Section: Monte Carlo Method

Monte Carlo method is a class of algorithms that relys on repeated random sampling to obtain numerical results. They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to use other approaches. In mathematics, it can be used to solve integration, simulation, optimization, inverse problems, etc.

[1] Monte Carlo Integration

Monte Carlo integration is a technique for numerical integration using random numbers. This method is particularly useful for higher-dimensional integrals.

[1.1] A simple illustration for 1D Monte Carlo integration

Let's recall Mean value Theorem for definite integrals:

Let $f : [a, b] \to \mathbb{R}$ be a continuous function. Then there exists c in (a, b) such that

$$f(c) = \frac{1}{b-a} \int_a^b f(x)\, dx.$$

In fact, $f(c)$ is the average value of the function f on the interval $[a, b]$. So naturally, we can rewrite the above formula as

$$\int_a^b f(x)\, dx = f(c)(b-a).$$

That is, the definite integration of a function f equals to its **average value** multiply by the **length of the interval**.

So, suppose we know the average value of a function on a given interval, we can then evaluate its definite integral based on the above formula. How to get the average value? A simple "guess" is to evaluate the average of function values at N randomly choosen points. Ideally, as N gets bigger, the average should get closer to the exact average.

[1.2] General Monte Carlo integration

Consider a function $f(x) : \Omega \to \mathbb{R}$ defined on $\Omega \subset \mathbb{R}^m$. We wish to calculate

$$I := \int_\Omega f(x)dx.$$

Let $V := \int_\Omega 1dx$ be the volume of Ω, and $x_1, x_2, \cdots, x_n \in \Omega$ are n random points chosen uniformly in Ω.

Then

$$I = \lim_{n\to\infty} V\frac{1}{n}\sum_{i=1}^{n} f(x_i).$$

Therefore, I can be aprroximated by

$$I \approx V\frac{1}{n}\sum_{i=1}^{n} f(x_i).$$

This can be proved by the law of large numbers.

References : Monte Carlo integration, Monte Carlo method

Example 1: Estimating π

We can use Monte Carlo Method to estimate π. Or in other words, estimate the area of the unit circle.

Consider the function

$$f(x) = \begin{cases} 1, & \text{if } \|x\|_2^2 < 1 \\ 0, & \text{otherwise} \end{cases}$$

and $\Omega = [0,1] \times [0,1]$ with $V = 1$.

Consider n random points $\{x_i\}_{i=1}^n$ in Ω. Then

$$\frac{\pi}{4} = I \approx \frac{1}{n} \sum_{i=1}^{n} f(x_i).$$

So

$$\pi \approx 4 \frac{1}{n} \sum_{i=1}^{n} f(x_i).$$

Or in other words,

$$\pi \approx 4 \times \frac{\text{\# of points that generated inside the quarter circle}}{\text{\# of total generated points in } [0,1] \times [0,1]}.$$

We now apply this method with different size of n.

```julia
# We need the LinearAlgebra Package to calculate norm
using LinearAlgebra
```

```julia
# We use rand(2) to construct a random point in [0,1]x[0,1]
x1 = rand(2)
```

```
2-element Array{Float64,1}:
 0.813279180270285
 0.18543267467098112
```

```julia
# We define a function for estimating pi with different size of n
function MonteCarloPi(n)
    # pi_mc: the number of points that is inside the quarter circle
    pi_mc = 0;

    # Run the loop for i=1...n
    for ii=1:n
        # Uniformly choose a random point in [0,1]x[0,1]
        x1 = rand(2);

        # Determine if the point lies in the quarter circle
        if(norm(x1,2)<1);
            pi_mc = pi_mc+1;
        end
    end

    #return the approximation of pi
    return 4*pi_mc/n
end
```

```
MonteCarloPi (generic function with 1 method)
```

```julia
# Now we try this method with different size of n and print them out
println("n\t","|\t","Estimation of pi");
println("--------+---------------")
for i=2:8
    n=10^i;
    println("10^",i,"\t|","\t",MonteCarloPi(n))
end
```

n	Estimation of pi
10^2	3.04
10^3	3.236
10^4	3.1516
10^5	3.12812
10^6	3.1415
10^7	3.1404532
10^8	3.14155884

[1.3] Quasi-Monte Carlo method

In example 1, we choose random points in $[0, 1]x[0, 1]$. But there might be the case that these randomly picked points are clustered in some region that results in a bad estimation of the average. So we might want to choose random points in the region, but not that random.

For example, we divide the region $[0, 1]x[0, 1]$ into 10 stripe, denoted by $[\frac{i-1}{10}, \frac{i}{10}]x[0, 1]$, $i = 1, 2, \cdots, 9$. Then every time we pick one random point from each of the stripe domain. In this way. we are sure that the points will distribute uniformly in the whole region, in the sense that we have the same number of points in each of the strip region.

Example 2

Continued by Example 1, for each $k = 1, 2, 3, \ldots, 10$, take a random point $y_k = [y_{k1}, y_{k2}]^\top$, consider

$$\hat{y_{k1}} = \frac{1}{10}(y_{k1} + (k-1)),$$
$$x_i^{(k)} = [\hat{y_{k1}}, y_{k2}]^\top,$$

and we have points $x_i^{(1)}, x_i^{(2)}, \cdots, x_i^{(10)}$.

In short, we devide $[0, 1] \times [0, 1]$ into 10 rectangles with the same area and choose one point randomly in each region, so we choose 10 points each time.

By doing this for $i = 1, \cdots, n$, we get $10n$ points in total.

Now apply Monte Carlo method to these $10n$ points.

```julia
function MonteCarloPi2(n)
    # pi_mc: the number of points that is inside the quarter circle
    pi_mc = 0;

    # Run the loop for 1...n
    for nn=1:n
        for ii=1:10
            # Uniformly choose a random point in [0,1]x[0,1]
            x1 = rand(2);

            # Do the transformation to the point
            x1[1] = (x1[1] + (ii-1))*0.1;

            # Determine if the point lies in the quarter circle
            if(norm(x1,2)<1);
                pi_mc = pi_mc+1;
            end
        end
    end

    # return the approximation of pi
    return (4*pi_mc/(10*n))
end
```

```
MonteCarloPi2 (generic function with 1 method)
```

```julia
# Now we try this method with different size of n and print them out
println("n\t","|\t","Estimation of pi");
println("--------+---------------")
for i=2:7
    n=10^i;
    println("10^",i,"\t|","\t",MonteCarloPi2(n))
end
```

n	Estimation of pi
10^2	3.06
10^3	3.1684
10^4	3.14796
10^5	3.140652
10^6	3.1418884
10^7	3.1417912

Example 3

Continued by Example 2. Now what if we do this to both x and y coordinate?

In other words, we devide $[0,1]\times[0,1]$ into 100 squares with the same area and choose 1 point in each region, then we get 100 points each time.

So in total, there are $100n$ points. We do Monte Carlo method to these points.

```julia
function MonteCarloPi3(n)
    # pi_mc: the number of points that is inside the quarter circle
    pi_mc = 0;

    # Run the loop for 1...n
    for nn=1:n
        for ii=1:10
            for jj=1:10
            # Uniformly choose a random point in [0,1]x[0,1]
            x1 = rand(2);

            # Do the transformation to the point
            x1[1] = (x1[1] + (ii-1))*0.1;
            x1[2] = (x1[2] + (jj-1))*0.1;

            # Determine if the point lies in the quarter circle
            if(norm(x1,2)<1);
                pi_mc = pi_mc+1;
            end
            end
        end
    end

    # return the approximation of pi
    return (4*pi_mc/(100*n))
end
```

```
MonteCarloPi3 (generic function with 1 method)
```

```julia
# Now we try this method with different size of n and print them out
println("n\t","|\t","Estimation of pi");
println("--------+---------------")
for i=2:6
    n=10^i;
    println("10^",i,"\t|","\t",MonteCarloPi3(n))
end
```

n	Estimation of pi
10^2	3.1436
10^3	3.13952
10^4	3.142208
10^5	3.1415808
10^6	3.14161188

Summary

1. We have shown how one can use Monte Carlo method to approximate π in Example 1. We see that as n becomes large, the solution seems to aprroximate π better. Here are some questions:
    - What is the convergence rate of the method? Is it fast/slow?
    - How can we estimate the error between the aprroximated solution and the real solution for each n?
2. In Example 2 and 3, we choose the points differently other than choosing a random point uniformly.
    - How does this affect the result?
    - Is this a good choice of points? Why?