run your program using triangulation τ_1 and τ_2. You should expect that the errors for both triangulations should be about the same.

Repeat the exercise with $N = 1,600,000$.

Your results should indicate that, by using a cumulative area vector, you can do uniform sampling and get better convergence, even if your underlying triangulation is nonuniform.

7 A Face-Off Between Monte Carlo and Quadrature Rules

It's time to look briefly at the question of the difference between estimating an integral by using quadrature rules or the Monte Carlo method. We will make a couple simple comparisons using a cost/benefit analysis. Our cost is N, the number of function evaluations. Our benefit is a low error. Since it's easy to control N, our analysis will ask simply, for a given triangulation τ, and a value of N, what is the comparison between the errors made by the two methods?

Recall that the power of quadrature rules partly lies in their exactness. If a function is polynomial, or well approximated by a polynomial, then a quadrature rule approach is likely to get a very good estimate quickly.

The Monte Carlo method has advantages of its own, but here we will concentrate on the fact that it essentially works about the same even when the function we are integrating has a discontinuous derivative, a jump, a singularity, or is highly oscillatory. It just doesn't care!

8 Program #3: Monte Carlo vs Quadrature Rules

Consider the following integrands:

$$\begin{aligned} f_1(x,y) &= \sqrt{|x-y|} \\ f_2(x,y) &= 1, \text{if} (4x-1)^2 + (4y-1)^2 \leq 1 \\ &= 0, \text{otherwise} \\ f_3(x,y) &= e^{2x+y} \\ f_4(x,y) &= \sin 8xy \end{aligned}$$

Use the uniform triangulation τ_1 with 16 triangles. Try the quadrature rules of order 1, 3, 4, 6 and 7 on each function and compare this to a Monte Carlo rule using the equivalent number of function evaluations. In the words, the last Monte Carlo calculation should use just 112 points.

With such a low number of points, Monte Carlo will have very poor accuracy. So the thing to observe is when the quadrature rule does well or not. If we move to a finer triangulation, all the Monte Carlo results will get better, but the bad quadrature results will become terrible.