Monte Carlo Estimation

Recall the average value of a function on $[a,b]$:

$$\text{Avg Value of } f(x) = \frac{1}{b-a}\int_a^b f(x)\,dx$$

So

$$\int_a^b f(x)\,dx = (b-a)(\text{Avg value of } f(x))$$

This suggests that one way to estimate $\int_a^b f(x)\,dx$ is by computing an estimate of the average value of $f(x)$ on $[a,b]$ and multiplying this by $(b-a)$.

One way to estimate the average value of $f(x)$ on $[a,b]$ is by

$$\frac{1}{n}\sum_{i=1}^{n} f(x_i) \approx \text{avg. value}$$

when the x_i are uniformly distributed in $[a,b]$. If the x_i are chosen at random, we have Monte Carlo Integration.

The name of this approach pays homage to a part of Monaco along the French Riviera known for gambling casinos.

So - to compute an estimate of $I = \int_a^b f(x)\,dx$

1. Generate n uniformly distributed random numbers in $[a,b]$
2. Compute $\text{Avg} = \frac{1}{n}\sum_{i=1}^{n} f(x_i)$
3. Compute $I = (b-a)(\text{Avg})$

This approach is not often used for integrals in one variable as other quadrature methods can often obtain better estimates with less computation (remember - function evaluations are expensive!)

It becomes more useful in higher dimensions, especially when the domain of integration is irregular.

Ex $\int_0^2 \int_{-1}^1 \int_0^3 (x^2y + z^2)\, dx\, dy\, dz$

```
n = 100,000;
X = 3*rand(n,1); y = 2*rand(n,1)-1; z = 2*rand(n,1);
V = (2-0)*(1-(-1))*(3-0);        # V = 12
I = V * sum(f(x,y,z))/n;
```

where `f = @(x,y,z) (x .^ 2) .*y + z .^ 2`

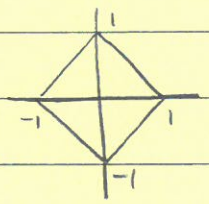
Gives 15.989, 15.969, 15.975, 16.021, 16.143, 15.964 in six trials. Exact value is 16.0.

In general, $I = \int_\Omega f(\vec{x})\, d\vec{x}$ is an integral over an n-dimensional region Ω. To estimate this, we use n random coordinates $\vec{x}_i$ uniformly distributed over Ω and compute

$$I \approx \frac{\text{measure of } \Omega}{n} \sum_{i=1}^{n} f(\vec{x}_i)$$

Monte Carlo Estimation

Ex Compute $\iint_\Omega e^{-x^2-y^2}\,dx\,dy$ where Ω is the interior of the diamond



Measure of Ω is 2 (square with sides of length $\sqrt{2}$)

Using the arrays x_i and y_i generated Computer Problem 10-1 #7b we can do

$$I \approx \frac{2}{n}\sum_{i=1}^{n} e^{-x_i^2-y_i^2}$$

Octave Code:

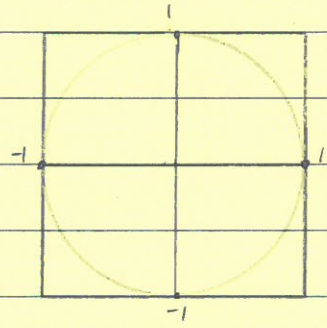
```
f = @(x,y) exp(-x.^2 - y.^2);
n = 100,000;
x = 2*rand(n,1) - 1;
y = 2*rand(n,1) - 1;
d = (x+y > -1 & x+y < 1 & x-y > -1 & x-y < 1);
I = 2 * sum(f(x(d), y(d))) / sum(d)
```

measure of domain — (sum(d) is the number of random numbers in the domain Ω)

yields 1.4657, 1.4636, 1.4637, 1.4618, 1.4651 in 5 trials

Monte Carlo Estimation

As another example we will estimate π using Monte Carlo sampling.



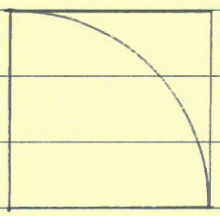
Consider the unit circle and the bounding box. The area of the circle is $A = \pi(1)^2 = \pi$ while the area of the box is $2 \times 2 = 4$.

Suppose we generate n random points uniformly distributed in the box $-1 < x, y < 1$ and then count the number of them that satisfy $\sqrt{x^2+y^2} < 1$. Then

$$\frac{\#\text{ points in circle}}{\#\text{ points generated}} \cong \frac{\text{area of circle}}{\text{area of square}} = \frac{\pi}{4}$$

So $$\pi \approx (4)\left(\frac{\#\text{ points where } x^2+y^2<1}{\#\text{ points}}\right)$$

Usually we just use the first quadrant so we are generating random points in $0 < x, y < 1$. The ratio of areas is still the same



So

$$\pi = (4)\left(\frac{\#\text{ points for which } x^2+y^2<1}{\#\text{ points generated}}\right)$$

In Octave

```
n = 100,000 ;  x = rand(n,1) ; y = rand(n,1);
d = x.^2 + y.^2 <1 ;
est = 4 * sum(d) / n ;
```

GORDON COLLEGE

MATHEMATICS AND COMPUTER SCIENCE

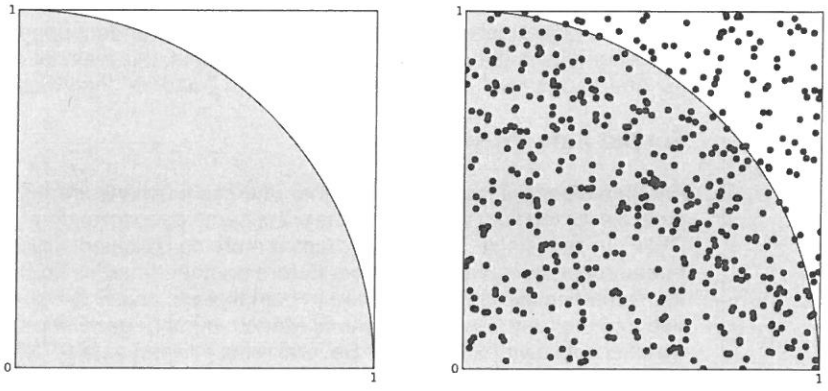
Parallel Computing with LittleFe

Model Problem: Estimating π via Monte Carlo Simulation

Problem overview

Let's begin by learning about a simple model problem. This problem was selected because it is embarrassingly parallel, meaning that nearly all the computation it requires can separated into segments of work that can be done completely independently of one another. While many important problems do not have this property, some do, and it gives us a good way to focus on the various types of parallelism.

Suppose we want to compute an estimate of π, the ratio of the circumference of a circle to its diameter. As you probably know, the area of a circle of radius r is computed by $A = \pi r^2$. A circle with $r = 1$ is called the *unit circle* and has area π, so the area of one fourth of the unit circle is just $\pi/4$. Below on the left we see a unit square, that is a square with sides of length one, with an embedded quarter unit circle. On the right is the same figure with 500 random points plotted within the square.



The ratio of the number of points inside the quarter circle to the total number of points in the square should ideally be the same as the ratio of the area of the quarter circle to the area of the square. This would mean

$$\pi \approx 4 \text{ (number of points in circle) / (number of points in square)}$$

which gives us a reasonable way to estimate π. The more random points we generate, the better our estimate will be. Generating random points in the unit square is easy; many pseudorandom number generators (PRNGs) return values in the interval [0,1) which is just what we need. The point (x,y) will be inside the unit circle if $x^2+y^2 < 1^2$. Pseudocode for the algorithm is

```
count = 0
for i = 0 to number_of_samples - 1 do
    x = random value from [0,1)
    y = random value from [0,1)
    if x * x + y * y < 1 then
        count = count + 1
endfor
estimate_of_pi = 4 * count / number_of_samples
```

As already noted, the more points we generate, the better our estimate will be. This is a natural place to exploit parallelism since the generation of each point is independent of the generation of all other points. We can create separate tasks that each compute a number of points and count the number within the circle, then add all of these together before estimating π.